A multiparty multi-session logic

Laura Bocchi[1], Romain Demangeon[2], and Nobuko Yoshida[3]
[1]University of Leicester, [2]Queen Mary, University of London, [3]Imperial College London

Abstract. Recent work on the enhancement of typing techniques for multiparty sessions with logical annotations enables, not only the validation of structural properties of the conversations and on the sorts of the messages, but also properties on the actual values exchanged. However, a specification and verification of mutual effects of multiple cross-session interactions are still an open problem. We introduce a multiparty logical system with virtual states that enables the tractable specification and validation of fine-grained inter-session correctness properties of processes participating in several interleaved sessions. We present a sound and relative complete static verification method, and justify its expressiveness by giving a sound and complete embedding into Hennessy-Milner logic.

1 Introduction

In extensively distributed computing environments, application scenarios often centre around structured conversations among multiple distributed participants; correctness properties often depend on the state of individual participants and span over multiple conversations and applications. A fundamental challenge is to establish an effective specification and verification method to ensure safety in distributed software where the notion of state plays a key role. A promising direction is the logical elaboration of types for programming languages [15]. Types offer a stable linkage between the fundamental dynamics of programs and their mathematical abstractions, serving as a highly effective basis for safety assurance. In the context of process algebras, approaches like [5, 11, 17] allow tractable (e.g., with respect to model checking techniques) validation of properties such as progress, session fidelity, and error freedom. Furthermore, they enable the specification of *global* properties of multiparty interactions, yet enabling modular *local* verification of each participant. The key idea is that a global specification (i.e., a global session type) is *projected* onto each endpoint, making its responsibilities explicit. When all endpoints conform to their projected specifications, the resulting conversation is guaranteed to conform to the original global specifications.

However, these approaches are confined to the specification of a *single* multiparty session and do not treat stateful specifications incorporating mutual effects of multiple sessions. This paper presents a simple but powerful extension of multiparty session specifications, by enriching the assertion language studied in [5] with capability to refer to *virtual states* local to each network principal. The resulting protocol specifications are called *multiparty stateful assertions* (*MPSA*s). Each principal in a network hence serves as unit of verification, in which the local state must conform to the virtual states of the *MPSA*s that principal implements. To see the kind of properties we are interested in, consider the following fragment of specification for the dialogue between a ticket allocation server (`S`) and its client (`C`), where the server allocates numbered tickets of increasing value to each client in consecutive, *separate* sessions:

$$\mathtt{S} \to \mathtt{C} : (y{:}\mathtt{Nat})\{y = \mathtt{S.x}\}\langle \mathtt{S.x{+}{+}} \rangle$$

The protocol between the server and each client is the single message-passing action where S sends C a message of type Nat. The description of this simple distributed application implies behavioural constraints of greater depth than the basic communication actions. The (sender-side) *predicate and effect* for the interaction step, $\{y = \mathtt{S.x}\}\langle \mathtt{S.x{+}{+}} \rangle$, asserts that the message y sent to each client must equal the current value of S.x, a state variable x allocated to the *principal* serving as S; and that the local effect of this message send is to increment S.x. In this way, S is specified to send incremental values across *consecutive* sessions.

The behaviour described above cannot be encoded by only using the primitives in [5] for single-session specifications. In fact, (1) to ensure inter-session properties one must discipline concurrent state updates with some mechanism of lock/unlock or atomic access/update, and (2) lock/unlock and atomic access/update can only be described as properties that span over multiple sessions.

To clarify the relevance of our work, we investigate how our specification corresponds to a Hennessy-Milner Logic (HML) formula [16]. We give the embedding of the behaviour of a role in a session into a formula: if a process and its state happen to perform reductions and updates matching the ones of the specification, the required predicates will hold. For instance, the formula corresponding to the behaviour of S from the previous example on channel s is:

$$\forall y : \mathtt{Nat}, [s_{\mathtt{C}}(y)](y = \mathtt{S.x} \wedge [\mathtt{S.x{+}{+}}]\mathtt{true})$$

where $[\ell]\phi$ means "if a process and its state perform the action ℓ, the resulting pair satisfies ϕ". Communications and state updates are both treated as actions of a labelled transition system. In § 6, we explain how specifications handling several roles in several sessions can be soundly and completely embedded, through the use of an *interleaving* of formulae, exploring all the possible orders in which the actions coming from different sessions can be performed, and ensuring that predicates are always satisfied.

Contribution We present a sound and relative complete validation method for *MPSA*s, based on statically-verifiable proof rules. The most distinctive feature with respect to the framework presented in [5] is the possibility of expressing properties that span several session, by referring to the states maintained by principals. Our analysis statically checks that a network, composed of several processes associated with their own states, satisfies a specification. Principals are associated to *invariants*, asserting properties on their state that are supposed to always hold, even when several sessions are executed in parallel. We prove that our analysis is sound and complete w.r.t. to the semantical satisfaction relation induced by the two labelled transition systems for processes and specifications: the actions performed by a typable process and its specification match. In addition, we justify the relevance of the stateful logical layer of our work by embedding it into Hennessy-Milner logic with predicates [1, 5]. Appendix lists auxiliary definitions, proof, and a complex use case from [20].

2 Multiparty assertions with virtual states

In the proposed framework, applications are built as the composition of units of design called *sessions*. Each type of session is specified as a *MPSA*, that is an abstract de-

scription of the interactions of the roles of a multiparty session. A *MPSA* specifies the conditions under which interactions can be done, the constraints on the communicated values and the effects on the *virtual state*.

The syntax of *MPSA*s is given in Figure 1 and extends [5] with the declaration of one virtual state for each role in the session, and with operations on the states called *updates*. *Global assertions* ($\mathcal{G}, \mathcal{G}', \ldots$) describe a multiparty session from a global perspective; and *local assertions* ($\mathcal{L}, \mathcal{L}', \ldots$) describe it from the perspective of one role.

A ::= true | false | $e_1 = e_2$ | $\neg A$ | $A_1 \wedge A_2$ | $\exists x.A$, S ::= bool | nat | .., U ::= S | $\langle\mathcal{L}\rangle$

$\mathcal{G}$ ::=	$\mathcal{L}$ ::=
$((\mathtt{p}_1 : [\tilde{\mathtt{x}}_1 : \tilde{S}_1]\{A_1\}, \ldots, \mathtt{p}_n : [\tilde{\mathtt{x}}_n : \tilde{S}_n]\{A_n\})).\mathcal{G}$	$[\tilde{\mathtt{x}} : \tilde{S}]\{A\}.\mathcal{L}$
$\mathtt{p} \to \mathtt{q} : \{l_i(x_i : U_i)\{A_i\}\langle E_i\rangle.\mathcal{G}_i\}_{i\in I}$	$\mathtt{p}!\{l_i(x_i : U_i)\{A_i\}\langle E_i\rangle.\mathcal{L}_i\}_{i\in I}$
$\mathcal{G}_1 \mid \mathcal{G}_2$	$\mathtt{p}?\{l_i(x_i : U_i)\{A_i\}\langle E_i\rangle.\mathcal{L}_i\}_{i\in I}$
$\mu\mathtt{t}\langle y : A'\rangle(x : S)\{A\}.\mathcal{G}$	$\mu\mathtt{t}\langle y : A'\rangle(x : S)\{A\}.\mathcal{L}$
$\mathtt{t}\langle y : A'\rangle$	$\mathtt{t}\langle y : A'\rangle$
end	end

Fig. 1. Global and local MPSAs

For expressing constraints we use *predicates* ($A, A', \ldots$) with decidable evaluation. We consider here the syntax of A in Figure 1, although we may use other predicates than equality in examples. Predicates are defined on *interaction variables*, modelling the content of a message exchanged by the roles in the session (as [5]), and on *state variables*, which are variables local to one role. Whereas the value of interaction variables does not change after they have been introduced, state variables can be updated a number of times. As a consequence, a predicate involving state variables may be true or false at different times during the session. Given a predicate A, we sometimes use the closure (using existential quantifiers) of the state variables in A, denoted with $\#A$, to keep only the persistent part of A, namely the part involving interaction variables.

Global Assertions Declaration $((\mathtt{p}_1 : [\tilde{\mathtt{x}}_1 : \tilde{S}_1]\{A_1\}, \ldots, \mathtt{p}_n : [\tilde{\mathtt{x}}_n : \tilde{S}_n]\{A_n\})).\mathcal{G}$ appears only once at the outset; it declares the roles $\mathtt{p}_1, .., \mathtt{p}_n$ involved in the session, and associates each role $\mathtt{p}_i$ to the signature $[\tilde{\mathtt{x}}_i : \tilde{S}_i]$ of its virtual state and to an assertion invariant A_i constraining $\tilde{\mathtt{x}}_i$. The declaration binds $\tilde{\mathtt{x}}_i$ in A_i and in $\mathcal{G}$; in $\mathcal{G}$ we denote $\mathtt{x} \in \tilde{\mathtt{x}}_i$ with $\mathtt{p}_i.\mathtt{x}$, as different roles can have state variables with the same name.

Interaction $\mathtt{p} \to \mathtt{q} : \{l_i(x_i : U_i)\{A_i\}\langle E_i\rangle.\mathcal{G}_i\}_{i\in I}$ models a message exchange where role p sends q one of the the branch labels l_i and an interaction variable x_i, with x_i binding its occurrences in A_i, E_i, and $\mathcal{G}_i$. A_i is the predicate which needs to hold for p to select l_i, and which may constrain the values to be sent for x_i. Note that A_i is at the same time an assumption for the receiver q and a constraint for the sender p (i.e., if A_i is violated then the blame is on p). E_i is the update prescribed on the virtual states of p and q. An update is a vector of assignments of the form $\mathtt{x} := e$, where x is updated by the result of evaluating e in the current state. We assume E does not contain two assignments to the same state variable, and is an atomic action.

$\mathcal{G}_1 \mid \mathcal{G}_2$ is for parallel composition. The recursive definition is guarded and defines a recursion parameter x initially set equal to a value satisfying the initialisation predicate A', with A being an invariant predicate. end is a termination.

We denote with $var(\mathcal{G})$ the set of (interaction/state) variables and recursion parameters in $\mathcal{G}$, and with $var(A)$ the free variables of A. We write $\mathtt{p} \in \mathcal{G}$ if p is a role of $\mathcal{G}$. The

set of variables that $\mathtt{p} \in \mathcal{G}$ *knows*, written $var(\mathcal{G}) \upharpoonright \mathtt{p}$, consists of: (i) the state variables in p's signature, (ii) the interaction variables sent or received by p in $\mathcal{G}$, and (iii) the parameters of the recursive definitions $\mu \mathtt{t}\langle y : A' \rangle (x : S)\{A\}.\mathcal{G}'$ in $\mathcal{G}$ such that p knows all the free variables in initialisation A', and all free variables in A'' for all $\mathtt{t}\langle y : A'' \rangle$ in $\mathcal{G}'$ (we assume each recursion parameter known by exactly two participants). We omit true predicates, empty vectors of variables/updates, and labels of single branches.

Example 1. Consider a session with two roles, C and S: C makes an offer x to S for buying a ticket; S either accepts or refuses the offer. In the former case C spends x credits and receives a ticket, and S earns x credits. Tickets are modelled as serial numbers; they must all be increasing numbers not exceeding 1000. $\mathcal{G}_T$ below specifies this scenario:

$$
\begin{array}{rl}
\mathcal{G}_T & = (\!(\mathtt{C} : [\mathtt{credit} : \mathtt{Nat}]\{A_\mathtt{C}\}, \mathtt{S} : [\mathtt{credit} : \mathtt{Nat}, \mathtt{count} : \mathtt{Nat}]\{A_\mathtt{S}\})\!). \\
& \quad \mathtt{C} \to \mathtt{S} : (x : \mathtt{Nat})\{\mathtt{C.credit} \geqslant x\}\langle \mathtt{C.credit} := \mathtt{C.credit} - x \rangle. \\
& \quad \mathtt{S} \to \mathtt{C} : \{\mathsf{ok}(y : \mathtt{Nat})\{\mathtt{S.count} < 1000 \wedge y = \mathtt{S.count}\}\langle E_{ok} \rangle, \\
& \qquad\qquad \mathsf{ko}\langle \mathtt{C.credit} := \mathtt{C.credit} + x \rangle \,\} \\
A_\mathtt{C} & = \{\mathtt{C.credit} \geqslant 0\},\ A_\mathtt{S} = \{\mathtt{S.credit} \geqslant 0 \wedge \mathtt{S.count} \geqslant 0 \wedge \mathtt{S.count} \leqslant 1000\} \\
E_{ok} & = \mathtt{S.credit} := \mathtt{S.credit} + x, \mathtt{S.count} := \mathtt{S.count} + 1
\end{array}
$$

C has state variable `credit`, and S has state variables `credit` and `S.count` (a counter for serial numbers). The assertion invariants $A_\mathtt{C}$ and $A_\mathtt{S}$ require the credit to be non-negative and the counter to not exceed the maximum number of tickets. The first interaction requires that the offer x does not exceed C's credit, and decrements the credit by x. S selects one of the two branches by either label ok or quit. The former branch can be selected only if $\mathtt{S.count} \leqslant 1000$.

Local Assertions Each local assertion $\mathcal{L}$ (Figure 1) refers to a specific role. The declaration of one role is $[\tilde{\mathtt{x}} : \tilde{S}]\{A\}.\mathcal{L}$. Assertion $\mathtt{p}!\{l_i(x_i : U_i)\{A_i\}\langle E_i \rangle.\mathcal{L}_i\}_{i \in I}$ models an interaction where the role sends p a branch label l_i and a message x_i. A_i and E_i are the predicate and update respectively. The branching is dual. The others are as in the global assertions, except that a local assertion cannot be multi-threaded.

Given a global assertion $\mathcal{G}$, we can automatically derive the local assertions for each role $\mathtt{p} \in \mathcal{G}$ by *projection*. The projection rules are as in [5], except that state declarations and updates are now considered. The projection of a predicate A on p in $\mathcal{G}$, written $A \upharpoonright \mathtt{p}$ is defined as $\exists \tilde{x}.A$ where $\tilde{x} = var(A) \backslash (var(\mathcal{G}) \upharpoonright \mathtt{p})$ (i.e., the existential closure of the variables that p does not know). The projection of an update E on p in $\mathcal{G}$, written $E \upharpoonright \mathtt{p}$ is the update E' containing only the assignments $\mathtt{p}_j.\mathtt{x}_i := e_j$ such that $\mathtt{p}_j = \mathtt{p}$.

A detailed presentation of the rules for global assertions is not necessary to understand the results in this paper, hence we only give an illustration through Example I.3. Henceforth, in $\mathcal{G} \upharpoonright \mathtt{p}$ we shall omit the $\mathtt{p}.\_$ prefix when referring to p's state variables.

Example 2. $\mathcal{L}_\mathtt{C}$ (resp. $\mathcal{L}_\mathtt{S}$) is the projection of $\mathcal{G}_T$ from Example I.2 on C (resp. S).

$$
\begin{array}{l}
\mathcal{L}_\mathtt{C} = [\mathtt{credit} : \mathtt{Nat}]\{A_\mathtt{C}\}.\mathtt{S}\,!\,(x : \mathtt{Nat})\{\mathtt{credit} \geqslant \mathtt{x}\}\langle \mathtt{credit} := \mathtt{credit} - x \rangle.\mathcal{L}'_\mathtt{C} \\
\mathcal{L}'_\mathtt{C} = \mathtt{S}\,?\{\mathsf{ok}(y : \mathtt{Nat})\{\exists \mathtt{S.count}.\mathtt{S.count} < 1000 \wedge y = \mathtt{S.count}\}.\mathsf{end}, \\
\qquad\quad \mathsf{ko}\langle \mathtt{credit} := \mathtt{credit} + x \rangle.\mathsf{end}\} \\
\mathcal{L}_\mathtt{S} = [\mathtt{credit} : \mathtt{Nat}, \mathtt{credit} : \mathtt{Nat}]\{A_\mathtt{S}\}.\mathtt{C}\,?(x : \mathtt{Nat})\{\exists \mathtt{C.credit}.\mathtt{C.credit} \geqslant x\}.\mathcal{L}'_\mathtt{S} \\
\mathcal{L}'_\mathtt{S} = \mathtt{C}\,!\,\{\mathsf{ok}(y : \mathtt{Nat})\{\mathtt{count} < 1000 \wedge y = \mathtt{count}\}\langle \mathtt{credit} := \mathtt{credit} + x, \mathtt{count} := \mathtt{count} + 1 \rangle.\mathsf{end}, \\
\qquad\quad \mathsf{ko}.\mathsf{end}\}
\end{array}
$$

The projection of the declaration of $\mathcal{G}_T$ on a role includes only the variables/invariant for that role. The projection of the first interaction of $\mathcal{G}_T$ on sender $\mathtt{C}$ (resp. receiver $\mathtt{S}$) is a send/select (resp. a receive/branch). The predicates/updates of the projections on a role are the projections of the predicates/updates on that role.[1] The continuation is projected similarly, proceeding point-wise for each branch. Sometimes the projected predicate provides useful information, e.g., $\exists \mathtt{S.count}.\mathtt{S.count} < 1000 \wedge y = \mathtt{S.count}$ provides $\mathtt{C}$ with precondition $y < 1000$.

Well-assertedness Our theory relies on the following consistency conditions to prevent assertions in which it is logically impossible for a role to meet the specified obligations.

History sensitivity (HS) Each role $\mathtt{p}$ must know all free variables in the predicates that $\mathtt{p}$ must guarantee (as in [5]), and must have enough information to perform the prescribed updates. To perform an update, $\mathtt{p}$ must know (i) which value to assign and (ii) when. The following assertion $\mathcal{G}_{HS}$ violates (i) as in the second interaction $\mathtt{r}$ has to update $\mathtt{x}_1$ with y without knowing y, and (ii) as $\mathtt{p}$ must update $\mathtt{x}_2$ not knowing whether/when the update should be done.

$$\mathcal{G}_{HS} = \mathtt{p} \to \mathtt{q} : (y{:}\mathtt{Nat}).\, \mathtt{q} \to \mathtt{r} : \{\mathsf{ok}(w{:}\mathtt{Nat})\langle \mathtt{r.x}_1 := y, \mathtt{p.x}_2 := y\rangle, \mathsf{ko}\}$$

Invariant Stability (IS) A global assertion $\mathcal{G}$ is stable w.r.t. an invariant A if each update performed in $\mathcal{G}$ does not invalidate A. We say that $\mathcal{G}$ satisfies invariant stability if it is stable w.r.t. the invariants of its state declaration. The assertion below does not satisfy invariant stability because if, e.g., $\mathtt{p.credit} = 10$ and $\mathtt{p}$ sends value 20 for x then the update $\langle \mathtt{p.credit} := \mathtt{p.credit} - x\rangle$ will invalidate the invariant $\mathtt{credit} \geqslant 0$.

$$((\mathtt{p} : [\mathtt{credit} : \mathtt{Nat}]\{\mathtt{credit} \geqslant 0\},\ \mathtt{q} : [\_])).\, \mathtt{q} \to \mathtt{p} : (x : \mathtt{Nat})\{x > 0\}\langle \mathtt{p.credit} := \mathtt{p.credit} - x\rangle$$

Hereafter, we assume assertions to be *well-asserted*, in the sense that they satisfy HS and IS. We extend well-assertedness to local assertions. Note that if $\mathcal{G}$ is a well-asserted and $\mathcal{G} \upharpoonright \mathtt{p}$ is defined then $\mathcal{G} \upharpoonright \mathtt{p}$ is well-asserted.

3 Multiparty networks with local states

We consider networks of interactional entities called *principals* linked by a common global transport, modelled as queues. Each principal runs a *located process*, that is a process with multiparty session primitives [2, 17] (to enable rigorous representation of conversation structures) and with a *local state*.

Syntax The syntax of networks and processes is given in Figure 2 and is a refined version of the multiparty session π-calculus from [2, 9] with local states. A local state σ maps a signature $[\tilde{\mathtt{x}} : \tilde{S}]$ of typed pairwise disjoint state variables $\tilde{\mathtt{x}}$ to their sorts. We denote the signature of σ as $\mathtt{s}(\sigma)$, and use the injective function $\mathtt{id}(\sigma)$ to map each local state to an identifier.

A network can be an empty network $\varnothing$, a located process $[P]\sigma$, a parallel composition of networks $N_1 \mid N_2$, a new session name $(\nu s)N$ which binds s in N, or a queue $s : h$ where h are messages in transit through session channel s. A network is *initial* if it has no new session names and queues, otherwise it is *runtime*. We denote the free session channels in N with $\mathsf{fn}(N)$, similarly for P with $\mathsf{fn}([P]\sigma) = \mathsf{fn}(P)$.

[1] Note that by HS the projection of a predicate on the sender of an interaction is always the predicate itself.

$$
\begin{array}{lllll}
N ::= \varnothing & P ::= \mathbf{0} & & & \mid P \mid Q \\
\mid [P]\sigma & \mid \overline{u}[\mathtt{n}](y).P & & & \mid (\mu X(x).P)\langle e\rangle \\
\mid N_1 \mid N_2 & \mid u[\mathtt{i}](y).P & & & \mid X\langle e\rangle \\
\mid (\nu s)N & \mid k[\mathtt{p},\mathtt{q}]!\{e_i \mapsto l_i\langle e_i'\rangle(x_i)\langle E_i\rangle; P_i\}_{i\in I} & & & \\
\mid s : h & \mid k[\mathtt{p},\mathtt{q}]?\{l_i(x_i)\langle E_i\rangle.P_i\}_{i\in I} & & &
\end{array}
$$

$$
\begin{array}{llll}
\sigma ::= [\tilde{\mathtt{x}} : \tilde{S}] \mapsto \tilde{S} & k ::= y \mid s & v ::= \mathtt{n} \mid s[\mathtt{p}] & E ::= \varnothing \\
h ::= \varnothing \mid (\mathtt{p},\mathtt{q}, l\langle v\rangle)\cdot h & u ::= y \mid a & e ::= v \mid e\ \mathtt{op}\ e & \mid E; \mathtt{x} := e
\end{array}
$$

$x, y, \ldots$ interaction variables $\quad \mathtt{x}, \mathtt{y}, \ldots$ state variables $\quad X, Y, \ldots$ process variables
$a, b, \ldots$ shared name $\quad s, s', \ldots$ session name $\quad \mathtt{n}, \mathtt{n}', \ldots$ constants

Fig. 2. Syntax of netowrks and processes

A process can be an idle process $\mathbf{0}$, a session request/accept, a guarded command [14][2], a branching, a parallel composition of processes, a recursive definition and invocation. Session request $\overline{u}[\mathtt{n}](y).P$ multicasts a request to each session accept process $u[\mathtt{i}](y).P$ (with $i \in \{2, .., n\}$) by synchronisation through a shared name u and continuing as P. Guarded command and branching processes represent communications through an established session k. Guarded command $k[\mathtt{p},\mathtt{q}]!\{e_i \mapsto l_i\langle e_i'\rangle(x_i)\langle E_i\rangle; P_i\}_{i\in I}$ acts as role $\mathtt{p}$ in session k and sends role $\mathtt{q}$ one of the labels l_i. The choice of the label is determined by boolean expressions e_i, assuming $\vee_{i\in I} e_i = \mathsf{true}$ and $i \neq j$ implies $e_i \wedge e_j = \mathsf{false}$. Each label l_i is sent with the corresponding expression e_i' which specifies the value for x_i, assuming e_i' and x_i have the same type. Branching $k[\mathtt{p},\mathtt{q}]?\{l_i(x_i)\langle E_i\rangle.P_i\}_{i\in I}$ plays role $\mathtt{q}$ in session k and is ready to receive from $\mathtt{p}$ one of the labels l_i and a value for the corresponding x_i, then behaves as P_i after instantiating x_i with the received value. In guarded command (resp. branching), the local state of the sender (resp. receiver) is updated according to update E_i; in both processes each x_i binds its occurrences in P_i and E_i.

Example 3. Processes $P_{\mathtt{S}}$ and $P_{\mathtt{C}}$ implement $\mathcal{L}_{\mathtt{S}}$ and $\mathcal{L}_{\mathtt{C}}$, respectively, from Example I.3.

$$
\begin{array}{l}
P_{\mathtt{S}} = a[\mathtt{2}](z).z[\mathtt{C},\mathtt{S}]?(x); P_{\mathtt{S}}' \qquad E_{ok} = \mathtt{count} := \mathtt{count} + 1, \mathtt{credit} := \mathtt{credit} + x \\
P_{\mathtt{S}}' = z[\mathtt{S},\mathtt{C}]!\{\{\mathtt{count} < 1000 \wedge x \geqslant 10\} \mapsto \mathsf{ok}\langle \mathtt{count}\rangle(y)\langle E_{ok}\rangle.\mathbf{0}, \\
\qquad\qquad \{\mathtt{count} \geqslant 1000 \vee x < 10\} \mapsto \mathsf{ko}.\mathbf{0}\} \\
P_{\mathtt{C}} = \overline{a}[\mathtt{2}](w).w[\mathtt{C},\mathtt{S}]!\langle 8\rangle(x)\langle \mathtt{credit} := \mathtt{credit} - x\rangle; P_{\mathtt{C}}' \\
P_{\mathtt{C}}' = w[\mathtt{S},\mathtt{C}]?\{\mathsf{ok}(y).\mathbf{0},\ \mathsf{ko}\langle \mathtt{credit} := \mathtt{credit} + x\rangle.\mathbf{0}\}
\end{array}
$$

We let $\mathtt{C} = \mathtt{1}$ and $\mathtt{S} = \mathtt{2}$. $P_{\mathtt{S}}$ accepts a request to participate to a session specified by $\mathcal{G}_T$ (assuming a has type $\mathcal{G}_T$) on channel z as role $\mathtt{2}$. In the established session z, the principal receives an offer x from the co-party. It follows a guarded command with two cases; if $\mathtt{count}$ has not reached its maximum value for serial numbers and the offer is greater than 10 then the first branch (ok) is taken and $\mathtt{count}$ is sent as y, otherwise the second branch (ko) is taken. Dually, $P_{\mathtt{C}}$ sends a request to participate to one instance of session $\mathcal{G}_T$ as the role $\mathtt{1}$. A principal may repeatedly execute a processes using recursion, or run concurrent instances of the same type of session (e.g., $[P_{\mathtt{S}} \mid P_{\mathtt{S}}]_\sigma$) or different types of session (e.g., $[P_{\mathtt{S}} \mid P_{\mathtt{C}}]_\sigma$) as discussed in Example I.5.

[2] This construct can be implemented using selection, if-then-else and lock-unlock. Although our theory is applicable to these primitives, we choose to make these low-level steps atomic for minimising the syntax.

