A Theory of Design by Contracts for Distributed Multiparty Interactions

Laura Bocchi[a], Kohei Honda[b], Emilio Tuosto[a], Nobuko Yoshida[c]

[a] *University of Leicester*
[b] *Queen Mary, University of London*
[c] *Imperial College London*

Abstract

Design by Contract (DbC) promotes reliable software development through elaboration of type signatures for sequential programs with logical predicates. This paper presents an assertion method, based on the π-calculus with full recursion, which generalises the notion of DbC to multiparty distributed interactions to enable effective specification and verification of distributed multiparty protocols. Centring on *global assertions* and their *projections* onto endpoint assertions, our method allows clear specifications for typed sessions, constraining the *content* of the exchanged messages, the *choice* of sub-conversations to follow, and *invariants* on recursions. The paper presents key theoretical foundations of this framework, including a sound compositional proof system for verifying processes against assertions.

Keywords: Distributed protocols, multiparty session types, Design-by-Contract, π-calculus, logic

Contents

☆

1. Introduction

This paper introduces an assertion method for the specification and verification of distributed multiparty protocols. We draw ideas from a framework known as Design-by-Contract (DbC), which is widely used in practice in the context of sequential programming [27, 22]. DbC [34] centres on the idea that systems should be built on the basis of precise contracts stipulated between users and sequential programs. Each contract consists of an elaboration of the type signatures of a program with pre/post-conditions and invariants. Consider the following type signature saying "method *foobar* should be invoked with an integer, and it will return (if ever) a string":

```
string foobar(int n)
```

The following elaboration of the type signature of *foobar* would allow us, for instance, to say "method *foobar* should be invoked with an integer n that is less than 10, and it will return (if ever) a string with size n":

```
string foobar(int n)
// precondition: n<10
// postcondition: size(result) = n
```

A type signature stipulates the key *interface* of a program. By associating type signatures with logical predicates, DbC enables a highly effective framework for specifying, validating and managing systems' behaviour. Such framework is usable throughout all phases of software development [38, 30, 32]. As a modelling and programming practice, DbC encourages engineers to make contracts among software modules precise [34, 23], and build a system on the basis of these contracts. The aim is to prevent defensive programming thus to improve reliability.

Reliability is a critical issue also in multi-organisational distributed applications, e.g., Web Services and financial protocols, where the interaction scenarios are much more complex than, say, request-reply. However, the traditional DbC-based approaches are limited to type signatures of sequential procedures. For designing, implementing and managing distributed applications, it is essential to have a tractable and rigorous description of how interactions should proceed through collaborations among participants: the format and the content of the messages, and how conversation structures are to unfold as communications take place. Such descriptions can be used as a basis of the whole range of engineering activities: high-level modelling, design, implementation, runtime management as well as legal and social practice including auditing and standardisation.

Unfortunately the current state of the art for the description of application-level distributed protocols is severely limited. As an example we consider financial protocols. The International Organization for Standardization (ISO) is currently working on a methodology called *universal financial industry message scheme* [46] for specifying and developing financial protocols. Financial protocols are typically used for deciding transactions of critical business and economical significance. The messages for such transactions should be sent and received following a strictly stipulated protocol structure agreed upon by all parties. It is important, both for business and legal concerns, that each party properly carries out the responsibility associated with its roles in the protocol. Currently the description of such a protocol is given only informally (except for the message formats), by combining informal charts and natural language sentences.

There are three main issues in current practice:

1. *It is imprecise:* descriptions of protocols are unclear, ambiguous, and legally unusable.
2. *It is incomplete:* only a small subset of traces from the whole protocol is captured by the description; it is not possible to describe the whole structure and essential constraints.
3. *It is informal:* the description cannot be used for formal reasoning about protocols; for checking their internal consistency; for verifying, either by hand or by machine, the conformance of endpoint programs against a stipulated protocol; for code generation; for testing; and for runtime communication monitoring.

We are currently lacking a methodology, backed up by a rigorous theory, which allows precise, complete and formal description of application-level distributed protocols suitable for all stages of software engineering.

In this paper we introduce a theory of assertions for distributed multiparty interactions, and present a method for specifying and verifying structures and constraints of distributed protocols inspired by the notion of Design-by-Contract (DbC). Adapting DbC to concurrency and communications poses new challenges due to the fact that, for instance, responsibilities are spread among distributed participants, and that each participant has a different knowledge of the system and a different perspective of obligations and guarantees.

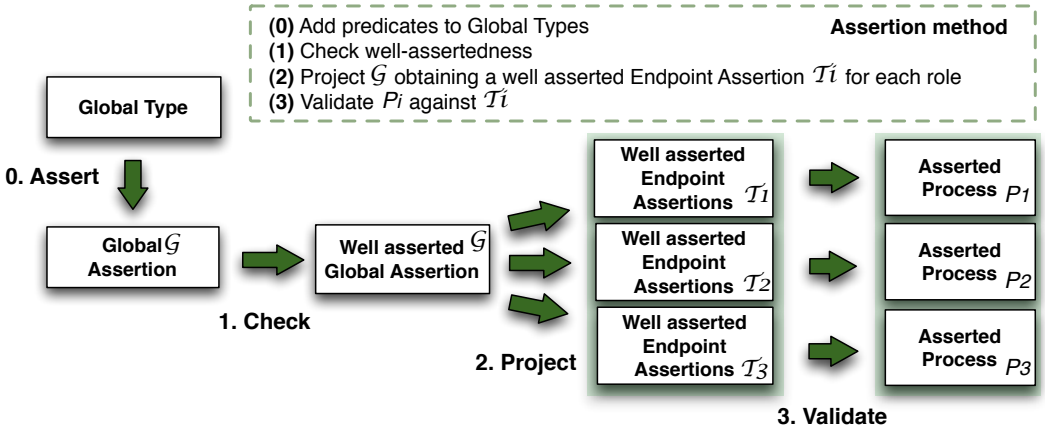

Figure 1: The assertion method

In previous work, it has been shown that multiparty session types can be used as a formal type signature for multiparty sessions [8, 4, 29]. A *session* is an abstraction unit consisting of a structured series of message exchanges among multiple participants. Conversations in distributed applications are organised in terms of sessions, which can possibly interleave in a single application. For example, a session for an electronic commerce can run interleaved with a session for a financial transaction to settle its payment. However, a session type does not, for example, allow to express constraints on the values of the exchanged messages.

Our theory centres on the notion of *global assertion*, which specifies a contract for the participants in a multiparty session by elaborating a session type with logical predicates. Just as in the traditional DbC, the use of logical predicates allows us to specify more refined protocols, regarding, among others, *content* of messages, how *choice* of sub-conversations is made based on preceding interactions, and what *invariants* may be obeyed in recursive interactions.

We present below the key ideas, which are illustrated in Figure 1.

0. Assert - A specification for a multiparty session is given as a *global assertion* $\mathcal{G}$, namely a type signature specifying the structure of the session annotated with logical predicates. A global assertion may be designed from scratch or by elaborating an existing global session type [29].

1. Check - A minimal semantic criterion, *well-assertedness of* $\mathcal{G}$, characterises consistent specifications with respect to the temporal flow of events, to avoid unsatisfiable specifications.

2. Project - $\mathcal{G}$ is projected onto the endpoints, yielding one *endpoint assertion* ($\mathcal{T}_i$) for each participant in the session. Each endpoint assertion specifies the behavioural responsibility, as well as the assumptions that the endpoint can rely on. The consistency of endpoint assertions is preserved by projection, so that well-assertedness can be checked only once at the global level.

3. Validate - Asserted processes, modelled with a variant of the π-calculus annotated with predicates, are verified against endpoint assertions through a sound and relatively complete compositional proof system. Noticeably, an asserted process can be validated against one

or more endpoint assertions, as sessions can interleave in a single application. For example, a session for an electronic commerce can run interleaved with a session for financial transaction to settle its payment.

Contributions. This article is the full version of the extended abstract published in [7]. Here we include the detailed definitions and explanations, additional results and examples, and complete proofs. We also expand the related work. Our contributions include an algorithmic validation of consistency of global assertions (Propositions 3.6 and 4.7); semantic foundations of global assertions through labelled transitions (Propositions 6.6 and 7.4); a sound proof system for the compositional validation of processes against assertions (Theorem 8.4), leading to *predicate error freedom* (Corollary 7.5). The latter result ensures that the process will meet its obligations assuming that the remaining parties do so.

Outline. § 2 gives an informal illustration of global assertions by two examples. § 3 presents the formal syntax of global assertions; § 3.2 defines consistency of global assertions in terms of two principles called *history sensitivity* and *temporal satisfiability*, § 3.3 and § 3.4 present a design-time checker for history sensitivity and temporal satisfiability, respectively. § 4 presents the formal syntax of endpoint assertions and the projection of global assertions onto the endpoints; § 4.1 defines consistency of endpoint assertions, which is proven to be preserved by projection. § 5 introduces our process language and validation of processes; § 5.1 defines the asynchronous π-calculus with session initialisation and extended with predicate annotations and run-time errors to notify violations of the constraints; § 5.2 presents the rules for static local validation against endpoint assertions which rely on a refinement relation on endpoint assertions. § 6 presents the semantics of assertions; § 6.1 defines the labelled transition systems for processes and for endpoint assertions, § 6.2 defines behavioural conformance of processes with assertions, and § 6.3 shows that if a process satisfies a specification then it also satisfies any weaker specification. § 7 presents our subject reduction result which yields a predicate error freedom theorem. Soundness is presented in § 8. § 9 concludes with further results and related work. Appendixes contain the proofs and detailed definitions.

2. DbC for Distributed Multiparty Interactions

In our theory contracts are specified as global assertions. A global assertion uses logical formulae to prescribe, for each interaction specified in the underlying session type, what the sending party must guarantee, and dually what the receiving party can rely on. Concretely:

1. Each message exchanged in a session is associated with a predicate which constrains the values carried in the message (e.g., "the seller will send an invoice to the buyer where the amount of product is equal to the amount previously specified by the buyer in the order"). The predicate is an obligation for the sender and a pre-condition for the receiver.
2. Each branch in a session is associated with a predicate which constrains the selection of that branch (e.g., "the seller can choose the 'sell' option for a product only if the ordered quantity does not exceed the stock"). The predicate is an obligation for the selector and a pre-condition for the other participant.
3. Each recursion in a session is associated with an invariant representing an obligation to be maintained by all parties at each repetition of the recursion (e.g., "while negotiating, seller and buyer maintain the price per unit about a fixed threshold").

The aim of this paper is to provide a design-time method for specification and validation of these constraints. Below we informally illustrate the key ideas of this specification method by showing two examples of global assertion using sequence diagrams annotated by logical formulae.

Example 2.1 (Buyer-Seller). Figure 2 specifies a simple session involving participants `Buyer` and `Seller`. The content of the messages exchanged by the participants is represented by the interaction variables *Order* and *Invoice*, both of type Int.

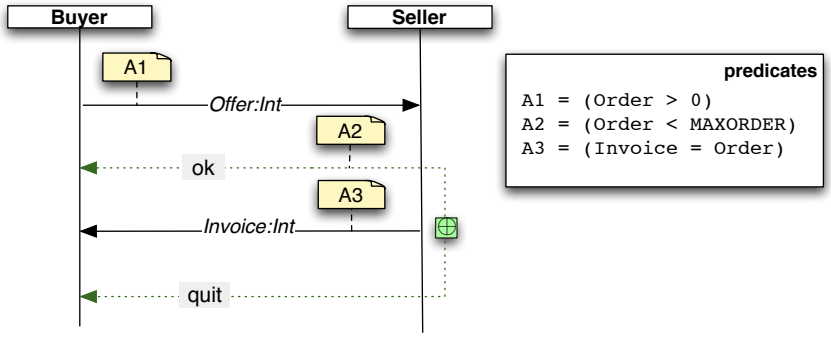

Figure 2: Global assertion for session "Buyer-Seller".

First `Buyer` asynchronously sends `Seller` an order. The message carries the variable *Order* representing the quantity of the order. Then `Seller` chooses between the branches `ok` or `quit`. In the `ok` branch `Seller` sends `Buyer` an invoice (i.e., a message that carries variable *Invoice* representing the dispatched quantity of product). In the `quit` branch the protocol terminates with no sale (e.g., `Seller` is out of stock). The predicates express constraints on the values that can be exchanged. The example uses three predicates. By $A1$, `Buyer` guarantees that the quantity of the order is greater that 0 and, dually, `Seller` can rely on this fact. By $A2$, `Seller` selects the branch `ok` only if the quantity requested by the order is less than the constant $MAXORDER$ (e.g., orders that are too large must be rejected). $A3$ specifies the relationship between the data previously sent in the order and the data being sent by the seller in the invoice.

Example 2.2 (Three Parties with Recursion). Figure 3 describes a multiparty session among the participants `Buyer`, `Seller`, and `Warehouse`. The content of the messages is represented by the interaction variables v_q, v_o of type Int, v_p, v_o of type Price, v_a, v_n of type Bool and v_k of type Update (e.g., the enumeration type $\{commit, cancel\}$ of commands to the warehouse database).

`Buyer` sends `Warehouse` a request v_q specifying the amount of product to buy; `Warehouse` sends `Seller` a boolean v_a representing the availability of the specified amount of product; `Seller` sends `Buyer` a boolean v_n representing the availability of product and a starting price v_p for the negotiation. Next, `Buyer` selects either to proceed with a recursive negotiation (branch ok) or to quit the protocol (branch quit). In the first case, `Buyer` sends `Seller` an offer v_o. `Seller` continues the negotiation (hag) or terminates the negotiation accepting (sell) or rejecting (quit) the offer. The recursion has one parameter p_v_o that is initially set to v_p (the initial price defined by `Seller`) and, upon recursive invocation, takes the value that v_o had in the previous

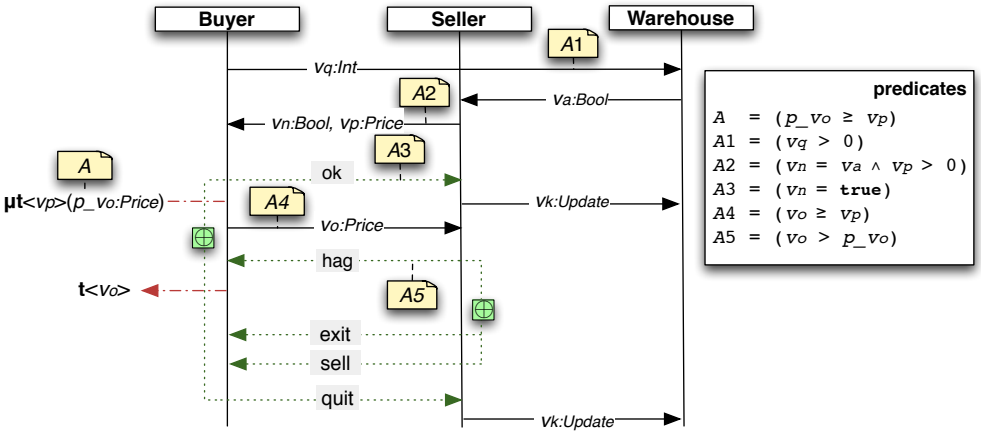

Figure 3: Global assertion for session "Three-Parties with Recursion".

recursion instance. This allows us to compare the current content of v_o with the one of the previous recursion instance (cf. assertion $A5$ below).