Operational semantics The LTS is generated from the rules in Figure **??** using the following labels: $\ell ::= \overline{a}[\mathtt{n}]\langle s\rangle \mid a[\mathtt{i}]\langle s\rangle \mid s[\mathtt{p},\mathtt{q}]!l\langle v\rangle \mid s[\mathtt{p},\mathtt{q}]?l\langle v\rangle \mid \tau$. We denote with $\sigma\,\mathtt{after}\,E$ the state σ after the update E_i. We write $e \downarrow v$ for a closed expression e when it evaluates to v.

$$[\overline{a}[\mathtt{n}](y).P]\sigma \xrightarrow{\overline{a}[\mathtt{n}]\langle s\rangle} [P[s/y]]\sigma \quad [a[\mathtt{i}](y).P]\sigma \xrightarrow{a[i]\langle s\rangle} [P[s/y]]\sigma \quad (s \notin \mathsf{fn}(P))$$

$$[s[\mathtt{p},\mathtt{q}]!\{e_i \mapsto l_i\langle e_i'\rangle(x_i)\langle E_i\rangle; P_i\}_{i\in I}]\sigma \xrightarrow{s[\mathtt{p},\mathtt{q}]!l_j\langle v\rangle} [P[v/x_j]]\sigma'$$
$$(j \in I \quad \sigma \models e_j' \downarrow v \quad \sigma \models e_j \quad \sigma' = \sigma\,\mathtt{after}\,E_j[v/x_j])$$

$$[s[\mathtt{p},\mathtt{q}]?\{l_i(x_i)\langle E_i\rangle.P_i\}_{i\in I}]\sigma \xrightarrow{s[\mathtt{p},\mathtt{q}]?l_j\langle v\rangle} [P_j[v/x_j]]\sigma' \qquad (j \in I \quad \sigma' = \sigma\,\mathtt{after}\,E_j[v/x_j])$$

$$\frac{[P_1]\sigma_1 \xrightarrow{\overline{a}[\mathtt{n}]\langle s\rangle} [P_1']\sigma_1 \quad [P_i]\sigma_i \xrightarrow{a[i]\langle s\rangle} [P_i']\sigma_i \quad (2 \leqslant i \leqslant n)}{[P_1]\sigma_1 \mid \cdots \mid [P_n]\sigma_n \xrightarrow{\tau} (\nu s)(s : \varnothing \mid [P_1']\sigma_1 \mid \cdots \mid [P_n']\sigma_n)}$$

$$\frac{[P]\sigma \xrightarrow{s[\mathtt{p},\mathtt{q}]!l_j\langle v\rangle} [P']\sigma'}{[P]\sigma \mid s : h \xrightarrow{\tau} [P']\sigma' \mid s : h \cdot (\mathtt{p},\mathtt{q},l_j\langle v\rangle)} \qquad \frac{[P]\sigma \xrightarrow{s[\mathtt{p},\mathtt{q}]?l_j\langle v\rangle} [P']\sigma'}{[P]\sigma \mid s : (\mathtt{p},\mathtt{q},l_j\langle v\rangle)\cdot h \xrightarrow{\tau} [P']\sigma' \mid s : h}$$

Fig. 3. Labelled transition for networks

The first and second rule are for requesting and accepting a session initialisation. The guarded command checks if condition e_j is satisfied in the current state σ, and sends a message consisting of one of the labels l_j and an expression e_j' (which is evaluated to a value v in state σ), updates σ according to E_j, and behaves as $P[v/x_j]$. Branching is symmetric. The synchronous session initialisation creates a new queue. We omit the standard context/structural equivalence rules.

4 Proof system for multiparty session logic with virtual states

In this section we outline how to obtain the syntactic validation of networks, written $\Gamma \vdash N \rhd \Sigma$. We assume processes typable, following [5], and rely on the environments:

$$\Gamma ::= \varnothing \mid \Gamma, a : \mathcal{G} \mid \Gamma, X : (x : S)\mathcal{L}_1 @ \mathtt{p}_1, \ldots, \mathcal{L}_n @ \mathtt{p}_n,\ \Delta ::= \varnothing \mid \Delta, s[\mathtt{p}] : \mathcal{L},\ \Sigma ::= \varnothing \mid \Sigma, [\Delta]\sigma$$

We call Σ a *global specification*. We also use an *assertion environment* $\mathcal{C}$, which is incrementally built by conjunction of the predicates occurring in the processes being validated, and model their assumptions. The closure $\#\mathcal{C}$ is defined similarly to $\#A$. We let $\mathsf{init}(P)$ be the set of pairs $(a, \mathtt{i})$ for which P includes a session accept/request on shared channel a as role $\mathtt{i}$ (i.e., the sessions/roles P may engage in).

Well-formedness We want to prevent processes from implementing two assertions that have different sorts for the same state variable. We say that P is *well-formed* w.r.t. Γ if for all $(a, \mathtt{i}), (a', \mathtt{j}) \in \mathsf{init}(P)$ such that $x : S$ (resp. $x : S'$) is in the signature of $\Gamma(a) \upharpoonright \mathtt{i}$ (resp. $\Gamma(a') \upharpoonright \mathtt{j}$), $S = S'$ holds. If P is well-formed w.r.t. Γ then its *signature* $\mathsf{s}(P, \Gamma)$ is defined as the union of the signatures of $\Gamma(a) \upharpoonright \mathtt{i}$ for all $(a, \mathtt{i}) \in \mathsf{init}(P)$.

Stability of Γ and principal invariant A process may implement a number of local assertions, whose signatures may have common state variables. Hence, it is not sufficient

to check the invariant stability of each assertion in isolation; we must check that each assertion does not invalidate the invariants of all the assertions that act on the same state.[3] We call the conjunction of all these invariants the *principal invariant*. More precisely, the principal invariant $\mathcal{I}$ of $[P]\sigma$ w.r.t. Γ is the conjunction of the assertion invariants of $\Gamma(a) \upharpoonright \mathtt{i}$ for each session $(a, \mathtt{i}) \in \mathsf{init}(P)$. We say that Γ *is stable w.r.t.* $\mathcal{I}$ if all assertions in the domain of Γ are invariant stable w.r.t. $\mathcal{I}$.

Validation rules Figure 4 illustrates the proof rules for initial networks and processes. Rule [N1] decomposes the validation of a network into the validations of each located process with its corresponding specifications (i.e., $\mathtt{id}(\sigma_p) = \mathtt{id}(\sigma_a)$); both local and virtual states must have signature $\mathtt{s}(P, \Gamma)$ and satisfy the principal invariant, and Γ' (i.e., a subset of Γ representing the knowledge of the specific principal) must be stable w.r.t. the principal invariant. [N4] uses a refinement relation between specifications: $(\Gamma', \Delta', \sigma') \Subset (\Gamma, \Delta, \sigma)$ if $(\Gamma', \Delta', \sigma')$ specifies a more refined behaviour than (Γ, Δ, σ), in that it poses more restrictions on the output actions and poses less restrictions on the input actions. The other rules for networks are standard. Rule [MACC] validates a session

$$\frac{\mathtt{id}(\sigma_p) = \mathtt{id}(\sigma_a) \quad \mathtt{s}(\sigma_p) = \mathtt{s}(\sigma_a) = \mathtt{s}(P, \Gamma) \qquad (\Gamma', \Delta', \sigma') \Subset (\Gamma, \Delta, \sigma)}{\sigma_p, \sigma_a \models \mathcal{I} \quad \Gamma' \subseteq \Gamma \quad \Gamma' \text{ stable w.r.t. } \mathcal{I} \quad \mathcal{I}; \mathcal{C}; \Gamma' \vdash P \rhd \Delta \qquad \mathcal{C} \supset \mathcal{C}' \quad \mathcal{C}'; \Gamma' \vdash N \rhd [\Delta']\sigma'}{\mathcal{C}; \Gamma \vdash [P]\sigma_p \rhd [\Delta]\sigma_a \qquad \mathcal{C}; \Gamma \vdash N \rhd [\Delta]\sigma} \ [\text{N1/N2}]$$

$$\frac{-}{\mathcal{C}; \Gamma \vdash \varnothing \rhd \varnothing} \qquad \frac{\mathcal{C}; \Gamma \vdash N \rhd \Sigma \qquad \Gamma \vdash N' \rhd \Sigma'}{\mathcal{C}; \Gamma \vdash N \mid N' \rhd \Sigma, \Sigma'} \ [\text{N3/N4}]$$

$$\frac{\mathcal{G} \upharpoonright \mathtt{i} = [\tilde{\mathtt{x}} : \tilde{S}]\{A\}.\mathcal{L} \qquad \mathcal{I}; \mathcal{C}; \Gamma, a : \mathcal{G} \vdash P \rhd y[\mathtt{i}] : \mathcal{L}, \Delta}{\mathcal{I}; \mathcal{C}; \Gamma, a : \mathcal{G} \vdash a[\mathtt{i}](y).P \rhd \Delta} \ [\text{MACC}]$$

$$\frac{\forall i \in I \quad \mathcal{I}; \mathcal{C} \wedge A_i; \Gamma \vdash P_i \rhd \Delta}{\mathcal{I}; \mathcal{C}; \Gamma \vdash s[\mathtt{p}, \mathtt{q}]?\{l_i(x_i)\langle E_i\rangle.P_i\}_{i \in I} \rhd \Delta, s[\mathtt{q}] : \mathtt{p}?\{l_i(x_i : U_i)\{A_i\}\langle E_i\rangle.\mathcal{L}_i\}_{i \in I}} \ [\text{BCH}]$$

$$\frac{\forall i \in I \exists j \in J \ \ l_i = l_j \quad \mathcal{I} \wedge \mathcal{C} \wedge e_i \supset (A_j \wedge (E_i = E_j))[e'_i/x_i] \qquad \mathcal{I}; \mathcal{C} \wedge \#e_i; \Gamma \vdash P[e'_i/x_i] \rhd \Delta, s[\mathtt{p}] : \mathcal{L}_j[e'_i/x_j]}{\mathcal{I}; \mathcal{C}; \Gamma \vdash s[\mathtt{p}, \mathtt{q}]!\{e_i \mapsto l_i\langle e'_i\rangle(x_i)\langle E_i\rangle; P_i\}_{i \in I} \rhd \Delta, s[\mathtt{p}] : \mathtt{q}!\{l_j(x_j : U_j)\{A_j\}\langle E_j\rangle.\mathcal{L}_j\}_{j \in J}} \ [\text{SEL}]$$

$$\frac{\mathcal{I}; \mathcal{C}; \Gamma \vdash P_1 \rhd \Delta_1 \quad \mathcal{I}; \mathcal{C}; \Gamma \vdash P_2 \rhd \Delta_2}{\mathcal{I}; \mathcal{C}; \Gamma \vdash P_1 \mid P_2 \rhd \Delta_1, \Delta_2} \qquad \frac{\Delta \text{ end only}}{\mathcal{I}; \mathcal{C}; \Gamma \vdash \mathbf{0} \rhd \Delta} \ [\text{PAR/END}]$$

$$\frac{\mathcal{L}_1[e/x], \ldots, \mathcal{L}_n[e/x] \ \text{ well-asserted}}{\mathcal{I}; \mathcal{C}; \Gamma, X : (x)\mathcal{L}_1 @ \mathtt{p}_1, \ldots, \mathcal{L}_n @ \mathtt{p}_n \vdash X\langle e\rangle \rhd s[\mathtt{p}_1] : \mathcal{L}_1[e/x], \ldots, s[\mathtt{p}_n] : \mathcal{L}_n[e/x]} \ [\text{VAR}]$$

$$\frac{\mathcal{I}; \mathcal{C}; \Gamma, X : (x)\mathcal{L}_1 @ \mathtt{p}_1, \ldots, \mathcal{L}_n @ \mathtt{p}_n \vdash P \rhd s[\mathtt{p}_1] : \mathcal{L}_1, \ldots, s[\mathtt{p}_n] : \mathcal{L}_n}{\mathcal{I}; \mathcal{C}; \Gamma \vdash (\mu X(x).P)\langle e\rangle \rhd s[\mathtt{p}_1] : \mathcal{L}_1[e/x], \ldots, s[\mathtt{p}_n] : \mathcal{L}_n[e/x]} \ [\text{REC}]$$

Fig. 4. Environments (top), proof rules for networks (centre) and proof rules for processes (bottom)

invitation provided that the invitation conforms to $\Gamma, a : \mathcal{G}$, and the continuation is validated with a new Δ extended with the new session. Note that $\mathcal{I} \supset A$ holds by invariant stability of Γ. The rule for session request is similar hence omitted. In rule [BCH] the continuations for each respective branch i are required to be still valid in the extended environments; namely $\mathcal{C}$ is extended to include the current predicate A_i. Note that by history sensitivity A_i does not include any free state variable. In rule [SEL] each branch

[3] This property is similar to non-interference in [19, 21].

i of the process must correspond to a branch j of the specification, with $I \subseteq J$. Expression e'_i must satisfy A_j under the assumption of condition e_i, of the invariant $\mathcal{I}$, and of the current assertion environment $\mathcal{C}$. $\mathcal{C}$ is extended in the premise with the closure $\#e_i$.[4] The other rules are similar to those in [5].

Example 4. Consider a principal who executes two parallel threads $P_{\mathtt{S}}$ and $P_{\mathtt{C}}$ from Example I.4, namely the principal sells a ticket and buys another kind of ticket from another principal (not modelled here). Assume the global specification is $[\varnothing]\sigma_a$ and $\sigma_p = \sigma_a = \{\mathtt{count} : \mathtt{Nat}, \mathtt{credit} : \mathtt{Nat}\} \mapsto \{10, 500\}$. We show the validation of $\mathsf{true}; \Gamma \vdash [P_{\mathtt{S}} \mid P_{\mathtt{C}}]\sigma_p \rhd [\varnothing]\sigma_a$ proceeding top-down using the rules in Figure 4. First, [N1] can be applied; note that $\mathcal{I} = A_{\mathtt{S}} \wedge A_{\mathtt{C}}$ is equivalent to $\{\mathtt{credit} \geqslant 0 \wedge \mathtt{count} \geqslant 0 \wedge \mathtt{count} \leqslant 1000\}$ and is satisfied by the local and virtual state. Next, rule [PAR] decomposes the derivation of two threads for $P_{\mathtt{S}}$ and $P_{\mathtt{C}}$ (we omit the illustration of the latter). We next apply [MACC] and [BRA] to the former thread:

$$\dfrac{\dfrac{\mathcal{I}; \{\exists \mathtt{C}.\mathtt{credit}.\mathtt{C}.\mathtt{credit} \geqslant x\}; \Gamma \vdash P'_{\mathtt{S}} \rhd z[\mathtt{S}] : \mathcal{L}'_{\mathtt{S}}}{\mathcal{I}; \mathsf{true}; \Gamma \vdash z[\mathtt{C}, \mathtt{S}]?(x).P'_{\mathtt{S}} \rhd z[\mathtt{S}] : \mathtt{C}?(x : \mathtt{Nat})\{\exists \mathtt{C}.\mathtt{credit}.\mathtt{C}.\mathtt{credit} \geqslant x\}.\mathcal{L}'_{\mathtt{S}}}\,[\textsc{Bch}]}{\mathcal{I}; \mathsf{true}; \Gamma \vdash P_{\mathtt{S}} \rhd \varnothing}\,[\textsc{Macc}]$$

Next, by [SEL], setting $e = \mathtt{count} < 1000 \wedge x \geqslant 10$, $A = \exists \mathtt{C}.\mathtt{credit}.\mathtt{C}.\mathtt{credit} \geqslant x$, and $E_{ok} = \mathtt{count} := \mathtt{count} + 1,\ \mathtt{credit} := \mathtt{credit} + x$:

$$\dfrac{\begin{array}{c}\mathcal{I} \wedge A \wedge e \supset (\mathtt{count} < 1000 \wedge y = \mathtt{count} \wedge E_{ok} = E_{ok})[\mathtt{count}/y] \qquad \mathcal{I}; A; \Gamma \vdash \mathbf{0} \rhd z[\mathtt{S}] : \mathsf{end} \\ \mathcal{I} \wedge A \wedge \neg e \supset \mathsf{true} \qquad \mathcal{I}; A; \Gamma \vdash \mathbf{0} \rhd z[\mathtt{S}] : \mathsf{end}\end{array}}{\begin{array}{r}\mathcal{I}; A; \Gamma \vdash z[\mathtt{S}, \mathtt{C}]!\{e \mapsto \mathsf{ok}\langle \mathtt{count}\rangle(y)\langle E_{ok}\rangle.\mathbf{0}, \neg e \mapsto \mathsf{ko}.\mathbf{0}\} \\ \models z[\mathtt{S}] : \mathtt{C}!\{\mathsf{ok}(y : \mathtt{Nat})\{\mathtt{count} < 1000 \wedge y = \mathtt{count}\}\langle E_{ok}\rangle.\mathsf{end},\ \mathsf{ko}.\mathsf{end}\}\end{array}}$$

where each line in the premise corresponds to a branch (i.e., ok and ko). Finally we apply $\lfloor$[END]$\rfloor$ to the second premise of each branch.

5 Soundness and completeness of the validation rules

We define a labelled transition relation for specifications $\langle \Gamma, \Sigma \rangle$ using the same labels as for networks. The main difference with the rules for networks is that predicates must be satisfied for the transition to occur. We illustrate below the most remarkable rules (the other rules are in Figure 8 in Appendix). The rule for session request:

$$\langle (a : \mathcal{G}, \Gamma); [\Delta]\sigma \rangle \xrightarrow{\overline{a}[\mathtt{n}]\langle s \rangle} \langle (a : \mathcal{G}, \Gamma); [\Delta, s[\mathtt{1}] : \mathcal{L}]\sigma \rangle \qquad (\mathcal{G} \upharpoonright \mathtt{1} = [\tilde{\mathtt{x}} : \tilde{S}]\{A\}.\mathcal{L},\ \ \sigma \models A)$$

extends Δ with the new session, given that $a : \mathcal{G}$ in Γ and the current state satisfies assertion invariant A. The rule for session accept is dual. The rule for selection/send:

$$\langle \Gamma; [\Delta, s[\mathtt{p}] : \mathtt{q}!\{l_i(x_i : U_i)\{A_i\}\langle E_i\rangle.\mathcal{L}_i\}_{i \in I}]\sigma \rangle \xrightarrow{s[\mathtt{p},\mathtt{q}]!l_j\langle \mathtt{n}\rangle} \langle \Gamma; [\Delta, s[\mathtt{p}] : \mathcal{L}_j[\mathtt{n}/x_j]]\sigma' \rangle$$
$$(j \in I,\ \sigma \models A[\mathtt{n}/x_j],\ \sigma' = \sigma\, \mathtt{after}\, E_j[\mathtt{n}/x_j])$$

moves to the continuation $\mathcal{L}_j$ of the selected branch with the updated state σ', given that the sent value $\mathtt{n}$ satisfies predicate A_j for branch j in the current state σ.

Semantic conformance is defined using *conditional simulation* [5], written $\precsim$, to relate networks N to specifications $\langle \Gamma; \Sigma \rangle$.

[4] $\lfloor$[BCH]$/\lfloor$[SEL]$\rfloor$ can be extended to delegation adding the following clause for $U_i = \langle \mathcal{L} \rangle$: ($\lfloor$[BCH]$\rfloor$) $\mathcal{I}; \mathcal{C} \wedge A_i; \Gamma \vdash P_i \rhd \Delta, s[\mathtt{q}] : \mathcal{L}_i, x_i : \mathcal{L}$, and ($\lfloor$[SEL]$\rfloor$) $\mathcal{I}; \mathcal{C} \wedge \#e_i; \Gamma \vdash P[e'_i/x_i] \rhd \Delta', s[\mathtt{p}] : \mathcal{L}_j[e'_i/x_j]$ with $\Delta = \Delta'', e_i : \mathcal{L}'_i$ and $\Delta' = \Delta''$.

Definition 1 (Conditional Simulation). A binary relation $\mathcal{R}$ over $\langle \Gamma; \Sigma \rangle$ is a *conditional simulation* if, for each $(N, \langle \Gamma, \Sigma \rangle) \in \mathcal{R}$, if $N \xrightarrow{\ell} N'$ with ℓ being:
(1) a branching then $\langle \Gamma; \Sigma \rangle$ is capable to move at the subject of ℓ, and if $\langle \Gamma; \Sigma \rangle \xrightarrow{\ell} \langle \Gamma; \Sigma' \rangle$ then $(N, \langle \Gamma; \Sigma' \rangle) \in \mathcal{R}$;
(2) a select, session request/accept, τ then $\langle \Gamma; \Sigma \rangle \xrightarrow{\ell} \langle \Gamma; \Sigma' \rangle$ and $(N', \langle \Gamma; \Sigma' \rangle) \in \mathcal{R}$.

Conditional simulation is as standard simulation for all types of actions except for branching, for which it requires N to be simulated only for legal values/labels (i.e., a process must conform to a given specification as long as its environment does so).

Definition 5 (Satisfaction). N satisfies Σ in Γ and $\mathcal{C}$, written $\mathcal{C}; \Gamma \models N \rhd \Sigma$, if for all closing substitutions $\tilde{\sigma}$ over N and Σ respecting Γ and $\mathcal{C}$, $N\tilde{\sigma} \precsim \langle \Gamma; \Sigma\tilde{\sigma} \rangle$

We write $\Gamma \models N \rhd \Sigma$ when $\mathcal{C}$ is true (e.g., for initial networks). Soundness and completeness for initial networks are stated below.

Theorem 6 (Soundness of Validation Rules). *Let N be an initial network. Then $\Gamma \vdash N \rhd \Sigma$ implies $\Gamma \models N \rhd \Sigma$*

Theorem 7 (Completeness of Validation Rules). *Let $N \equiv \prod_{i \in I} [P_i]\sigma_{pi}$ be an initial network, with $\mathcal{I}_i$ principal invariant of $[P_i]\sigma_{pi}$, and $\Sigma = \prod_{i \in I} [\Delta_i]\sigma_{ai}$ be a specification. Assume that for all $i \in I$:* (1) $\mathtt{id}(\sigma_{pi}) = \mathtt{id}(\sigma_{ai})$, (2) $var(\mathcal{I}_i) \subseteq \mathtt{s}(\sigma_{pi}) = \mathtt{s}(\sigma_{ai})$, *and* (3) $\mathcal{I}_i$ *equivalent to* true. *If $\Gamma \models N \rhd \Sigma$ then $\Gamma \vdash N \rhd \Sigma$*

(1–2) are for symmetry between N and Σ. (3) is necessary since the principals in N can make updates that differ from those made by the corresponding specifications in Σ; this may not compromise the observable behaviour of N with respect to Σ, but N may invalidate some principal invariant which would make the thesis false.

6 Embedding into Hennessy-Milner Logic

In order to compare the expressiveness of our system to existing logical frameworks, we propose an embedding of the local environment associated to a principal into an HML formula. Our proof rules can be seen as the superposition of two analyses: a session type system and a logical layer. The former ensures that a process is able to perform some visible actions and could be easily encoded in HML (for instance, by using a "surely/then" modality [1]). We focus on the embedding of the latter, namely on *predicate safety*, ensuring that stateful predicates will remain satisfied. As a result, the completeness result of Theorem 8 is given relatively to a unasserted typing result ($\mathbf{Er}(\Delta)$ is Δ where all logical predicates and updates have been removed). Formal details and proof sketches are given in Appendix H.

The LTS associated to our HML consider as actions both the communications of the process and the updates of the state. Yet, we also explain how a further *pure* encoding can translate state updates into interactions. Our embedding is given by:

$$\|\mathtt{q}!\{l_i(x_i : S_i)\{A_i\}\langle E_i\rangle.\mathcal{L}_i\}_{i\in I}\|^{s[\mathtt{p}]} = \textstyle\bigwedge_{i\in I} \forall x_i : S_i, [\overline{s[\mathtt{p},\mathtt{q}]}(x_i)](A_i \wedge [E_i]\|\mathcal{L}_i\|^{s[\mathtt{p}]})$$
$$\|\mathtt{q}?\{l_j(x_j : S_j)\{A_j\}\langle E_j\rangle.\mathcal{L}_j\}_{j\in J}\|^{s[\mathtt{p}]} = \textstyle\bigwedge_{j\in J} \forall x_j : S_j, [s[\mathtt{q},\mathtt{p}](x_j)](A_j \Rightarrow \|\mathcal{L}_j\|^{s[\mathtt{p}]})$$

Predicates are required to hold for output actions and used as premises for implications for input actions. To obtain soundness for typing judgements involving specifications, we have to introduce *interleavings* of formulae, treating the fact that for one process playing several roles in several sessions, the actions could be interleaved in different way. Interleaving is not a new operator *per se* and can be seen as syntactic sugar, describing shuffling of must modalities for formulas. $\texttt{Inter}(\Delta, \Gamma)$ stands for the embedding of the environments Δ and Γ.

Theorem 8 (Preciseness).
If $\mathcal{I}; \mathcal{C}; \Gamma \vdash P \rhd \Delta$ *and* $\sigma \models \mathcal{I}$, *then:* $P, \sigma \models (\mathcal{I} \wedge \mathcal{C} \Rightarrow \texttt{Inter}(\Delta, \Gamma))$.
If $\sigma \models \mathcal{I}$, $\vdash P \rhd \mathbf{Er}(\Delta)$ *and* $P, \sigma \models (\mathcal{I} \wedge \mathcal{C} \Rightarrow \texttt{Inter}(\Delta, \Gamma))$ *then* $\mathcal{I}; \mathcal{C}; \Gamma \vdash P \rhd \Delta$.

The embedding of recursive types is challenging, as it involves describing by a finite (yet recursive [12]) formula all the possible infinite interleavings. We explain a method to obtain this result in Appendix H.

7 Related work and further topics

The preceding integrations of session types with logical constraints include [11], based on concurrent constraints ensuring bi-linear usage of channels, and [5], based on logical annotations on interactions, do not treat stateful properties. The combination of types and logical assertions referring to local state newly proposed in this paper enable fine-grained specifications and validation, which are not possible in [5, 11].

The expressiveness of the session type-based analyses has been greatly extend these past few years. On one side, the conversation calculus [7], contracts [10] and dynamic multirole session types [13] have opened the way to the modelling of protocols complex in their shapes, by describing more accurately how sessions can be joined or left, who is allowed participate. On the other side, works such as [5, 8] improved the way interactions inside a session are decribed: in [5], an assertion framework ensure logical property on the communicated values, in [8], a security analysis guarantees that the coherence of the information flow is preserved. Our work improves the session type analyses in both directions: by proposing a division of the process being tested into separate principals that can join one or several sessions independently when conditions are matched and manage their own state, and by giving a description, inside each session, of the internal state of each participant and the property it should satisfy.

The refinement types for channels (e.g. [3]) specify value dependency with logical constraints. For example, one might write $?(x : \mathsf{int}, !\{y : \mathsf{int} \mid y > x\})$ using the notations from [15]. It specifies a dependency at a *single point* (channel), unable to describe a constraint for a series of interactions among multiple channels. Our theory, based on multiparty sessions, can verify processes against a contract globally agreed by multiple distributed peers. The work [6] investigates a relationship between a dual intuitionistic linear logic and binary session types, and shows that the former defines a proof system for a session calculus which can automatically characterise and guarantee a session fidelity and global progress. The work [23] further extends [6] to the dependent type theory in order to include processes that communicate data values in functional languages. A recent work [22] encodes dynamic features in [13] in a dependently typed language for secure distributed programming. None of the above works treat either virtual states or logical specifications for interleaved multiparty sessions.

Another future direction is to link between our static analysis and a dynamic monitor-based approach. Using our local specification as a monitor at each end-point, incoming and outgoing messages can be verified and filtered. We are currently working on this topic with [20] based on the logic developed in this paper.

References

1. M. Berger, K. Honda, and N. Yoshida. Completeness and logical full abstraction in modal logics for typed mobile processes. In *ICALP (2)*, volume 5126 of *LNCS*, pages 99–111. Springer, 2008.
2. L. Bettini et al. Global progress in dynamically interleaved multiparty sessions. In *CONCUR*, volume 5201 of *LNCS*, pages 418–433. Springer, 2008.
3. K. Bhargavan, C. Fournet, and A. D. Gordon. Modular verification of security protocol code by typing. In *POPL*, pages 445–456, 2010.
4. L. Bocchi, R. Demangeon, and N. Yoshida. A multiparty multi-session logic (extended report). http://www.cs.le.ac.uk/people/lb148/statefulassertions.html.
5. 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, 2010.
6. L. Caires and F. Pfenning. Session types as intuitionistic linear propositions. In *CONCUR'10*, volume 6269 of *LNCS*, pages 222–236. Springer-Verlag, 2010.
7. L. Caires and H. T. Vieira. Conversation types. In *ESOP*, volume 5502 of *LNCS*, pages 285–300. Springer, 2009.
8. S. Capecchi, I. Castellani, and M. Dezani-Ciancaglini. Information flow safety in multiparty sessions. In *EXPRESS*, volume 64 of *EPTCS*, pages 16–30, 2011.
9. M. Carbone, K. Honda, and N. Yoshida. Structured interactional exceptions in session types. In *CONCUR*, volume 5201 of *LNCS*, pages 402–417. Springer, 2008.
10. G. Castagna and L. Padovani. Contracts for mobile processes. In *CONCUR 2009*, number 5710 in LNCS, pages 211–228, 2009.
11. M. Coppo and M. Dezani-Ciancaglini. Structured communications with concurrent constraints. In *TGC*, pages 104–125, 2008.
12. M. Dam. Ctl* and ectl* as fragments of the modal mu-calculus. *TCS*, 126(1):77–96, 1994.
13. P.-M. Deniélou and N. Yoshida. Dynamic multirole session types. In *POPL*, pages 435–446, 2011.
14. E. W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Commun. ACM*, 18:453–457, August 1975.
15. T. Freeman and F. Pfenning. Refinement types for ML. *SIGPLAN Not.*, 26(6):268–277, 1991.
16. M. Hennessy and R. Milner. On observing nondeterminism and concurrency. In *ICALP*, pages 299–309, 1980.
17. K. Honda, N. Yoshida, and M. Carbone. Multiparty Asynchronous Session Types. In *POPL'08*, pages 273–284. ACM, 2008.
18. K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. In G. C. Necula and P. Wadler, editors, *POPL*, pages 273–284. ACM, 2008.
19. L. Lamport and F. B. Schneider. The "hoare logic" of csp, and all that. *ACM Trans. Program. Lang. Syst.*, 6:281–296, April 1984.
20. Ocean Observatories Initiative (OOI). `http://www.oceanleadership.org/programs-and-partnerships/ocean-observing/ooi/`.
21. S. S. Owicki and D. Gries. An axiomatic proof technique for parallel programs i. *Acta Inf.*, 6:319–340, 1976.