The recursion invariant A states that p_v_o is always greater or equal than v_p; by $A1$ `Buyer` guarantees that the amount of product is greater than 0 and dually, `Warehouse` relies upon it; by $A2$ `Seller` guarantees that the availability is the same specified by `Warehouse` and that the initial price is greater than 0. By $A3$ `Buyer` guarantees that the protocol continues (i.e., branch `ok` is selected) only if there is enough product (i.e., $v_n = \mathsf{true}$); by $A4$ `Buyer` guarantees that the offer is always greater or equal than v_p, by $A5$ `Seller` guarantees that the negotiation continues only if `Buyer` has increased the offer with respect to the previous recursion instance (or with respect to the initialisation value when executing the first instance).

3. Global Assertions

In this section we present the syntax and consistency criteria for *global assertions*. We also illustrate an effective method to check for the consistency of global assertions.

3.1. Syntax of Global Assertions

First, we fix the model of the predicates, called *underlying logic*, for which we assume the decidability of the validity of closed formulae. We use the following syntax of logical formulae, often called *predicates*.

$$\begin{array}{lll} A, B & ::= & \mathsf{true} \mid \mathsf{false} \mid \quad \phi(e_1, \ldots, e_n) \mid A \wedge B \mid \neg A \mid \exists v(A) \\ e_1, e_2 & ::= & \mathtt{n} \mid e_1 \ \mathtt{op} \ e_2 \end{array}$$

where e_i ranges over expressions and ϕ ranges over pre-defined atomic predicates with fixed arities and types [33, §2.8]. Constants (e.g., Bool, Int, etc) are denoted with `n` and `op` ranges over pre-defined binary expressions operators. We denote the set of *free variables* of A with $var(A)$, similarly for $var(e)$.

Global assertions (ranged over by $\mathcal{G}, \mathcal{G}', \ldots$) elaborate global session types in [29] with predicates. The syntax is given below:

$$
\begin{array}{rcl}
\mathcal{G} & ::= & \mathtt{p} \to \mathtt{p}' : k\,(\tilde{v} : \tilde{S})\{A\}.\mathcal{G} \\
& | & \mathtt{p} \to \mathtt{p}' : k\,\{\{A_j\} l_j : \mathcal{G}_j\}_{j \in J} \\
& | & \mu \mathbf{t} \langle \tilde{u} : A' \rangle (\tilde{v} : \tilde{S})\{A\}.\mathcal{G} \\
& | & \mathbf{t} \langle \tilde{u} : A' \rangle \\
& | & \mathcal{G}, \mathcal{G}' \\
& | & \mathsf{end}
\end{array}
$$

- $\mathtt{p}, \mathtt{p}', ..$ are *participants*,
- $k, k', ..$ are *channels*,
- $u, v, ..$ are *interaction variables*,
- $S, S', ..$ are *sorts*.

Interaction $\mathtt{p} \to \mathtt{p}' : k\,(\tilde{v} : \tilde{S})\{A\}.\mathcal{G}$ describes a communication between a sender $\mathtt{p}$ and a receiver $\mathtt{p}'$ via the k^{th} session channel (k is a natural number), followed by $\mathcal{G}$. The variables in the vector $\tilde{v}$ are called *interaction variables* and bind their occurrences in A and $\mathcal{G}$; interaction variables are sorted by *sorts* S (Bool, Int, ...) that denote types for first-order message values (i.e., do not include channels). The predicate A constrains the content of $\tilde{v}$: the sender $\mathtt{p}$ *guarantees* A and the receiver $\mathtt{p}'$ *relies on* A (like in the rely-guarantee paradigm [31]).

Branching $\mathtt{p} \to \mathtt{p}' : k\,\{\{A_j\} l_j : \mathcal{G}_j\}_{j \in J}$ allows the selector $\mathtt{p}$ to send to participant $\mathtt{p}'$, through k, a label l_i from $\{l_j\}_{j \in J}$ (J is a finite set of indexes). In this case $\mathtt{p}$ guarantees A_i (upon which $\mathtt{p}'$ can rely). Once l_i is selected, $\mathcal{G}_i$ is to be executed by all parties.

Recursive assertions (cf. [19]) take the form

$$\mu \mathbf{t} \langle \tilde{u} : A' \rangle (\tilde{v} : \tilde{S})\{A\}.\mathcal{G} \tag{3.1}$$

(where $\mathbf{t}$ is an *assertion variable*) and specify how a recursive session should be carried out through interactions among participants. The vector of formal parameters $\tilde{v}$ consists of pairwise distinct variables, and $\tilde{S}$ is a vector of sorts of the same length of $\tilde{v}$ (each v_i in $\tilde{v}$ has sort S_i of $\tilde{S}$). The expression $\langle \tilde{u} : A' \rangle$ in (3.1) denotes the set of all the vectors of values satisfying A' and settles the initial conditions on formal parameters $\tilde{v}$; the vectors $\tilde{v}$ and $\tilde{u}$ have the same length (each u_i corresponds to v_i in $\tilde{v}$ and has sort S_i). Assertion A is the *recursion invariant* which specifies the condition that needs to be obeyed at each *recursion instantiation*, namely each occurrence of the form $\mathbf{t} \langle \tilde{u} : A' \rangle$ in the recursion body $\mathcal{G}$; expression $\langle \tilde{u} : A' \rangle$ denotes the set of vectors of values satisfying A', as in (3.1), and constrains the values of $\tilde{v}$ in the next recursion instance. A recursive assertion can be unfolded to an infinite tree, as in the *equi-recursive* view on recursive types [41]. The free occurrences in A and $\mathcal{G}$ of parameters in $\tilde{v}$ are bound in the recursive assertion. We impose that each recursion instantiation in $\mathcal{G}$ is guarded by prefixes, i.e. the underlying recursive types should be contractive and, for simplicity, assume that there is at least a recursion instantiation in $\mathcal{G}$. In recursive assertions and recursion instantiations, when convenient, we use $\langle \tilde{e} \rangle$ as an abbreviation for $\langle \tilde{u} : \tilde{u} = \tilde{e} \rangle$ (with all variables in $\tilde{u}$ fresh).

Composition $\mathcal{G}, \mathcal{G}'$ represents the parallel interactions specified by $\mathcal{G}$ and $\mathcal{G}'$, while end represents the termination. Sorts and trailing occurrences of end are often omitted.

We write $\mathtt{p} \in \mathcal{G}$ when $\mathtt{p}$ occurs in $\mathcal{G}$. For simplicity, we avoid linearity-check [4] by assuming that each channel in $\mathcal{G}$ is used (maybe repeatedly) only between two parties: one party for input/branching and by the other for output/selection.

Example 3.1 (Global Assertion for "Buyer-Seller"). The protocol illustrated in Example 2.1 is modelled by the global assertion $\mathcal{G}_{BS}$ below, where k_1, k_2, and k_3 are communication channels:

$$
\begin{array}{rcl}
\mathcal{G}_{BS} & = & \mathtt{Buyer} \to \mathtt{Seller} : k_1\,(Order : \mathsf{Int})\{A1\}. \\
& & \quad \mathtt{Seller} \to \mathtt{Buyer} : k_2\{\{A2\}\mathbf{ok} : \mathcal{G}_{pay}, \\
& & \qquad\qquad\qquad\qquad \{\mathsf{true}\}\mathbf{quit} : \mathsf{end}\ \} \\
\mathcal{G}_{pay} & = & \mathtt{Seller} \to \mathtt{Buyer} : k_3\,(Invoice : \mathsf{Int})\{A3\}.\mathsf{end}
\end{array}
$$

$$
\begin{array}{l}
A1 \stackrel{\text{def}}{=} (Order > 0) \\
A2 \stackrel{\text{def}}{=} (Order < MAXORDER) \\
A3 \stackrel{\text{def}}{=} (Invoice = Order)
\end{array}
$$

Example 3.2 (Global Assertion for "Three Parties with Recursion"). The protocol illustrated in Example 2.2 is modelled by the global assertion $\mathcal{G}_{neg}$ below:

$$
\begin{array}{rcl}
\mathcal{G}_{neg} & = & \texttt{Buyer} \to \texttt{Warehouse}\colon k_1\,(v_q : \mathsf{Int})\{A1\}. \\
 & & \texttt{Warehouse} \to \texttt{Seller}\colon k_2\,(v_a : \mathsf{Bool})\{\mathsf{true}\}. \\
 & & \texttt{Seller} \to \texttt{Buyer}\colon k_3\,(v_n, v_p : \mathsf{Bool}, \mathsf{Price})\{A2\}. \\
 & & \texttt{Buyer} \to \texttt{Seller}\colon k_4\{\{A3\}\mathbf{ok} : \mathcal{G}_{ok},\ \{\mathsf{true}\}\mathbf{quit} : \mathcal{G}_{end}\} \\
\mathcal{G}_{ok} & = & \texttt{Seller} \to \texttt{Warehouse}\colon k_5\,(v_k : \mathsf{Update})\{\mathsf{true}\}. \\
 & & \mu\mathbf{t}\langle v_p\rangle(p\_v_o : \mathsf{Price})\{A\}. \\
 & & \texttt{Buyer} \to \texttt{Seller}\colon k_6\,(v_o : \mathsf{Price})\{A4\}. \\
 & & \texttt{Seller} \to \texttt{Buyer}\colon k_7\{\{A5\}\mathbf{hag}\colon \mathbf{t}\langle v_o\rangle, \\
 & & \qquad\qquad \{\mathsf{true}\}\mathbf{exit}\colon \mathsf{end}, \\
 & & \qquad\qquad \{\mathsf{true}\}\mathbf{sell}\colon \mathsf{end}\ \} \\
\mathcal{G}_{end} & = & \texttt{Seller} \to \texttt{Warehouse}\colon k_5\,(v_k : \mathsf{Update})\{\mathsf{true}\}.\mathsf{end}
\end{array}
$$

$$
\begin{array}{rcl}
A & \stackrel{\text{def}}{=} & (p\_v_o \geq v_p) \\
A1 & \stackrel{\text{def}}{=} & (v_q > 0) \\
A2 & \stackrel{\text{def}}{=} & (v_n = v_a \wedge v_p > 0) \\
A3 & \stackrel{\text{def}}{=} & (v_n > 0) \\
A4 & \stackrel{\text{def}}{=} & (v_o \geq v_p) \\
A5 & \stackrel{\text{def}}{=} & (v_o > p\_v_o)
\end{array}
$$

where $k_1 \ldots k_7$ are (linear) channels and the recursion parameter p_v_o (initially set to v_p) denotes the offer of `Buyer` in the previous recursion instance.

3.2. Well Asserted Global Assertions

When setting up global assertions as a contract among multiple participants, we should prevent inconsistent specifications, such as those in which it is logically impossible for a participant to meet the specified obligations. Below we define consistency of global assertions in terms of two constraints on the predicates in terms of two principles: *history sensitivity* and *temporal satisfiability*.

Let $I(\mathcal{G})$ be the set of variables occurring in $\mathcal{G}$ (i.e., the variables exchanged in the interactions and the recursion parameters). A participant $\mathtt{p}$ *knows* a variable $v \in I(\mathcal{G})$ if either

- v occurs in an interaction of $\mathcal{G}$ involving $\mathtt{p}$ as a sender or receiver,
- v is a recursion parameter of recursive assertion $\mu\mathbf{t}\langle\tilde{u} : A'\rangle(\tilde{v}\colon \tilde{S})\{A\}.\mathcal{G}'$ occurring in $\mathcal{G}$ and $\mathtt{p}$ knows (1) all the variables in $var(A')$, and (2) all variables in $var(A'')$ for each recursive instantiation $\mathbf{t}\langle\tilde{u} : A''\rangle$ in $\mathcal{G}'$.

$I(\mathcal{G}) \upharpoonright \mathtt{p}$ denotes the set of variables of $\mathcal{G}$ that $\mathtt{p} \in \mathcal{G}$ knows; this relation can be computed effectively (see Appendix A).

History sensitivity: "*A predicate guaranteed by a participant* $\mathtt{p}$ *can only contain those interaction variables that* $\mathtt{p}$ *knows.*"

Temporal satisfiability: "*For each possible set of values satisfying A and, for each predicate* A' *appearing after A, it is possible to find values satisfying* A'".

History sensitivity requires each participant, say $\mathtt{p}$, to know all the variables involved in the predicates on which $\mathtt{p}$ has an obligation.