22. N. Swamy, J. Chen, C. Fournet, P.-Y. Strub, K. Bharagavan, and J. Yang. Secure distributed programming with value-dependent types. In *ICFP*. ACM, 2011.
23. B. Toninho, L. Caires, and F. Pfenning. Dependent session types via intuitionistic linear type theory. In *PPDP'11*. ACM, 2011.
24. N. Yoshida, P.-M. Deniélou, A. Bejleri, and R. Hu. Parameterised multiparty session types. In *FoSSaCs'10*, volume 6014 of *LNCS*, pages 128–145. Springer, 2010.

A Auxiliary Definitions

Definition 2 (Refinement). *A binary relation $\mathcal{R}$ over (Γ, Δ, σ) is a* refinement relation *if $(\Gamma_1, \Delta_1, \sigma_1)\mathcal{R}(\Gamma_2, \Delta_2, \sigma_2)$ implies one of the following conditions holds*

- $\langle \Gamma_1; [\Delta_1]\sigma_1 \rangle \xrightarrow{\ell} \langle \Gamma_1; [\Delta_1']\sigma_1' \rangle$ *with ℓ being a selection action then* $\langle \Gamma_2; [\Delta_2]\sigma_2 \rangle \xrightarrow{\ell} \langle \Gamma_2; [\Delta_2]\sigma_2' \rangle$ *with* $(\Gamma_1, \Delta_1', \sigma_1')\mathcal{R}(\Gamma_2, \Delta_2', \sigma_2')$.
- $\langle \Gamma_2; [\Delta_2]\sigma_2 \rangle \xrightarrow{\ell} \langle \Gamma_2; [\Delta_2']\sigma_2' \rangle$ *with ℓ being a branching action, then* $\langle \Gamma_1; [\Delta_1]\sigma_1 \rangle \xrightarrow{\ell} \langle \Gamma_1; [\Delta_1']\sigma_1' \rangle$ *with* $(\Gamma_1, \Delta_1', \sigma_1')\mathcal{R}(\Gamma_2, \Delta_2', \sigma_2')$.

If $(\Gamma_1, \Delta_1, \sigma_1)\mathcal{R}(\Gamma_2, \Delta_2, \sigma_2)$ for some refinement relation $\mathcal{R}$, we say $(\Gamma_1, \Delta_1, \sigma_1)$ is a refinement *of $(\Gamma_2, \Delta_2, \sigma_2)$ (written $(\Gamma_1, \Delta_1, \sigma_1) \ni (\Gamma_2, \Delta_2, \sigma_2)$).*

Definition 3 (Algorithm for the derivation of $\mathsf{init}(\mathtt{P})$).

- *if* $P = \overline{a}[\mathtt{n}](y).P'$ *then* $\mathsf{init}(P) = (a, \mathtt{1}) \cup \mathsf{init}(P')$
- *if* $P = a[\mathtt{i}](y).P'$ *then* $\mathsf{init}(P) = (a, \mathtt{i}) \cup \mathsf{init}(P')$
- *if* $P = s[\mathtt{p}, \mathtt{q}]!\{e_i \mapsto \langle e_i'(x_i)\langle E_i \rangle.P_i\}_{i \in I}$ *then* $\mathsf{init}(P) = \bigcup_{i \in I} \mathsf{init}(P_i)$
- *if* $P = s[\mathtt{p}, \mathtt{q}]?\{(x_i)\langle E_i \rangle.P_i\}_{i \in I}$ *then* $\mathsf{init}(P) = \bigcup_{i \in I} \mathsf{init}(P_i)$
- *if* $P = P_1 \mid P_2$ *then* $\mathsf{init}(P) = \mathsf{init}(P_1) \cup \mathsf{init}(P_2)$
- *if* $P = \mu X\langle y : A' \rangle(x : S)\{A\}.P'$ *then* $\mathsf{init}(P) = \mathsf{init}(P')$
- *if* $P = X\langle y : A' \rangle$ *or* $P = \mathbf{0}\rangle$ *then* $\mathsf{init}(P) = \varnothing$

Definition 4 (Algorithm for the derivation of the principal invariant). *Given Γ and $[P]\sigma$ the principal invariant, denoted with $\mathcal{I}(\Gamma, [P]\sigma)$ or simply $\mathcal{I}$ is defined as:*

$$\wedge_{(a,\mathtt{i}) \in \mathsf{init}(P)} A_i \;\; s.t. \;\; \Gamma(a) \upharpoonright \mathtt{i} = [\tilde{\mathtt{x}} : \tilde{S}]\{A_i\}.\mathcal{L}$$

Definition 5 (Projection). Assume $\mathtt{p}, \mathtt{q}, \mathtt{r} \in \mathcal{G}$ and $\mathtt{p} \neq \mathtt{q}$. The projection of $\mathcal{G}$ on $\mathtt{r} \in \mathcal{G}$, written $\mathcal{G} \upharpoonright \mathtt{r}$, is defined as follows.

(1) $(((\mathtt{p}_1 : [\tilde{\mathtt{x}}_1 : \tilde{S}_1]\{A_1\}, \ldots, \mathtt{p}_n : [\tilde{\mathtt{x}}_n : \tilde{S}_n]\{A_n\}.\mathcal{G}))) \upharpoonright \mathtt{r} = [\tilde{\mathtt{x}}_i : \tilde{S}_i]\{A_i\}.(\mathcal{G} \upharpoonright \mathtt{p}_i)$ if $\mathtt{r} = \mathtt{p}_i$

(2) $(\mathtt{p} \to \mathtt{q} : \{l_i(x_i : U_i)\{A_i\}\langle E_i \rangle.\mathcal{G}_i\}_{i \in I}) \upharpoonright \mathtt{r} =$

$$\begin{cases} \mathtt{q}!\{l_i(x_i : U_i)\{A_i\}\langle E_i \upharpoonright \mathtt{r} \rangle.(\mathcal{G}_i \upharpoonright \mathtt{p})\}_{i \in I} & \text{if } \mathtt{r} = \mathtt{p} \neq \mathtt{q}, \\ \mathtt{p}?\{l_i(x_i : U_i)\{A_i \upharpoonright \mathtt{r}\}\langle E_i \upharpoonright \mathtt{r} \rangle.(\mathcal{G}_i \upharpoonright \mathtt{q})\}_{i \in I} & \text{if } \mathtt{r} = \mathtt{q} \neq \mathtt{p}, \\ \mathcal{G}_1 \upharpoonright \mathtt{r} & \text{if } \mathtt{r} \neq \mathtt{q}, \mathtt{p} \end{cases}$$

(3) $(\mathcal{G}_1 \mid \mathcal{G}_2) \upharpoonright \mathtt{r} = \begin{cases} \mathcal{G}_i \upharpoonright \mathtt{r} & \text{if } \mathtt{r} \in \mathcal{G}_i \text{ and } \mathtt{r} \notin \mathcal{G}_j, i \neq j \in \{1, 2\} \\ \mathsf{end} & \text{if } \mathtt{r} \notin \mathcal{G}_1 \text{ and } \mathtt{r} \notin \mathcal{G}_2. \end{cases}$

(4) $(\mu \mathsf{t}\langle y : A' \rangle(x : S)\{A\}.\mathcal{G}) \upharpoonright \mathtt{r} = \begin{cases} \mu \mathsf{t}\langle y : A' \upharpoonright \mathtt{r} \rangle(x : S)\{A \upharpoonright \mathtt{r}\}.\mathcal{G} \upharpoonright \mathtt{r} & \text{if } \mathtt{r} \in \mathcal{G} \\ \mathsf{end} & \text{if } \mathtt{r} \notin \mathcal{G} \end{cases}$

(5) $\mathsf{t}\langle y : A' \rangle \upharpoonright \mathtt{r} = \mathsf{t}\langle y : A' \upharpoonright \mathtt{r} \rangle$

B Congruence, Reduction and Labelled Transitions

Figure 5 presents the full congruence rules for networks and processes, where the asynchronous messages are considered upon permutation. Figure 6 and Figure 8 illustrate the full transition rules for networks and specifications. Figure 6 models silent actions as reductions from Figure 7. In the paper we have represented silent actions explicitly in the LTS for a more concise presentation.

$$N \mid \varnothing \equiv N \qquad N_1 \mid N_2 \equiv N_1 \mid N_2 \qquad (N_1 \mid N_2) \mid N_3 \equiv N_1 \mid (N_2 \mid N_3) \text{ if } a \notin \mathsf{fn}(N)$$

$$(\nu s)\varnothing \equiv \varnothing \qquad (\nu s)(\nu s')N \equiv (\nu s')(\nu s)N \qquad (\nu s)N \mid N' \equiv (\nu s)(N \mid N') \quad \text{if } s \notin \mathsf{fn}(N)$$

$$s : h_1 \cdot (\mathtt{p}_1, \mathtt{p}_2, l\langle v\rangle) \cdot (\mathtt{q}_1, \mathtt{q}_2, l'\langle v'\rangle) \cdot h_2 \equiv s : h_1 \cdot (\mathtt{q}_1, \mathtt{q}_2, l'\langle v'\rangle) \cdot (\mathtt{p}_1, \mathtt{p}_2, l\langle v\rangle) \cdot h_2 \quad *\text{if } \mathtt{p}_1 \neq \mathtt{q}_1 \text{ or } \mathtt{p}_2 \neq \mathtt{q}_2$$

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

$$(\mu X(x).P)\langle e\rangle \equiv P[\mu X(x).P/X][e/x] \text{ where } X\langle e'\rangle[\mu X(x).P/X] \stackrel{\text{def}}{=} (\mu X(x).P)\langle e'\rangle$$

Fig. 5. Structural congruence for networks (top) and processes (bottom)

$$[\overline{a}[\mathtt{n}](y).P]\sigma \xrightarrow{\overline{a}[\mathtt{n}]\langle s\rangle} [P[s/y]]\sigma \quad [a[\mathtt{i}](y).P]\sigma \xrightarrow{a[i]\langle s\rangle} [P[s/y]]\sigma \quad (s \notin \mathsf{fn}(P))$$

$$[s[\mathtt{p},\mathtt{q}]!\{e_i \mapsto l_i\langle e_i'\rangle(x_i)\langle E_i\rangle; P_i\}_{i\in I}]\sigma \xrightarrow{s[\mathtt{p},\mathtt{q}]!l_j\langle v\rangle} [P[v/x_j]]\sigma' \quad (j \in I \quad \sigma \models e_j' \downarrow v \wedge e_j \quad \sigma' = \sigma \,\mathtt{after}\, E_j[v/x_j])$$

$$[s[\mathtt{p},\mathtt{q}]?\{l_i(x_i)\langle E_i\rangle.P_i\}_{i\in I}]\sigma \xrightarrow{s[\mathtt{p},\mathtt{q}]?l_j\langle v\rangle} [P_j[v/x_j]]\sigma' \quad (j \in I \quad \sigma' = \sigma \,\mathtt{after}\, E_j[v/x_j])$$

$$\frac{N \longrightarrow N'}{N \xrightarrow{\tau} N'} \qquad \frac{[P]\sigma \xrightarrow{\ell} [P']\sigma \quad (s \notin \mathsf{fn}(P))}{[\mathcal{E}[P]]\sigma \xrightarrow{\ell} [\mathcal{E}[P']]\sigma} \qquad \frac{N \equiv N_0 \quad N_0 \xrightarrow{\ell} N_0' \quad N_0' \equiv N' \quad \mathsf{fn}(\ell) \notin bn(\mathcal{E}[\_])}{\mathcal{E}[N] \xrightarrow{\ell} \mathcal{E}[N']}$$

Fig. 6. Labelled transition for networks

C Effective Checking of Well Assertedness

Checking History Sensitivity We uses the environments $\mathcal{E}$ defined by the following grammar:

$$\mathcal{E} := \varnothing \mid \mathcal{E}, x@\mathtt{p} \mid \mathcal{E}, x@\mathtt{L} \mid \mathcal{E}, \mathtt{t} : x@\mathtt{L}$$

Expressions of the form $y@\mathtt{p}$ assign a state variable y to a role, and $y@\mathtt{L}$ assigns an interaction variable or recursion parameters to a *location* $\mathtt{L}$. A location is a set $\{\mathtt{p}, \mathtt{p}'\}$ of the two roles who know x. The checker relies on the annotation of the recursion parameters with their locations and we assume recursion parameter to be known by only two roles. We denote the domain of $\mathcal{E}$ with $dom(\mathcal{E})$. We write $\mathcal{E} \vdash x@\mathtt{p}$ when $\mathtt{p} = \mathcal{E}(x)$, $\mathtt{p} \in \mathcal{E}(x)$, or $\mathcal{E}(\mathtt{t}) = x@\mathtt{L}$ and $\mathtt{p} \in \mathtt{L}$.

The rules in Figure 9 enforce history sensitivity by restricting the set of variables that can be used in each predicate and update. The first two rules require that each role knows all the interaction variables of the predicate to be checked at its side. The other rules are straightforward. Note that the rules are purely syntactic, hence the verification of history sensitivity of $\mathcal{G}$ is a linear-time problem.

Checking Invariant Stability This principle requires that each update performed in a global assertion does not invalidate the assertion invariants in the declaration of that global assertion. To check for invariant stability, we use the function defined below, where A_{inv} is the invariant, and A_{bag} is a set of preconditions built as the incremental conjunction of interaction predicates (initially true).

Definition 6 (Invariant Stability). Let $\mathcal{G}$ be a global specification, A_{inv} and A_{pre} two predicates. $\mathtt{IS}(\mathcal{G},\ A_{inv},\ A_{bag})$ is given by:

$$[\overline{a}[\mathtt{n}](y).P_1 \mid Q_1]\sigma_1 \mid \prod_{2 \leqslant i \leqslant n}[a[\mathtt{i}](y_i).P_i \mid Q_i]\sigma_i \longrightarrow (\nu s)(s : \varnothing \mid \prod_{1 \leqslant i \leqslant n}[P_i[s/y_i] \mid Q_i]\sigma_i)$$

$$\frac{j \in I \qquad \sigma \models e_j \qquad \sigma \models e'_j \downarrow v \qquad \sigma' = \sigma\,\mathtt{after}\,E_j[v/x_j]}{[s[\mathtt{p},\mathtt{q}]!\{e_i \mapsto l_i\langle e'_i\rangle(x_i)\langle E_i\rangle; P_i\}_{i\in I} \mid Q]\sigma \mid s : h \longrightarrow [P_j[v/x_j] \mid Q]\sigma' \mid s : h \cdot (\mathtt{p},\mathtt{q}, l_j\langle v\rangle)}$$

$$\frac{j \in I \qquad \sigma' = \sigma\,\mathtt{after}\,E_j[v/x_j]}{[s[\mathtt{p},\mathtt{q}]?\{l_i(x_i)\langle E_i\rangle.P_i\}_{i\in I} \mid Q]\sigma \mid s : (\mathtt{p},\mathtt{q}, l_j\langle v\rangle) \cdot h \longrightarrow [P_j[v/x_j] \mid Q]\sigma' \mid s : h}$$

$$\frac{P \equiv P_0 \quad P_0 \longrightarrow P'_0 \quad P'_0 \equiv P'}{P \longrightarrow P'} \qquad \frac{N \equiv N_0 \quad N_0 \longrightarrow N'_0 \quad N' \equiv N'_0}{\mathcal{E}[N] \longrightarrow \mathcal{E}[N']}$$

Fig. 7. Reduction for networks

1. $\mathtt{IS}(\mathtt{p} \to \mathtt{q} : \{l_i(x_i : U_i)\{A_i\}\langle E_i\rangle.\mathcal{G}_i\}_{i\in I},\ A_{inv},\ A_{bag}) =$
$\begin{cases} \bigwedge_{i\in I} \mathtt{IS}(\mathcal{G}_i,\ A_{inv},\ A_{bag} \wedge \#A_i) & \text{if } \bigvee_{i\in I} A_{inv} \wedge A_{bag} \wedge A_i \supset A_{inv}\,\mathtt{after}\,E_i \\ \mathsf{false} & \text{otherwise} \end{cases}$
2. $\mathtt{IS}(\mathcal{G}_1 \mid \mathcal{G}_2,\ A_{inv},\ A_{bag}) = STS(\mathcal{G}_1, A_{inv}, A_{bag}) \wedge STS(\mathcal{G}_2, A_{inv}, A_{bag})$
3. $\mathtt{IS}(\mu \mathtt{t}\langle u : A'\rangle(x : S)\{A\}.\mathcal{G}',\ A_{inv},\ A_{bag}) = \mathtt{IS}(\mathcal{G}',\ A_{inv},\ A_{bag} \wedge A)$
4. $\mathtt{IS}(t\langle u : A'\rangle,\ A_{inv},\ A_{bag}) = \mathtt{IS}(\mathsf{end},\ A_{inv},\ A_{bag}) = \mathsf{true}$

The function $\mathtt{IS}(\_,\ \_,\ \_)$ can be adapted for local assertions. We use the same notation as it will be clear from the context whether we are checking global or local assertions.

Definition 7 (Invariant Stability for Local Assertions). Let $\mathcal{L}$ be a global specification, A_{inv} and A_{pre} two predicates. $\mathtt{IS}(\mathcal{L},\ A_{inv},\ A_{bag})$ is given by:

1. $\mathtt{IS}(\mathtt{p}!\{l_i(x_i : U_i)\{A_i\}\langle E_i\rangle.\mathcal{L}_i\}_{i\in I},\ A_{inv},\ A_{bag}) =$
$\begin{cases} \bigwedge_{i\in I} \mathtt{IS}(\mathcal{L}_i,\ A_{inv},\ A_{bag} \wedge \#A_i) & \text{if } \bigvee_{i\in I} A_{inv} \wedge A_{bag} \wedge A_i \supset A_{inv}\,\mathtt{after}\,E_i \\ \mathsf{false} & \text{otherwise} \end{cases}$
2. $\mathtt{IS}(\mathtt{p}?\{l_i(x_i : U_i)\{A_i\}\langle E_i\rangle.\mathcal{L}_i\}_{i\in I},\ A_{inv},\ A_{bag}) =$
$\begin{cases} \bigwedge_{i\in I} \mathtt{IS}(\mathcal{L}_i,\ A_{inv},\ A_{bag} \wedge \#A_i) & \text{if } \bigvee_{i\in I} A_{inv} \wedge A_{bag} \wedge A_i \supset A_{inv}\,\mathtt{after}\,E_i \\ \mathsf{false} & \text{otherwise} \end{cases}$
3. $\mathtt{IS}(\mu \mathtt{t}\langle u : A'\rangle(x : S)\{A\}.\mathcal{L}',\ A_{inv},\ A_{bag}) = \mathtt{IS}(\mathcal{L}',\ A_{inv},\ A_{bag} \wedge A)$
4. $\mathtt{IS}(t\langle u : A'\rangle,\ A_{inv},\ A_{bag}) = \mathtt{IS}(\mathsf{end},\ A_{inv},\ A_{bag}) = \mathsf{true}$

D Message Assertions

We introduce the definitions for processes with queues. The aim is to take into account, in the proof of soundness of the validation rules, the mechanisms of message exchange of runtime processes. We use *message assertions* which abstract messages in queues.

Definition 9 (Message Assertions). The syntax of endpoint assertions is extended as follows:

$$\mathcal{L} ::= \ldots \mid \mathcal{M} \mid \mathcal{M}; \mathcal{L} \qquad\qquad \mathcal{M} ::= \mathtt{p}!l\langle v\rangle \mid \mathcal{M}; \mathcal{M}'$$

We call $\mathcal{M}$ a *message assertion*.

In Definition 9, $\mathtt{p}!l\langle v\rangle$ represents a label/value $l\langle v\rangle$ in the queue for participant $\mathtt{p}$, and $\mathcal{M}; \mathcal{M}'$ represents a queue with multiple elements.

Figure 10 presents the additional validation rules for runtime processes (to extend the rules in Figure 4).

Figure 11 presents the additional transition rules for message assertions.

$$\frac{\langle \Gamma; \Sigma_1\rangle \xrightarrow{\ell} \langle \Gamma; \Sigma_1'\rangle}{\langle \Gamma; \Sigma_1, \Sigma_2\rangle \xrightarrow{\ell} \langle \Gamma; \Sigma_1', \Sigma_2\rangle} \quad \frac{\langle \Gamma; \Sigma_2\rangle \xrightarrow{\ell} \langle \Gamma; \Sigma_2'\rangle}{\langle \Gamma; \Sigma_1, \Sigma_2\rangle \xrightarrow{\ell} \langle \Gamma; \Sigma_1, \Sigma_2'\rangle} \quad \lfloor \text{TR-CTX1/TR-CTX2} \rfloor$$

$$\frac{-}{\langle \Gamma; \Sigma\rangle \xrightarrow{\tau} \langle \Gamma; \Sigma\rangle} \quad \lfloor \text{TR-TAU} \rfloor$$

$$\frac{\mathcal{G} \upharpoonright \mathtt{1} = [\tilde{\mathtt{x}} : \tilde{S}]\{A\}.\mathcal{L} \qquad \sigma \models A}{\langle (a : \mathcal{G}, \Gamma); [\Delta]\sigma\rangle \xrightarrow{\overline{a}[\mathtt{n}]\langle s\rangle} \langle (a : \mathcal{G}, \Gamma); [\Delta, s[\mathtt{1}] : \mathcal{L}]\sigma\rangle} \quad \lfloor \text{TR-A-MREQ} \rfloor$$

$$\frac{\mathcal{G} \upharpoonright \mathtt{i} = [\tilde{\mathtt{x}} : \tilde{S}]\{A\}.\mathcal{L} \qquad \sigma \models A}{\langle (a : \mathcal{G}, \Gamma); [\Delta]\sigma\rangle \xrightarrow{a[i]\langle s\rangle} \langle (a : \mathcal{G}, \Gamma); [\Delta, s[\mathtt{i}] : \mathcal{L}]\sigma\rangle} \quad \lfloor \text{TR-A-MACC} \rfloor$$

$$\lfloor \text{TR-A-BCH} \rfloor$$
$$\frac{j \in I \qquad \sigma \models A[\mathtt{n}/x_j] \qquad \sigma' = \sigma \,\mathtt{after}\, E_j[\mathtt{n}/x_j]}{\langle \Gamma; [\Delta, s[\mathtt{p}] : \mathtt{q}?\{l_i(x_i : U_i)\{A_i\}\langle E_i\rangle.\mathcal{L}_i\}_{i\in I}]\sigma\rangle \xrightarrow{s[\mathtt{q},\mathtt{p}]?l_j\langle \mathtt{n}\rangle} \langle \Gamma; [\Delta, s[\mathtt{p}] : \mathcal{L}_j[\mathtt{n}/x_j]]\sigma'\rangle}$$

$$\lfloor \text{TR-A-SEL} \rfloor$$
$$\frac{j \in I \qquad \sigma \models A[\mathtt{n}/x_j] \qquad \sigma' = \sigma \,\mathtt{after}\, E_j[\mathtt{n}/x_j]}{\langle \Gamma; [\Delta, s[\mathtt{p}] : \mathtt{q}!\{l_i(x_i : U_i)\{A_i\}\langle E_i\rangle.\mathcal{L}_i\}_{i\in I}]\sigma\rangle \xrightarrow{s[\mathtt{p},\mathtt{q}]!l_j\langle \mathtt{n}\rangle} \langle \Gamma; [\Delta, s[\mathtt{p}] : \mathcal{L}_j[\mathtt{n}/x_j]]\sigma'\rangle}$$

$$\lfloor \text{TR-A-DELBCH} \rfloor$$
$$\frac{j \in I \qquad \sigma \models A_j \qquad \sigma' = \sigma \,\mathtt{after}\, E_j \quad U_j = \langle\mathcal{L}\rangle}{\langle \Gamma; [\Delta, s[\mathtt{p}] : \mathtt{q}?\{l_i(x_i : U_i)\{A_i\}\langle E_i\rangle.\mathcal{L}_i\}_{i\in I}]\sigma\rangle \xrightarrow{s[\mathtt{q},\mathtt{p}]?l_j\langle t[\mathtt{r}]\rangle} \langle \Gamma; [\Delta, s[\mathtt{p}] : \mathcal{L}_j, t[\mathtt{r}] : \mathcal{L}]\sigma'\rangle}$$

$$\lfloor \text{TR-A-DELSEL} \rfloor$$
$$\frac{j \in I \qquad \sigma \models A_j \qquad \sigma' = \sigma \,\mathtt{after}\, E_j \quad U_j = \langle\mathcal{L}\rangle}{\langle \Gamma; [\Delta, s[\mathtt{p}] : \mathtt{q}!\{l_i(x_i : U_i)\{A_i\}\langle E_i\rangle.\mathcal{L}_i\}_{i\in I}, t[\mathtt{r}] : \mathcal{L}]\sigma\rangle \xrightarrow{s[\mathtt{p},\mathtt{q}]!l_j\langle t[\mathtt{r}]\rangle} \langle \Gamma; [\Delta, s[\mathtt{p}] : \mathcal{L}_j]\sigma'\rangle}$$

Fig. 8. Labelled transition for specifications

E Soundness

E.1 Auxiliary Lemmas

This section contains auxiliary lemmas for soundness. The proofs of Lemma 1, Lemma 2 can be found below. The proofs of Lemma 3 and Lemma 4 are similar to the ones in [5] (hence omitted) as the stated properties do not directly involve the state.

Substitution The substitution lemma uses the following lemma saying that any substitution of a free variable with a value in a local assertion preserves well-assertedness.