Example 3.3 (History sensitivity). *The following global assertion violates history sensitivity*

$$\mathtt{p_A} \to \mathtt{p_B}\colon k_1(u : \mathit{Int})\{\mathsf{true}\}.\ \mathtt{p_B} \to \mathtt{p_C}\colon k_2\,(v : \mathit{Int})\{\mathsf{true}\}.\ \mathtt{p_C} \to \mathtt{p_A}\colon k_3\,(z : \mathit{Int})\{z < u\}.\ \mathsf{end}$$

since $\mathtt{p_C}$ *has to send* z *such that* $z < u$ *without knowing the value of* u*. We model below a similar (although not equivalent) conversation that satisfies history sensitivity:*

$$\mathtt{p_A} \to \mathtt{p_B}\colon k_1(u : \mathit{Int})\{\mathsf{true}\}.\ \mathtt{p_B} \to \mathtt{p_C}\colon k_2\,(v : \mathit{Int})\{v < u\}.\ \mathtt{p_C} \to \mathtt{p_A}\colon k_3\,(z : \mathit{Int})\{z < v\}.\ \mathsf{end}$$

where $\mathtt{p_C}$ *is able to choose a non-violating value for* z *since she knows* v *and she relies on* v *being smaller than* u*.*

Temporal satisfiability requires that a process can always find a valid forward path at each interaction point until it meets the end (if ever). This property is violated, for example, when a global assertions has a trivially false predicate (e.g., $\mathtt{p_A} \to \mathtt{p_B}\colon k_1(v : \mathsf{Int})\{\mathsf{false}\}.\mathcal{G}$). However, it is not sufficient to check the satisfiability of each single predicate to ensure temporal satisfiability. Rather, we should check that no predicates can become unsatisfiable when considered in the temporal sequence of the conversation flow. In other words, for every set of values satisfying a predicate occurring at some point of the protocol, there must exist at least one set of values satisfying all predicates that will occur later in the protocol.

Example 3.4 (Temporal satisfiability). $\mathcal{G}_{TS\_no}$ *below violates temporal satisfiability:*

$$\mathcal{G}_{TS\_no} = \mathtt{p_A} \to \mathtt{p_B}\colon k_1(v : \mathit{Int})\{v > 6\}.\mathtt{p_B} \to \mathtt{p_A}\colon k_2\,(z : \mathit{Int})\{v > z \wedge z > 6\}.\ \mathsf{end}$$

In fact, if $\mathtt{p_A}$ *sends* $v = 7$*, which satisfies* $v > 6$*, then* $\mathtt{p_B}$ *will not be able to find a value that satisfies* $7 > z \wedge z > 6$*. Global assertion* $\mathcal{G}_{TS\_yes}$ *models a similar (although not equivalent) conversation that satisfies temporal satisfiability:*

$$\mathcal{G}_{TS\_yes} = \mathtt{p_A} \to \mathtt{p_B}\colon k_1(v : \mathit{Int})\{v > 7\}.\mathtt{p_B} \to \mathtt{p_A}\colon k_2\,(z : \mathit{Int})\{v > z \wedge z > 6\}.\ \mathsf{end}$$

In $\mathcal{G}_{TS\_yes}$*, predicate* $v > 7$ *guarantees that there is at least one non violating value for* z*.*

Definition 3.5 (Well-assertedness). *Assertions satisfying history sensitivity and temporal satisfiability are called* well-asserted*.*

Proposition 3.6 (Decidability of Well-assertedness). *Checking well-assertedness of global assertions is decidable if the underlying logic is decidable.*

In the rest of this section we present a method for effectively checking the well-assertedness of global assertions. We first introduce a simple compositional checker for history sensitivity (§ 3.3), then we give the compositional rules for checking temporal satisfiability (§ 3.4).

3.3. Effective Checking of History-Sensitivity

The rules for checking history-sensitivity are shown in Figure 4, where for readability we omit sort annotations of the interaction variables. The checker uses the environments $\mathcal{E}$ defined by the following grammar

$$\mathcal{E} \ ::= \ \emptyset \ \mid \ \mathcal{E}, v \,@\, \mathtt{L} \ \mid \ \mathcal{E}, \mathbf{t} : v_1 \,@\, \mathtt{L}_1, \ldots, v_n \,@\, \mathtt{L}_n$$

where in expressions of the form $v \,@\, \mathtt{L}$, $\mathtt{L}$ is a *location*, namely a set $\{\mathtt{p}, \mathtt{p}'\}$ of two participants who know v. We assume that $\mathcal{E}$ uniquely associates (*i*) a location $\mathtt{L}$ to interaction variables and (*ii*) a location $\mathtt{L_i}$ to each recursion parameter v_i in the recursive assertion defining $\mathbf{t}$. We denote

$$\frac{\mathcal{E}, \tilde{v} @ \{\mathtt{p}, \mathtt{p}'\} \vdash \mathcal{G} \quad \forall u \in var(A) \setminus \tilde{v}, \mathcal{E} \vdash u @ \mathtt{p}}{\mathcal{E} \vdash \mathtt{p} \to \mathtt{p}' : k\,(\tilde{v})\{A\}.\mathcal{G}} \qquad \frac{\forall j \in J,\ \mathcal{E} \vdash \mathcal{G}_j \quad \forall u \in \bigcup_{j\in J} var(A_j), \mathcal{E} \vdash u @ \mathtt{p}}{\mathcal{E} \vdash \mathtt{p} \to \mathtt{p}' : k\,\{\{A_j\} l_j :\ \mathcal{G}_j\}_{j\in J}}$$

$$\frac{\mathcal{E} \vdash \mathcal{G} \quad \mathcal{E} \vdash \mathcal{G}'}{\mathcal{E} \vdash \mathcal{G}, \mathcal{G}'} \qquad \frac{}{\mathcal{E} \vdash \mathsf{end}} \qquad \frac{dom(\mathcal{E}) \cup \{v_1, \ldots, v_n\} \supseteq var(A) \setminus \tilde{u}}{\mathcal{E}, \mathbf{t} : v_1 @ \mathtt{L}_1 \ldots v_n @ \mathtt{L}_n \vdash \mathbf{t}\langle \tilde{u} : A \rangle}$$

$$\frac{\mathcal{E}, \mathbf{t} : v_1 @ \mathtt{L}_1 \ldots v_n @ \mathtt{L}_n \vdash \mathcal{G} \quad dom(\mathcal{E}) \supseteq var(A) \setminus \tilde{u} \quad dom(\mathcal{E}) \cup \{v_1, \ldots, v_n\} \supseteq var(A')}{\mathcal{E} \vdash \mu\mathbf{t}\langle \tilde{u} : A \rangle (v_1 @ \mathtt{L}_1, \ldots, v_n @ \mathtt{L}_n)\{A'\}.\mathcal{G}}$$

Figure 4: Checker for history sensitivity on global assertions

the domain of $\mathcal{E}$ with $dom(\mathcal{E})$. For recursive assertions, the checker relies on the annotations $\mu\mathbf{t}\langle \tilde{u} : A' \rangle (\tilde{v}_1 @ \mathtt{L}_1 \ldots \tilde{v}_n @ \mathtt{L}_n)\{A\}.\mathcal{G}$ assigning a location to each recursion parameter (an algorithm for automatic annotation is presented in Appendix A).

Hereafter, we write $\mathcal{E} \vdash v @ \mathtt{p}$ when $\mathtt{p} \in \mathcal{E}(v)$.

The rules in Figure 4 enforce history-sensitivity by disciplining the usage of assertion variables and restricting the set of interaction variables that can be used in each predicate. The first two rules require that the sender/selector p must know all the interaction variables of the predicate. 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.

3.4. Effective Checking of Temporal-Satisfiability

The checker for temporal-satisfiability is presented in Definition 3.7. We pre-annotate each occurrence of assertion variables $\mathbf{t}$ with $\mathbf{t}_{A(\tilde{v})}$ where A is the invariant of the recursion binding $\mathbf{t}$ and $\tilde{v}$ are the corresponding formal parameters. This annotation is always possible if $\mathcal{G}$ does not have free assertion variables.

Definition 3.7 (Temporal-Satisfiability of Global Assertions). We recursively define a boolean function $\mathtt{GSat}(\mathcal{G}, A)$ as follows:

1. if $\mathcal{G} = \mathtt{p}_1 \to \mathtt{p}_2 : k\,(\tilde{v} : \tilde{S})\{A'\}.\mathcal{G}'$ then
$$\begin{cases} \mathtt{GSat}(\mathcal{G}, A) = \mathtt{GSat}(\mathcal{G}', A \wedge A'), & \text{if } A \supset \exists \tilde{v}(A') \\ \mathtt{GSat}(\mathcal{G}, A) = \mathsf{false}, & \text{otherwise} \end{cases}$$
2. if $\mathcal{G} = \mathtt{p}_1 \to \mathtt{p}_2 : k\,\{\{A_j\} l_j :\ \mathcal{G}_j\}_{j\in J}$ then
$$\begin{cases} \mathtt{GSat}(\mathcal{G}, A) = \bigwedge_{j\in J} \mathtt{GSat}(\mathcal{G}_j, A \wedge A_j), & \text{if } A \supset (\bigvee_{j\in J} A_j) \\ \mathtt{GSat}(\mathcal{G}, A) = \mathsf{false}, & \text{otherwise} \end{cases}$$
3. if $\mathcal{G} = \mathcal{G}_1, \mathcal{G}_2$ then $\mathtt{GSat}(\mathcal{G}, A) = \mathtt{GSat}(\mathcal{G}_1, A) \wedge \mathtt{GSat}(\mathcal{G}_2, A)$
4. if $\mathcal{G} = \mu\mathbf{t}\langle \tilde{u} : A' \rangle (\tilde{v} : \tilde{S})\{B\}.\mathcal{G}'$ then
$$\begin{cases} \mathtt{GSat}(\mathcal{G}, A) = \mathtt{GSat}(\mathcal{G}', A \wedge B), & \text{if } A \supset \exists \tilde{u}(A') \text{ and } A \wedge A' \supset (B[\tilde{u}/\tilde{v}]) \\ \mathtt{GSat}(\mathcal{G}, A) = \mathsf{false}, & \text{otherwise} \end{cases}$$
5. if $\mathcal{G} = \mathbf{t}_{B(\tilde{v})}\langle \tilde{u} : A' \rangle$ then $\mathtt{GSat}(\mathcal{G}, A) = (A \supset \exists \tilde{u}(A')) \wedge (A \wedge A' \supset (B[\tilde{u}/\tilde{v}]))$
6. if $\mathcal{G} = \mathsf{end}$ then $\mathtt{GSat}(\mathcal{G}, A) = \mathsf{true}$

$\texttt{GSat}(\mathcal{G},A)$ incrementally builds in A the conjunction of all the predicates that precede the current interaction predicate. In (1) we require that for all the values that satisfy A, there exists a set of values for the interaction variables $\tilde{v}$ that satisfy the current predicate A'. In (2) $\texttt{GSat}(\mathcal{G},A)$ takes predicates of branching points disjunctively and requires that (for all the values that satisfy A) there exists at least one branch that can be chosen (i.e., the corresponding predicate A_j is true). Notice that the protocol

$$\texttt{Alice} \to \texttt{Bob}\colon k_1\,(v:\mathsf{Int})\{v>0\}.\ \texttt{Bob} \to \texttt{Alice}\colon k_2\,\{\{v<0\}l_1\colon \mathcal{G}_1, \{v>0\}l_2\colon \mathcal{G}_2\}$$

is well-asserted even if only its second branch is satisfiable. We do not specify any relationship among the predicates in a branching, such as a XOR relationship to enforce determinism in potential paths, in order to allow abstract (but consistent) specifications. For the same reason, well-assertedness does not prohibit unreachable branches. Note that global assertions are trivially satisfiable in a bad environment (i.e., $\texttt{GSat}(\mathcal{G},\mathsf{false}) = \mathsf{true}$).

Example 3.8 (Evaluation of GSat). *We illustrate the evaluation of* $\texttt{GSat}(\mathcal{G}_{TS\_rec},\mathsf{true})$ *where* $\mathcal{G}_{TS\_rec}$ *is an extension of global assertion* $\mathcal{G}_{TS\_yes}$ *of Example 3.4 with recursion, where the predicate of the second interaction has been modified to include the recursion parameter.*

$$\begin{array}{lcl}
\mathcal{G}_{TS_rec} & = & \mu\mathbf{t}\langle u' : u' = 7\rangle(u : \mathit{Int})\{u \geq 7\}.\mathcal{G}' \\
\mathcal{G}' & = & \mathtt{p_A} \to \mathtt{p_B}\colon k_1(v : \mathit{Int})\{v > u\}.\mathcal{G}'' \\
\mathcal{G}'' & = & \mathtt{p_B} \to \mathtt{p_A}\colon k_2\,(z : \mathit{Int})\{v > z \wedge z > 6\}.\ \mathbf{t}\langle u+1\rangle
\end{array}$$

To evaluate $\texttt{GSat}(\mathcal{G}_{TS\_rec},\mathsf{true})$*, first we apply case (4) of Definition 3.7. The first condition*

$$\mathsf{true} \supset \exists u'(u' = 7) \quad \textit{and} \quad \mathsf{true} \wedge (u' = 7) \supset (u' \geq 7)$$

of (4) is satisfied thus the evaluation is reduced to (the evaluation of) $\texttt{GSat}(\mathcal{G}', u \geq 7)$*. Next we apply case (1) of Definition 3.7. The first condition of (1) is satisfied since*

$$(u \geq 7) \supset \exists v(v > u)$$

and the evaluation is reduced to $\texttt{GSat}(\mathcal{G}'', (u \geq 7) \wedge (v > u))$*. Next we apply again case (1) of Definition 3.7. The first condition of (1) is satisfied*

$$(u \geq 7) \wedge (v > u) \supset \exists z((v > z) \wedge (z > 6))$$

and the evaluation is reduced to $\texttt{GSat}(\mathbf{t}\langle u' : u' = u+1\rangle, A)$*, where* $A = (u \geq 7) \wedge (v > u) \wedge (v > z) \wedge (z > 6)$*, which is* true *by (5) since* $A \supset \exists u'(u' = u+1)$ *and* $A \wedge (u' = u+1) \supset (u' > 7)$*.*

The fixed shape of the implications (e.g. no alternating quantifiers) suggests that validation can be done efficiently [42]. Since the algorithm is compositional it can be integrated with the checker in Figure 4.

$\mathcal{G}$ is *well asserted* if it satisfies history sensitivity (i.e., Figure 4) and $\texttt{GSat}(\mathcal{G},\mathsf{true}) = \mathsf{true}$.