Lemma 10. *Let $\mathcal{L}$ be a well-asserted local assertion (and well-typed wrt the underlying typing discipline), $x : U$ be an interaction variable, $v : U$ be a value of the same type as x. If $\mathcal{C}[v/x]$ admits solutions then $\mathcal{L}[v/x]$ is well-asserted.*

Proof. History sensitivity is clearly not affected by the substitution of an interaction variable with a value, as it is based on the notion of knowledge and a value is obviously known by any participant. For invariant stability, assume $\mathcal{L}[v/x]$. Since $\mathcal{L}$ is invariant stable by hypothesis, the checker $\mathtt{IS}(\_, \_, \_)$ (Definition 7) will return false for $\mathcal{L}[v/x]$ because of the otherwise case is met in (1) or (2). In both cases, if the predicate $(A_{inv} \wedge$

$$\frac{\mathcal{E}, \tilde{\mathtt{x}}_1@\mathtt{p}_1, \ldots, \tilde{\mathtt{x}}_n@\mathtt{p}_n \vdash \mathcal{G} \quad \forall i \in \{1, \ldots, n\},\ var(A_i) \subseteq \tilde{\mathtt{x}}_i}{\mathcal{E} \vdash (\!(\mathtt{p}_1 : [\tilde{\mathtt{x}}_1]\{A_1\}, \ldots, \mathtt{p}_n : [\tilde{\mathtt{x}}_n]\{A_n\})\!).\mathcal{G}}$$

$$\frac{\forall i \in I, \quad \mathcal{E}, x_i@\{\mathtt{p}, \mathtt{q}\} \vdash \mathcal{G}_i \quad \forall y \in (var(A_i) \cup var(E_i))\backslash x_i,\ \mathcal{E} \vdash y@\mathtt{p} \quad \forall y \in var(E_i)\backslash x_i,\ \mathcal{E} \vdash y@\mathtt{q}}{\mathcal{E} \vdash \mathtt{p} \to \mathtt{q} : \{l_i(x_i)\{A_i\}\langle E_i\rangle.\mathcal{G}_i\}_{i \in I}}$$

$$\frac{\mathcal{E} \vdash \mathcal{G} \quad \mathcal{E} \vdash \mathcal{G}'}{\mathcal{E} \vdash \mathcal{G}, \mathcal{G}'} \qquad \frac{}{\mathcal{E} \vdash \mathsf{end}} \qquad \frac{\forall y \in var(e),\ \forall \mathtt{r} \in \mathtt{L},\ \mathcal{E} \vdash y@\mathtt{r}}{\mathcal{E}, \mathtt{t} : x@\mathtt{L} \vdash \mathtt{t}\langle e\rangle}$$

$$\frac{\mathcal{E}, \mathtt{t} : x@\mathtt{L} \vdash \mathcal{G} \quad dom(\mathcal{E}) \supseteq var(A)\backslash x}{\mathcal{E} \vdash \mu\mathtt{t}\langle e\rangle(x@\mathtt{L})\{A\}.\mathcal{G}}$$

Fig. 9. Syntactic checker for history sensitivity

$$\frac{\mathcal{C}; \Gamma \vdash N \rhd [\Delta_1, s[\mathtt{1}] : \mathcal{L}_1]\sigma_1, \ldots, [\Delta_n, s[\mathtt{n}] : \mathcal{L}_n]\sigma_n \qquad \{s[\mathtt{i}] : \mathcal{L}_i\}_{1 \geqslant i \geqslant n} \text{ coherent}}{\mathcal{C}; \Gamma \vdash (\nu s)N \rhd [\Delta_1]\sigma_1, \ldots, [\Delta_n]\sigma_n} \lfloor \textsc{Cres} \rfloor$$

$$\mathcal{C}; \Gamma \vdash s : \varnothing \rhd \{s[\mathtt{i}] : \varnothing\}_{1 \geqslant i \geqslant n} \lfloor \textsc{Qnil} \rfloor$$

$$\frac{\mathcal{C}; \Gamma \vdash s : h \rhd [\Delta, s[\mathtt{p}] : \mathcal{L}]\sigma, \Sigma}{\mathcal{C}; \Gamma \vdash s : h \cdot (\mathtt{p}, \mathtt{q}, l\langle v\rangle) \rhd [\Delta, s[\mathtt{p}] : \mathtt{q}!l\langle v\rangle; \mathcal{L}]\sigma, \Sigma} \lfloor \textsc{Qval} \rfloor$$

Fig. 10. Additional Proof Rules for Runtime Networks and Processes

$A_{bag} \wedge A_i \supset A_{inv}\,\mathtt{after}\,E_i)[v/x]$ is false then also the its (stronger) unsubstituted version is false, which makes $\mathcal{L}$ not invariant stable contradicting the hypothesis.

Lemma 1 (Substitution). *Let $\mathcal{I}; \mathcal{C}; \Gamma \vdash P \rhd \Delta$ with Δ well-asserted and $x : U$ be an interaction variable and $v : U$ be a value. If $x \in \mathsf{fn}(P)$ then $\mathcal{I}; \mathcal{C}[v/x]; \Gamma \vdash P[v/x] \rhd \Delta[v/x]$ and $\Delta[v/x]$ is well-asserted.*

Proof. The proof is by on the validation rules. We proceed by case analysis o the rules in Figure 4.

Case $\lfloor \textsc{Sel} \rfloor$. We set $P = s[\mathtt{p}, \mathtt{q}]!\{e_i \mapsto l_i\langle e_i'\rangle(x_i)\langle E_i\rangle; P_i\}_{i \in I}$ and $\Delta = \Delta', s[\mathtt{p}] : \mathtt{q}!\{l_j(x_j : U_j)\{A_j\}\langle E_j\rangle.\mathcal{L}_j\}_{j \in J}$. We first assume $x : S$. By $\lfloor \textsc{Sel} \rfloor$:

$$\frac{\begin{array}{l}\forall i \in I \exists j \in J\ l_i = l_j \quad \mathcal{I} \wedge \mathcal{C} \wedge e_i \supset (A_j \wedge (E_i = E_j))[e_i'/x_i] \\ \mathcal{I}; \mathcal{C} \wedge \#e_i; \Gamma \vdash P_i[e_i'/x_i] \rhd \Delta', s[\mathtt{p}] : \mathcal{L}_j[e_i'/x_j]\end{array}}{\mathcal{I}; \mathcal{C}; \Gamma \vdash P \rhd \Delta', s[\mathtt{p}] : \mathtt{q}!\{l_j(x_j : U_j)\{A_j\}\langle E_j\rangle.\mathcal{L}_j\}_{j \in J}} \tag{1}$$

Without loss of generality we assume $x \notin \{x_j\}_{j \in J}$. The first premise of (1) entails the following predicate

$$(\mathcal{I} \wedge \mathcal{C} \wedge e_i \supset (A_j \wedge (E_i = E_j))[e_i'/x_j])[v/x] \tag{2}$$

since the free occurrences of x (if any) in the first premise of (1) are supposed to be universally quantified. By definition, (2) is equivalent to

$$(\mathcal{I} \wedge \mathcal{C} \wedge e_i)[v/x] \supset (A_j \wedge (E_i = E_j))[e_i'/x_j][v/x] \tag{3}$$

Moreover, by inductive hypothesis, we have

$$\mathcal{I}; \mathcal{C}[v/x]; \Gamma \vdash P_i[e_i'/x_i][v/x] \rhd (\Delta', s[\mathtt{p}_1] : \mathcal{L}_j)[v/x] \tag{4}$$

By applying $\lfloor \textsc{Sel} \rfloor$ with premises (3) and (5) we obtain the thesis. If $x : \langle \mathcal{L} \rangle$ the case is similar except x does not need to be substituted to the predicates.

$$\frac{}{\langle \Gamma, [\Delta, s[\mathtt{p}] : \mathtt{q}!l\langle v\rangle; \mathcal{L}]\sigma\rangle \xrightarrow{s[\mathtt{p},\mathtt{q}]!l\langle v\rangle} \langle \Gamma, [\Delta, s[\mathtt{p}] : \mathcal{L}]\sigma\rangle} \quad \lfloor \text{T1} \rfloor$$

$$\frac{j \in I \qquad \sigma \models A_j[\mathtt{n}/x_j] \downarrow \mathsf{true} \quad \sigma' = \sigma \,\mathtt{after}\, E_j[\mathtt{n}/x_j]}{\langle \Gamma; [s[\mathtt{p}] : \mathtt{q}!\{l_i(x_i : U_i)\{A_i\}\langle E_i\rangle.\mathcal{L}_i\}_{i\in I}]\sigma\rangle \xrightarrow{\tau} \langle \Gamma; [s[\mathtt{p}] : \mathtt{q}!l_j\langle \mathtt{n}\rangle; \mathcal{L}_j[v/x_j]]\sigma'\rangle} \quad \lfloor \text{T2} \rfloor$$

$$\frac{j \in I \qquad \sigma \models A_j[\mathtt{n}/x_j] \downarrow \mathsf{true} \quad \sigma'' = \sigma' \,\mathtt{after}\, E_j[\mathtt{n}/x_j]}{\begin{array}{r}\langle \Gamma; [s[\mathtt{p}] : \mathtt{q}!l_j\langle \mathtt{n}\rangle; \mathcal{L}]\sigma, [s[\mathtt{q}] : \mathtt{p}?\{l_i(x_i : U_i)\{A_i\}\langle E_i\rangle.\mathcal{L}_i\}_{i\in I}]\sigma'\rangle \\ \xrightarrow{\tau} \langle \Gamma; [s[\mathtt{p}] : \mathcal{L}]\sigma, [\mathcal{L}_j[\mathtt{n}/x_j]]\sigma''\rangle\end{array}} \quad \lfloor \text{T3} \rfloor$$

$$\frac{j \in I \qquad \sigma \models A_j \downarrow \mathsf{true} \quad \sigma' = \sigma \,\mathtt{after}\, E_j \quad U_j = \langle \mathcal{L}\rangle}{\begin{array}{r}\langle \Gamma; [s[\mathtt{p}] : \mathtt{q}!\{l_i(x_i : U_i)\{A_i\}\langle E_i\rangle.\mathcal{L}_i\}_{i\in I}, t[\mathtt{r}] : \mathcal{L}]\sigma\rangle \\ \xrightarrow{\tau} \langle \Gamma; [s[\mathtt{p}] : \mathtt{q}!l_j\langle t[\mathtt{r}]\rangle; \mathcal{L}_j]\sigma'\rangle\end{array}} \quad \lfloor \text{T4} \rfloor$$

$$\frac{j \in I \qquad \sigma \models A_j \downarrow \mathsf{true} \quad \sigma'' = \sigma' \,\mathtt{after}\, E_j \quad U_j = \langle \mathcal{L}\rangle}{\begin{array}{r}\langle \Gamma; [s[\mathtt{p}] : \mathtt{q}!l_j\langle v\rangle; \mathcal{L}', t[\mathtt{r}] : \mathcal{L}]\sigma, [s[\mathtt{q}] : \mathtt{p}?\{l_i(x_i : U_i)\{A_i\}\langle E_i\rangle.\mathcal{L}_i\}_{i\in I}]\sigma'\rangle \\ \xrightarrow{\tau} \langle \Gamma; [s[\mathtt{p}] : \mathcal{L}']\sigma, [\mathcal{L}_j, t[\mathtt{r}] : \mathcal{L}]\sigma''\rangle\end{array}} \quad \lfloor \text{T5} \rfloor$$

Fig. 11. Labelled transition for message assertions

Case $\lfloor$BCH$\rfloor$. We set $P = s[\mathtt{p},\mathtt{q}]?\{l_i(x_i)\langle E_i\rangle.P_i\}_{i\in I}$ and $\Delta = \Delta', s[\mathtt{p}_2] : \mathtt{p}_1?\{l_i(x_i : U_i)\{A_i\}\langle E_i\rangle.\mathcal{L}_j\}_{i\in I}$. By rule $\lfloor$BCH$\rfloor$ (we omit the case for delegation acceptance as it is similar) and assume $x : S'$:

$$\frac{\mathcal{I}; \mathcal{C} \wedge A_i; \Gamma \vdash P_i \rhd \Delta, s[\mathtt{p}_2] : \mathcal{L}_i}{\mathcal{I}; \mathcal{C}; \Gamma \vdash P \rhd \Delta', s[\mathtt{p}_2] : \mathtt{p}_1?\{l_i(x_i : U_i)\{A_i\}\langle E_i\rangle.\mathcal{L}_j\}_{i\in I}}$$

Without loss of generality we assume $x \notin \{x_i\}_{i\in I}$. By induction

$$\mathcal{I}; (\mathcal{C} \wedge A_i)[v/x]; \Gamma \vdash P_i[v/x] \rhd (\Delta', s[\mathtt{p}_2] : \mathtt{p}_1?\{l_i(x_i : S_i)\{A_i\}\langle E_i\rangle.\mathcal{L}_j\}_{i\in I})[v/x] \tag{5}$$

By applying (5) as a premise for $\lfloor$BCH$\rfloor$ we obtain the thesis.

Case $\lfloor$MREQ$\rfloor$ *(resp.* $\lfloor$MACC$\rfloor$*).* This case follows straightforwardly by induction. The case for $\lfloor$MACC$\rfloor$ is similar.

Case $\lfloor$VAR$\rfloor$. We set $P = X\langle e\rangle$. By $\lfloor$VAR$\rfloor$

$$\frac{\mathcal{L}_1[e/y]...\mathcal{L}_n[e/y] \text{ well-asserted}}{\mathcal{I}; \mathcal{C}; \Gamma, X : (y : S')\mathcal{L}_1 @ \mathtt{p}_1 .. \mathcal{L}_n @ \mathtt{p}_n \vdash X\langle e\rangle \rhd \Delta', s[\mathtt{p}_1] : \mathcal{L}_1[e/y], .., s[\mathtt{p}_n] : \mathcal{L}_n[e/y]}$$

Without loss of generality we assume $x \neq y$. Since $\mathcal{L}_1[e/y]...\mathcal{L}_n[e/y]$ are well-typed wrt the underlying typing discipline, $x : S, y : S'$ and $v : S$ then $\mathcal{L}_1[e/y][v/x]...\mathcal{L}_n[e/y][v/x]$ are also well-typed. $\mathcal{L}_1[e/y][v/x]...\mathcal{L}_n[e/y][v/x]$ are well-asserted by Lemma 10. By applying $\mathcal{L}_1[e/y][v/x]...\mathcal{L}_n[e/y][v/x]$ as a premise of $\lfloor$VAR$\rfloor$ we obtain the thesis.

Remaining Cases The other case are straightforward.

Evaluation

Lemma 2 (Evaluation). *If* $\mathcal{I};\mathcal{C};\Gamma \vdash P(e) \rhd \Delta(e)$ *and* $\sigma \models e \downarrow v$ *for a* σ *s.t.* $\sigma \models \mathcal{I}$ *then we have* $\mathcal{I};\mathcal{C};\Gamma \vdash P[e/v] \rhd \Delta[e/v]$.

Proof. The proof is by induction on the validation rules. We proceed by case analysis. By decidability of underlying logic, we can write $\sigma \models A[e/x] \downarrow \mathsf{true}$ when a closed formula $A[e/x]$ evaluates to true. Note that if we further have $e \downarrow$ then we have $A[v/x] \downarrow \mathsf{true}$.

Case [SEL]. If $P(e) = s[\mathtt{p},\mathtt{q}]!\{e_i \mapsto l_i\langle e_i'\rangle(x_j)\langle E_j\rangle; P_j(e)\}_{i\in J}$ then $P(v) = s[\mathtt{p},\mathtt{q}]!\{e_i \mapsto l_i\langle e_i'\rangle(x_j)\langle E_j\rangle; P_j(v)\}_{i\in J}$ and

$$\Delta(e) = \Delta = \Delta'(e), s[\mathtt{p}_1] : \mathtt{p}_2!\{l_i(x_i : S_i)\{A_i(e)\}\langle E_i(e)\rangle.\mathcal{L}_j(e)\}_{i\in I}$$

with and $\mathcal{I} \wedge \mathcal{C} \wedge e_i \supset (A_j \wedge (E_i = E_j))[e/x_j]$. Notice that $\mathcal{I} \wedge \mathcal{C} \wedge e_i \supset (A_j \wedge (E_i = E_j))[e/x_j]$ is equivalent to

$$\mathcal{I} \wedge \mathcal{C} \wedge e_i \supset (A_j \wedge (E_i = E_j))[v/x_j] \tag{6}$$

By inductive hypothesis

$$\mathcal{I};\mathcal{C};\Gamma \vdash P'[v/e] \rhd \Delta'[v/e], s[\mathtt{p}] : \mathcal{L}[v/e] \tag{7}$$

By applying (6) and (7) to the validation rule [SEL] the lemma holds for this case.

Recursion Invocation If $P(e) = X\langle e\rangle$ (since $P(e)$ is well-formed against Δ by hypothesis) then $P(v) = X\langle v\rangle$. Since the substituted specification is still well-asserted (as it does not contain expressions) then $P(v)$ is well-formed against $\Delta[v/x]$ by rule [VAR].

Remaining cases The remaining cases are similar to the previous ones or straightforward by induction.

Other lemmas

Lemma 3 (Assertion Reduction and Coherence). *If* Δ *is coherent and* $\langle\Gamma; [\Delta]\sigma\rangle \xrightarrow{\tau} \langle\Gamma'; [\Delta']\sigma'\rangle$ *then* Δ' *is again coherent.*

Lemma 4 (Subject Congruence). *If* $\mathcal{I};\mathsf{true};\Gamma \vdash P_1 \rhd \Delta$ *and* $P_1 \equiv P_2$ *then* $\mathcal{I};\mathsf{true};\Gamma \vdash P_2 \rhd \Delta$

E.2 Soundness Proof

Theorem 6 (Soundness for Initial Networks) follows immediately from Lemma 6 (Soundness for open Networks), via Lemma 5. Lemma 5 shows there is a conditional simulation between the closing substitution of each single open validated located process and its corresponding specification. Recall that in the derivation of an open located process $\mathcal{C}$ may not be true, and we take a closing substitution consistent with $\mathcal{C}$ and Γ).

Lemma 5. *Let $\mathcal{R}$ be a relation collecting all pairs of the form $([P\tilde{\sigma}]\sigma_p; \langle \Gamma\tilde{\sigma}; [\Delta\tilde{\sigma}]\sigma_a\rangle)$ such that $\mathcal{I}; \mathcal{C}; \Gamma \vdash P \rhd \Delta$ where:*

1. *$[P]\sigma_p$ is a sub-term of a multi-step $\xrightarrow{\ell}$-derivative of a located process,*
2. *$[\Delta]\sigma_a$ is an assertion assignment with state,*
3. *$(\_)\tilde{\sigma}$ is a closing substitution of interaction variables consistent with $\mathcal{C}$, Γ and Δ,*
4. *$\sigma_p \models \mathcal{I}$ and $\sigma_a \models \mathcal{I}$*

Then $\mathcal{R}$ is a conditional stateful simulation.

Proof. We show that $\mathcal{R}$ is a conditional stateful simulation by induction on the depth of the validation tree. We proceed by case analysis of the last rule applied.

Case [MREQ] *(resp.* [MACC]*).* In this case P is defined as $\overline{a}[\mathtt{n}](y).P'$. The last derivation rule for P is [MREQ] where $\Gamma = \Gamma', a : \mathcal{G}$

$$\frac{\mathcal{G} \upharpoonright \mathtt{1} = [\tilde{\mathtt{x}} : \tilde{S}]\{A\}.\mathcal{L} \qquad \mathcal{I}; \mathcal{C}; \Gamma \vdash P' \rhd y[\mathtt{1}] : \mathcal{L}, \Delta}{\mathcal{I}; \mathcal{C}; \Gamma \vdash \overline{a}[\mathtt{n}](y).P' \rhd \Delta} \tag{8}$$

The only possible transition of $P\tilde{\sigma} = \overline{a}[\mathtt{n}](y).P'\tilde{\sigma}$ is by [TR-MREQ] in Figure 6. Notice that by $\sigma_a \models \mathcal{I}$ (condition (4) in the hypothesis).

By [TR-MREQ]:

$$[\overline{a}[\mathtt{n}](y).P'\tilde{\sigma}]_{\sigma_p} \xrightarrow{\overline{a}[\mathtt{n}]\langle s\rangle} [P'\tilde{\sigma}[s/y]]_{\sigma_p}$$

$\langle \Gamma\tilde{\sigma}; [\Delta\tilde{\sigma}]\sigma_a\rangle$ can move by [TR-A-MREQ] in Figure 8 since the first premise of [TR-A-MREQ] follows immediately from $\mathcal{G} \upharpoonright \mathtt{1} = [\tilde{\mathtt{x}} : \tilde{S}]\{A\}.\mathcal{L}$ (by first premise of (8)), and the second premise of [TR-A-MREQ] (i.e., $\sigma_a \models A$) holds by definition of $\mathcal{I}$:

$$\langle \Gamma\tilde{\sigma}; [\Delta\tilde{\sigma}]\sigma_a\rangle \xrightarrow{\overline{a}[n]\langle s\rangle} \langle \Gamma\tilde{\sigma}; [\Delta\tilde{\sigma}, s[\mathtt{1}] : \mathcal{L}\tilde{\sigma}]\sigma_a\rangle$$

$([P'\tilde{\sigma}]\sigma_p; \langle \Gamma\tilde{\sigma}; [\Delta\tilde{\sigma}, s[\mathtt{1}] : \mathcal{L}\tilde{\sigma}]\sigma_a\rangle) \in R$ by applying Lemma 1 to the third premise of (8), observing that the conditions (1÷4) are preserved. $\mathcal{R}$ is a conditional stateful simulation by induction. The case for [MACC] is similar.

Case [BCH]. In this case P is defined as $s?\{l_i(x_i)\langle E_i\rangle.P_i\}_{i\in I}$. The last derivation rule for P is [BCH]:

$$\frac{\begin{array}{c}\forall i \in I \quad \mathcal{I}; \mathcal{C} \wedge A_i; \Gamma \vdash P_i \rhd \Delta' \\ \text{if } U_i = \langle \mathcal{L}\rangle \text{ then } \Delta' = \Delta, s[\mathtt{q}] : \mathcal{L}_i, x_i : \mathcal{L} \text{ otw } \Delta' = \Delta, s[\mathtt{q}] : \mathcal{L}_i\end{array}}{\mathcal{I}; \mathcal{C}; \Gamma \vdash s[\mathtt{p}, \mathtt{q}]?\{l_i(x_i)\langle E_i\rangle.P_i\}_{i\in I} \rhd \Delta', s[\mathtt{q}] : \mathtt{p}?\{l_i(x_i : U_i)\{A_i\}\langle E_i\rangle.\mathcal{L}_i\}_{i\in I}} \tag{9}$$

The possible transitions of $P\tilde{\sigma}$ are with label $s[\mathtt{p},\mathtt{q}]?l_j\langle v\rangle$ for some v by [TR-BCH] in Figure 6:

$$[s[\mathtt{p},\mathtt{q}]?\{l_i(x_i)\langle E_i\tilde{\sigma}\rangle.P_i\tilde{\sigma}\}_{i\in I}]\sigma_p \xrightarrow{s[\mathtt{p},\mathtt{q}]?l_j\langle v\rangle} [P_j\tilde{\sigma}[v/x_j]]\sigma'_p \quad \sigma'_p = \sigma_p \,\mathtt{after}\, E_j$$

By the shape of the specification in (9), $s[\mathtt{q}] : \mathtt{p}?\{l_i(x_i : U_i)\{A_i\}\langle E_i\rangle.\mathcal{L}_i\}_{i\in I}$ is able to make a move at subject $s?$. By definition of conditional simulation we are interested in inspecting only the case in which the specification can make a step with label $s[\mathtt{p},\mathtt{q}]?l_j\langle v\rangle$. If the specification moves with label $s[\mathtt{p},\mathtt{q}]?l_j\langle v\rangle$ we have two cases:

- Case $U_j = S$. Hence, by [TR-A-BCH] in Figure 8:

  $$\langle\Gamma\tilde{\sigma}; [s[\mathtt{q}] : \mathtt{p}?\{l_i(x_i : U_i)\{A_i\tilde{\sigma}\}\langle E_i\tilde{\sigma}\rangle.\mathcal{L}_i\tilde{\sigma}\}_{i\in I}]\sigma_a\rangle \xrightarrow{s[\mathtt{p},\mathtt{q}]?l_j\langle v\rangle} \langle\Gamma\tilde{\sigma}; [s[\mathtt{q}] : \mathcal{L}_j\tilde{\sigma}]\sigma'_a\rangle$$

  with

  $$\sigma'_a = \sigma_a \,\mathtt{after}\, E_j \tag{10}$$

  $([P_j\tilde{\sigma}]\sigma'_p; \langle\Gamma\tilde{\sigma}; \langle\Delta\tilde{\sigma}, s[\mathtt{q}] : \mathcal{L}_j\tilde{\sigma}\rangle\sigma'_a\rangle) \in R$ holds observing that the conditions (1÷4) are preserved. Notice that condition (4) follows by invariant stability. $\mathcal{R}$ is a conditional stateful simulation by induction.
- Case $U_j = \langle\mathcal{L}\rangle$. In this case $v = t[\mathtt{r}]$. This case is similar to the previous one except Δ moves by [TR-A-DELBCH] in Figure 8:

  $$\langle\Gamma\tilde{\sigma}; [s[\mathtt{q}] : \mathtt{p}?\{l_i(x_i : U_i)\{A_i\tilde{\sigma}\}\langle E_i\tilde{\sigma}\rangle.\mathcal{L}_i\tilde{\sigma}\}_{i\in I}]\sigma_a\rangle \xrightarrow{s[\mathtt{p},\mathtt{q}]?l_j\langle t[\mathtt{r}]\rangle} \langle\Gamma\tilde{\sigma}; [s[\mathtt{q}] : \mathcal{L}_j\tilde{\sigma}, t[\mathtt{r}] : \mathcal{L}]\sigma'_a\rangle$$

  with

  $$\sigma'_a = \sigma_a \,\mathtt{after}\, E_j \tag{11}$$

Case [SEL]. In this case P is defined as $s[\mathtt{p},\mathtt{q}]!\{e_i \mapsto l_i\langle e'_i\rangle(x_i)\langle E_i\rangle; P_i\}_{i\in I}$. The last derivation rule for P is [SEL]:

$$\frac{\begin{array}{c}\forall i \in I \exists j \in J \text{ s.t. } x_i = x_j \quad l_i = l_j \quad \mathcal{I} \wedge \mathcal{C} \wedge e_i \supset (A_j \wedge (E_i = E_j))[e'_i/x_i] \\ \mathcal{I}; \mathcal{C} \wedge \#e_i; \Gamma \vdash P[e'_i/x_i] \rhd \Delta', s[\mathtt{p}] : \mathcal{L}_j[e'_i/x_i] \\ \text{if } U_i = \langle\mathcal{L}\rangle \text{ then } \Delta = \Delta'', e'_i : \mathcal{L}'_i \text{ and } \Delta' = \Delta'' \text{ otw } \Delta' = \Delta\end{array}}{\mathcal{I}; \mathcal{C}; \Gamma \vdash s[\mathtt{p},\mathtt{q}]!\{e_i \mapsto l_i\langle e'_i\rangle(x_i)\langle E_i\rangle; P_i\}_{i\in I} \rhd \Delta, s[\mathtt{p}] : \mathtt{q}!\{l_j(x_j : U_j)\{A_j\}\langle E_j\rangle.\mathcal{L}_j\}_{j\in J}} \tag{12}$$

The only possible transition is by [TR-SEL] in Figure 6. Assume first that the value sent is of type S. By [TR-SEL] then $P\tilde{\sigma}$ performs the following transition

$$[s[\mathtt{p},\mathtt{q}]!l_i(x_i)\langle E_i\tilde{\sigma}\rangle.P_i\tilde{\sigma}]_{\sigma_p} \xrightarrow{s[\mathtt{p},\mathtt{q}]!l_i\langle v\rangle} [P_i\tilde{\sigma}[v/x_i]]_{\sigma'_p} \quad \sigma'_p = \sigma_p \,\mathtt{after}\, E_i$$

Notice that

$$\sigma_a \models A_j\tilde{\sigma}[v/x_i] \tag{13}$$

following by $x_i = x_j, \sigma_a \models \mathcal{I}$ (condition (4) in the hypothesis), $\mathcal{I} \wedge \mathcal{C} \wedge e_i \supset A_j[v/x_i]$ (second premise of (12) above), and the fact that $\_\tilde{\sigma}$ is consistent with $\mathcal{C}$ (condition (3) in the hypothesis).

By (13) as premise of rule [TR-A-SEL] in Figure 8:

$$\langle \Gamma\tilde{\sigma}; [\Delta\tilde{\sigma}, s[\mathtt{p}] : \mathtt{q}!\{l_j(x_j : U_j)\{A_j\tilde{\sigma}\}\langle E_j\tilde{\sigma}\rangle.\mathcal{L}_j\tilde{\sigma}\}_{j\in J}]\sigma_a\rangle \xrightarrow{s[\mathtt{p},\mathtt{q}]!l_j\langle v\rangle} \langle \Gamma\tilde{\sigma}; [\Delta\tilde{\sigma}, s[\mathtt{p}] : \mathcal{L}_j]\sigma'_a\rangle$$

with $l_i = l_j$ and

$$\sigma'_a = \sigma_a \,\mathtt{after}\, E_j\tilde{\sigma} \tag{14}$$

$([P_i\tilde{\sigma}]_{\sigma_p}; \langle \Gamma\tilde{\sigma}; [\Delta\tilde{\sigma}, s[\mathtt{p}] : \mathcal{L}_j\tilde{\sigma}]\sigma'_a\rangle) \in R$, observing that the conditions $(1 \div 4)$ are preserved. Notice that case (4) holds by invariant stability. $\mathcal{R}$ is a conditional stateful simulation by induction.

This case in which x_j is a session channel is similar to the previous one, except transition [TR-A-SEL] of P has a corresponding [TR-A-DELSEL] of $\langle \Gamma\tilde{\sigma}; [\Delta\tilde{\sigma}]\sigma_a\rangle$.

Case [EMPTY]. We can set $P = \mathbf{0}$; the property holds since there are no transitions.

Case [PAR]. A parallel process $P = P_1 \mid P_2$ can only make either independent actions or reductions involving only either P_1 or P_2 (no reductions due to communication between P_1 and P_2 as only one role can be played by one principal in each session instance). This case is direct from the induction hypothesis.