4. Endpoint Assertions and Projection

Endpoint assertions, ranged over by $\mathcal{T}, \mathcal{T}', \ldots$, specify the behavioural contract of a session from the perspective of a single participant. The grammar is given as follows:

$$\begin{array}{lcl}
\mathcal{T} & ::= & k!(\tilde{v}\colon \tilde{S})\{A\};\mathcal{T} \mid k?(\tilde{v}\colon \tilde{S})\{A\};\mathcal{T} \mid k\&\{\{A_i\}l_i\colon \mathcal{T}_i\}_{i\in I} \mid k\oplus\{\{A_j\}l_j\colon \mathcal{T}_j\}_{j\in I} \\
 & \mid & \mu\mathbf{t}\langle\tilde{u} : A'\rangle(\tilde{v} : \tilde{S})\{A\}.\mathcal{T} \mid \mathbf{t}\langle\tilde{u} : A'\rangle \mid \mathsf{end}
\end{array}$$

In $k!(\tilde{v}\colon \tilde{S})\{A\};\mathcal{T}$ the sender guarantees that the values sent via k (denoted by $\tilde{S}$-sorted variables $\tilde{v}$) satisfy A, then behaves as $\mathcal{T}$; dually for the receiver $k?(\tilde{v}\colon \tilde{S})\{A\};\mathcal{T}$.

In $k \oplus \{\{A_j\}l_j\colon \mathcal{T}_j\}_{j\in I}$ the selector guarantees A_j when choosing l_j on k; dually assertion $k\&\{\{A_j\}l_j\colon \mathcal{T}_j\}_{i\in I}$ states that A_j can be assumed when branching at k on a label l_j. Assertion $\mu\mathbf{t}\langle\tilde{u} : A'\rangle(\tilde{v} : \tilde{S})\{A\}.\mathcal{T}$ constrains parameters $\tilde{v}$ of types $\tilde{S}$ which initially take values $\tilde{u}$ (also of types $\tilde{S}$) that satisfy A'; the invariant of the recursion is A.

Definition 4.1 (Closed and Open Endpoint Assertion). An endpoint assertion is *closed* (resp. *open*) if any of its predicates does not (resp. it may) contain free variables.

Let $\mathcal{G}$ be a global assertion, A be a predicate, and $\mathtt{p} \in \mathcal{G}$ (i.e., $\mathtt{p}$ is a participant occurring in $\mathcal{G}$). The *projection of A on $\mathtt{p}$ with respect to $\mathcal{G}$*, written $A \upharpoonright_{\mathcal{G}} \mathtt{p}$, is a predicate obtained by existentially quantifying all the variables of A that $\mathtt{p}$ does not know, and is defined as $\exists\tilde{v}_{ext}(A)$ where $\tilde{v}_{ext}$ is the vector of variables in $var(A) \setminus I(\mathcal{G})\upharpoonright\mathtt{p}$. Hereafter, we assume $\mathcal{G}$ understood and write $\_\upharpoonright\mathtt{p}$ instead of $\_\upharpoonright_{\mathcal{G}}\mathtt{p}$. The *projection* function in Definition 4.2 yields an endpoint assertions from a global assertion and a predicate A_P according to a participant $\mathtt{p}$. The predicate A_P models the set of assumptions on which $\mathtt{p}$ can rely when engaging in the endpoint assertion.

Definition 4.2 (Projection). *Given $\mathcal{G}$ and A_P, the* projection of $\mathcal{G}$ on a participant $\mathtt{p}$ with respect to A_P *is denoted by $(\mathcal{G})\downarrow^{A_P}_{\mathtt{p}}$ and recursively defined as follows, assuming $\mathtt{p}_1 \neq \mathtt{p}_2$.*

$$(1)\quad (\mathtt{p}_1 \to \mathtt{p}_2\colon k\,(\tilde{v}\colon \tilde{S})\{A\}.\mathcal{G}')\downarrow^{A_P}_{\mathtt{p}} = \begin{cases} k!(\tilde{v}\colon \tilde{S})\{A\}.(\mathcal{G}')\downarrow^{A\wedge A_P}_{\mathtt{p}} & \text{if } \mathtt{p}=\mathtt{p}_1 \\ k?(\tilde{v}\colon \tilde{S})\{(A\wedge A_P)\upharpoonright\mathtt{p}\}.(\mathcal{G}')\downarrow^{A\wedge A_P}_{\mathtt{p}} & \text{if } \mathtt{p}=\mathtt{p}_2 \\ (\mathcal{G}')\downarrow^{A\wedge A_P}_{\mathtt{p}} & \text{otherwise} \end{cases}$$

$$(2)\quad (\mathtt{p}_1 \to \mathtt{p}_2\colon k\,\{\{A_i\}l_i\colon \mathcal{G}_i\}_{i\in I})\downarrow^{A_P}_{\mathtt{p}} = \begin{cases} k\oplus\{\{A_i\}l_i\colon (\mathcal{G}_i)\downarrow^{A_i\wedge A_P}_{\mathtt{p}}\}_{i\in I} & \text{if } \mathtt{p}=\mathtt{p}_1 \\ k\&\{\{(A_i\wedge A_P)\upharpoonright\mathtt{p}\}l_i\colon (\mathcal{G}_i)\downarrow^{A_i\wedge A_P}_{\mathtt{p}}\}_{i\in I} & \text{if } \mathtt{p}=\mathtt{p}_2 \\ (\mathcal{G}_1)\downarrow^{A_P\wedge\bigvee_{j\in I}A_j}_{\mathtt{p}}\ (=(\mathcal{G}_i)\downarrow^{A_P\wedge\bigvee_{j\in I}A_j}_{\mathtt{p}}) & \text{otherwise} \end{cases}$$

$$(3)\quad (\mathcal{G}_1,\mathcal{G}_2)\downarrow^{A_P}_{\mathtt{p}} = (\mathcal{G}_i)\downarrow^{A_P}_{\mathtt{p}} \qquad \text{if } \mathtt{p}\in\mathcal{G}_i \text{ and } \mathtt{p}\notin\mathcal{G}_j, i\neq j\in\{1,2\}$$

$$(4)\quad (\mu\mathbf{t}\langle\tilde{u} : A'\rangle(\tilde{v}\colon \tilde{S})\{A\}.\mathcal{G})\downarrow^{A_P}_{\mathtt{p}} = \begin{cases} \mu\mathbf{t}\langle\tilde{u} : A'\upharpoonright\mathtt{p}\rangle(\tilde{v}\colon S)\{A\upharpoonright\mathtt{p}\}.(\mathcal{G})\downarrow^{A_P}_{\mathtt{p}} & \text{if } \mathtt{p}\in\mathcal{G} \\ \mathsf{end} & \text{if } \mathtt{p}\notin\mathcal{G} \end{cases}$$

$$(5)\quad (\mathbf{t}\langle\tilde{u} : A'\rangle)\downarrow^{A_P}_{\mathtt{p}} = \mathbf{t}\langle\tilde{u} : A'\upharpoonright\mathtt{p}\rangle$$

$$(6)\quad (\mathsf{end})\downarrow^{A_P}_{\mathtt{p}} = \mathsf{end}$$

If no side condition applies, $(\mathcal{G})\downarrow^{A}_{\mathtt{p}}$ is undefined. The projection *of $\mathcal{G}$ on $\mathtt{p}$, denoted $\mathcal{G}\upharpoonright\mathtt{p}$, is given as $(\mathcal{G})\downarrow^{\mathsf{true}}_{\mathtt{p}}$.*

Clause (1) in Definition 4.2 projects value passing interactions; the projection for the sender $\mathtt{p}_1$ does not change A, which is consistent with the fact that $\mathtt{p}_1$ knows all variables in A (i.e., $A\upharpoonright\mathtt{p} = A$) when $\mathcal{G}$ is well-asserted (by history sensitivity of $\mathcal{G}$). The projection for the receiver $\mathtt{p}_2$ is more delicate. Consider the following well-asserted global assertion:

$$\begin{array}{l} \mathtt{Seller}\to\mathtt{Buyer}\colon k_1\,(cost : \mathsf{Int})\{cost > 10\}. \\ \mathtt{Buyer}\to\mathtt{Bank}\colon k_2\,(pay : \mathsf{Int})\{pay \geq cost\}.\ \mathsf{end} \end{array}$$

The predicate $pay \geq cost$ is meaningless to `Bank` since `Bank` does not know *cost*; rather the projection on `Bank` should be

$$k_2?(pay : \mathsf{Int})\{\exists cost(cost > 10 \wedge pay \geq cost)\}$$

which incorporates the constraint between `Buyer` and `Seller`. In this way, we give `Bank` a stronger pre-condition by using a predicate of the global assertions (without revealing `Bank` the actual value of *cost*). More generally, (1) projects all the past predicates incorporating also the constraints on interactions in which $\mathtt{p}_2$ does not participate. Existential quantification is used to close the predicate with respect to the variables that the participant does not know. The projection (1) provides $\mathtt{p}_2$ with the strongest precondition, avoiding the burden of defensive programming (e.g. the programmer of `Bank` can concentrate on the case $pay \geq 10$).

The case (2) projects branching interactions and it is similar to value passing. The "otherwise" case says the projection should be the same for all branches following [29]. In (3), each participant is in at most a single global assertion to ensure each local assertion is single threaded. In (4), the projection on `p` is the recursive assertion itself with its predicate projected on `p` by existential quantification, similarly in (5). In (6), the projection of any global assertion $\mathcal{G}$ on a participant that does not participate to any interaction/branching in $\mathcal{G}$ is end. Remarkably, by (6) the projection of end on any participant is end.

Example 4.3 (Projection of "Buyer-Seller"). We define below the projections of $\mathcal{G}_{BS}$ (Example 3.1) on the two participants $\mathcal{T}_{Buyer} = \mathcal{G}_{BS} \upharpoonright \mathtt{Buyer}$, and $\mathcal{T}_{Seller} = \mathcal{G}_{BS} \upharpoonright \mathtt{Seller}$:

$$\begin{array}{rcl} \mathcal{T}_{Buyer} & = & k_1!(Order : \mathsf{Int})\{A1\}; \\ & & k_2\&\{\{A1 \wedge A2\}\mathbf{ok}\colon k_3?(Invoice : \mathsf{Int})\{A1 \wedge A2 \wedge A3\},\ \{A1\}\mathbf{quit}\colon \mathsf{end}\} \\ \mathcal{T}_{Seller} & = & k_1?(Order : \mathsf{Int})\{A1\}; \\ & & k_2 \oplus \{\{A2\}\mathbf{ok}\colon k_3!(Invoice : \mathsf{Int})\{A3\},\ \{\mathsf{true}\}\mathbf{quit}\colon \mathsf{end}\} \end{array}$$

where $A1 \stackrel{\text{def}}{=} (Order > 0)$, $A2 \stackrel{\text{def}}{=} (Order < MAXORDER)$, and $A3 \stackrel{\text{def}}{=} (Invoice = Order)$ (cf. Example 3.1). The projections in Example 4.3 do not introduce existential quantifiers since both participants know all the variables involved in the session. Remarkably, since `Buyer` knows *Order*, the predicates added by the projection do not add relevant information. In fact, $\mathcal{T}_{Buyer}$ could be simplified as follows:

$$k_1!(Order : \mathsf{Int})\{A1\};k_2\&\{\{A2\}\mathbf{ok}\colon k_3?(Invoice : \mathsf{Int})\{A3\},\ \{\mathsf{true}\}\mathbf{quit}\colon \mathsf{end}\}$$

More generally, it is not necessary to incorporate a previously occurred predicate A in the projection of a receive/branching on `p`, say B, when the truth of B does not depend on A. For example, if `p` knows all the variables of a previous predicate A then `p` does not acquire new information from the relations stated in A because `p` knows the actual values assigned to each variable when they are introduced. Another example is when the free variables in A are unrelated to the free variables of the current predicate, as illustrated in Example 4.4.

Example 4.4 (Projection of "Three Parties with Recursion"). We define below the projections of $\mathcal{G}_{neg}$ (Example 3.2) on the three participants $\mathcal{T}_{Warehouse} = \mathcal{G}_{neg} \upharpoonright \mathtt{Warehouse}$, $\mathcal{T}_{Buyer} = \mathcal{G}_{neg} \upharpoonright$

Buyer, and $\mathcal{T}_{Seller} = \mathcal{G}_{neg} \upharpoonright \texttt{Seller}$:

$$
\begin{array}{lcl}
\mathcal{T}_{Warehouse} & = & k_1?(v_q : \mathsf{Int})\{A1\}; k_2!(v_a : \mathsf{Bool})\{\mathsf{true}\} \\
& & k_5?(v_k : \mathsf{Update})\{A \wedge A_2\}; \mathsf{end} \\
\\
\mathcal{T}_{Buyer} & = & k_1!(v_q : \mathsf{Int})\{A1\}; k_3?(v_n, v_p : \mathsf{Bool}, \mathsf{Price})\{A2'\}; \\
& & k_4 \oplus \{\{A3\}\mathbf{ok}\colon \mathcal{T}^{ok}_{Buyer},\ \{\mathsf{true}\}\mathbf{quit}\colon \mathsf{end}\} \\
\mathcal{T}^{ok}_{Buyer} & = & \mu \mathbf{t}\langle v_p \rangle (p\_v_o : \mathsf{Price})\{A\}. \\
& & k_6!(v_o : \mathsf{Price})\{A4\}; \\
& & k_7 \& \{\{A5\}\mathbf{hag}\colon \mathbf{t}\langle v_o \rangle,\ \{\mathsf{true}\}\mathbf{exit}\colon \mathsf{end},\ \{\mathsf{true}\}\mathbf{sell}\colon \mathsf{end}\} \\
\\
\mathcal{T}_{Seller} & = & k_2?(v_a : \mathsf{Bool})\{\mathsf{true}\}; k_3!(v_n, v_p : \mathsf{Bool}, \mathsf{Price})\{A2\}; \\
& & k_4 \& \{\{A3\}\mathbf{ok}\colon \mathcal{T}^{ok}_{Seller},\ \{\mathsf{true}\}\mathbf{quit}\colon k_5!(v_k : \mathsf{Update})\{\mathsf{true}\}\} \\
\mathcal{T}^{ok}_{Seller} & = & k_5!(v_k : \mathsf{Update})\{\mathsf{true}\}; \\
& & \mu \mathbf{t}\langle v_p \rangle (p\_v_o : \mathsf{Price})\{A\}. \\
& & k_6?(v_o : \mathsf{Price})\{A4\}; \\
& & k_7 \oplus \{\{A5\}\mathbf{hag}\colon \mathbf{t}\langle v_o \rangle,\ \{\mathsf{true}\}\mathbf{exit}\colon \mathsf{end},\ \{\mathsf{true}\}\mathbf{sell}\colon \mathsf{end}\}
\end{array}
$$

where $A \stackrel{\text{def}}{=} (p\_v_o \geq 100)$, $A1 \stackrel{\text{def}}{=} (v_q > 0)$, $A2 \stackrel{\text{def}}{=} (v_n = v_a \wedge v_p > 0)$, $A2' \stackrel{\text{def}}{=} (\exists v_n.A2)$, $A3 \stackrel{\text{def}}{=} (v_n = \mathsf{true})$, $A4 \stackrel{\text{def}}{=} (v_o \geq v_p)$, and $A5 \stackrel{\text{def}}{=} (v_o > p\_v_o)$. In the initialisation of the recursive parameters $\langle v_p \rangle$ is a shortcut for $\langle u : u = v_p \rangle$. Notice that the projection of $A2$ on Seller is $A2'$ where the variable v_n, which is not known by Seller, is closed with the existential quantifier.