Case [VAR]. We set P to be $X\langle e\rangle$ with $\Gamma(X) = (x : S)\mathcal{L}_1 \,@\, \mathtt{p}_1..\mathcal{L}_n \,@\, \mathtt{p}_n$. $P\tilde{\sigma}$ is a process such that $\mathcal{I}; \mathcal{C}\tilde{\sigma}; \Gamma\tilde{\sigma} \vdash P\tilde{\sigma}[e/x] \triangleright \Delta\tilde{\sigma}$ where $\Delta\tilde{\sigma} = \Delta_0\tilde{\sigma}, s[1] : \mathcal{L}_1[e/x], ..., s[\mathtt{n}] : \mathcal{L}_n[e/x]$ is the closure of the endpoint assertion of P. The property follows from the cases for the other process types.

Cases [REC]. This case is proved by the standard syntactic approximation of a recursion. We can assume, in all derivations for processes in P, the application of [REC] only occurs in (the last steps of) a derivation. Assume that we have

$$\mathcal{I}; \mathcal{C}; \Gamma, X : (x : S)\mathcal{L}_1@\mathtt{p}_1..\mathcal{L}_n@\mathtt{p}_n \models P \triangleright s[\mathtt{p}_1] : \mathcal{L}_1..s[\mathtt{p}_n] : \mathcal{L}_n \tag{15}$$

Further we also assume

$$\mathcal{I}; \mathcal{C}; \Gamma, X : (x : S)\mathcal{L}_1@\mathtt{p}_1..\mathcal{L}_n@\mathtt{p}_n \models Q \triangleright \Delta \tag{16}$$

Let y range over interaction names and session channels. In the following we often use the notation for the substitution $Q[(y)R/X]$ which replaces each occurrence of $X\langle e\rangle$ with $R[e/y]$. Using well-guardedness of process variables in [18], we first approximate the recursion by the following hierarchy:

$$P^0 \stackrel{def}{=} P' \sim \mathbf{0} \quad P^1 \stackrel{def}{=} P[(x)P^0/X] \quad \ldots \quad P^{n+1} \stackrel{def}{=} P[(x)P^n/X]$$

Above P^0 is chosen as the process which is typed by the same typing as P and which has no visible action. For example, choosing a and s to be fresh, $P^0 \stackrel{def}{=} (\nu a : \mathcal{G})(a[2](y).P')$ then $P^0 \sim \mathbf{0}$. We also set $P^\omega = \mu X\langle y\rangle(x).P'$ to be the recursively defined agent itself.

In the conclusion of [REC] we abstract the process variable X by the μ construct. Instead, we replace each X in Q with $(y)P^0, (y)P^1, \ldots, (y)P^n$, and finally $(y)P^\omega$. We call the result $Q^0, Q^1, \ldots Q^n$, and Q^ω, where Q^ω is nothing but the term in the conclusion (after one-time unfolding which does not change the behaviour).

Now suppose that $\mathcal{I}; \mathcal{C}; \Gamma \vdash S \rhd \Delta$ is derivable and that $\mathcal{I}; \mathcal{C}_0; \Gamma_0 \vdash S_0 \rhd \Delta_0$ occurs in its derivation, hence $S0$ occurs in S. Suppose that also $\mathcal{I}; \mathcal{C}_0; \Gamma_0 \vdash S'_0 \rhd \Delta_0$. We can replace the occurrence of S_0 in S by S'_0, with the result written S', such that $\mathcal{I}; \mathcal{C}; \Gamma \vdash S' \rhd \Delta$ is derivable.

Using property, we first note that, for any $\langle \Gamma; [\Delta]\sigma_a \rangle$ and $\mathcal{C}$, we have $\mathcal{C}; \Gamma \models P^0 \rhd \Delta$. Thus we apply this to (15) and replace X in P by $(x)P^0$:

$$\mathcal{C}; \Gamma \models P^1 \rhd s[\mathtt{p}_1] : \mathcal{L}_1, \ldots, s[\mathtt{p}_n] : \mathcal{L}_n$$

This can again be used for (15) (noting the environment Γ can always be taken as widely as possible in [VAR]): $\mathcal{C}; \Gamma \models P^2 \rhd s[\mathtt{p}_1] : \mathcal{L}_1, \ldots, s[\mathtt{p}_n] : \mathcal{L}_n$. In this way we know that for an arbitrary n: $\mathcal{C}; \Gamma \models P^n \rhd s[\mathtt{p}_1] : \mathcal{L}_1, \ldots, s[\mathtt{p}_n] : \mathcal{L}_n$.

By applying this to (15) , we obtain:

$$\mathcal{C}; \Gamma \models Q^n \rhd \Delta$$

for an arbitrary n. Now assume, for simplicity, that there are no free variables in Q (hence in Q^n) and therefore $\mathcal{C} = \mathsf{true}$ (the reasoning is precisely the same by applying a closing substitution). We can then construct a relation taking each node in the transitions from Q^ω and relating it to the derivative of $\langle \Gamma; [\Delta]\sigma_a \rangle$, by observing that assertions transitions are always deterministic for the given process and its transition derivatives. Let the resulting relation be $\mathcal{R}$. Since any finite trace of Q^ω is in some Q^n, the conditions of conditional simulation hold at each step.

Lemma 6 (Soundness for Open Networks). *Let N be a network. Then $\mathcal{C}; \Gamma \vdash N \rhd \Sigma$ implies $\mathcal{C}; \Gamma \models N \rhd \Sigma$*

Proof. Let $\mathcal{R}$ be a relation collecting all pairs of the form $(N; \Sigma)$ such that $\mathcal{C}; \Gamma \vdash N \rhd \Sigma$ where: (i) N is a sub-term of a multi-step $\xrightarrow{\ell}$-derivative of an initial network, (ii) Σ is a specification. Proceeding by induction on the length of the derivation tree. We proceed by case analysis of the validation rules for networks in Figure 4 and in Figure 10. Subject reduction for silent actions (Lemma 9)

Case [N1]. If [N1] is applied then $N = [P_i]\sigma_a$, $\Sigma = [\Delta']\sigma_p$, and

$$\mathcal{I}; \mathcal{C}; \Gamma \vdash P \rhd \Delta$$

This case follows by Lemma 5, observing that by premise fourth condition ($\sigma_a \models \mathcal{I}$ and $\sigma_p \models \mathcal{I}$) holds by premise of [N1].

Case [N2]. This case follows by definition of refinement.

Case [N3]. This case is immediate since $N = \varnothing$ thus cannot make any transition.

Case [N4]. A parallel network $N = N_1 \mid N_2$ can make either independent actions or reductions. The case for independent actions is direct from the induction hypothesis. If the reduction takes place by interaction, then we use Lemma 9.

Cases [QNIL], [QVAL]. Queues do not have transitions. The behaviours of queues are taken into account as part of τ-actions in the case for [N3] above.

Case [CRES]. This case follows by Lemma 3.

F Subject Reduction Proofs

Lemma 7. *If $N \longrightarrow N'$ then one of the following cases hold:*

1. $N \equiv \mathcal{E}[\prod_{i\in\{1..n\}}[P_i]\sigma_i]$ *with* $P_1 = \overline{a}[\mathtt{n}](y_1).P_1' \mid Q_1$ *and* $P_i = a[\mathtt{i}](y_i).P_i' \mid Q_i$ *s.t.*
$$[P_1]\sigma_1 \xrightarrow{\overline{a}[\mathtt{n}]\langle s\rangle} [P_1' \mid Q_1]\sigma_1 \qquad [P_i]\sigma_i \xrightarrow{\overline{a}[\mathtt{i}]\langle s\rangle} [P_i' \mid Q_i]\sigma_i$$
and $N' = \mathcal{E}[(\nu s)(s : \varnothing \mid \prod_{i\in\{1..n\}}[P_i' \mid Q_i]\sigma_i)]$
2. $N \equiv \mathcal{E}[[P]\sigma \mid s : h]$ *s.t.* $[P]\sigma \xrightarrow{s[\mathtt{p},\mathtt{q}]!l\langle v\rangle} [P']\sigma$ *and* $N' \equiv \mathcal{E}[[P']\sigma \mid s : h \cdot (\mathtt{p},\mathtt{q}, l\langle v\rangle)]$
3. $N \equiv \mathcal{E}[[P]\sigma \mid s : (\mathtt{p},\mathtt{q}, l\langle v\rangle) \cdot h]$ *s.t.* $[P]\sigma \xrightarrow{s[\mathtt{p},\mathtt{q}]?l\langle v\rangle} [P']\sigma$ *and* $N' \equiv \mathcal{E}[[P']\sigma \mid s : h]$

Proof. Immediate from the corresponding reduction rules.

Lemma 8. *1. If $[P]\sigma \xrightarrow{a[\mathtt{n}]\langle s\rangle} [P']\sigma$ and $\mathcal{C};\Gamma \vdash [P]\sigma \rhd \Sigma$ then $\Sigma = [\Delta]\sigma'$ for some Δ, σ', and $\mathcal{C};\Gamma \vdash [P']\sigma \rhd [\Delta, s[\mathtt{1}] : \mathcal{L}]\sigma'$*

2. *If $[P]\sigma \xrightarrow{a[\mathtt{i}]\langle s\rangle} [P']\sigma$ and $\mathcal{C};\Gamma \vdash [P]\sigma \rhd \Sigma$ then $\Sigma = [\Delta]\sigma'$ for some Δ, σ', and $\mathcal{C};\Gamma \vdash [P']\sigma \rhd [\Delta, s[\mathtt{i}] : \mathcal{L}]\sigma'$*
3. *If $[P]\sigma_p \xrightarrow{s[\mathtt{p},\mathtt{q}]!l\langle v\rangle} [P']\sigma_p'$ and $\mathcal{C};\Gamma \vdash [P]\sigma_p \mid s : h \rhd \Sigma$ then $\Sigma = [\Delta]\sigma_a$ for some Δ, σ_a and $\mathcal{C};\Gamma \vdash [P']\sigma_p \mid s : h \cdot (\mathtt{p},\mathtt{q}, l\langle v\rangle) \rhd [\Delta']\sigma_a'$ s.t. $\langle\Gamma; [\Delta]\sigma_a\rangle \xrightarrow{\tau} \langle\Gamma; [\Delta']\sigma_a'\rangle$*
4. *If $[P]\sigma_p \xrightarrow{s[\mathtt{p},\mathtt{q}]?l\langle v\rangle} [P']\sigma_p'$ and $\mathcal{C};\Gamma \vdash [P]\sigma \mid s : (\mathtt{p},\mathtt{q}, l\langle v\rangle) \cdot h \rhd \Sigma$ then $\Sigma = [\Delta]\sigma'$ for some Δ, σ', and either*
   - *Σ can move at subject $s?[\mathtt{p},\mathtt{q}]$ but cannot move with label $s[\mathtt{p},\mathtt{q}]?l\langle v\rangle$*
   - *$\mathcal{C};\Gamma \vdash [P']\sigma_p \mid s : h \rhd [\Delta']\sigma_a$ s.t. $\langle\Gamma; [\Delta]\sigma_a\rangle \xrightarrow{\tau} \langle\Gamma; [\Delta']\sigma_a'\rangle$.*

Proof. (1) and (2) are immediate. Below we show the cases (3) and (4).

Case (3) Suppose we have $\mathcal{I}; \mathsf{true}; \Gamma \vdash P \mid s : h \rhd \Delta$. We safely assume the last rule applied is [PAR], thus we can assume $\Delta = \Delta_0, \Delta_1$ for some Δ_0 and Δ_1, and

$$\mathcal{I};\mathcal{C};\Gamma \vdash P \rhd \Delta_0 \tag{17}$$

Now consider the transition $[P]\sigma_p \xrightarrow{s[\mathtt{p},\mathtt{q}]!l_i\langle v\rangle} [P']\sigma_p'$, by (17) we observe Δ_0 has the shape

$$\Delta_0 = s[\mathtt{p}] : \mathtt{q}!\{l_j(x_j : U_j)\langle E_j\rangle\{A_j\}; \mathcal{L}_j\}_{j\in J}\}, \Delta_{00}$$

and that P' can be typed by Δ_0' such that:

$$\Delta_0' = s[\mathtt{p}] : \mathcal{L}_j[v/x_i], \Delta_{00} \tag{18}$$

Now the assertion Δ_1 for the queue has the shape, omitting the vacuous "end": $\Delta_1 = s : \mathcal{M}$ hence the addition of the values to this queue, $s : h \cdot (\mathtt{p}, \mathtt{q}, l_i\langle v\rangle)$, must have the endpoint assertion:

$$\Delta_1' = s[\mathtt{p}] : \mathtt{q}!l_i\langle v\rangle; \mathcal{M} \tag{19}$$

Setting $\Delta' = \Delta_0', \Delta_1'$, we know $\mathcal{I}; \mathsf{true}; \Gamma \vdash P' \mid s : h \cdot (\mathtt{p}, \mathtt{q}, l_i\langle v\rangle) \rhd \Delta'$. By (18) and (19) we obtain

$$\Delta_0', \Delta_1' = s[\mathtt{p}] : \mathtt{q}!l_i\langle v\rangle; \mathcal{L}_j[v/x_i], \Delta_{00}, \Delta_1$$

and

$$\langle \Gamma; \Delta_0, \Delta_1\rangle \xrightarrow{\tau} \langle \Gamma; \Delta_0', \Delta_1'\rangle$$

and the only change is at the assertion assignment at $s[\mathtt{p}]$, as required.

Case (4). Suppose we have $\mathcal{I}; \mathsf{true}; \Gamma \vdash P \mid s : h \cdot (\mathtt{p}, \mathtt{q}, l_i\langle v\rangle) \rhd \Delta$. Again we safely assume the last rule applied is [PAR]. Thus we can assume, for some Δ_0 and Δ_1:

$$\mathcal{I}; \mathsf{true}; \Gamma \vdash s : h \cdot (\mathtt{p}, \mathtt{q}, l_i\langle v\rangle) \rhd \Delta_1$$

with $\Delta = \Delta_0, \Delta_1$, and

$$\mathcal{I}; \mathsf{true}; \Gamma \vdash P \rhd \Delta_0 \tag{20}$$

Now consider the transition

$$[P]\sigma_p \xrightarrow{s[\mathtt{p},\mathtt{q}]?l_j\langle v\rangle} [P']\sigma_p' \tag{21}$$

As before, we can infer, from (20) and (21) the shape of Δ_0 as follows,

$$\Delta_0 = s[\mathtt{q}] : \mathtt{p}?\{l_j(x_i : U_i)\langle E_i\rangle\{A_i\}; \mathcal{L}_i\}_{i\in I}, \Delta_{00}$$

for some $\mathtt{p}$; and that P' can be validated against Δ_0' given as

$$\Delta_0' = s[\mathtt{q}] : \mathcal{L}[v/x_j], \Delta_{00} \tag{22}$$

Now the assertion Δ_1 for the queue has the shape (again omitting "end"-only assertions):

$$\Delta_1 = s[\mathtt{q}] : \mathtt{p}?l_j\langle v\rangle; \mathcal{M} \tag{23}$$

which, if we take off the values (hence for the queue $s : h$), we obtain:

$$\Delta_1' = s[\mathtt{q}] : \mathcal{M} \tag{24}$$

Note this is symmetric to the case (1) above. As before, setting $\Delta' = \Delta_0', \Delta_1'$, we know: $\mathcal{I}; \mathcal{C}; \Gamma \vdash P' \mid s : h \rhd \Delta'$. By (22) and (24) we obtain

$$\begin{array}{l} \Delta_0', \Delta_1' = s[\mathtt{q}] : \mathcal{L}_j[v/x_j],\ \Delta_{00},\ \Delta_1 \\ \qquad \xleftarrow{\tau} \Delta_0, \Delta_1 \end{array}$$

The only change from Δ to Δ' is at the type assignment at $s[\mathtt{q}]$, as required.

For convenience of the case analysis we explicitly write $P \xrightarrow{\tau_s} P'$ if $P \xrightarrow{\tau} P'$ is derived by the reduction rules for free session channels.

Lemma 9 (Subject Reduction for Silent Actions). *Suppose* $\Gamma \vdash N \rhd \Sigma$.

1. *if* $N \xrightarrow{\tau} N'$ *then* $\Gamma \vdash N' \rhd \Sigma$ *again*
2. *if* $N \xrightarrow{\tau_s} N'$ *then there exists* Σ' *s.t.* $\langle \Gamma, \Sigma \rangle \xrightarrow{\tau} \langle \Gamma, \Sigma' \rangle$ *and* $\Gamma \vdash N' \rhd \Sigma'$.

Proof. If $N \xrightarrow{\tau} N'$ then each of the cases of Lemma 7 are possible, we inspect them one by one.

Case (1): Session Initiation. By Lemma 7 (1) and (2) we set $N = [P_1]\sigma_1 \mid \prod_{2 \geqslant i \geqslant n} [P_i]\sigma_i$ where $P_1 = \overline{a}[\mathtt{n}](y_1).P_1' \mid Q_1$ and $Q_i = a[\mathtt{i}](y_i).P_i' \mid Q_i$. As given in Lemma 7 (2) the actions of P_i compensate each others and correspond to reduction $N \longrightarrow (\nu s)(s : \varnothing \mid \prod_{1 \geqslant n \geqslant n} [P_i'[s[\mathtt{i}]/y_i] \mid Q_i]\sigma_i)$ by the first rule in Figure 7.

Since $\mathcal{E}[\_]$ is a reduction context we can safely set

$$\Gamma \vdash [a[\mathtt{i}](y_i)\{A_i\}.P_i]\sigma_i \rhd [\Delta_i]\sigma_i'$$

so that $[\Delta_1]\sigma_1', ..., [\Delta_n]\sigma_n' = \Sigma$.

Hence, by premise of validation rule [MACC] we have, with $\Gamma(a) = \mathcal{G}$,

$$\Gamma \vdash [P_i]\sigma_i \rhd [\Delta_i, s[\mathtt{i}] : \mathcal{L}_i]\sigma_i'$$

with $\mathcal{G} \upharpoonright_{\mathtt{i}} = \delta\{A_i\}\mathcal{L}_i$ (similarly for role 1).

Since $\{\mathcal{G} \upharpoonright_{\mathtt{i}}\}_{i \in I}$ is obviously coherent then $\Gamma \vdash (\nu s)(\prod_i P_i'[s\mathtt{i}]/y] \mid Q_i) \rhd \Delta$ as required.

Case (2): Select. By Lemma 7 (3) we set $N \equiv \mathcal{E}[[P]\sigma_p \mid s : h]$ with $[P]_{\sigma_p} \xrightarrow{s[\mathtt{p},\mathtt{q}]!l\langle v \rangle} [P']_{\sigma_p'}$. As above we can safely set $\mathcal{I}; \mathsf{true}; \Gamma \vdash [Q]_{\sigma_p'} \mid s : \tilde{h} \rhd \Delta$. By Lemma 8 we can infer $\mathcal{I}; \mathsf{true}; \Gamma \vdash [P']_{\sigma_p'} \mid s : \tilde{h} \cdot (\mathtt{p}, \mathtt{q}, l\langle v \rangle) \rhd \Delta'$ such that $\langle \Gamma, [\Delta]\sigma_a \rangle \xrightarrow{\tau} \langle \Gamma, [\Delta']\sigma_a' \rangle$. Since N reduces to N' by τ-transition rather than τ_s transition, we know that s is hidden in N. Assume therefore without loss of generality

$$N \equiv C[(\nu s)([P]_\sigma \mid s : h \mid M)] \qquad \mathcal{I}; \mathsf{true}; \Gamma \vdash [P]_\sigma \mid s : h \mid M \rhd \Sigma_1$$

with Σ_1 coherent and $\Sigma_1 = \Sigma, \Sigma_{01}$. By Lemma 3 and $\Sigma_1 \longrightarrow \Sigma', \Sigma_{01}$ we know Σ', Σ_{01} is also coherent, hence done.

Case (3): Branch. The argument exactly follows case (2) above except using Lemma 7 (3) and Lemma 8 (4) instead of Lemma 7 (2) and Lemma 8 (3), respectively.

Case τ_s. Proceeds as above but without restricting s.

Lemma 10. *If* $\mathcal{I}; \mathcal{C}; \Gamma \vdash P \rhd \Delta$ *and* $[P]\sigma_p \xrightarrow{\ell} [P']\sigma_p'$ *for some* ℓ, P', σ_p *and* σ_p *s.t.* $\sigma_p \models \mathcal{I}$ *then:*

– *if ℓ is a branching action then $\langle \Gamma; [\Delta]\sigma_p \rangle$ is able to move at subject of ℓ, and if $\langle \Gamma; [\Delta]\sigma_p \rangle \xrightarrow{\ell} \langle \Gamma; [\Delta']\sigma'_a \rangle$ then we have $\mathcal{I}; \mathsf{true}; \Gamma \vdash P' \rhd \Delta'$.*
– *if ℓ is not a branching nor a τ action then $\langle \Gamma; [\Delta]\sigma_a \rangle \xrightarrow{\ell} \langle \Gamma; [\Delta']\sigma'_a \rangle$ then we have $\mathcal{I}; \mathsf{true}; \Gamma \vdash P' \rhd \Delta'$.*

Proof. The proof is by induction on the validation rules .We proceed by the case analysis depending on the last rule used for deriving this judgement. We assume processes are closed. Further below notice $\mathcal{C}$ in the conclusion of each rule should be true by our assumption.

Case [SEL]. In this case, we derive $\mathcal{I}; \mathsf{true}; \Gamma \vdash P \rhd \Delta$ with:

$$P = s[\mathtt{p}, \mathtt{q}]!\{e_i \mapsto l_i\langle e'_i \rangle(x_i)\langle E_i \rangle; P_i\}_{i \in j} \text{ and } \Delta = \Delta_0, s[\mathtt{p}] : \mathtt{q}!\{l_j(x_j : U_j)\{A_j\}\langle E_j \rangle; \mathcal{L}_j\}_{j \in J} \tag{25}$$

P can move only by [TR-SEL] in Figure 6: $[P]\sigma_p \xrightarrow{\ell} [P_i[v/x_i]]\sigma'_p$ with $\ell = s[\mathtt{p}, \mathtt{q}]!l_i\langle v \rangle$, $\sigma'_p = \sigma_p \, \mathtt{after} \, E_i$, and

$$\sigma_p \models e_i \downarrow v \tag{26}$$

since P is closed the only free variables in e_i are state variables defined in σ_p. By the first premise of validation rule [SEL] we have:

$$\mathcal{I} \supset A_j[e'_i/x_j] \tag{27}$$

By (26) and (27) we infer $\mathcal{I} \supset A_j[v/x_j]$.

From $\mathcal{I} \supset A[v/x_j]$, since $\sigma_a \models \mathcal{I}$ we have $\sigma_a \models A[v/x_j]$ hence $\langle \Gamma; [\Delta]\sigma_a \rangle$ can move by [TR-SEL]:

$$\langle \Gamma; [\Delta]\sigma_a \rangle \xrightarrow{\ell} \langle \Gamma; [\Delta_0, s[\mathtt{p}] : \mathcal{L}_j[v/x_j]]\sigma'_a \rangle$$

with $\sigma'_a = \sigma_a \, \mathtt{after} \, E_i$

By the third premise of validation [SEL] we have

$$\mathcal{I}; \mathsf{true}; \Gamma \vdash P_j[e'_i/x_i] \rhd \Delta_0, s[\mathtt{p}] : \mathcal{L}_j[e'_i/x_j] \tag{28}$$

By Evaluation Lemma, (28) immediately gives $\mathcal{I}; \mathsf{true}; \Gamma \vdash P_j[v/x_j] \rhd \Delta_0, s[\mathtt{p}] : \mathcal{L}[v/x_j]$ as required. This case holds by induction observing that $\sigma'_p \models \mathcal{I}$ and $\sigma'_a \models \mathcal{I}$ by invariant stability.

Case [BCH]. In this case the conclusion is $\mathcal{I}; \mathsf{true}; \Gamma \vdash P \rhd \Delta$ with:

$$P = s[\mathtt{p}, \mathtt{q}]?\{l_i(x_i)\langle E_i \rangle.P_i\}_{i \in I} \quad \text{and} \quad \Delta = \Delta_0, s[\mathtt{p}] : \mathtt{q}?\{l_i(x_i : U_i)\{A_i\}\langle E_i \rangle.\mathcal{L}_i\}_{i \in I} \tag{29}$$

By the shape of P we can set $\ell = s[\mathtt{q}, \mathtt{p}]?l_j\langle v \rangle$. P can move only by [TR-BCH], obtaining $[P]\sigma_p \xrightarrow{\ell} [P_j[v/x_j]]\sigma'_p$, with $\sigma'_p = \sigma_p \, \mathtt{after} \, E_j$. By the shape of Δ from the validation rule [BCH] we have that Δ is able to move at subject of ℓ. In case $\langle \Gamma; [\Delta]\sigma_a \rangle$ can move with label ℓ we have by [TR-BCH]:

$$\langle \Gamma, [\Delta]\sigma_a \rangle \xrightarrow{\ell} \langle \Gamma; [\Delta_0, s[\mathtt{p}] : \mathcal{L}[v/x_j]]\sigma'_a \rangle$$

with $\sigma'_a = \sigma_a \,\texttt{after}\, E_j$. for which $\sigma_a \models A[v/x_j]$. Now the premise of validation rule [BCH]:

$$\mathcal{I}; \mathsf{true} \wedge A_j; \Gamma \vdash P_j \rhd \Delta_0, s[\mathtt{p}] : \mathcal{L}$$

By Substitution Lemma we obtain

$$\mathcal{I}; \mathsf{true} \wedge A_j[v/x_j]; \Gamma \vdash P_j[v/x_j] \rhd \Delta_0, s[\mathtt{p}] : \mathcal{L}[v/x_j]$$

Since by history sensitivity A_j does not contain free state variables the it is possible to evaluate it. By $A_j[v/x_j] \downarrow \mathsf{true}$ and by validation rule [BCH] we obtain $\mathcal{I}; \mathsf{true}; \Gamma \vdash P_j[v/x_j] \rhd \Delta_0, s[\mathtt{p}] : \mathcal{L}[v/x_j]$ as required. This case holds by induction observing that $\sigma'_p \models \mathcal{I}$and $\sigma'_a \models \mathcal{I}$ by invariant stability.

Case [MREQ]. In this case we have $\mathcal{I}; \mathsf{true}; \Gamma \vdash P \rhd \Delta$ such that, combining with the premises of the rule [MREQ] we have: $P = \overline{a}[n](y).Q$ and $\mathcal{I}; \mathsf{true}; \Gamma \vdash Q \rhd \Delta, s[\mathtt{1}] : \mathcal{L}$ where

$$\Gamma(a) = \mathcal{G} \text{ and } \mathcal{G} \upharpoonright_1 = \delta\{A\}.\mathcal{L} \text{ and } \mathcal{I} \supset A \tag{30}$$

By the shape of P we can set $\ell = \overline{a}[\mathtt{n}]\langle s \rangle$ and $P \xrightarrow{\ell} Q$. By (30) we have $\mathcal{I} \supset A$ and by hypothesis $\sigma_a \models \mathcal{I}$ hence $\sigma_a \models A$. Therefore the following transition is possible using [TR-A-MREQ]: $\langle \Gamma, [\Delta]\sigma_p \rangle \xrightarrow{\ell} \langle \Gamma, [\Delta, s[\mathtt{1}] : \mathcal{L}]\sigma_a \rangle$ as required.

Case [MACC]. Similar to the case [MCAST] above.

Case [PAR]. Immediate, since the visible transition for $P \mid Q$ is reducible to the same action by either P or Q, and because the resulting assertion environments (one result of the visible transition) can again be composed, because linear compatibility only depends on channel names and participant names.

Case[REC]. This case follows from applying induction on the unfolding of P and folding it back after the transition.

Other cases. In each case, direct from the induction hypothesis.

Case [VAR]. Immediate since in this case there is no reduction from P.

Lemma 11. *If* $\mathsf{true}; \Gamma \vdash N \rhd \Sigma$ *and* $N \xrightarrow{\ell} N'$ *and* $\langle \Gamma; \Sigma \rangle \xrightarrow{\ell} \langle \Gamma; \Sigma' \rangle$ *where* $\ell \neq \tau$*, then we have* $\mathsf{true}; \Gamma \vdash N' \rhd \Sigma'$

Proof. The proof is by induction on the validation rules. The case for [N1] follows by Lemma 11. The cases for [N2] and [N3] are straightforward. We show below the case for [N4].