Noticeably, the predicate of the last interaction of $\mathcal{T}_{warehouse}$ in Example 4.4 can be simplified by substituting $A \wedge A2$ with true since A and A_2 do not add information on v_k.

For readability, in the examples of this paper we will omit unnecessary predicates from the projections. In Definition 4.2, we opted for simplicity rather than efficiency. An alternative definition of projection which implements the optimization (omitting unnecessary predicates) can be found in Appendix B.1.

4.1. Well-Assertedness of Endpoint Assertions

Well-assertedness can be defined on endpoint assertions as for global assertions, characterising the same two principles discussed in §3.2.

An endpoint assertion for endpoint p satisfies history-sensitivity if for each predicate A that annotates a send or a select interaction, p knows all $v \in var(A)$. To check for history-sensitivity of endpoint assertions is vacuous since all actions are now executed by a single participant. The decision procedure for the temporal-satisfiability of endpoint assertions is similar to the one in Definition 3.7.

Definition 4.5 (Temporal-Satisfiability of Endpoint Assertions). We define a boolean function $\texttt{LSat}(\mathcal{T}, A)$ as follows:

1. If $\mathcal{T} = k!(\tilde{v} : S)\{A'\}; \mathcal{T}'$ or $\mathcal{T} = k?(\tilde{v} : S)\{A'\}; \mathcal{T}'$ then
$$\begin{cases} \texttt{LSat}(\mathcal{T}, A) = \texttt{LSat}(\mathcal{T}', A \wedge A'), & \text{if } A \supset \exists \tilde{v}(A') \\ \texttt{LSat}(\mathcal{T}, A) = \mathsf{false}, & \text{otherwise} \end{cases}$$
2. If $\mathcal{T} = k \oplus \{\{A_j\} l_j : \mathcal{T}_j\}_{j \in J}$ or $\mathcal{T} = k \& \{\{A_j\} l_j : \mathcal{T}_j\}_{j \in J}$ then
$$\begin{cases} \texttt{LSat}(\mathcal{T}, A) = \bigwedge_{j \in J} \texttt{LSat}(\mathcal{T}_j, A \wedge A_j), & \text{if } A \supset (\bigvee_{j \in J} A_j) \\ \texttt{LSat}(\mathcal{T}, A) = \mathsf{false}, & \text{otherwise} \end{cases}$$

3. If $\mathcal{T} = \mu \mathbf{t}\langle \tilde{u} : A' \rangle (\tilde{v} : S)\{B\}.\mathcal{T}'$ then
$$\begin{cases} \mathtt{LSat}(\mathcal{T}, A) = \mathtt{LSat}(\mathcal{T}', A \wedge B), & \text{if } A \supset \exists \tilde{u}(A') \text{ and } A \wedge A' \supset B[\tilde{u}/\tilde{v}] \\ \mathtt{LSat}(\mathcal{T}, A) = \mathsf{false}, & \text{otherwise} \end{cases}$$
4. If $\mathcal{T} = \mathbf{t}_{B(\tilde{v})}\langle \tilde{u} : A' \rangle$ then $\mathtt{LSat}(\mathcal{T}, A) = (A \supset \exists \tilde{u}(A')) \wedge (A \wedge A' \supset B[\tilde{u}/\tilde{v}])$
5. If $\mathcal{T} = \mathsf{end}$ then $\mathtt{LSat}(\mathcal{T}, A) = \mathsf{true}$

We say $\mathcal{T}$ satisfies temporal satisfiability if $\mathtt{LSat}(\mathcal{T}, \mathsf{true}) = \mathsf{true}$.

We say $\mathcal{T}$ is *well-asserted* if it satisfies history sensitivity and temporal satisfiability. In Proposition 4.7 we show that projection preserves well-assertedness. This results relies on the following lemma.

Lemma 4.6. *Let $\mathcal{G}_{root}$ be a well-asserted global assertion and $\mathtt{p} \in \mathcal{G}_{root}$. For every sub-term $\mathcal{G}$ of $\mathcal{G}_{root}$ and for every predicate $A_{\mathcal{G}}$ such that $var(A_{\mathcal{G}}) \subseteq I(\mathcal{G}_{root}) \setminus I(\mathcal{G})$*

$$\mathtt{GSat}(\mathcal{G}, A_{\mathcal{G}}) \supset \mathtt{LSat}(\mathcal{G} \upharpoonright \mathtt{p}, A_{\mathcal{G}} \upharpoonright_{\mathcal{G}root} \mathtt{p})$$

PROOF: See Appendix B.2. □

Recall that $A_{\mathcal{G}} \upharpoonright_{\mathcal{G}root} \mathtt{p} = \exists \tilde{v}_{ext}(A_{\mathcal{G}})$ where $\tilde{v}_{ext} = var(A_{\mathcal{G}}) \setminus I(\mathcal{G}_{root} \upharpoonright p)$. By Lemma 4.6, `GSat` (with an assumption on the predicate $A_{\mathcal{G}}$) implies `LSat` (with an assumption on the predicate $\exists \tilde{v}_{ext}(A_{\mathcal{G}})$); noticeably, such assumption is always satisfied if $A_{\mathcal{G}}$ (resp. $\exists \tilde{v}_{ext}(A_{\mathcal{G}})$) is obtained as an incremental conjunction of predicates as that operated by `GSat` (resp. `LSat`) from $\mathcal{G}_{root}$ to $\mathcal{G}$.

Proposition 4.7 (Projections). *Let $\mathcal{G}$ be a well-asserted global assertion. Then for each $\mathtt{p} \in \mathcal{G}$, if $\mathcal{G} \upharpoonright \mathtt{p}$ is defined then $\mathcal{G} \upharpoonright \mathtt{p}$ is also well-asserted.*

PROOF: The case for history-sensitivity is trivial, considering that the predicates annotating send/select interactions of projection $\mathcal{G} \upharpoonright \mathtt{p}$ are exactly those that appear in the corresponding interactions of $\mathcal{G}$ (and $\mathtt{p}$ knows all the interaction variables therein, by well-assertedness of $\mathcal{G}$). The case for temporal satisfiability immediately follows from Lemma 4.6. □

5. Compositional Validation

In § 5.1 we present the language for protocol implementation. We use the π-calculus with multiparty sessions [29] augmented with predicates for checking (both outgoing and incoming) communications and errors to notify a run-time violation of the contract. The reduction rules for asserted processes are also given in § 5.1. The validation system for asserted processes is presented in § 5.2.

5.1. The Asserted π-Calculus

The syntax of *asserted programs* or simply *programs* ($P, Q, \ldots$) is given below.

$P ::= \overline{a}_{[2..\mathtt{n}]}(\tilde{s}).P$	request	$\mid$ if e then P else Q	cond
$\mid a_{[\mathtt{p}]}(\tilde{s}).P$	accept	$\mid (\nu a : \mathcal{G})P$	hide
$\mid s!\langle\tilde{e}\rangle(\tilde{v})\{A\};P$	send	$\mid P \mid Q$	parallel
$\mid s?(\tilde{v})\{A\};P$	receive	$\mid \mu X\langle\tilde{e}\tilde{t}\rangle(\tilde{v}\tilde{s}).P$	rec def
$\mid s \lhd \{A\}l;P$	selection	$\mid X\langle\tilde{e}\tilde{s}\rangle$	rec call
$\mid s \rhd \{\{A_i\}l_i : P_i\}_{i\in I}$	branch	$\mid \mathbf{0}$	idle

$\overline{a}_{[2..\mathtt{n}]}(\tilde{s}).P$ multicasts a session initiation request to each $a_{[\mathtt{p}]}(\tilde{s}).P$ (with $2 \leq \mathtt{p} \leq n$) by multiparty synchronisation through a *shared name* a. Send, receive, and selection, all through a *session channel* s, are associated with a predicate. Branch associates a predicate to each label. In the conditional process, e is a boolean expression. The others are standard. Binders for programs are defined as follows: $\overline{a}[2..n](\tilde{s}).P$ and $a[\mathtt{p}](\tilde{s}).P$ bind $\tilde{s}$ in P, $s!\langle\tilde{e}\rangle(\tilde{v})\{A\};P$ and $s?(\tilde{v})\{A\};P$ bind $\tilde{v}$ in P, $(\nu a : \mathcal{G})P$ binds a in P, and $\mu X\langle\tilde{e}\tilde{t}\rangle(\tilde{v}\tilde{s}).P$ binds $X, \tilde{v}, \tilde{s}$ in P. We denote the set of free program names of P as $\mathrm{fpv}(P)$ and we assume that for every recursion definition $\mu X\langle\tilde{e}\tilde{t}\rangle(\tilde{v}\tilde{s}).P$, $X \in \mathrm{fpv}(P)$ (i.e., the recursion body includes at least one call). P is *closed* if it is without free variables.

(Asserted) runtime processes or simply *processes* are obtained extending programs with the runtime constructs below:

$$\ldots \mid s : \tilde{h} \mid \mathsf{errH} \mid \mathsf{errT} \mid (\nu\tilde{s})P \quad \text{with} \quad h ::= l \mid \tilde{\mathtt{n}}$$

Process $s : h_1..h_n$ represents messages in transit through a session channel s, assuming asynchronous in-order delivery as in TCP, with each h_i denoting either a branching label l or a vector of values $\tilde{\mathtt{n}}$. The empty queue is written $s : \emptyset$. Processes errH and errT denote two kinds of runtime assertion violation: errH (for "error here") indicates a predicate violation by the process itself; and errT ("error there") a violation by the environment (e.g., another party). Process $(\nu\tilde{s})P$ binds $\tilde{s}$ in P, and $(P \mid Q)$ is for parallel composition.

We denote the set of free (resp. bound) names (i.e., channel names, and shared names) of P with $\mathrm{fn}(P)$ (resp. $\mathrm{bn}(P)$).

We define below the *structural congruence* on programs and runtime processes as the smallest relation closed under the following equations and including alpha-renaming.

$$P \mid \mathbf{0} \equiv P \qquad (\nu a : \mathcal{G})\mathbf{0} \equiv \mathbf{0} \qquad (\nu\tilde{s})\mathbf{0} \equiv \mathbf{0} \qquad P \mid Q \equiv Q \mid P \qquad (P \mid Q) \mid R \equiv P \mid (Q \mid R)$$

$$(\nu a : \mathcal{G})(\nu a' : \mathcal{G}')P \equiv (\nu a' : \mathcal{G}')(\nu a : \mathcal{G})P \qquad (\nu a : \mathcal{G})P \mid Q \equiv (\nu a : \mathcal{G})(P \mid Q) \quad \text{if } a \notin \mathrm{fn}(Q)$$

$$(\nu\tilde{s})(\nu\tilde{s}')P \equiv (\nu\tilde{s}')(\nu\tilde{s})P \qquad (\nu\tilde{s})P \mid Q \equiv (\nu\tilde{s})(P \mid Q) \quad \text{if } \tilde{s} \notin \mathrm{fn}(P)$$

$$\mu X\langle\tilde{e}\tilde{t}\rangle(\tilde{v}\tilde{s}).P \equiv P[\mu X(\tilde{v}\tilde{s}).P/X][\tilde{e}\tilde{t}/\tilde{v}\tilde{s}] \text{ where } X\langle\tilde{e}'\tilde{s}'\rangle[\mu X(\tilde{v}\tilde{s}).P/X] \stackrel{\text{def}}{=} \mu X\langle\tilde{e}'\tilde{s}'\rangle(\tilde{v}\tilde{s}).P$$

Example 5.1 (Programs for Three Parties with Recursion). We define a program implementing global assertion $\mathcal{G}_{neg}$ in Examples 3.2 and 4.4, where $\tilde{s} = s_1, \ldots, s_7$ (for simplicity s_i denotes s_{k_i}) and `Warehouse`, `Buyer`, and `Seller` are participants 1, 2, and 3, respectively: $P_{neg} = \overline{a}_{[1,3]}(\tilde{s}).P_2 \mid a_{[1]}(\tilde{s}).P_1 \mid a_{[3]}(\tilde{s}).P_3$. In P_{neg}, `Buyer` invites the other participants; the role of each

participant is implemented below:

$$
\begin{array}{lcl}
P_1 & = & s_1?(v_q)\{A1\};\ s_2!\langle inStock(v_q)\rangle(v_a)\{\mathsf{true}\};\ s_5?(v_k)\{\mathsf{true}\};\ \mathbf{0} \\
P_2 & = & s_1!\langle 10\rangle(v_q)\{A1\};\ s_3?(v_n,v_p)\{A2'\}; \\
 & & \mathsf{if}\ (v_n = \mathsf{true})\ \mathsf{then}\ (s_3 \lhd \{A3\}\mathbf{ok}; P_2^{ok})\ \mathsf{else}\ (s_3 \lhd \{\mathsf{true}\}\mathbf{quit}; \mathbf{0}) \\
P_2^{ok} & = & \mu X\langle v_p, \tilde{s}\rangle(p\_v_o, \tilde{s}).\ s_6!\langle p\_v_o + 1\rangle(v_o)\{A4\}; \\
 & & s_7 \rhd \{\{A5\}\mathbf{hag}:\ X\langle v_o, \tilde{s}\rangle,\ \{\mathsf{true}\}\mathbf{exit}:\ \mathbf{0},\ \{\mathsf{true}\}\mathbf{sell}:\ \mathbf{0}\} \\
P_3 & = & s_2?(v_a)\{\mathsf{true}\};\ s_3!\langle v_a, 100\rangle(v_n,v_p)\{A2\}; \\
 & & s_4 \rhd \{\{A3\}\mathbf{ok}:\ P_3^{ok},\ \{\mathsf{true}\}\mathbf{quit}:\ s_5!\langle restore\rangle(v_k)\{\mathsf{true}\};\ \mathbf{0}\} \\
P_3^{ok} & = & s_5!\langle commit\rangle(v_k)\{\mathsf{true}\}; \mu X\langle v_p, \tilde{s}\rangle(p\_v_o, \tilde{s}).s_6?(v_o)\{A3\}; \\
 & & \mathsf{if}\ (v_o \geq 200)\ \mathsf{then}\ (s_7 \lhd \{\mathsf{true}\}\mathbf{sell}; \mathbf{0})\ \mathsf{else} \\
 & & \quad \mathsf{if}\ (v_o > p\_v_o)\ \mathsf{then}\ (s_7 \lhd \{A5\}\mathbf{hag}; X\langle v_o, \tilde{s}\rangle)\ \mathsf{else}\ (s_7 \lhd \{\mathsf{true}\}\mathbf{exit}; \mathbf{0})
\end{array}
$$

Above, the warehouse program P_1 uses the locally implemented function *inStock*, from Int to Bool, to check if the given quantity of product is available. P_2 intends to buy 10 items of product and, in the recursion body the offer increments of one the price of the previous recursion instance. P_3 always proposes 100 as a starting bid. Specific policies are defined for the selection of the branches in the conditional statement of P_3 (e.g., **sell** is selected if the offer is greater then or equal to 200); such policies are not part of the contract and do not need to be known by the other participants, although they must be compatible with the predicates of the selections in the corresponding assertions.

$$
\overline{a}_{[2..n]}(\tilde{s}).P_1 \mid a_{[2]}(\tilde{s}).P_2 \mid \ldots \mid a_{[n]}(\tilde{s}).P_n \to (\nu\tilde{s})(P_1 \mid P_2 \mid \ldots \mid P_n \mid s_1 : \emptyset \mid \ldots \mid s_n : \emptyset) \quad \text{[R-LINK]}
$$

$$
s!\langle\tilde{e}\rangle(\tilde{v})\{A\}; P \mid s : \tilde{h} \to P[\tilde{\mathfrak{n}}/\tilde{v}] \mid s : \tilde{h} \cdot \tilde{\mathfrak{n}} \qquad (\tilde{e} \downarrow \tilde{\mathfrak{n}} \text{ and } A[\tilde{\mathfrak{n}}/\tilde{v}] \downarrow \mathsf{true}) \quad \text{[R-SEND]}
$$

$$
s?(\tilde{v})\{A\}; P \mid s : \tilde{\mathfrak{n}} \cdot \tilde{h} \to P[\tilde{\mathfrak{n}}/\tilde{v}] \mid s : \tilde{h} \qquad (A[\tilde{\mathfrak{n}}/\tilde{v}] \downarrow \mathsf{true}) \quad \text{[R-RECV]}
$$

$$
s \rhd \{\{A_i\}l_i : P_i\}_{i\in I} \mid s : l_j \cdot \tilde{h} \to P_j \mid s : \tilde{h} \qquad (j \in I \text{ and } A_j \downarrow \mathsf{true}) \quad \text{[R-BRANCH]}
$$

$$
s \lhd \{A\}l : P \mid s : \tilde{h} \to P \mid s : \tilde{h} \cdot l \qquad (A \downarrow \mathsf{true}) \quad \text{[R-SELECT]}
$$

$$
\mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \to P\ \ (e \downarrow \mathsf{true}) \quad \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \to Q\ \ (e \downarrow \mathsf{false}) \quad \text{[R-IF]}
$$

$$
s!\langle\tilde{e}\rangle(\tilde{v})\{A\}; P \to \mathsf{errH} \quad (\tilde{e} \downarrow \tilde{\mathfrak{n}} \text{ and } A[\tilde{\mathfrak{n}}/\tilde{v}] \downarrow \mathsf{false}) \quad \text{[R-SENDERR]}
$$

$$
s?(\tilde{v})\{A\}; P \mid s : \tilde{\mathfrak{n}} \cdot \tilde{h} \to \mathsf{errT} \mid s : \tilde{h} \quad (A[\tilde{\mathfrak{n}}/\tilde{v}] \downarrow \mathsf{false}) \quad \text{[R-RECVERR]}
$$

$$
s \rhd \{\{A_i\}l_i : P_i\}_{i\in I} \mid s : l_j \cdot \tilde{h} \to \mathsf{errT} \mid s : \tilde{h} \quad (j \in I \text{ and } A_j \downarrow \mathsf{false}) \quad \text{[R-BRANCHERR]}
$$

$$
s \lhd \{A\}l : P \to \mathsf{errH} \quad (A \downarrow \mathsf{false}) \quad \text{[R-SELECTERR]}
$$

$$
\frac{P \to P'}{(\nu a : \mathcal{G})P \to (\nu a : \mathcal{G})P'} \qquad \frac{P \to P'}{(\nu s)P \to (\nu s)P'} \qquad \frac{P \to P'}{P \mid Q \to P' \mid Q} \qquad \frac{P \to Q' \quad P \equiv P' \quad Q \equiv Q'}{P \to Q}
$$

[R-ARES/R-SRES/R-PAR/R-STR]

Figure 5: Reduction: non-error cases (top) - error cases (centre) - context rules (bottom)

The reduction semantics with predicate checking is the smallest relation on runtime processes closed under the rules in Figure 5.

The satisfaction of the predicate is checked at each communication action: *send*, *receive*, *selection* and *branching*, where we write $A \downarrow \mathsf{true}$ (resp. $\tilde{e} \downarrow \tilde{\mathtt{n}}$) for a closed formula A (resp. expression $\tilde{e}$) when it evaluates to true (resp. $\tilde{\mathtt{n}}$). When initiating a session, [R-LINK] establishes a session through multiparty synchronisation, generating queues and hiding all session channels. The remaining rules are standard, modelling communications in a session via queues [29, 4].

The validation rules presented in § 5.2 do not rely on the predicates which annotate programs. We use such annotations to model the runtime checking that the contents of incoming and outgoing messages respect the contract. This runtime checking activity is modelled by the reduction rules in Figure 5. Here, we use annotated programs for consistency and symmetry. An extension of our theory where runtime checking is embedded in external dedicated monitors can be found in [17].

5.2. Validation Rules

For validation, we use judgements of the form

$$\mathcal{C}; \Gamma \vdash P \triangleright \Delta$$

which reads: *"under $\mathcal{C}$ and Γ, program P is validated against Δ"*; $\mathcal{C}$, Γ and Δ are defined as follows:

$$\begin{array}{lcll} \mathcal{C} & ::= & \mathsf{true} \mid \mathcal{C} \wedge A & \\ \Delta & ::= & \emptyset \mid \Delta, \tilde{s} : \mathcal{T} @ \mathtt{p} & \\ \Gamma & ::= & \emptyset \mid \Gamma, a : \mathcal{G} \mid \Gamma, X : (\tilde{v} : \tilde{S}) \mathcal{T}_1 @ \mathtt{p}_1 \ldots \mathcal{T}_n @ \mathtt{p}_n & \end{array}$$

$\mathcal{C}$ is an *assertion environment* which incrementally records the conjunction of predicates; hereafter, $\Gamma \vdash P \triangleright \Delta$ abbreviates $\mathsf{true}; \Gamma \vdash P \triangleright \Delta$.

Δ is an *endpoint assertion assignment* which maps the channels for each session, say $\tilde{s}$, to a well-asserted endpoint assertion located at a participant, say $\mathcal{T} @ \mathtt{p}$. Let Δ_1 and Δ_2 be endpoint assertion assignments, their disjoint union Δ_1, Δ_2 is endpoint assertion assignment Δ such that

$$\Delta(\tilde{s}) = \begin{cases} \Delta_1(\tilde{s}) & \text{if } \tilde{s} \in dom(\Delta_1) \text{ and } \forall \tilde{s}' \in dom(\Delta_2),\ \tilde{s} \cap \tilde{s}' = \emptyset \\ \Delta_2(\tilde{s}) & \text{if } \tilde{s} \in dom(\Delta_2) \text{ and } \forall \tilde{s}' \in dom(\Delta_1),\ \tilde{s} \cap \tilde{s}' = \emptyset \\ \bot & \text{otherwise} \end{cases}$$

Hereafter, when writing Δ_1, Δ_2 we assume $\Delta = \Delta_1, \Delta_2$ is defined.

Γ is a *global assertion assignment* which maps shared names to well-asserted global assertions and process variables to the specification of their parameters. We write $\Gamma \vdash a : \mathcal{G}$ when Γ assigns $\mathcal{G}$ to a, and $\Gamma \vdash X : (\tilde{v} : \tilde{S}) \mathcal{T}_1 @ \mathtt{p}_1 \ldots \mathcal{T}_n @ \mathtt{p}_n$ when Γ maps X to the vector of endpoint assertions $\mathcal{T}_1 @ \mathtt{p}_1 \ldots \mathcal{T}_n @ \mathtt{p}_n$ with recursion parameters $\tilde{v}$ sorted by $\tilde{S}$.

The validation rules are given in Figure 6. In each rule, we assume all occurring (global/endpoint) assertions to be well-asserted. We illustrate the key rules.

Rule [SND] validates that participant $\mathtt{p}$ sends values $\tilde{e}$ on session channel k, *provided* that $\tilde{e}$ satisfy the predicate under the current assertion environment, and that the continuation is valid once $\tilde{v}$ gets replaced by $\tilde{e}$. Dually, rule [RCV] validates a value input against the continuation of the endpoint assertion under the extended assertion environment $\mathcal{C} \wedge A$ (i.e., the process can rely on A for the received values after the input). Rules [SEL] and [BRA] are similar. Rules [MACC] and [MCAST]

$$\frac{\mathcal{C} \supset A[\tilde{e}/\tilde{v}] \quad \mathcal{C};\Gamma \vdash P[\tilde{e}/\tilde{v}] \triangleright \Delta, \tilde{s}\colon \mathcal{T}[\tilde{e}/\tilde{v}]\,@\,\mathtt{p}}{\mathcal{C};\Gamma \vdash s_k!\langle \tilde{e}\rangle(\tilde{v})\{A\}; P \triangleright \Delta, \tilde{s}\colon k!(\tilde{v}:\tilde{S})\{A\}; \mathcal{T}\,@\,\mathtt{p}}\,[\textsc{Snd}]$$

$$\frac{\mathcal{C} \wedge A; \Gamma \vdash P \triangleright \Delta,\ \tilde{s}\colon \mathcal{T}\,@\,\mathtt{p}}{\mathcal{C};\Gamma \vdash s_k?(\tilde{v})\{A\}; P \triangleright \Delta,\ \tilde{s}\colon k?(\tilde{v}:\tilde{S})\{A\}; \mathcal{T}\,@\,\mathtt{p}}\,[\textsc{Rcv}]$$

$$\frac{\mathcal{C} \supset A_j \quad \mathcal{C};\Gamma \vdash P \triangleright \Delta, \tilde{s}\colon \mathcal{T}_j\,@\,\mathtt{p} \quad j \in I}{\mathcal{C};\Gamma \vdash s_k \lhd \{A_j\}l_j : P \triangleright \Delta,\ \tilde{s}\colon k \oplus \{\{A_i\}l_i : \mathcal{T}_i\}_{i\in I}\,@\,\mathtt{p}}\,[\textsc{Sel}]$$

$$\frac{\mathcal{C} \wedge A_i; \Gamma \vdash P_i \triangleright \Delta, \tilde{s}\colon \mathcal{T}_i\,@\,\mathtt{p} \quad \forall i \in I}{\mathcal{C};\Gamma \vdash\ s_k \rhd \{\{A_i\}l_i :\ P_i\}_{i\in I} \triangleright \Delta, \tilde{s}\colon k \& \{\{A_i\}l_i : \mathcal{T}_i\}_{i\in I}\,@\,\mathtt{p}}\,[\textsc{Bra}]$$

$$\frac{\mathcal{C};\Gamma \vdash P \triangleright \Delta, \tilde{s}\colon (\Gamma(a) \upharpoonright \mathtt{p})\,@\,\mathtt{p} \quad \mathtt{p} \gtrsim 1}{\mathcal{C};\Gamma \vdash a_{[\mathtt{p}]}(\tilde{s}).P \triangleright \Delta}\,[\textsc{Macc}] \qquad \frac{\mathcal{C};\Gamma \vdash P \triangleright \Delta, \tilde{s}\colon (\Gamma(a) \upharpoonright 1)\,@\,1}{\mathcal{C};\Gamma \vdash \overline{a}_{[2..\mathtt{n}]}(\tilde{s}).P \triangleright \Delta}\,[\textsc{Mcast}]$$

$$\frac{\mathcal{C} \wedge e; \Gamma \vdash P \triangleright \Delta \quad \mathcal{C} \wedge \neg e; \Gamma \vdash Q \triangleright \Delta}{\mathcal{C};\Gamma \vdash \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \triangleright \Delta}\,[\textsc{If}]$$

$$\frac{\mathcal{C};\Gamma \vdash P \triangleright \Delta \quad \mathcal{C};\Gamma \vdash Q \triangleright \Delta'}{\mathcal{C};\Gamma \vdash P \mid Q \triangleright \Delta, \Delta'}\,[\textsc{Conc}] \qquad \frac{\Delta\ \mathsf{end\ only}}{\mathcal{C};\Gamma \vdash \mathbf{0} \triangleright \Delta}\,[\textsc{Idle}]$$

$$\frac{\mathcal{C};\Gamma, a\colon \mathcal{G} \vdash P \triangleright \Delta \quad a \notin \mathrm{fn}(\mathcal{C},\Gamma,\Delta)}{\mathcal{C};\Gamma \vdash (\nu a : \mathcal{G})P \triangleright \Delta}\,[\textsc{Hide}] \qquad \frac{\mathcal{C}';\Gamma' \vdash P \triangleright \Delta' \quad \Gamma' \subseteq \Gamma \quad \mathcal{C} \supset \mathcal{C}' \quad \Delta' \ni \Delta}{\mathcal{C};\Gamma \vdash P \triangleright \Delta}\,[\textsc{Conseq}]$$

$$\frac{\mathcal{T}_1[\tilde{e}/\tilde{v}], \ldots, \mathcal{T}_n[\tilde{e}/\tilde{v}] \quad \text{well-asserted and well-typed under } \Gamma, \tilde{v} : \tilde{S}}{\mathcal{C}\ ;\ \Gamma, X : (\tilde{v} : \tilde{S})\mathcal{T}_1\,@\,\mathtt{p}_1 \ldots \mathcal{T}_n\,@\,\mathtt{p}_n \vdash X\langle \tilde{e}\tilde{s}_1 \ldots \tilde{s}_n\rangle \triangleright \tilde{s}_1 : \mathcal{T}_1[\tilde{e}/\tilde{v}]\,@\,\mathtt{p}_1, \ldots, \tilde{s}_n : \mathcal{T}_n[\tilde{e}/\tilde{v}]\,@\,\mathtt{p}_n}\,[\textsc{Var}]$$

$$\frac{\mathcal{C}\ ;\ \Gamma, X : (\tilde{v} : \tilde{S})\mathcal{T}_1\,@\,\mathtt{p}_1 \ldots \mathcal{T}_n\,@\,\mathtt{p}_n \vdash P \triangleright \tilde{s}_1 : \mathcal{T}_1\,@\,\mathtt{p}_1 \ldots \tilde{s}_n : \mathcal{T}_n\,@\,\mathtt{p}_n}{\mathcal{C}\ ;\ \Gamma \vdash \mu X\langle \tilde{e}\tilde{s}_1 \ldots \tilde{s}_n\rangle(\tilde{v}\tilde{s}_1 \ldots \tilde{s}_n).P \triangleright\ \tilde{s}_1 : \mathcal{T}_1[\tilde{e}/\tilde{v}]\,@\,\mathtt{p}_1 \ldots \tilde{s}_n : \mathcal{T}_n[\tilde{e}/\tilde{v}]\,@\,\mathtt{p}_n}\,[\textsc{Rec}]$$

Figure 6: Validation rules for programs

for session acceptance and request validate the continuation against the projection of the global assertion onto that participant ($\mathtt{n}$ is the number of participants in $\Gamma(a)$ and $\mathtt{p}$ is one of them).