Suppose the conclusion is $\mathsf{true}; \Gamma \vdash N \rhd [\Delta]\sigma$ which is derived from

$$\mathsf{true}; \Gamma_0 \vdash N \rhd [\Delta_0]\sigma_0 \tag{31}$$

with $\Gamma_0, \Delta_0, \sigma_0 \ni \Gamma, \Delta, \sigma$. Now first suppose the concerned visible action ℓ is neither a receive action nor a branching. Now suppose $N \xrightarrow{\ell} N'$. By induction hypothesis and

by (31), $\langle \Gamma_0; [\Delta_0]\sigma_0 \rangle \xrightarrow{\ell} \langle \Gamma_0'; [\Delta_0']\sigma_0' \rangle$ for some Γ_0', Δ_0' and σ_0' for which we have, by induction hypothesis

$$\mathsf{true}; \Gamma_0' \vdash N' \rhd [\Delta_0']\sigma_0' \tag{32}$$

Since the assertion transition is deterministic and by definition of refinement $\Gamma_0', \Delta_0', \sigma_0' \ni \Gamma', \Delta', \sigma'$, by (32) we can use [N4] to reach the thesis.

G Completeness

Assuming $\Gamma \models [P]\sigma_p \rhd [\Delta_0]\sigma_a$, we introduce generation rules (Figure 12) to obtain a formula Δ parametric with respect to a number of predicate variables. Then we show that there exist a substitution ξ of the predicate variables in Δ such that: (1) $\mathsf{true}; \Gamma \vdash [P]\sigma_p \rhd [\Delta\xi]\sigma_a$ (i.e., provability of validation rules, Lemma 13), and (2) $\Delta\xi \ni \Delta_0$ (completeness via refinement, Lemma 14). The thesis is a consequence of (2) and of validation rule [N4].

Remark 1. We consider a more general formulation of our framework where the updates are expressed as predicates namely $E \stackrel{\text{def}}{=} E' \wedge \mathtt{x}' : A$ where $\mathtt{x}'$ denotes $\mathtt{x}$ after the update and A can refer to the values in the current state (e.g., $\mathtt{x}$). We can express the formulation in the paper, i.e., $E \stackrel{\text{def}}{=} E'; \mathtt{x} := e$ as $E_{gen} \stackrel{\text{def}}{=} E'_{gen} \wedge \{\mathtt{x}' = e\}$. Also, we consider recursive definitions that can have more than one recursion parameter.

G.1 Predicate Variables and Extended Predicates

The generation rules for principal formulae use sequents with predicate variables with fixed arities. We need predicate variables since we cannot rely on a specific predicate when we stipulate a constraint on an input value: if we concretize it, we may lose principality (i.e., the principal formula is not the strongest), since the stronger an input constraint is, the stronger a related output constraint is. Similarly we use predicate variables also for the updates of the branching.

Definition 8 (Extended predicates). *Extended predicates are defined as predicates where* predicate variables *can occur; predicate variables are ranged over by $\phi(x)$ and are meant to be replaced by normal predicates A such that $\mathsf{fn}(A) \subseteq x$ (similarly for $\phi(\mathtt{x})$).*

G.2 Generalised Sequent

We use the following sequents towards completeness, all using predicate variables.

1. $\mathcal{C}; \Gamma_0 \vdash_\star P \blacktriangleright \Delta$. This is used for generation of principal formulae and reads:
   "Under $\mathcal{C}$ as constraints on values and Γ_0 as public contracts for shared names, P has the principal formula Δ".
   Note predicates in these assertions use predicate variables, defined in Definiton 8.

2. $\mathcal{C}; \Gamma \vdash^{ext} N \rhd \Sigma$. This is the same provability as the one obtained using the validation rules in Figure 4 *except* using an extended syntax of predicates incorporating predicate variables (in both predicates and updates of the branching).

3. $\mathcal{C}; \Gamma \models^{ext} [P]\sigma_p \rhd [\Delta]\sigma_a$. Again this is the same satisfiability as we defined in Definition 5 *except* using the syntax of predicates incorporating predicate variables (the semantics of predicate variables is taken in the standard way, taking satisfiability under all closing substitutions).

In brief (1) is the sequent for generation described in G.4 while (2) and (3) are the sequents for validation/satisfiability obtained extending the logic with predicate variables.

As more clear later, for all possible concrete substitutions, (2) implies the normal provability and (3) implies satisfiability.

G.3 Two Merge Operations

This subsection is a technical discussion introducing and studying two merge operations used in the generation rules. This subsection is technical, needed only for the proofs of completeness, hence may as well be skipped until the proof of the theorem.

Convention 11 (shape of recursive assertions). In this subsection and henceforth we assume two recursive assertions to be merged are always in the same shape. Since the shape of recursive assertions to be generated rely on the shape of recursions in the original process, this assumption means semantically neutral assumption (up to a simple transformation) of recursions in processes. Since generated formulae are equivalent for different shapes of recursions, this does not lose generality.

Merging Assertions (1)

Definition 12 (Merge). Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two local assertions. The function $\between$ takes two local assertions and *merges* them; it is recursively defined as follows:

- $\mathtt{p}!\{l_i(x_i : U_i)\{A_i\}\langle E_i^1\rangle; \mathcal{L}_i\}_{i\in I} \between \mathtt{p}!\{l_i(x_i : U_i)\{B_i\}\langle E_i^2\rangle; \mathcal{L}'_i\}_{i\in I} \stackrel{def}{=} \mathtt{p}!\{l_i(x_i : U_i)\{A_i \vee B_i\}\langle E_i^1 \vee E_i^2\rangle; \mathcal{L}_i \between \mathcal{L}'_i\}_{i\in I}$
- $\mathtt{p}?\{l_i(x_i : U_i)\{A_i\}\langle E_i^1\rangle; \mathcal{L}_i\}_{i\in I} \between \mathtt{p}?\{l_i(x_i : U_i)\{B_i\}\langle E_i^2\rangle; \mathcal{L}'_i\}_{i\in I} \stackrel{def}{=} \mathtt{p}?\{l_i(x_i : U_i)\{A_i \wedge B_i\}\langle E_i^1 \wedge E_i^2\rangle; \mathcal{L}_i \between \mathcal{L}'_i\}_{i\in I}$
- $\mu \mathsf{t}\langle \tilde{y_1} : A\rangle(\tilde{x_1} : \tilde{U_1})\{\mathsf{true}\}.\mathcal{L}'_1 \between \mu \mathsf{t}\langle \tilde{y_2} : B\rangle(\tilde{x_2} : \tilde{U_2})\{\mathsf{true}\}.\mathcal{L}'_1$
  $\stackrel{def}{=} \mu \mathsf{t}\langle \tilde{y_1}, \tilde{y_2} : A \wedge B\rangle(\tilde{x_1}, \tilde{x_2} : \tilde{U_1}\tilde{U_2})\{\mathsf{true}\}.\mathcal{L}'_1 \between \mathcal{L}'_2$
  where we assume $\tilde{x_1} \neq \tilde{x_2}$ and $\tilde{y_1} \neq \tilde{y_2}$.
- $\mathsf{t}\langle \tilde{x_1} : A\rangle \between \mathsf{t}\langle \tilde{x_2} : B\rangle \stackrel{def}{=} \mathsf{t}\langle \tilde{x_1}\tilde{x_2} : A \wedge B\rangle$ where we assume $\tilde{x_1} \neq \tilde{x_2}$.
- $\mathsf{end} \between \mathsf{end} = \mathsf{end}$.

Definition 13. The merge operation is extended to assignments Δ in the obvious way, i.e. $\Delta_1 \between \Delta_2$ is the pointwise merge of Δ_1 and Δ_2.

Lemma 14. *1.* $(\Gamma, \Delta_1, \sigma) \ni (\Gamma, \Delta_1 \between \Delta_2, \sigma)$.
2. If $(\Gamma, \Delta_1, \sigma) \ni (\Gamma, \Delta', \sigma)$ *and* $(\Gamma, \Delta_2, \sigma) \ni (\Gamma, \Delta', \sigma)$ *then* $(\Gamma, \Delta_1 \between \Delta_2, \sigma) \ni (\Gamma, \Delta', \sigma)$.

Proof. The proof of (1) and (2) is straightforward by induction under arbitrary closing substitution.

Merging Assertions (2) We also need a refined merging function when considering the guarded command, to take into account the conditions.

Definition 15 (Parametric Merge). The parametric merge of assertions $\{\mathcal{L}_i\}_{i\in I}$ wrt conditions $\{e_i\}_{i\in I}$ (written $\biguplus\{e_i, \mathcal{L}_i\}_{i\in I}$) is defined recursively as follows:

- $\biguplus\{e_i, \mathcal{L}_i\}_{i\in I}$ with $\mathcal{L}_i = \mathtt{p}!\{l_j(x_j : U_j)\{A^i_j\}\langle E^i_j\rangle; \mathcal{L}^i_j\}_{j\in J}$ is defined as

$$\mathtt{p}!\{l_j(x_j : U_j)\{\vee_{i\in I}(\#e_i \wedge A^i_j)\}\langle \#e_i \supset E^i_j\rangle; \biguplus\{e_i, \mathcal{L}^i_j\}_{i\in I}\}_{j\in J}$$

- $\biguplus\{e_i, \mathcal{L}_i\}_{i\in I}$ with $\mathcal{L}_i = \mathtt{p}?\{l_i(x_j : U_j)\{A^i_j\}\langle E^i_j\rangle; \mathcal{L}^i_j\}_{j\in J}$ is defined as

$$\mathtt{p}?\{l_j(x_j : U_j)\{\vee_{i\in I}(\#e_i \supset A^i_j)\}\langle \#e_i \supset E^i_j\rangle; \biguplus\{e_i, \mathcal{L}^i_j\}_{i\in I}\}_{j\in J}$$

- $\mu\mathtt{t}\langle\tilde{y_1} : A\rangle(\tilde{x_1} : \tilde{U_1})\{\mathsf{true}\}.\mathcal{L}'_1 \uplus \mu\mathtt{t}\langle\tilde{y_2} : B\rangle(\tilde{x_2} : \tilde{U_2})\{\mathsf{true}\}.\mathcal{L}'_1$
  $\stackrel{def}{=} \mu\mathtt{t}\langle\tilde{y_1}, \tilde{y_2} : A \wedge B\rangle(\tilde{x_1}\tilde{x_2} : \tilde{U_1}\tilde{U_2})\{\mathsf{true}\}.\mathcal{L}'_1 \uplus \mathcal{L}'_2$
  where we assume $\tilde{x_1} \neq \tilde{x_2}$ and $\tilde{y_1} \neq \tilde{y_2}$.
- $\mathtt{t}\langle\tilde{x_1} : A\rangle \uplus \mathtt{t}\langle\tilde{x_2} : B\rangle \stackrel{def}{=} \mathtt{t}\langle\tilde{x_1}\tilde{x_2} : A \wedge B\rangle$ where we assume $\tilde{x_1} \neq \tilde{x_2}$.

Definition 16. The parametric merge operation is extended to assignments Δ in the obvious way, i.e. the pointwise merge of each Δ_i.

Lemma 17. *The following properties of parametric $\biguplus$ hold:*

$$\Delta_i \ni \biguplus\{e_i, \Delta_i\}_{i\in I}$$

$$\biguplus\{e_i, \Delta_i\}_{i\in I} \ni \Delta_i \text{ if } e_i \downarrow \mathsf{true}$$

G.4 Generation of Principal Assertions

The rules use judgements of the form

$$\mathcal{C}; \Gamma_0 \vdash_\star P \blacktriangleright \Delta \tag{33}$$

The rules in Figure 12 (cf. page 34) induce an algorithm that takes in input $\Gamma_0, \mathcal{C}$ and P and generates "most general" Δ for P under the conditions $\mathcal{C}$ and assignment Γ_0. We remark that the principal general assertion of a program may not exist, in which case the algorithm is supposed to return 'error'. However, if the process is well typed wrt the underlying typing discipline then a principal formula will be generated.

Without loss of generality, we assume the standard bound variable convention. The generation rules use the merge and parametric merge.

Each rule is naturally obtained, where under the left-hand side environment we derive the right-hand side principal formulae for processes inductively.

In the rule for selection, observe that P may have different branches corresponding to the same branch l_j of the local assertion (although with different conditions). We set J as the set of indexes of non replicated labels, with the cardinality of J, denoted with $|J|$ being the number of partitions of I collecting all indexes s.t. $l_1 = l_2$. We denote each of such partitions with $H(j)$.

As a local assertion cannot have duplicated branches, we have to create one branch for the principal formula (say l_j) that types all the branches (say l_i) of the process

such that $l_j = l_i$. Notice also that by well-typedness of P, for each $h1, h2 \in H(j)$, $x_{h1} = x_{h2}$. The extension with delegation does not present further challenges proceeds as in [5] except the predicates and updates are treated as in the case of selection (where x_i is not in the free variables of updates and predicates). The same holds for session receive.

In the rule for the input action we introduce the notation $\exists_{out}\tilde{x}.\Delta$ for the *existential closure of interaction variables* on each assertion in Δ where $\exists_{out}\tilde{x}.\mathcal{L}$ is closing with existential quantifiers $\tilde{x}$ in the predicates for selection and recursion in $\mathcal{L}$.

We use an annotation on which role the process is defining/instantiating.

Definition 9. *$\exists_{out}\tilde{x}.\mathcal{L}$ is defined as: (1) $\mathtt{p}!\{l_i(x_i : U_i)\{\exists\tilde{x}.A_i\}\langle E_i\rangle.\mathcal{L}_i\}_{i\in I}$ if $\mathcal{L} = \mathtt{p}!\{l_i(x_i : U_i)\{A_i\}\langle E_i\rangle.\mathcal{L}_i\}_{i\in I}$, (2) $\mu\mathtt{t}\langle y : A'\rangle(x : S)\{\exists\tilde{x}A\}.\mathcal{L}'$ if $\mathcal{L} = \mu\mathtt{t}\langle y : A'\rangle(x : S)\{A\}.\mathcal{L}'$, (3) $\mathcal{L}$ otherwise.*

The following proposition holds:

Proposition 1. *For any assertion $\mathcal{L}$ and any vector of pairwise dijoint interaction variables $\tilde{x}$*

$$\mathcal{L} \ni \exists\tilde{x}.(\mathcal{L})$$

Proof. Trivially from the definition of refinement and existential closure, observing that the existential closure weakens only the predicate of selections and recursion.

G.5 Completeness

Lemma 12. *$\mathcal{I}; \mathcal{C}, \Gamma \vdash^{ext} P \rhd \Delta \quad \supset \quad \mathcal{I}; \mathcal{C}\xi, \Gamma\xi \vdash P \rhd \Delta\xi$. for each ξ such that $\Gamma\xi$ and $\Delta\xi$ are well-asserted.*

Lemma 13 (Provability by Validation Rules). *If $\mathcal{C}; \Gamma_0 \vdash_\star P \blacktriangleright \Delta$, then $\mathcal{C}, \Gamma_0 \vdash^{\mathsf{ext}} [P]\sigma_p \rhd [\Delta]\sigma_a$.*

Proof. We show each generation rule in Figure 12 is an instance of the corresponding extension of the validation rules in Figure 4 with predicate variables: if the assumption is read as a sequent with $\vdash^{ext}$ rather than $\vdash_\star$, then the same holds for the conclusion, which is enough for the soundness of the each extended validation rule.

⌊SEL⌋ By hypothesis, for all distinguished labels l_i in P (with I being the branch indexes of P) we have $\mathcal{C} \wedge \#e_i; \Gamma_0 \vdash_\star P_i \rhd \Delta_i, y[\mathtt{p}] : \mathcal{L}_i$. Consider one generic $i \in I$, by induction from the derivation of P:

$$\mathcal{I}; \mathcal{C} \wedge \#e_i; \Gamma_0 \vdash^{ext} P_i \rhd \Delta_{rest}, y[\mathtt{p}] : \mathcal{L}_j[e'_i/x_j] \tag{34}$$

We now show that the first precondition of validation rule ⌊SEL⌋ when trying to validate P against the generated formula is

$$\mathcal{I} \wedge \mathcal{C} \wedge e_i \supset A_j \wedge (E_j = \wedge_{h\in H(j)} e_h \supset E_h)[e'_i/x_j] \tag{35}$$

with

$$A_j \stackrel{\text{def}}{=} \mathcal{C} \wedge \vee_{h\in H(j)}(e_h \wedge x_j = e'_h)$$

$$\frac{\mathcal{C}; \Gamma_0 \vdash_\star P \blacktriangleright \Delta, y[\mathtt{i}] : \mathcal{L} \qquad \Gamma_0(a) = \mathcal{G} \quad \mathcal{G} \upharpoonright \mathtt{i} = [\tilde{\mathtt{x}} : \tilde{S}]\{A\}.\mathcal{L}}{\mathcal{C}; \Gamma_0 \vdash_\star a[\mathtt{i}](y).P \blacktriangleright \Delta} \lfloor \textsc{Macc} \rfloor$$

$$\frac{\mathcal{C}; \Gamma_0 \vdash_\star P \blacktriangleright \Delta, y[\mathtt{1}] : \mathcal{L} \qquad \Gamma_0(a) = \mathcal{G} \quad \mathcal{G} \upharpoonright \mathtt{1} = [\tilde{\mathtt{x}} : \tilde{S}]\{A\}.\mathcal{L}}{\mathcal{C}; \Gamma_0 \vdash_\star \overline{a}[\mathtt{n}](y).P \blacktriangleright \Delta} \lfloor \textsc{Mcast} \rfloor$$

$$\frac{\begin{array}{c} H(j) \text{ with } j \in J \text{ is one of the } |J| \text{ partitions of } I \text{ collecting all indexes s.t. } \forall i_1, i_2 \in H(j)\ l_1 = l_2 \\ A_j = \mathcal{C} \wedge \vee_{h \in H(j)} (e_h \wedge x_j = e'_h) \\ E_j = \vee_{h \in H(j)} (e_h \supset E_h) \\ \mathcal{C} \wedge \#e_i; \Gamma_0 \vdash_\star P_i \blacktriangleright \Delta_i, k[\mathtt{p}] : \mathcal{L}_i \end{array}}{\begin{array}{c} \mathcal{C}; \Gamma_0 \vdash_\star k[\mathtt{p}, \mathtt{q}]!\{e_i \mapsto l_i\langle e'_i\rangle(x_i)\langle E_i\rangle; P_i\}_{i \in I} \\ \blacktriangleright (\Cup\{\#e_i, \Delta_i\}_{i \in I}),\ k[\mathtt{p}] : \mathtt{q}!\{l_j(x_j)\{A_j\}\langle E_j\rangle.\ \Cup\{\#e_i, \mathcal{L}_i\}_{i \in H(j)}\}_{j \in J} \end{array}} \lfloor \textsc{Sel} \rfloor$$

$$\frac{\mathcal{C} \wedge \Phi(x_i); \Gamma_0 \vdash_\star P_i \blacktriangleright \Delta_i, k[\mathtt{p}] : \mathcal{L}_i}{\mathcal{C}; \Gamma_0 \vdash_\star k[\mathtt{p}, \mathtt{q}]?\{l_i(x_i)\langle E_i\rangle.P_i\}_{i \in I} \blacktriangleright \exists_{out} \tilde{x_i} (\Cup\{\Delta_i\}_{i \in I}), k[\mathtt{p}] : \mathtt{q}?\{l_i(x_i)\{\phi_i(x_i)\}\langle \phi'_i\rangle; \mathcal{L}_i\}_{i \in I}} \lfloor \textsc{Bch} \rfloor$$

$$\frac{\mathcal{C}; \Gamma_0 \vdash_\star P_i \blacktriangleright \Delta_i \quad (i = 1, 2)}{\mathcal{C}; \Gamma_0 \vdash_\star P_1 \mid P_2 \blacktriangleright \Delta_1, \Delta_2} \lfloor \textsc{Par} \rfloor \qquad \frac{\Delta \ \ \mathsf{end\ only}}{\mathcal{C}; \Gamma_0 \vdash_\star \mathbf{0} \blacktriangleright \Delta} \lfloor \textsc{inact} \rfloor$$

$$\frac{}{\mathcal{C}; \Gamma_0, X : (x : S)\mathtt{t}_i^X \ @\ \mathtt{p}_\mathtt{i} .. \mathtt{t}_n^X \ @\ \mathtt{p}_\mathtt{n} \vdash_\star X_n^i\langle e\rangle \blacktriangleright s[\mathtt{p}_i] : \mathtt{t}_i^X\langle y : y = e \ \wedge\ \mathcal{C}\rangle} \lfloor \textsc{Var} \rfloor$$

$$\frac{\mathcal{C}; \Gamma_0, X : (x : S)\mathtt{t}_1^X \ @\ \mathtt{p}_1 .. \mathtt{t}_n^X \ @\ \mathtt{p}_\mathtt{n} \vdash_\star P \blacktriangleright \Delta, s[\mathtt{p}_i] : \mathcal{L}_i}{\mathcal{C}\ ;\ \Gamma_0 \vdash_\star \mu X_n^i\langle e\rangle(x).P \blacktriangleright \Delta, s[\mathtt{p}_i] : \mu \mathtt{t}_X\langle y : y = e\rangle(x)\{\mathsf{true}\}.\mathcal{L}_i} \lfloor \textsc{Rec} \rfloor$$

Fig. 12. Generation rules for programs (see Definition 9 for $\exists_{out}$).

holds with $e'_i = e'_h$ for some $h \in H$. First of all notice that the conclusion $(E_j = \wedge_{h \in H(j)} e_h \supset E_h)$ holds since exactly one e_h is true (at least one is true by premise $\vee_h e_h$ and only one is true by well typedness of P). Thus (35) can be simplified as follows, making A_j explicit:

$$\mathcal{I} \wedge \mathcal{C} \wedge e_i \supset \mathcal{C} \wedge \vee_{h \in H(j)} (e_h \wedge x_j = e'_h)[e'_i / x_j] \tag{36}$$

which is equivalent to (considering only the predicate for the branch for which $e'_i = e'_h$ and $e_i = e_h$ and ... denoting the remaining predicates)

$$\mathcal{I} \wedge \mathcal{C} \wedge e_i \supset \mathcal{C} \wedge ((e_h \wedge e'_i = e'_h) \vee \ldots) \tag{37}$$

which is true (with $\mathcal{I}$ true).

From (34), using premise (37) for validation rule $\lfloor$SEL$\rfloor$, and then using the fact that $\Delta_i \ni \Cup\{\#e_i, \Delta_i\}_{i \in I}$ and $\mathcal{L}_i \ni \Cup\{\#e_i, \mathcal{L}_i\}_{i \in H(j)}$ by Lemma 17 to apply validation rule $\lfloor$N4$\rfloor$ we obtain the thesis.

$\lfloor$BCH$\rfloor$ Easy by inductive hypothesis and straightforward application of extended validation rule $\lfloor$BCH$\rfloor$. For the existential elimination, observe that:

1. all occurrences of the abstracted variable are in send/select and recursion instantiation; and

2. all recursion instantiation is used in send/select inside the recursion body.

Thus existential elimination only anti-refines the given assertion, hence done.

[MCAST] By inductive hypothesis from $\mathcal{C}; \Gamma_0 \vdash_\star P \blacktriangleright \Delta, y[\mathtt{1}] : \mathcal{L}_1$ it follows

$$\mathcal{I}; \mathcal{C}; \Gamma_0 \vdash^{ext} P \rhd \Delta, y[\mathtt{1}] : \mathcal{L}_1 \tag{38}$$

where $\Gamma_0(a) \upharpoonright \mathtt{1} = [\tilde{\mathtt{x}} : \tilde{S}]\{A\}.\mathcal{L}_1$.

By [MCAST] of extended validation, we have

$$\mathcal{I}; \mathcal{C}; \Gamma_0 \vdash^{ext} \overline{a}[\mathtt{n}]\langle s \rangle.P \rhd \Delta \tag{39}$$

which, because we have $\Gamma_0, a : \mathcal{G} = \Gamma_0$, is equivalent to,

$$\mathcal{C}; \Gamma_0, a : \mathcal{G} \vdash^{ext} P \rhd \Delta \tag{40}$$

as required.

[PAR] By induction.

[INACT] Immediate from the corresponding validation rules.

[REC] To prove this case, we consider substitution instance of the assumption of the rule, with t instantiated into the corresponding recursive assertion in the conclusion. By this we can apply the original (validation) rule for recursion, hence as required.

This exhausts all cases.

Definition 10. *We say that ξ is a concretising substitution if no predicate variables occur in its codomain.*

Lemma 14 (Completeness via Refinement). *Let $\mathcal{C}; \Gamma_0 \models [P]\sigma_p \rhd [\Delta_0]\sigma_a$ be an open judgment (and P be well-typed, wrt the underlying typing discipline, against the type obtained by erasing all predicates and updates from Δ). Let $\mathcal{C}; \Gamma_0 \vdash_\star P \rhd \Delta$ be the generated formula. Assume that :* (1) $\mathtt{id}(\sigma_p) = \mathtt{id}(\sigma_a)$, (2) $var(\mathcal{I}) \subseteq \mathtt{s}(\sigma_p) = \mathtt{s}(\sigma_a)$, *and* (3) $\mathcal{I}$ *equivalent to* true. *There exists a concretising substitution ξ such that for any closing substitution $\tilde{\sigma}$ consistent with $\mathcal{C}$, $\Delta\xi\tilde{\sigma}$ is well-asserted and $\Delta\xi\tilde{\sigma} \Supset \Delta_0$.*

Proof. By induction on the size of the process (we use the size of processes rather than direct structural induction since we need to reason up to substitutions, even though we can in effect use rule induction). In the proof below, we use typed labelled transition for open processes, which stands for the family of its instantiations into closed processes as defined before.

Case [MACC]. We assume $P = a[\mathtt{i}](y).P'$. By hypothesis

$$\mathcal{C}; \Gamma_0 \models [a[\mathtt{i}](y).P']\sigma_p \rhd [\Delta_0]\sigma_a \tag{41}$$

$$\mathcal{C}; \Gamma_0 \vdash_\star a[\mathtt{i}](y).P' \blacktriangleright \Delta \tag{42}$$

By (41) and one step of conditional simulation

$$\mathcal{C}; \Gamma_0 \models [P']\sigma_p \rhd [\Delta_0, s[\mathtt{i}] : \mathcal{L}]\sigma_a \quad \Gamma_0(a) \upharpoonright_i = [\tilde{\mathtt{x}} : \tilde{S}]\{A\}.\mathcal{L}$$

By (42) and the application of generation rule [MACC]

$$\mathcal{C}; \Gamma_0 \vdash_\star P' \blacktriangleright \Delta, y[\mathtt{i}] : \mathcal{L} \quad \Gamma_0(a) \upharpoonright_i = [\tilde{\mathtt{x}} : \tilde{S}]\{A\}.\mathcal{L}$$

By induction $\Delta, y[\mathtt{i}] : \mathcal{L}\xi\tilde{\sigma} \Supset \Delta_0, s[\mathtt{i}] : \mathcal{L}$ hence by definition of refinement follows $\Delta\xi\tilde{\sigma} \Supset \Delta_0$ for some $\tilde{\sigma}, \xi$.

Case [SEL]. We assume $P = k[\mathtt{p},\mathtt{q}]!\{e_i \mapsto l_i\langle e'_i\rangle(x_i)\langle E_i\rangle.P_i\}_{i\in I}$. Below for brevity we use labelled transition for open processes, which stands for the family of its instantiations into closed processes as defined before. We do not mention these substitutions since for each substitution the same reasoning applies. By hypothesis

$$\begin{array}{l}\mathcal{C};\Gamma_0 \models [s[\mathtt{p},\mathtt{q}]!\{e_i \mapsto l_i\langle e'_i\rangle(x_i)\langle E_i\rangle.P_i\}_{i\in I}]\sigma_p \rhd \\ \qquad [\Delta_0^{rest}, s[\mathtt{p}] : \mathtt{q}!\{(x_j : U_j)\{A'_j\}\langle E'_j\rangle.\mathcal{L}_j\}_{j\in J}]\sigma_a\end{array} \tag{43}$$

where we note that E'_i may differ from the corresponding E'_j. After one step of conditional simulation with $\ell = s[\mathtt{p},\mathtt{q}]!l_i\langle v\rangle$ and (43)

$$\mathcal{C};\Gamma_0 \models [P_i[v/x_i]]\sigma'_p \rhd [\Delta_0^{rest}, s[\mathtt{p}] : \mathcal{L}_j[v/x_j]]\sigma'_a$$

where $\sigma'_p = \sigma_p\,\mathtt{after}\,E_i$ and $\sigma'_a = \sigma_a\,\mathtt{after}\,E'_j$.

By generation:

$$\begin{array}{l}\mathcal{C};\Gamma_0 \vdash_\star s[\mathtt{p},\mathtt{q}]!\{e_i \mapsto l_i\langle e'_i\rangle(x_i)\langle E_i\rangle.P_i\}_{i\in I} \blacktriangleright \\ \qquad \text{Ш}\{\#e_i, \Delta_i\}, y[\mathtt{p}] : \mathtt{q}!\{(x_j : U_j)\{A_j\}\langle E_j\rangle.\mathcal{L}^\star_j\}_{j\in J}\end{array} \tag{44}$$

with $E_j \stackrel{\text{def}}{=} \vee_{h\in H(j)} e_h \supset E_h$, $A_j \stackrel{\text{def}}{=} \mathcal{C} \wedge \vee_{h(j)\in H}(e_h \wedge x_j = e'_h)$ and $\mathcal{L}^\star_j \stackrel{\text{def}}{=} \text{Ш}\{\#e_i, \Delta_i\}_{i\in H(j)}$.

By generation rule [SEL] and (44)

$$\mathcal{C} \wedge \#e_i;\Gamma_0 \vdash_\star P_i \rhd \Delta, y[\mathtt{p}] : \mathcal{L}_i^{gen}$$

By Lemma 17 $\text{Ш}\{\#e_i, \Delta_i\}_{i\in I} \ni \Delta_i$ if $\#e_i \downarrow \mathsf{true}$ and by induction $\Delta_i \ni \Delta_0^{rest}$ for all $i \in I$, thus

$$\text{Ш}\{\#e_i, \Delta_i\}_{i\in I} \ni \Delta_0^{rest} \qquad \text{if } \#e_i \downarrow \mathsf{true} \text{ for each } i \in I \tag{45}$$

Also, by induction for some $\xi, \tilde{\sigma}$

$$y[\mathtt{p}] : \mathcal{L}_i^{gen}\xi\tilde{\sigma} \ni s[\mathtt{p}] : \mathcal{L}_i[v/x_i] \tag{46}$$

The thesis

$$\begin{array}{l}\text{Ш}\{e_i, \Delta_i\}_{i\in I}, y[\mathtt{p}] : \mathtt{q}!\{(x_i : U_i)\{A_j\}\langle E_i\rangle.\mathcal{L}^\star_i\}_{i\in I}\xi\tilde{\sigma} \\ \ni \Delta_0^{rest}, s[\mathtt{p}] : s[\mathtt{p}] : \mathtt{q}!\{(x_j : U_j)\{A'_j\}\langle E'_j\rangle.\mathcal{L}_j\}_{j\in J}\end{array}$$

follows observing that if $y[\mathtt{p}] : \mathtt{q}!\{(x_i : U_i)\{\mathcal{C} \wedge e_i \wedge x_i = e'_i\}\langle E_i\rangle.\mathcal{L}^\star_i$ can make a step with label ℓ, then $A_j \stackrel{\text{def}}{=} \mathcal{C} \wedge \vee_{h(j)\in H}(e_h \wedge x_j = e'_h)$ is also true. Namely, e_h must be true for some h. Since by definition of P two conditions cannot be true at the same time then only e_h is true. This means that P can make also a step with label ℓ thus by conditional simulation $s[\mathtt{p}] : s[\mathtt{p}] : \mathtt{q}!\{(x_j : U_j)\{A'_j\}\langle E'_j\rangle.\mathcal{L}_j\}_{j\in J}$ can also do a step with label ℓ. Finally, from (46), Lemma 17 and the fact that $e_h \downarrow \mathsf{true}$ (thus $\#e_h \downarrow \mathsf{true}$) we obtain that also the continuations preserve the refinement relationship, namely $\mathcal{L}^\star_i\xi\tilde{\sigma} \ni \mathcal{L}_j$.

Case [BCH]. We assume $P = k[\mathtt{p},\mathtt{q}]?\{(x_i)\langle E_i\rangle.P_i\}_{i\in I}$. By hypothesis

$$\begin{array}{l}\mathcal{C};\Gamma_0 \models [s[\mathtt{p},\mathtt{q}]?\{(x_i)\langle E_i\rangle.P_i\}_{i\in I}]\sigma_p \rhd \\ \qquad [\Delta_0^{rest}, s[\mathtt{q}] : \mathtt{p}?\{(x_i : U_i)\{A_i\}\langle E_i'\rangle.\mathcal{L}_i\}_{i\in I}]\sigma_a\end{array} \tag{47}$$

where we note that E_i' may differ from E_i.

$$\begin{array}{l}\mathcal{C};\Gamma_0 \vdash_\star s[\mathtt{p},\mathtt{q}]?\{(x_i)\langle E_i\rangle.P_i\}_{i\in I} \blacktriangleright \\ \qquad \exists\tilde{x}\uplus\Delta_i, y[\mathtt{q}] : \mathtt{p}?\{(x_i : U_i)\{\phi(x_i)\}\langle\phi'\rangle.\mathcal{L}_i^\star\}_{i\in I}\end{array} \tag{48}$$

After one step of conditional simulation and (47)

$$\mathcal{C};\Gamma_0 \models [P_i[v/x_i]]\sigma_p' \rhd [\Delta_0^{rest}, s[\mathtt{q}] : \mathcal{L}_j[v/x_i]]\sigma_a'$$

where $\sigma_p' = \sigma_p\,\mathtt{after}\,E_i$ and $\sigma_a' = \sigma_a\,\mathtt{after}\,E_i'$.

Finally, $\uplus\{e_i,\Delta_i\}, y[\mathtt{q}] : \mathtt{p}?\{(x_i : U_i)\{\psi(x_i)\}\langle E_i\rangle.\mathcal{L}_i^\star\}_{i\in I}$ is well asserted. History sensitivity follows and temporal satisfiability follow from the fact that $\mathcal{L}^\star$ is well asserted thus admits possible computations also for the values satisfying A_i if A_i admits solutions. We know that A_i admits solutions for some branch i because Δ_0 is temporal satisfiable. Invariant satisfiability is trivially satisfied since $\mathcal{I} = \mathsf{true}$. Updatability is preserved since P is able to perform the prescribed update.

By generation rule [BCH] and (48)

$$\mathcal{C}\wedge\phi(x_i);\Gamma_0 \vdash_\star P_i \rhd \Delta_i, y[\mathtt{q}] : \mathcal{L}_i^\star$$

By Lemma 14 $\uplus\Delta_i \ni \Delta_i$, by Proposition 1 and definition of closure used in Figure 12, $\exists\tilde{x}.\uplus\Delta_i \ni \Delta_i$ and by induction $\Delta_i \ni \Delta_0^{rest}$, thus

$$\exists\tilde{x}.\uplus\Delta_i \ni \Delta_0^{rest} \tag{49}$$

Also, by induction for some $\xi,\tilde{\sigma}$

$$y[\mathtt{q}] : \mathcal{L}_i^\star\xi\tilde{\sigma} \ni s[\mathtt{q}] : \mathcal{L}_i[v/x_i] \tag{50}$$

The thesis

$$\begin{array}{l}\exists\tilde{x}.\uplus\Delta_i, y[\mathtt{q}] : \mathtt{p}?\{(x_i : U_i)\{\phi(x_i)\}\langle\phi'\rangle.\mathcal{L}_i^\star\}_{i\in I}\xi\tilde{\sigma} \\ \ni \Delta_0^{rest}, s[\mathtt{q}] : \mathtt{p}?\{(x_i : U_i)\{A_i\}\langle E_i'\rangle.\mathcal{L}_i\}_{i\in I}\end{array}$$

for some $\xi\tilde{\sigma}$ follows from (49) observing that if $s[\mathtt{q}] : \mathtt{p}?\{(x_i : U_i)\{A_i\}\langle E_i'\rangle.\mathcal{L}_i\}_{i\in I}$ can make a step with label ℓ then $y[\mathtt{q}] : \mathtt{p}?\{(x_i : U_i)\{\psi(x_i)\}\langle\psi'\rangle.\mathcal{L}_i^\star$ can also do a step with label ℓ where we substitute $\phi(x_i)$ with A_i and ϕ' with E_i' since the continuations still preserve the refinement by (50).

Case [PAR]. Assuming $\mathcal{C};\Gamma_0 \models P1 \mid P2\sigma_p \rhd \Delta1, \Delta2$ with $\Delta1$ and $\Delta2$ disjoint, we have $\mathcal{C};\Gamma_0 \models [P1]\sigma_p \rhd \Delta1$ and $\mathcal{C};\Gamma_0 \models P1 \rhd \Delta1$.

By inductive hypothesis $\mathcal{C};\Gamma_0 \vdash_\star P1 \rhd \Delta'1$ and $\mathcal{C};\Gamma_0 \vdash_\star P2 \rhd \Delta'2$ where $\Delta'1 \ni \Delta1$ and $\Delta'2 \ni \Delta2$. Therefore $\Delta'1, \Delta'2 \ni \Delta1, \Delta2$.

Case [VAR]. Straightforward.

Case [REC]. We present an informal argument (the formal case is very similar to the corresponding case in [5]: we know by induction the assumption gives the strongest assertion. Hence its instantiation by an appropriate substitution for the assertion variables concerned, gives the strongest assertion (recall these variables are introduced at the time of the [VAR]). If the recursive process in the conclusion ever satisfies an assertion, then P in the assumption also satisfies the assertion if the assertion variables are instantiated into the corresponding recursive assertions (through the unfolding). Applying this observation to both the satisfying assertion and the strongest assertion, we can reason, for each finite step, transitions from (the finite unfoldings of) the strongest assertion refines (the finite unfoldings of) the satisfying assertion.

H HML Embedding

Embedding We use a standard HML with the first-order predicates as in [1]. These predicates, denoted by A in the following are to the ones appearing in assertions, defined in Figure 1. The LTS associated to our HML consider as actions, denoted by ℓ, both the communications of the process and the updates of the state. As a consequence $P, \sigma \xrightarrow{\ell} P', \sigma'$ if either $P \xrightarrow{\ell} P'$ and $\sigma' = \sigma$ or $P = P'$ and $\sigma' = \sigma\,\texttt{after}\,\ell$. We use ϕ to denote HML-formulae, which are built from predicates, implications, universal quantifiers, conjunctions and *must* modalities. We remark that the logic used in this *safety embedding* is positive: if we remove the implication symbol, there is no negation, no existential quantifier, no disjunction and no may modality. Additionally, the implication will always appear as $A \Rightarrow \phi$ meaning that modalities never appear in the negative side.

$$\phi ::= \texttt{true} \mid \phi \wedge \phi \mid \phi \Rightarrow \phi \mid [\ell]\phi \mid A \mid \forall x : S.\phi \quad \ell ::= s[\texttt{p},\texttt{q}](x) \mid \overline{s[\texttt{p},\texttt{q}]}(x) \mid E$$

$$\frac{P, \sigma \models \phi_1 \qquad P, \sigma \models \phi_2}{P, \sigma \models \phi_1 \wedge \phi_2} \qquad \frac{}{P, \sigma \models \texttt{true}}$$

$$\frac{\text{if } P, \sigma \models \phi_1 \text{ then } P, \sigma \models \phi_2}{P, \sigma \models \phi_1 \Rightarrow \phi_2} \qquad \frac{\text{For all } P', \sigma' \text{ s.t. } P, \sigma \xrightarrow{\ell} P', \sigma', P', \sigma' \models \phi}{P \models [\ell]\phi}$$

$$\frac{\sigma \vdash_{bool} A}{P, \sigma \models A} \qquad \frac{\text{For all values } v \text{ of type } T, P, \sigma \models \phi[v/x]}{P, \sigma \models \forall x : T.\phi}$$

Fig. 13. Logical rules

The satisfactions rules (Figure 14) are fairly standard, for a pair P, σ to satisfy a predicate A, A has to hold w.r.t. to σ, denoted by $\sigma \vdash_{bool} A$, meaning that $\sigma(A)$ is a tautology for the boolean logic.

The embedding of local types we propose is parametrised with a session channel $s[\texttt{p}]$. Assertions appearing in input prefixes are embedded as premises in implications, and assertions in output prefixes have to be satisfied, yielding:

$$\|\texttt{q}!\{l_i(x_i : S_i)\{A_i\}\langle E_i\rangle.\mathcal{L}_i\}_{i\in I}\|^{s[\texttt{p}]} = \textstyle\bigwedge_{i\in I} \forall x_i : S_i, [\overline{s[\texttt{p},\texttt{q}]}(x_i)](A_i \wedge [E_i]\|\mathcal{L}_i\|^{s[\texttt{p}]})$$
$$\|\texttt{q}?\{l_j(x_j : S_j)\{A_j\}\langle E_j\rangle.\mathcal{L}_j\}_{j\in J}\|^{s[\texttt{p}]} = \textstyle\bigwedge_{j\in J} \forall x_j : S_j, [s[\texttt{q},\texttt{p}](x_j)](A_j \Rightarrow \|\mathcal{L}_j\|^{s[\texttt{p}]})$$

$$\frac{P,\sigma \models \phi_1 \qquad P,\sigma \models \phi_2}{P,\sigma \models \phi_1 \wedge \phi_2} \qquad\qquad \frac{}{P,\sigma \models \texttt{true}}$$

$$\frac{\text{if } P,\sigma \models \phi_1 \text{ then } P,\sigma \models \phi_2}{P,\sigma \models \phi_1 \Rightarrow \phi_2} \qquad\qquad \frac{\text{For all } P',\sigma' \text{ s.t. } P,\sigma \xrightarrow{\ell} P',\sigma', P',\sigma' \models \phi}{P \models [\ell]\phi}$$

$$\frac{\sigma \vdash_{bool} A}{P,\sigma \models A} \qquad\qquad \frac{\text{For all values } v \text{ of type } T, P,\sigma \models \phi[v/x]}{P,\sigma \models \forall x : T.\phi}$$

Fig. 14. Logical rules

The embedding of selection, is a conjunction of formulae corresponding to the branches: for each value sent on the session channel, predicates should be satisfied and, if the state is updated, the embedding of the continuation should hold. For branching types, the assertion is used as an hypothesis and no update appear.

Soundness To obtain soundness for typing judgements involving specifications, we have to introduce *interleavings* of formulae, treating the fact that one process can play several roles in several sessions. As a simple example both $s[\mathtt{p}_1,\mathtt{p}_2]?(x).k![\mathtt{q}_1,\mathtt{q}_2]\langle 10\rangle$ and $k![\mathtt{q}_1,\mathtt{q}_2]\langle 10\rangle.s[\mathtt{p}_1,\mathtt{p}_2]?(x)$ can be typed with $s[\mathtt{p}_2] : \mathtt{p}_1?(x : \mathtt{Nat}).\mathsf{end}$, $k[\mathtt{q}_1] : \mathtt{q}_2!(y : \mathtt{Nat}).\mathsf{end}$.

Interleaving is not a new operator *per se* and can be seen as syntactic sugar, describing shuffling of must modalities. The main rule for interleaving is: $[\ell_1]\phi_1 \rtimes [\ell_2]\phi_2 = [\ell_1](\phi_1 \rtimes [\ell_2]\phi_2) \wedge [\ell_2]([\ell_1]\phi_1 \wedge \phi_2)$. When interleaving two or more formulae containing modalities, we obtain a conjunction of formulae, each one representing a different way of organising all modalities in a way preserving their initial orders. Informally, the interleaving of $[1][2]$ and $[A][B]$ is $[1][2][A][B] \wedge [A][B][1][2] \wedge [1][A][2][B] \wedge [A][1][B][2] \wedge [1][A][B][2] \wedge [A][1][2][B]$.

We encode a pair Δ, Γ into a complex formula $\mathtt{Inter}(\Delta,\Gamma)$, defined as the interleaving of the formulae obtained by encoding the local types of Δ on their corresponding channels and the formulae corresponding to Γ, built as follows: for each channel $a : \mathtt{I}(\mathcal{G})$, if some $s[\mathtt{p}]$ is received on a, the resulting process should satisfy the encoding on $s[\mathtt{p}]$ of the projection of $\mathcal{G}$ on $\mathtt{p}$:

$$\begin{aligned}&\mathtt{Inter}(s_1[\mathtt{p}_1],\ldots,s_n[\mathtt{p}_n]; a_1 : \mathtt{I}(\mathcal{G}_1),\ldots,a_m : \mathtt{I}(\mathcal{G}_m))\\ &\qquad = \|T_1\|^{s_1[\mathtt{p}_1]} \rtimes \ldots \rtimes \|T_n\|^{s_n[\mathtt{p}_n]} \rtimes \phi_1 \rtimes \ldots \rtimes \phi_m\end{aligned}$$

where $\phi_i = \forall s_i'.\forall \mathtt{p}_i'.[a_i(s_i'[\mathtt{p}_i])]\|\mathcal{G}_i \upharpoonright \mathtt{p}_i'\|^{s_i'[\mathtt{p}_i']}$.

The preliminaries lemmas concerning logics need to be proved. Lemma 15 states that a process cannot perform an action on a channel that does bot appear in its type. Lemma 16 observes that parallel composition with processes that does not perform any action does not change the set of formulae a process satisfies. Lemma 17 states that satisfaction of assertion is stable by reduction and Lemma 18 enforces the stability of satisfaction judgement by well-typed substitutions.

Lemma 15 (Type safety). *If $\mathcal{I};\mathcal{C};\Gamma \vdash P : \Delta$ and $s[\mathtt{p}] \notin \Delta \cup \Gamma$, then there is no P' s.t. $P,\sigma \xrightarrow{\ell_s} P',\sigma$ for any action ℓ_s of the form $s![\mathtt{p},\mathtt{q}]\langle l.v\rangle$ or $s![\mathtt{q},\mathtt{p}]l(x)$.*

Similarly, if $a : \mathtt{I}(\mathcal{G}) \notin \Gamma$*, there is no* $P', s[\mathtt{p}]$ *s.t.* $P, \sigma \xrightarrow{a(s[\mathtt{p}])} P', \sigma$.

The direct corollary that will be used later is that a process typed with an empty Δ cannot make any action.

Proof. Easily done by induction on the typing rules, noticing that $\mathbf{0}$ requires an empty typing context to be typed.

Lemma 16 (Trivial Composition). *If* $P_1, \sigma \models \phi$ *and* P_2 *cannot make any action, then* $P_1 \mid P_2, \sigma \models \phi$.

Proof. By structural induction over ϕ, the only interesting case being $\phi = [\alpha_s]\phi'$. It is done by supposing that $P_1 \mid P_2 \xrightarrow{\alpha_s} P'$, we immediately notice that $P_1 \xrightarrow{\alpha_s} P_1'$ and $P' = P_1' \mid P_2$, as P_2 cannot make any action. We use the induction hypothesis to conclude.

Lemma 17 (Stability of assertions). *If* $P, \sigma \models A$ *and* $P \xrightarrow{\ell} P'$*, then* $P', \sigma \models A$.

Proof. According to logics rules $P \models A$ if $\sigma(A)$ is a tautology. Thus we directly have the stronger result, If $P, \sigma \models A$, for all P', $P', \sigma \models A$.

Lemma 18 (Satisfaction substitution). *If* $P, \sigma \models \phi$ *and* $x : S, v : S$ *are not bound in* P, σ *and* ϕ*, then* $P[v/x], \sigma \models \phi[v/x]$.

Proof. By induction on ϕ, as our processes and formulas abide a Barendregt convention, the case $\forall y.\phi$ is easy as $y \neq x$ and $yneqv$ the only interesting case being assertion and must modality:

- Case $\phi = A$. The logic rules notifies that $\sigma(A)$ is a tautology, so any instantiation of its free variable should be so. Thus $\sigma(A)\{v/x\}$ is a tautology and any process (in particular $P\{v/x\}$) and the state σ form a pair that satisfies it.
- Case $\phi = [\alpha]\phi'$. We prove, by induction on the reduction rules, that if $P \xrightarrow{\alpha} P'$, then $P\{v/x\} \xrightarrow{\alpha\{v/x\}} P'\{v/x\}$ and use the induction hypothesis.

We state, thanks to the previous lemmas, the following 'simple' soundness, for simple local types:

Proposition 2 (Simple Soundness). *If* $\mathcal{I}, \mathcal{C}, \varnothing \vdash P \rhd s[\mathtt{p}] : \mathcal{L}$ *and* $\sigma \models \mathcal{I}$*, then* $(P, \sigma) \models \mathcal{C} \Rightarrow \|\mathcal{L}\|^{s[\mathtt{p}]}$.

The above proposition is proved by induction on the typing judgement.
Unasserted types are built from:

$$\mathcal{L} ::= \quad \mathtt{p}?\{l_i(x_i : U_i)E_i.\mathcal{L}_i\}_{i\in I} \mid \mathtt{p}!\{l_i(x_i : S_i)E_i.\mathcal{L}_i\}_{i\in I} \mid \mathsf{end}$$

The multiplicative parallel rule is given as:

$$\frac{\mathcal{I}; \mathcal{C}; \Gamma_1 \vdash P_1 : \Delta_1 \qquad \mathcal{I}; \mathcal{C}; \Gamma_2 \vdash P_2 : \Delta_2}{\mathcal{I}; \mathcal{C}; \Gamma_1, \Gamma_2 \vdash P_1 \mid P_2 : \Delta_1, \Delta_2}$$

Proposition 3 (Simple Completeness). *For all $\mathcal{L}$ and $\mathcal{I}$, if $\vdash P : s[p] : \mathbf{Er}(\mathcal{L})$, $\sigma \models \mathcal{I}$ and $P, \sigma \models \mathcal{C} \Rightarrow \|\mathcal{L}\|^{s[\mathtt{p}]}$ then $\mathcal{C}; \mathcal{I}; \vdash P \rhd s[p] : \mathcal{L}$.*

Proof. By induction on the typing judgement $\vdash P : s[\mathtt{p}] : \mathbf{Er}(\mathcal{L})$:

- Case *branching*. We have $\mathcal{L} = \mathtt{p}_0?\{l_i(x_i : U_i)\{A_i\}\langle E_i\rangle.\mathcal{L}_i\}_{i\in I}$. Let $i \in I$ and suppose $\mathcal{C}$ holds and $\sigma \models \mathcal{I}$. We have from the hypothesis $\vdash P \rhd \mathtt{p}_0?\{l_i(x_i : U_i).\mathcal{L}_i\}_{i\in I}$. The unasserted typing rules give $P = s?\{(x_i : S_i)\{A_i\}.P_i\}_{i\in I}$, and $\vdash P_i \rhd s[\mathtt{p}] : \mathcal{L}_i$. We know that $P, \sigma \models \mathcal{C} \implies \|\mathcal{L}\|^s[\mathcal{I}]$, which is:

  $$P, \sigma \models \bigwedge_{i\in I} \forall x_i.[\overline{s}(x_i)](A_i \implies [E_i]\|\mathcal{L}_i\|^s[\mathcal{I}] \wedge (A_i \wedge \mathcal{I} \wedge \mathcal{C}) \implies \mathcal{I}\,\mathtt{after}\,E_i)$$

  . We know from the shape of P, given above, and the reduction rules, that P can perform $s(x_i)$ to $P_i, \sigma \models (A_i \implies [E_i]\|\mathcal{L}_i\|^{s[\mathtt{p}],\mathcal{I}} \wedge (A_i \wedge \mathcal{I}) \implies \mathcal{I}\,\mathtt{after}\,E_i)$. We see that σ can perform E_i to $\sigma\,\mathtt{after}\,E_i$, meaning that we have $P_i, \sigma\,\mathtt{after}\,E_i \models (A_i \implies \|\mathcal{L}_i\|^{s[\mathtt{p}],\mathcal{I}})$, so we invoke Lemma **??**, proving that $sigma\,\mathtt{after}\,E_i \models \mathcal{I}$ in order to use the induction hypothesis to get $\mathcal{C} \wedge A_i; \mathcal{I}; \varnothing \vdash P_i \rhd \mathcal{L}_i$ and $\sigma\,\mathtt{after}\,E_i \models \mathcal{I}$.To sum up, for all i, $\mathcal{C}, A_i \vdash P_i \rhd s[\mathtt{p}] : \mathcal{L}_i$. Additionaly we have $(A_i \wedge \mathcal{I} \wedge \mathcal{C}) \implies \mathcal{I}\,\mathtt{after}\,E_i$ and this allows us to use the proof rule for branching to prove $\mathcal{C}; \mathcal{I}; \varnothing \vdash P \rhd s[p] : \mathcal{L}$.
- Case *selection*. We have $\mathcal{L} = \mathtt{p}_0?\{l_i(x_i : U_i)\{A_i\}\langle E_i\rangle.\mathcal{L}_i\}_{i\in I}$. Suppose $\mathcal{C}$ holds and $\sigma \models \mathcal{I}$. We have from the hypothesis $\vdash P \rhd s[\mathtt{p}] : \mathtt{p}_0?\{l_i(x_i : U_i).\mathcal{L}_i\}_{i\in I}$. The unasserted typing rules give $P = s!\{\{A_j\}\langle P_j\rangle(x_j : S_j)E_j\}$, and $\vdash P_j \rhd s[\mathtt{p}] : \mathcal{L}_j$. We know that $P, \sigma \models \mathcal{C} \implies \|\mathcal{L}\|^s[\mathcal{I}]$, which is $P \models \bigwedge_{i\in I} \forall x_i.[\overline{s}(x_i)]A_i \wedge \|\mathcal{L}_i\|^{s[\mathtt{p}],\mathcal{I}} \wedge (A_i \wedge \mathcal{I} \wedge \mathcal{C}) \implies \mathcal{I}\,\mathtt{after}\,E_i)$. In particular, as $\mathcal{C}$ holds, $P \models [\overline{s}(x_j)]A_j \wedge \|\mathcal{L}_j\|^{s[\mathtt{p}],\mathcal{I}} \wedge (A_j \wedge \mathcal{I}) \implies \mathcal{I}\,\mathtt{after}\,E_j)$ We know from the shape of P, given above, and the reduction rules, that P can perform $\overline{s}(x_j)$ to $P_j \models (A_j \wedge \|\mathcal{L}_j\|^s)$, meaning that A_j holds. Also, σ can perform E_j to $\sigma\,\mathtt{after}\,E_j$. To sum up, we have $\mathcal{C} \wedge \mathcal{I} \implies A_j, P_j \models \mathcal{C} \implies \|\mathcal{L}_j\|^{s[\mathtt{p}],\mathcal{I}}$ and $\vdash P_j \rhd s[\mathtt{p}] : \mathcal{L}_j$, Lemma **??** allows us to use the induction hypothesis to get $\mathcal{C} \vdash P_j \rhd : \mathcal{L}_j$ and this allows us to use the proof rule for selection to prove $\mathcal{C} \vdash P \rhd s[p] : \mathcal{L}$.
- Case *parallel*. No assertion appear in the parallel rule and we can use Lemmas 15 and 16 to state that exactly one side of the parallel composition satisfies the formula (along with the same state σ). As a consequence, we use the induction hypothesis twice and conclude.
- Case *end*. $\mathcal{L} = \mathsf{end}$, so this case is trivial.

Here are additional definitions for interleaving:

$$[\ell_1]\phi_1 \rtimes (\phi_{2,1} \wedge \phi_{2,2}) = [\ell_1](\phi_1 \rtimes \phi_{2,1}) \wedge [\ell_1](\phi_1 \rtimes \phi_{2,2})$$

$$\phi \rtimes \mathtt{true} = \phi \qquad\qquad \phi \rtimes (\phi_1 \wedge \phi_2) = (\phi \rtimes \phi_1) \wedge (\phi \rtimes \phi_2)$$

$$(\phi_1 \wedge \phi_2) \rtimes \phi = (\phi_1 \rtimes \phi) \wedge (\phi_2 \rtimes \phi) \qquad\qquad \forall x : T.\phi_1 \rtimes \phi_2$$

$$(A \Rightarrow \phi_1) \rtimes \phi_2 = A \Rightarrow (\phi_1 \rtimes \phi_2)$$

The following Lemmas are used in the proofs of soundness and completeness to handle interleavings.

Lemma 19 (Shuffling correctness).