Rule [If] validates a conditional against Δ if each branch is validated against Δ under the extended environment $\mathcal{C} \wedge e$ or $\mathcal{C} \wedge \neg e$, as in the corresponding rule in Hoare logic. It is possible that one of these two assertion environments is not satisfiable. In this case, in every successive send, selection or recursion invocation the entailment of the current predicate from the assertion environment will be trivially true. Notice that the branch validated under the unsatisfiable assertion environment will never be executed. Rule [Conc] takes a disjoint union of two endpoint assertion assignments, and rule [Idle] takes Δ which only contains end as endpoint assertions. Rule [hide] is standard, assuming a is not specified in $\mathcal{C}$.

Rule [Var] validates an instantiation of X with expressions against the result of performing the corresponding substitutions over endpoint assertions associated to X (in the environment). In [Rec], a recursive program is validated if the recursion body P is validated against the given endpoint assertions for its zero or more sessions, under the same endpoint assumptions assigned to the process variable X. The validity of this rule hinges on the partial correctness nature of the semantics of the judgement.

Rule [Conseq] uses the *refinement* relation $\ni$ on endpoint assertions. If $\mathcal{T} \ni \mathcal{T}'$, $\mathcal{T}$ specifies a *more refined behaviour* than $\mathcal{T}'$, in that $\mathcal{T}$ strengthens the predicates for send/selection, so it emits/selects less; and weakens those for receive/branching, so it can receive/accept more. Example 5.2 illustrates this intuition.

Example 5.2 (Refinement). Below, endpoint assertion $\mathcal{T}_s$ refines $\mathcal{T}_w$ (i.e., $\mathcal{T}_s \Subset \mathcal{T}_w$):

$$\begin{array}{lcl} \mathcal{T}_s & = & k_1!(v:\mathsf{Int})\{v>10\}; \quad k_2?(z:\mathsf{Int})\{z>0\}; \quad k_3\&\{\{\mathsf{true}\}\mathbf{l1}:\mathcal{T}_1, \{v>100\}\mathbf{l2}:\mathcal{T}_2\} \\ \mathcal{T}_w & = & k_1!(v:\mathsf{Int})\{v>0\}; \quad k_2?(z:\mathsf{Int})\{z>10\}; \quad k_3\&\{\{v>5\}\mathbf{l1}:\mathcal{T}_1\} \end{array}$$

$\mathcal{T}_s$ has a stronger obligation on the sent value v, a weaker reliance on the received value z, a weaker guarantee at **l1**, and offers one additional branch. The formal definition of refinement adopts the standard definition of unfolding of recursive assertions given below.

Definition 5.3 (Unfolding of Endpoint Assertions). *Assume the following recursive assertion to be well-asserted: $\mathcal{T} = \mu\mathbf{t}\langle\tilde{u}:A\rangle(\tilde{v}:\tilde{S}).\mathcal{T}'$ (omitting the invariant since it does not affect the unfolding). Its one-time unfolding is*

$$\begin{array}{l} \mathsf{unfold}(\mathcal{T}) = (\mathcal{T}'[\mu\mathbf{t}(\tilde{v}:\tilde{S}).\mathcal{T}'/\mathbf{t}], A[\tilde{v}/\tilde{u}]) \\ \textit{with} \quad \mathbf{t}\langle\tilde{u}:A'\rangle[\mu\mathbf{t}(\tilde{v}:\tilde{S}).\mathcal{T}'/\mathbf{t}] \stackrel{def}{=} \mu\mathbf{t}\langle\tilde{u}:A'\rangle(\tilde{v}:\tilde{S}).\mathcal{T}' \end{array}$$

The rest being homomorphic (and returning $A = \mathsf{true}$).

The result of an unfolding is an open process with respect to the recursion parameters $\tilde{v}$; we annotate it with the initialisation predicate A, which must be considered when computing the closure (i.e., by checking that $A[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{true}$ when substituting $\tilde{\mathtt{n}}$). The formal definition of refinement is given below.

Definition 5.4 (Refinement). A binary relation $\mathcal{R}$ over closed well-asserted endpoint assertions is a *refinement relation* if $\mathcal{T}_1\mathcal{R}\mathcal{T}_2$ implies one of the following conditions holds

- $\mathcal{T}_1 = k!(\tilde{v}:\tilde{S})\{A_1\};\mathcal{T}_1'$ and $\mathcal{T}_2 = k!(\tilde{v}:\tilde{S})\{A_2\};\mathcal{T}_2'$ s.t. $A_1 \supset A_2$ and $\mathcal{T}_1'\sigma\mathcal{R}\mathcal{T}_2'\sigma$ for each $\sigma = [\tilde{\mathtt{n}}/\tilde{v}]$ with $A_1\sigma \downarrow \mathsf{true}$.
- $\mathcal{T}_1 = k?(\tilde{v}:\tilde{S})\{A_1\};\mathcal{T}_1'$ and $\mathcal{T}_2 = k?(\tilde{v}:\tilde{S})\{A_2\};\mathcal{T}_2'$ s.t. $A_2 \supset A_1$ and $\mathcal{T}_1'\sigma\mathcal{R}\mathcal{T}_2'\sigma$ for each $\sigma = [\tilde{\mathtt{n}}/\tilde{v}]$ with $A_2\sigma \downarrow \mathsf{true}$.
- $\mathcal{T}_1 = k \oplus \{\{A_{1i}\}l_{1i}:\mathcal{T}_{1i}\}_{i\in I}$ and $\mathcal{T}_2 = k \oplus \{\{A_{2j}\}l_{2j}:\mathcal{T}_{2j}\}_{j\in J}$ s.t. $\forall i \in I$, $l_{1i} = l_{2j}$, $A_{1i} \supset A_{2i}$ and $\mathcal{T}_{1i}\mathcal{R}\mathcal{T}_{2i}$.
- $\mathcal{T}_1 = k\&\{\{A_{1i}\}l_{1i}:\mathcal{T}_{1i}\}_{i\in I}$ and $\mathcal{T}_2 = k\&\{\{A_{2j}\}l_{2j}:\mathcal{T}_{2j}\}_{j\in J}$ s.t. $\forall i \in I$, $l_{1i} = l_{2j}$, $A_{2j} \supset A_{1j}$ and $\mathcal{T}_{1j}\mathcal{R}\mathcal{T}_{2j}$.
- $\forall i,j \in \{1,2\}$ s.t. $i \neq j$, $\mathcal{T}_i = \mu\mathbf{t}\langle\tilde{u}:A'\rangle(\tilde{v}:\tilde{S})\{A\}.\mathcal{T}_i'$, $\mathcal{T}_j = \mathcal{T}_j'[\tilde{\mathtt{n}}/\tilde{v}]$ where $(\mathcal{T}_j', A) = \mathsf{unfold}(\mathcal{T}_i)$, and $A[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{true}$.

where predicates are evaluated in fixed environments $\mathcal{C}$, Γ and Δ.
If $\mathcal{T}_1\mathcal{R}\mathcal{T}_2$ for some refinement relation $\mathcal{R}$, we say $\mathcal{T}_1$ is a *refinement* of $\mathcal{T}_2$ (written $\mathcal{T}_1 \Subset \mathcal{T}_2$).

Definition 5.5 (Refinement of Assertion Assignments). We define refinement of assertions assignments (i.e., $\Delta \Subset \Delta'$) as follows:

- $\Delta \Subset \Delta$
- $\Delta, \tilde{s}:\mathsf{end}\,@\,\mathtt{p} \Subset \Delta$
- $\tilde{s}:\mathcal{T}\,@\,\mathtt{p} \Subset \tilde{s}:\mathcal{T}'\,@\,\mathtt{p}$ if $\mathcal{T} \Subset \mathcal{T}'$

- $\Delta_1, \Delta_2 \Subset \Delta_1', \Delta_2'$ if $\Delta_i \Subset \Delta_i'$, for $i = 1,2$

Example 5.6 (Validation of "Buyer-Seller"). We present an implementation P of $\mathcal{G}_{BS}$ from Example 3.1 and we illustrate a fragment of its validation against the projections $\mathcal{T}_{Buyer} = \mathcal{G}_{BS} \upharpoonright \texttt{Buyer}$ and $\mathcal{T}_{Seller} = \mathcal{G}_{BS} \upharpoonright \texttt{Seller}$ from Example 4.3, that we summarise below for convenience:

$$\begin{array}{lcl} \mathcal{T}_{Buyer} & = & k_1!(Order : \mathsf{Int})\{A1\}; \mathcal{T}'_{Buyer} \\ \mathcal{T}'_{Buyer} & = & k_2 \& \{\{A1\}\mathbf{ok}\colon k_3?(Invoice : \mathsf{Int})\{A3\},\ \{\mathsf{true}\}\mathbf{quit}\colon \mathsf{end}\} \\ \mathcal{T}_{Seller} & = & k_1?(Order : \mathsf{Int})\{A1\}; \mathcal{T}'_{Seller} \\ \mathcal{T}'_{Seller} & = & k_2 \oplus \{\{A2\}\mathbf{ok}\colon k_3!(Invoice : \mathsf{Int})\{A3\},\ \{\mathsf{true}\}\mathbf{quit}\colon \mathsf{end}\} \end{array} \quad \left| \begin{array}{l} A1 = (Order > 0) \\ A2 = (Order < MAXORDER) \\ A3 = (Invoice = Order) \end{array} \right.$$

Program P is defined as follows; for simplicity we write s_i for s_{k_i} and $s_1, s_2, s_3 = \tilde{s}$

$$\begin{array}{lcl} P & = & (\nu a : \mathcal{G}_{BS})(\overline{a}_{[2]}(\tilde{s}).P_{Buyer} \mid a_{[2]}(\tilde{s}).P_{Seller}) \\ \\ P_{Buyer} & = & s_1!\langle 100\rangle(Order)\{A1 \wedge Order < 200\}; P'_{Buyer} \\ P'_{Buyer} & = & s_2 \rhd \{\{A1\}\mathbf{ok} : s_3?(Invoice)\{A3\}; \mathbf{0}, \{\mathsf{true}\}\mathbf{quit} : \mathbf{0}\} \\ \\ P_{Seller} & = & s_1?(Order)\{A1\}; P'_{Seller} \\ P'_{Seller} & = & \mathsf{if}\ (Order > 10 \wedge A2)\ \mathsf{then}\ s_2 \lhd \{A2\}\mathbf{ok} : s_3!\langle Order\rangle(Invoice)\{A3\};\ \mathbf{0} \\ & & \mathsf{else}\ s_2 \lhd \{\mathsf{true}\}\mathbf{quit} : \mathbf{0} \end{array}$$

Notice that the predicate of the first interaction of P_{Buyer} is strengthened with respect to $\mathcal{T}_{Buyer}$, that is `Buyer` guarantees a stronger postcondition on $Order$. In P'_{Seller} the party expresses a precise condition $Order > 10 \wedge A2$ to determine the branch to select. The first fragment of validation for P is the following:

$$\dfrac{\dfrac{\dfrac{\mathsf{true}, a : \mathcal{G}_{BS} \vdash P_{Buyer} \rhd \tilde{s} : \mathcal{T}_{Buyer} \mathbin{@} 1}{\mathsf{true}, a : \mathcal{G}_{BS} \vdash \overline{a}_{[2]}(s).P_{Buyer} \rhd \emptyset}\ [\textsc{Mcast}] \quad \dfrac{\mathsf{true}, a : \mathcal{G}_{BS} \vdash P_{Seller} \rhd \tilde{s} : \mathcal{T}_{Seller} \mathbin{@} 2}{\mathsf{true}, a : \mathcal{G}_{BS} \vdash a_{[2]}(s).P_{Seller} \rhd \emptyset}\ [\textsc{Macc}]}{\mathsf{true}, a : \mathcal{G}_{BS} \vdash \overline{a}_{[2]}(s).P_{Buyer} \mid a_{[2]}(s).P_{Seller} \rhd \emptyset}\ [\textsc{Conc}]}{\mathsf{true}, \emptyset \vdash (\nu a : \mathcal{G}_{BS})(\overline{a}_{[2]}(s).P_{Buyer} \mid a_{[2]}(s).P_{Seller}) \rhd \emptyset}\ [\textsc{Hide}]$$