Let P_1, P_2, ϕ_1, ϕ_2*, if* $P_1 \models \phi_1$ *and* $P_2 \models \phi_2$ *and if* $\texttt{free}(\phi_1) \cap \texttt{free}(P_2) = \texttt{free}(\phi_2) \cap \texttt{free}(P_1) = \texttt{free}(P_1) \cap \texttt{free}(P_2) = \texttt{free}(\phi_1) \cap \texttt{free}(\phi_2) = \varnothing$*, then* $P_1 \mid P_2 \models \phi_1 \rtimes \phi_2$.

Conversely, if $P_1 \mid P_2 \models \phi_1 \rtimes \phi_2$, $\texttt{free}(\phi_1) \cap \texttt{free}(P_2) = \texttt{free}(\phi_2) \cap \texttt{free}(P_1) = \texttt{free}(P_1) \cap \texttt{free}(P_2) = \texttt{free}(\phi_1) \cap \texttt{free}(\phi_2) = \varnothing$, $\texttt{free}(\phi_1) \subseteq \texttt{free}(P_1)$ *and* $\texttt{free}(\phi_2) \subseteq \texttt{free}(P_2)$.

Proof. We proceed by double structural induction over the pair (ϕ_1, ϕ_2).

- The most interesting case is when both formula are modalities: $\phi_1 = [\alpha_1]\phi_1'$ and $\phi_2 = [\alpha_2]\phi_2'$. The formula $\phi_1 \rtimes \phi_2$ is $[\alpha_1](\phi_1' \rtimes \phi_2) \wedge [\alpha_2](\phi_1' \rtimes \phi_2)$. We proves that $P_1 \mid P_2$ satisfies the first formula (the other part is similar). First the condition of $\texttt{free}(P_2) \cap \texttt{free}(\phi_1)$ ensures that there is no P_2' s.t. $P_2 \xrightarrow{\alpha_1} P_2'$. As a consequence, if $P_1 \mid P_2 \xrightarrow{alpha_1} P'$, it means that $P_1 \xrightarrow{\alpha_1} P_1'$. By hypothesis, $P_1' \models \phi_1'$ and we use the induction hypothesis to get $P_1' \mid P_2 \models (\phi_1' \rtimes \phi_2)$.
- Other cases are treated by destructing one construct, following the definition, and using the induction hypothesis.

Lemma 20 (Description of free names). *If* $\mathcal{C}, \mathcal{I}, \Gamma \vdash P : \Delta$ *then* $\texttt{free}(P) \subseteq \texttt{free}(\Delta) \cup \texttt{free}(\Gamma)$

Proof. Easily done by induction on the typing judgement.

Lemma 21 (Nature of an interleaving).

Let $\Delta = \{s_k[\mathtt{p}_k] : \mathtt{q}_k \overset{!}{?} \{ \begin{smallmatrix} l_i(x_i) \\ e_i \mapsto l_i\langle e_i'\rangle(x_i) \end{smallmatrix} \{A_i\}\langle E_i\rangle.T_{k,i}\}_{i\in I}\}_k$ *and* $\Gamma = \{a_j : \mathtt{I}(\mathcal{G}_j)\}_j$ *be well-formed, then the formula* $\texttt{Inter}(\Delta, \Gamma)$ *is equivalent to a formula guarded by several* $\forall$ *operators guarding a conjunction of formulae, each one starting with a modality, and this modalities are in bijection with the pairs of* $(s_k[\begin{smallmatrix} \mathtt{p}_k, \mathtt{q}_k \\ \mathtt{q}_k, \mathtt{p}_k \end{smallmatrix}], l_{k,i})$ *and* $(a_j, \varnothing)$.

Proof. By induction over the pair Γ, Δ, using the definition of the $\rtimes$ operator.

Do not understand at all and symbol is too strong

Proposition 4 (Soundness). *If* $\mathcal{I}; \mathcal{C}; \Gamma \vdash P \triangleright \Delta$ *and* $\sigma \models \mathcal{I}$*, then:* $P, \sigma \models (\mathcal{I} \wedge \mathcal{C} \Rightarrow \texttt{Inter}(\Delta, \Gamma))$.

Proof. By induction on the typing judgment:

- Case *selection*. We have $\Delta = \Delta', s[\mathtt{p}] : \mathtt{p}_0?\{l_i(x_i : S_i)\{A_i\}\langle E_i\rangle.\mathcal{L}_i\}_{i\in I}$ and $P = s[\mathtt{p}, \mathtt{p}_0]?\{(x_i : S_i)\{A_i\}.P_i\}_{i\in I}$. We use Lemma 21 to state the formula we are trying to validate using P is a conjunction on several formulas, all beginning with a different modality from the pairs $(s_k[\mathtt{p}_k], l_{k,i})$ and $(a_j, \varnothing)x$. As P is only able

to perform an action $s[\mathtt{p}, \mathtt{p}_0]?$, all formulas starting with a modality associated to a different name are automatically satisfied, and we have to prove that for each i:

$$P, \sigma \models \mathcal{C} \implies \|T_i\|^{s[\mathtt{p}]}, \mathtt{S} \rtimes \bigtimes_{s_k[\mathtt{p}_k]:T_k \in \Delta'} \|T_k\|^{s_k[\mathtt{p}_k], \mathcal{I}}, \mathtt{S} \rtimes$$

$$\bigtimes_{a_j:\mathcal{G}_j[\mathtt{p}_j] \in \Gamma} \forall s_j.[a_j(s_j[\mathtt{p}_j])] \|\mathcal{G}_j|_{p_j}\|^{s_j[\mathtt{p}_j], \mathcal{I}}, \mathtt{S}$$

. We conclude in a way similar to the one followed in the proof of Proposition 2.

- Case *branching*. We have $P = s[\mathtt{p}, \mathtt{p}_0]!\{\{i \in I\}\langle P_i\rangle(x_i : S_i)\}$. We use Lemma 21 to state the formula we are trying to validate using P is a conjunction on several formulas, all beginning with a different prefix. As P is only able to perform an action $s[\mathtt{p}, \mathtt{p}_0]!$, all formulas starting with a different modality are automatically satisfied, and we have to prove . We conclude using the proof of Proposition 2.
- Case *session reception*. We have $P = a(s).P'$ and $\Gamma = a : \mathcal{G}[\mathtt{p}], \Gamma'$. We use Lemma 21 to state the formula we are trying to validate using P is a conjunction on several formulas, all beginning with a different modality. As P is only able to perform an action on a, all formulas starting with a modality associated to a different name are automatically satisfied, and we have to prove that P satisfies $\forall s[\mathtt{p}], [a(s)]\mathtt{Inter}(\Gamma'; \Delta, s : \mathcal{G}|_{\mathtt{p}})$. As P is able to receive $s[\mathtt{p}]$ on a, we use the induction hypothesis to conclude.
- Case *parallel composition*. Easily done by using Lemmas 19 and 20 and the fact that both Γ and Δ are split multiplicatively in the rule for parallel composition we use.
- Case *end* is trivial.

Completeness The erasing operator $\mathbf{Er}(\mathcal{L})$, which translates an asserted type into its unasserted counterpart is straightforwardly defined: we remove every assertion A from the local types. Unasserted typing rules for the judgements $\vdash P \triangleright \Delta$ are easily deduced from the proof rules. Our completeness result is:

Proposition 5 (Completeness).
If $\sigma \models \mathcal{I}$, $\vdash P \triangleright \mathbf{Er}(\Delta)$ *and* $P, \sigma \models (\mathcal{I} \wedge \mathcal{C} \Rightarrow \mathtt{Inter}(\Delta, \Gamma))$ *then* $\mathcal{I}; \mathcal{C}; \Gamma \vdash P \triangleright \Delta$.

Proof. By induction on the unasserted typing judgment, case branching and selection are treated in a way similar to the proof of Proposition 3, parallel composition is done using Lemmas 19 and 20.

Embedding to Pure HML We are able to embed a stateful satisfaction relation $P, \sigma \models \phi$ into a satisfaction relation $P' \models \phi'$ of a standard π-calculus with first-order values, by encoding the σ into a π-process:

$$\begin{array}{l}\|x_1 \mapsto v_1, \ldots, x_n \mapsto v_n\|_{\mathtt{p}} = \quad \overline{a_1}(v_1) \mid \ldots \mid \overline{a_n}(v_n) \mid \\ \quad !x_1(e).a_1(y_1) \ldots a_n(y_n).(\overline{a_1}(\mathtt{eval}(e[y_1 \ldots y_n/x_1 \ldots x_n])) \mid \overline{a_2}(y_2) \mid \ldots \mid \overline{a_n}(y_n)) \mid \ldots \mid \\ \quad !x_n(e).a_1(y_1) \ldots a_n(y_n).(\overline{a_1}(y_1) \mid \ldots \mid \overline{a_{n-1}}(y_{n-1}) \mid \overline{a_n}(\mathtt{eval}(e[y_1 \ldots y_n/x_1 \ldots x_n])))\end{array}$$

For each variable x_i in the domain of the stateσ, we add an output prefix emitting its content on the channel a_i and we add a replicated module that waits for an update e

at x_i, then capture the value of all variables of the current state, replace the variable x_i by evaluating e by `eval`, and then makes available the other ones. Soundness and completeness allow us to state that HML formulae for pairs state/process can be seen as pure HML formulas on the π-processes.

Embedding for state σ is given by the following:

$$\begin{array}{c}\|x_1 \mapsto v_1, \ldots, x_n \mapsto v_n\|_{\mathtt{p}} = \overline{a_1}(v_1) \mid \ldots \mid \overline{a_n}(v_n) \mid \\ !x_1(e).a_1(y_1)\ldots a_n(y_n).(\overline{a_1}(\mathtt{eval}(e\{y_1 \ldots y_n/x_1 \ldots x_n\})) \mid \overline{a_2}(y_2) \mid \ldots \mid \overline{a_n}(y_n)) \mid \\ \ldots \\ !x_n(e).a_1(y_1)\ldots a_n(y_n).(\overline{a_1}(y_1) \mid \ldots \mid \overline{a_{n-1}}(y_{n-1}) \mid \overline{a_n}(\mathtt{eval}(e\{y_1 \ldots y_n/x_1 \ldots x_n\})))\end{array}$$

For each variable x_i in the domain of the stateσ, we add an output prefix emitting its content on the channel a_i and we add a replicated module that waits for an update e at x_i, then capture the value of all variable of the current state, replace the variable x_i by evaluating e w.r.t. the values of the state, and then makes available the other variables.

The embedding for the formula is given by the following:

$$\|[E]\phi\|_{\mathtt{p}} = [\|E\|_{\mathtt{p}}]\|\phi\|_{\mathtt{p}} \qquad \|A\|_{\mathtt{p}} = [\overline{x_1}(v_1)]\ldots[\overline{x_n}(v_n)]A\{v_1, \ldots, v_n/$$

$$x_1, \ldots, x_n\} \text{ where the state variables of } A \text{ are } x_1, \ldots, x_n$$

The following theorem ensures that the encoding is sound and complete.

Proposition 6 (Preciseness).
If $P, \sigma \models \phi$, then $\|P\|_{\mathtt{p}} \mid \|\sigma\|_{\mathtt{p}} \models \|\phi\|_{\mathtt{p}}$.
If $\|P\|_{\mathtt{p}} \mid \|\sigma\|_{\mathtt{p}} \models \|\phi\|_{\mathtt{p}}$ then $P, \sigma \models \phi$

Proof. Easily done by induction on the formulas, remarking first that σ and $\|\sigma\|_{\mathtt{p}}$ have the same LTS.

Embedding Recursion Recursion is absent from the previous embeddings, but can actually be encoded, at the cost of much technical details, we give here a brief sketch of how we proceed. For this purpose, we add to our HML syntax the recursion operators, $\mu X.\phi$ and X (similar to the one present in the μ-calculus [12]).

The main difficulty lies in the interaction between interleaving and recursion: loops coming from different sessions can be interleaved in many different way, and the difficult task is to compute the finite formula which is equivalent to this interleaving.

As a small example consider the following session environment (interactions are replaced by integer labels): $s_1[\mathtt{p}_1] : \mu X.1.2.X, s_2[\mathtt{p}_2] : \mu Y.3.4.Y$. The simplest HML formula describing all possible interleavings is:

$$\begin{array}{c}\mu A.([1]\mu B.([2]A \wedge [3]\mu C.([4]B \wedge [2]([1]C \wedge [4].A))) \wedge \\ [3]\mu D.([4].A \wedge [1]\mu E.([2]D \wedge [4]([2]A \wedge [3]E))))\end{array}$$

We use the following method to obtain a matching HML formula. We use a translation through finite automata. Here is a sketch of the method, which takes as arguments a set session environment Δ:

1. Encode every session judgement $s_i[\mathtt{p}_i] : T_i$ of Δ independently into a formula ϕ_i, conforming to previous embedding and the definitions $\|\mu X.T\|^{s[\mathtt{p}]} = \mu X \|T\|^{s[\mathtt{p}]}$.
2. Translate every formula ϕ_i into a finite automata $\mathcal{A}_i$, one state corresponds to a point between two modalities or a μX in the formula, one transition correponds to either $[\ell](A \wedge [E]\circ)$ (output) or $[\ell](A \Rightarrow \circ)$ (input). Every automata is *directed* with a source state corresponding to the head of the formula and leaf states corresponding to recursion variables (or end of protocols).
3. Compute the automata $\mathcal{A}$, the parallel composition of all the $\mathcal{A}_i$, which is still *directed*.
4. Expand the automata $\mathcal{A}$, in order to obtain an equivalent branch automata, that is, an automata such that there is a root (the starting state) and transitions form a tree (back transitions are allowed but only on the same branch). This could be done by recursively replacing sub-automata with several copies of this sub-automata.
5. Translate back the automata into a formula, every state with more than two incoming transition is encoded as a recursion operator.

One our example, step 1 gives the formulas $\mu X.[1][2].X$ and $\mu Y.[3][4].Y$. Step 2 gives for each formula an automata with 2 states (initial and between $[1]$ (resp. $[3]$) and $[2]$ (resp. $[4]$)). Step 3 gives an automata with 4 states: the inital one, one after $[1]$, one after $[3]$, one after both $[1]$ and $[3]$. This automata is diamond-shaped, and, as a result, not tree-shaped. Step 4 yields an automata with 7 states, which is then translated in the formula described above.

The preciseness proof relies on the fact that the operation described in 3. and 4. give equivalent automata, and that two formulas translated to two equivalent automata are equivalent for the HML satisfaction relation.

The following theorem ensures that the encoding is sound and complete.

Proposition 7 (Preciseness).
If $P, \sigma \models \phi$, *then* $\|P\|_{\mathtt{p}} \mid \|\sigma\|_{\mathtt{p}} \models \|\phi\|_{\mathtt{p}}$.
If $\|P\|_{\mathtt{p}} \mid \|\sigma\|_{\mathtt{p}} \models \|\phi\|_{\mathtt{p}}$ *then* $P, \sigma \models \phi$

I Example: OOI Instrument Control

I.1 Motivating example: Instrument Command Usecase

To demonstrate our framework, we use a scenario based on the Instrument Command (IC) Usecase from the Ocean Observatories Initiative (OOI) [20]. The OOI is an NSF program to provide long-term infrastructure for delivering scientific data from a large network of ocean sensor systems to on-shore research stations around the US. Through the paper we will show how to validate process, engaging in multiple simultaneous instances of the type of session illustrated below. In the IC usecase, a user `U` obtains capabilities to use a particular instrument `I` from the service registry `R`. `U` initiates the session sending `R` an `InterfaceId` message which states the desired measurement type. `R` replies with the maximum number of measurements that `U` is permitted to make in this session. The process enters the main session loop; in each recursion step `U` has the choice of sending `I` the next `Command` via the `more` branch case, or to end the session via the `quit` case. The global specification declares that roles `U` and `R` must have `credit` and `load` state variables, respectively. `credit` is used to meter the usage of instruments

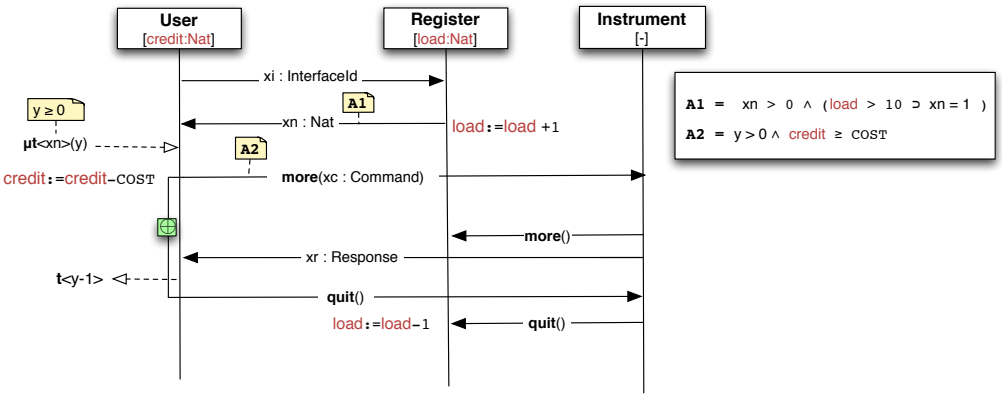

Fig. 15. Instrument control: an example of stateful specification

by each principal, and `load` records the total current load of the instruments, which serves multiple users concurrently. Predicate A_1 ensures that `R` permits `U` to make $x_n > 0$ commands; however, if the current `load` is greater than a certain threshold (fixed as 10 in this example), then `U` is only permitted to make a single command. The subsequent update increments the `load` counter. Next, the recursion μt is annotated with a "loop counter" y, initialised to x_n, and the invariant predicate $y \geqslant 0$. The idea is that one command can be issued in each recursion step and `C` should not issue more than the permitted number of commands; the nested recursion variable t is accordingly annotated with $y - 1$. Within the recursion, the `more` message from `U` to `I` is guarded by the predicate that $y > 0$ and $\texttt{credit} \geqslant \texttt{COST}$, where `COST` is the constant number of credits needed to perform one command; the associated update is to decrease `credit` by `COST`. The `quit` message to `R` at the end of the session has the effect of decrementing `load`.

I.2 Global specification

$\mathcal{G}_{\text{IC}}$ is the global assertion for the protocol illustrated in § I.1 (where we added assertion invariants for `C` and `R`).

$$
\begin{aligned}
\mathcal{G}_{\text{IC}} = {} & ((\texttt{C} : [\texttt{credit} : \texttt{Nat}]\{\texttt{credit} \geqslant 0\},\ \texttt{R} : [\texttt{load} : \texttt{Nat}]\{\texttt{load} \geqslant 0\},\ \texttt{I} : [\_]\{\mathsf{true}\})). \\
& \texttt{C} \to \texttt{R} : (x_i : \texttt{InterfaceId}). \\
& \texttt{R} \to \texttt{C} : (x_n : \texttt{Nat})\{x_n > 0 \wedge (\texttt{R.load} > 10 \supset x_n = 1)\}\langle \texttt{R.load} := \texttt{R.load} + 1\rangle. \\
& \mu \texttt{t}\langle x_n\rangle(y : \texttt{Nat})\{y \geqslant 0\}.\texttt{C} \to \texttt{I} : \{ \\
& \quad \texttt{more}(x_c : \texttt{Command})\{y > 0 \wedge \texttt{C.credit} \geqslant \texttt{COST}\}\langle \texttt{C.credit} := \texttt{C.credit} - \texttt{COST}\rangle.\mathcal{G}_{\text{com}}, \\
& \quad \texttt{quit}().\mathcal{G}_{\text{end}}\}
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{G}_{\text{com}} &= \texttt{I} \to \texttt{C} : (x_r : \texttt{Response})\{\mathsf{true}\}.\texttt{I} \to \texttt{R} : \texttt{more}().\texttt{t}\langle y - 1\rangle \\
\mathcal{G}_{\text{end}} &= \texttt{I} \to \texttt{R} : \texttt{quit}()\langle \texttt{R.load} := \texttt{R.load} - 1\rangle
\end{aligned}
$$

$\mathcal{G}_{\text{IC}}$ specifies non-trivial dependencies between message behaviour and virtual state, which reflect the past and concurrent behaviours of the principal in other sessions.

I.3 Local assertion for R

$\mathcal{L}_{\mathtt{R}}$ is the projection of $\mathcal{G}_{\text{IC}}$ in I.2 on role R. The projection makes use of a standard branch mergeability, which is an extension following e.g. [24].

$$\begin{aligned}
\mathcal{L}_{\mathtt{R}} &= [\mathtt{load} : \mathtt{Nat}]\{\mathtt{load} \geqslant 0\}.\mathcal{L}'_{\mathtt{R}} \qquad \mathcal{L}'_{\mathtt{R}} = \mathtt{C}\,?(x_i : \mathtt{InterfaceId}).\mathcal{L}''_{\mathtt{R}} \\
\mathcal{L}''_{\mathtt{R}} &= \mathtt{C}\,!\,(x_n{:}\mathtt{Nat})\{x_n > 0 \wedge (\mathtt{load} > 10 \supset x_n = 1)\}\langle \mathtt{load} := \mathtt{load} + 1\rangle.\mathcal{L}'''_{\mathtt{R}} \\
\mathcal{L}'''_{\mathtt{R}} &= \mu \mathtt{t}\langle x_n\rangle(y : \mathtt{Nat})\{y \geqslant 0\}.\mathtt{I}\,?\{\mathtt{more}().\mathtt{t}\langle y-1\rangle, \mathtt{quit}()\langle \mathtt{load} := \mathtt{load} - 1\rangle\}
\end{aligned}$$

Notice that the recursion invariant in $\mathcal{L}'''_{\mathtt{R}}$ would actually be, by definition, $\{\exists.\mathtt{C}.\mathtt{credit}.y \geqslant 0 \wedge \mathtt{C}.\mathtt{credit} \geqslant \mathtt{COST}\}$.

I.4 Process

Process $\mathsf{P}_{\mathtt{R}}$ accepts a request to engage in a session specified by global specification $\mathcal{G}_{\text{IC}}$ with the role of R. $\mathsf{P}_{\mathtt{R}}$ is implementing the registry. The other roles involved in the session are user C, and instrument I. We omit the updates when empty, and the labels when there is only one branch.

$$\begin{aligned}
\mathsf{P}_{\mathtt{R}} &= a(z[\mathtt{R}] : \mathcal{G}_{\text{IC}}).\mathsf{P}'_{\mathtt{R}} \qquad \mathsf{P}'_{\mathtt{R}} = z[\mathtt{C},\mathtt{R}]?(x_{id});\mathsf{P}''_{\mathtt{R}} \\
\mathsf{P}''_{\mathtt{R}} &= z[\mathtt{R},\mathtt{C}]!\{\{\mathtt{load} > 10\} \mapsto \langle 1\rangle(x_1)\langle \mathtt{load} := \mathtt{load} + 1\rangle.\mathsf{P}''_{\mathtt{R},1} \\
&\qquad\qquad \{\mathtt{load} \leqslant 10\} \mapsto \langle 2\rangle(x_2)\langle \mathtt{load} := \mathtt{load} + 1\rangle.\mathsf{P}''_{\mathtt{R},2} \\
\mathsf{P}''_{\mathtt{R},n} &= (\mu X(y).z[\mathtt{I},\mathtt{R}]?\{\mathtt{more}().t\langle y-1\rangle,\ \mathtt{quit}()\langle \mathtt{load} := \mathtt{load} - 1\rangle.\mathbf{0}\})\langle x_n\rangle
\end{aligned}$$

In $\mathsf{P}'_{\mathtt{R}}$, R receives the identifier x_{id} from C, and tests if state variable load is greater than the threshold of 10. If so, it sends the value 1; otherwise it sends 2. For brevity, we have parametrised the definitions of $\mathsf{P}''_{\mathtt{R}}$ and $\mathsf{P}'''_{\mathtt{R}}$ by $n \in \{1, 2\}$, with n used to initialise the later recursion parameter y. Each guarded-case leads to the appropriate $\mathsf{P}''_{\mathtt{R},n}$, which increments load. R then enters the recursion, following I through the command-loop. R uses a branching inside the recursion: if more is received enters another recursion; otherwise, load is decremented.

I.5 Validation of two threads

We show selected extracts from the validation of a principal executing two parallel threads, each behaving as $P_{\mathtt{R}}$ in I.4. We set $\Gamma = a : \mathcal{G}_{\text{IC}}$, namely the principal can receive invitations to act in $\mathcal{G}_{\text{IC}}$, hence $\mathcal{I}$ is defined as $\mathtt{load} \geqslant 0$. We illustrate the validation of $[P_{\mathtt{R}} \mid P_{\mathtt{R}}]\sigma_p$ proceeding top-down and using the proof rules in Figure 4. The first rule to be applied is [N1]:

$$\frac{\sigma_p, \sigma_a \models \mathtt{load} \geqslant 0 \quad \mathtt{load} \geqslant 0; \mathsf{true}; \Gamma \vdash P_{\mathtt{R}} \mid P_{\mathtt{R}} \rhd \varnothing}{\mathsf{true}; \Gamma \vdash [P_{\mathtt{R}} \mid P_{\mathtt{R}}]\sigma_p \rhd [\varnothing]\sigma_a}$$

The process can be validated under the assumption that it runs in a state which satisfies the invariant $\mathtt{load} \geqslant 0$ (e.g., $\sigma_p(\mathtt{load}) = \sigma_a(\mathtt{load}) = 3$). Next we apply [PAR] which decomposes the derivation into the derivation of two identical threads: $\mathtt{load} \geqslant 0; \mathsf{true}; \Gamma \vdash P_{\mathtt{R}} \rhd \varnothing$. We show only the derivation of one thread. The rule would apply in the same way even if the two parallel processes engaged in different types of conversations. Next we apply rule [MACC]:

$$\frac{\mathcal{G}_{\text{IC}} \upharpoonright_{\mathtt{R}} = [\mathtt{load} : \mathsf{Nat}]\{\mathtt{load} \geqslant 0\}.\mathcal{L}'_{\mathtt{R}} \qquad \mathcal{I}; \mathsf{true}; \Gamma \vdash \mathsf{P}'_{\mathtt{R}} \rhd z[\mathtt{R}] : \mathcal{L}'_{\mathtt{R}}}{\mathcal{I}; \mathsf{true}; \Gamma \vdash a[\mathtt{R}](z)\{\mathsf{true}\}.\mathsf{P}'_{\mathtt{R}} \rhd \varnothing}$$

where by definition $\mathcal{I}$ entails the session invariant for R. The third premise is equivalent

to $\mathcal{I}; \mathsf{true}; \Gamma \vdash z[\mathtt{C}, \mathtt{R}]?(x_{id} : \mathtt{InterfaceId}).\mathsf{P}''_{\mathtt{R}} \rhd z[\mathtt{R}] : \mathcal{L}'_{\mathtt{R}}$. After the application of rules $\lfloor_{\text{BCH}}\rfloor$ we derive:

$$\frac{\begin{array}{ll}\mathcal{I} \wedge \mathsf{true} \wedge \mathtt{load} > 10 \supset A[1/x_n] & \mathcal{I}; \mathsf{true}; \Gamma \vdash \mathsf{P}''_{\mathtt{R},1} \rhd z[\mathtt{R}] : \mathcal{L}'''_{\mathtt{R}} \\ \mathcal{I} \wedge \mathsf{true} \wedge \mathtt{load} \leqslant 10 \supset A[2/x_n] & \mathcal{I}; \mathsf{true}; \Gamma \vdash \mathsf{P}''_{\mathtt{R},2} \rhd z[\mathtt{R}] : \mathcal{L}'''_{\mathtt{R}}\end{array}}{\mathcal{I}; \mathsf{true}; \Gamma \vdash z[\mathtt{A}, \mathtt{C}]!\{\{\mathtt{load} > 10\} \mapsto \langle 1\rangle(x_n)\langle E\rangle.\mathsf{P}''_{\mathtt{R},1}, \{\mathtt{load} \leqslant 10\} \mapsto \langle 2\rangle(x_n)\langle E\rangle.\mathsf{P}''_{\mathtt{R},2}\} \rhd z[\mathtt{R}] : \mathcal{L}''_{\mathtt{R}}}$$

where we recall A is defined as $\{x_n > 0 \wedge (\mathtt{R.load} > 10 \supset x_n = 1)\}$ and E is $\langle \mathtt{load} := \mathtt{load} + 1\rangle$. Each branch in the premise can rely on the condition $\mathtt{load} > 10$ or $\neg(\mathtt{load} > 10)$. In the case $\mathtt{load} > 10$ it is necessary that x_n takes value 1; this is checked in the first premise for the first branch.