We illustrate below the validation of the left-hand side branch (omitting the branch for P_{Seller}), where $A1' = A1 \wedge Order < 200$ and $\sigma = [100/Order]$. The rule [Conseq] is applied to suit the predicate $A1'$ of the process with the predicate $A1$ of the endpoint assertion; the rule can be applied since $A1' \supset A1$ therefore $k_1!(Order : \mathsf{Int})\{A1'\}; \mathcal{T}'_{Buyer} \Subset k_1!(Order : \mathsf{Int})\{A1\}; \mathcal{T}'_{Buyer}$ by Definition 5.4. The dots stand for the validation of the quitting branch that we omit.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{-}{A1 \wedge A3\sigma, a : \mathcal{G}_{BS} \vdash \mathbf{0} \rhd \tilde{s} : \mathsf{end} \mathbin{@} 1}\ [\textsc{Idle}]}{A1\sigma, a : \mathcal{G}_{BS} \vdash s_3?(Invoice)\{A3\sigma\}; \mathbf{0} \rhd \tilde{s} : k_3?(Invoice : \mathsf{Int})\{A3\sigma\}; \mathsf{end} \mathbin{@} 1}\ [\textsc{Rcv}] \quad \dots}{\mathsf{true} \supset 100 \geq 100 \wedge 100 < 200 \qquad \mathsf{true}, a : \mathcal{G}_{BS} \vdash P'_{Buyer}\sigma \rhd \tilde{s} : \mathcal{T}'_{Buyer}\sigma \mathbin{@} 1}\ [\textsc{Bra}]}{\mathsf{true}, a : \mathcal{G}_{BS} \vdash s_1!\langle 100\rangle(Order)\{A1'\}; P'_{Buyer} \rhd \tilde{s} : k_1!(Order : \mathsf{Int})\{A1'\}; \mathcal{T}'_{Buyer} \mathbin{@} 1}\ [\textsc{Snd}]}{\mathsf{true}, a : \mathcal{G}_{BS} \vdash s_1!\langle 100\rangle(Order)\{A1'\}; P'_{Buyer} \rhd \tilde{s} : k_1!(Order : \mathsf{Int})\{A1\}; \mathcal{T}'_{Buyer} \mathbin{@} 1}\ [\textsc{Conseq}]$$

Example 5.7 (Validating Seller Process - an Example with Recursion). The validation of P_{neg} from Example 5.1 against $\Delta = \tilde{s} : \mathcal{T}_{Warehouse} \mathbin{@} 1, \tilde{s} : \mathcal{T}_{Buyer} \mathbin{@} 2, \tilde{s} : \mathcal{T}_{Seller} \mathbin{@} 3$ is decomposed in the validations of each single participant using the rule [Conc]. In this example we focus on a fragment of P_3, the `Seller` part of P_{neg}:

$$\mathsf{true}, \emptyset \vdash P_3 \rhd \tilde{s} : \mathcal{T}_{Seller} \mathbin{@} 3 \tag{5.1}$$

$\mathcal{T}_{Seller}$ and P_3 are summarised below for convenience, with each $s_1,\ldots,s_7$ of P_{neg} corresponding to a channel $k_1,\ldots,k_7$ of $\mathcal{T}_{neg}$.

$$
\begin{array}{lcl}
\mathcal{T}_{Seller} & = & k_2?(v_a:\mathsf{Bool})\{\mathsf{true}\};k_3!(v_n,v_p:\mathsf{Bool},\mathsf{Price})\{A2\};\\
 & & k_4\&\{\{A3\}\mathbf{ok}\colon\ \mathcal{T}^{ok},\ \{\mathsf{true}\}\mathbf{quit}\colon\ k_5!(v_k:\mathsf{Update})\{\mathsf{true}\}\}\\
\mathcal{T}_{ok} & = & k_5!(v_k:\mathsf{Update})\{\mathsf{true}\};\mathcal{T}_{rec}\\
\mathcal{T}_{rec} & = & \mu\mathbf{t}\langle v_p\rangle(p\_v_o:\mathsf{Price})\{A\}.\mathcal{T}_{body}\\
\mathcal{T}_{body} & = & k_6?(v_o:\mathsf{Price})\{A4\};\mathcal{T}_{hag}\\
\mathcal{T}_{hag} & = & k_7\oplus\{\{A5\}\mathbf{hag}\colon\ \mathbf{t}\langle v_o\rangle,\ \{\mathsf{true}\}\mathbf{exit}\colon\ \mathsf{end},\ \{\mathsf{true}\}\mathbf{sell}\colon\ \mathsf{end}\}
\end{array}
$$

$$
\begin{array}{lcl}
P_3 & = & s_2?(v_a)\{\mathsf{true}\};\ s_3!\langle v_a,100\rangle(v_n,v_p)\{A2\};\\
 & & s_4\rhd\{\{A3\}\mathbf{ok}\colon\ P^{ok},\ \{\mathsf{true}\}\mathbf{quit}\colon\ s_5!\langle restore\rangle(v_k)\{\mathsf{true}\};\ \mathbf{0}\}\\
P_{ok} & = & s_5!\langle commit\rangle(v_k)\{\mathsf{true}\};P_{rec}\\
P_{rec} & = & \mu X\langle v_p,\tilde{s}\rangle(p\_v_o,\tilde{s}).P_{body}\\
P_{body} & = & s_6?(v_o)\{A4\};P_{hag}\\
P_{hag} & = & \mathsf{if}\ (v_o\geq 200)\ \mathsf{then}\ (s_7\lhd\{\mathsf{true}\}\mathbf{sell};\mathbf{0})\ \mathsf{else}\ P_{else}\\
P_{else} & = & \mathsf{if}\ (v_o>p\_v_o)\ \mathsf{then}\ (s_7\lhd\{\mathsf{true}\}\mathbf{hag};X\langle v_o,\tilde{s}\rangle)\ \mathsf{else}\ (s_7\lhd\{\mathsf{true}\}\mathbf{exit};\mathbf{0})
\end{array}
$$

By applying [RCV], [SND], and [BRA] the validation of P_3 is reduced to the validation of $\mathsf{true},\emptyset\vdash\mathbf{0}\rhd\mathsf{end}$ (straightforward by [IDLE]) and by $A3,\emptyset\vdash P_{ok}\rhd\mathcal{T}_{ok}$ which by [SND] is reduced into

$$A3,\emptyset\vdash P_{rec}\rhd\mathcal{T}_{rec} \tag{5.2}$$

The one-time unfolding of $\mathcal{T}_{rec}$ is $\mathsf{unfold}(\mathcal{T}_{rec})=(\mathcal{T}_{Body}\sigma,p\_v_o=v_p)$ where $\sigma=[\mu\mathbf{t}\ldots/\mathbf{t}]$. By Definition 5.4, $\mathcal{T}_{rec}\sigma[v_p/p\_v_o]\ni\mathcal{T}_{rec}$. By [CONSEQ], (5.2) follows from:

$$A3,\emptyset\vdash P_2\rhd\mathcal{T}_{rec}\sigma[v_p/p\_v_o]\ @\ 3 \tag{5.3}$$

By rule [REC], (5.3) follows from

$$A3,\Gamma\vdash s_6?(v_o)\{A4\};\mathsf{if}\ldots\rhd\tilde{s}:\mathcal{T}_{body}\sigma\ @\ 3\qquad(\Gamma=X:(p\_v_o:\mathsf{Price})\mathcal{T}_{body}\sigma\ @\ 3) \tag{5.4}$$

We show below the derivation tree for (5.4), where we let $\mathcal{C}=A3\wedge A4\wedge\neg(v_0\geq 200)$.

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathcal{T}_{body}\sigma[v_o/p\_v_o]\ \text{well asserted}}}{\mathcal{T}_{body}\sigma[v_o/p\_v_o]\ni\mu\mathbf{t}\langle v_o\rangle(p\_v_o:\mathsf{Price})\{A\}.\mathcal{T}_{body}\quad\mathcal{C},\Gamma\vdash X\langle v_o,\tilde{s}\rangle\rhd\tilde{s}:\mathcal{T}_{body}\sigma[v_o/p\_v_o]\ @\ 3}\ [\textsc{Var}]}{\mathcal{C},\Gamma\vdash X\langle v_o,\tilde{s}\rangle\rhd\tilde{s}:\mu\mathbf{t}\langle v_o\rangle(p\_v_o:\mathsf{Price})\{A\};\mathcal{T}_{body}\ @\ 3}\ [\textsc{Conseq}]}{\mathcal{C}\wedge v_o>p\_v_o\supset v_o>p\_v_o\quad\mathcal{C},\Gamma\vdash X\langle v_o,\tilde{s}\rangle\rhd\tilde{s}:\mathbf{t}\langle v_o\rangle\sigma\ @\ 3}\ (\text{substituting})}{\mathcal{C}\wedge v_o>p\_v_o,\Gamma\vdash s_4\lhd\{v_o>p\_v_o\}\mathbf{hag};X\langle v_o,\tilde{s}\rangle\rhd\tilde{s}:\mathcal{T}_{hag}\sigma\ @\ 3\qquad\ldots}\ [\textsc{Sel}]}{\ldots\qquad\mathcal{C},\Gamma\vdash\mathsf{if}\ (v_o>p\_v_o)\ \mathsf{then}\ (s_7\lhd\{\mathsf{true}\}\mathbf{hag};X\langle v_o,\tilde{s}\rangle)\ \mathsf{else}\ (s_7\lhd\{\mathsf{true}\}\mathbf{exit};\mathbf{0})\rhd\tilde{s}:\mathcal{T}_{hag}\sigma\ @\ 3}\ [\textsc{If}]}{A3\wedge A4,\Gamma\vdash\mathsf{if}\ (v_o\geq 200)\ \mathsf{then}\ (s_7\lhd\{\mathsf{true}\}\mathbf{sell}:\mathbf{0})\ \mathsf{else}\ (P_{else})\rhd\tilde{s}:\mathcal{T}_{hag}\sigma\ @\ 3}\ [\textsc{If}]}{A3,\Gamma\vdash s_6?(v_o)\{A4\};\mathsf{if}\ldots\rhd\tilde{s}:k_6?(v_o:\mathsf{Price})\{A4\};\mathcal{T}_{hag}\sigma\ @\ 3}\ [\textsc{Rcv}]
$$

The … on the premise of (both occurrences of) rule [IF] indicate the missing validation of the first and second branch, respectively. We focus instead on the branch containing the recursive instantiation. In [CONSEQ] the endpoint assertion is unfolded again and, as before by Definition 5.4, the unfolding is a refinement.

6. Semantics of Assertions

A desirable soundness condition is that if a process can be validated against an assertion assignment then its behaviour conforms with the behaviour of that assignment. In order to define the semantics of judgement, we compare the semantics of processes with that of endpoint assertions. To this purpose we define the labelled transition systems for asserted processes and for assertions, and we give the rules for semantic conformance, or *satisfaction*, of asserted processes against endpoint assertion assignment.

6.1. Labelled Transition Systems for Processes and Assertions

The labelled transition relation for asserted processes uses the following labels

$$\alpha ::= \overline{a}[2..n](\tilde{s}) \mid a[i](\tilde{s}) \mid s!\tilde{\mathtt{n}} \mid s?\tilde{\mathtt{n}} \mid \mid s \lhd l \mid s \rhd l \mid \tau$$

for session requesting/accepting, value sending/receiving, selection, branching, and the silent action, respectively. We write $P \xrightarrow{\alpha} Q$ when P has a one-step transition α to Q.

$$\overline{a}_{[2..\mathtt{n}]}(\tilde{s}).P \xrightarrow{\overline{a}[2..n](\tilde{s})} P \qquad a_{[\mathtt{i}]}(\tilde{s}).P \xrightarrow{a[i](\tilde{s})} P \qquad \text{[LinkOut]/ [LinkIn]}$$

$$\frac{A[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{true}}{s_k!\langle\tilde{\mathtt{n}}\rangle(\tilde{v})\{A\};P \xrightarrow{s_k!\tilde{\mathtt{n}}} P[\tilde{\mathtt{n}}/\tilde{v}]} \qquad \frac{A[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{true}}{s_k?(\tilde{v})\{A\};P \xrightarrow{s_k?\tilde{\mathtt{n}}} P[\tilde{\mathtt{n}}/\tilde{v}]} \qquad \text{[Send]/ [Recv]}$$

$$\frac{A \downarrow \mathsf{true}}{s_k \lhd \{A\}l : P \xrightarrow{s_k \lhd l} P} \qquad \frac{(A_j \downarrow \mathsf{true})_{j\in I}}{s_k \rhd \{\{A_i\}l_i : P_i\}_{i\in I} \xrightarrow{s_k \rhd l_j} P_j} \qquad \text{[Sel]/ [Branch]}$$

$$\frac{A[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{false}}{s_k!\langle\tilde{\mathtt{n}}\rangle(\tilde{v})\{A\};P \xrightarrow{\tau} \mathsf{errH}} \qquad \frac{A[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{false}}{s_k?(\tilde{v})\{A\};P \xrightarrow{s_k?\tilde{\mathtt{n}}} \mathsf{errT}} \qquad \text{[SendErr]/ [RecvErr]}$$

$$\frac{A \downarrow \mathsf{false}}{s_k \lhd \{A\}l : P \xrightarrow{\tau} \mathsf{errH}} \qquad \frac{(A_j \downarrow \mathsf{false})_{j\in I}}{s_k \rhd \{\{A_i\}l_i : P_i\}_{i\in I} \xrightarrow{s_k \rhd l_j} \mathsf{errT}} \qquad \text{[LabelErr]/ [BranchErr]}$$

$$\frac{P \to Q}{P \xrightarrow{\tau} Q} \qquad \frac{P \xrightarrow{\alpha} P' \quad \mathrm{bn}(\alpha) \cap \mathrm{fn}(Q) = \emptyset}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \qquad \text{[Tau]/[Par]}$$

$$\frac{P \xrightarrow{\alpha} P'}{(\nu a : \mathcal{G})P \xrightarrow{\alpha} (\nu a : \mathcal{G})P'} \qquad \frac{P \xrightarrow{\alpha} P'}{(\nu \tilde{s})P \xrightarrow{\alpha} (\nu \tilde{s})P'} \qquad \text{[ARes]/[SRes]}$$

$$\frac{P' \xrightarrow{\alpha} Q' \quad P \equiv P' \quad Q \equiv Q'}{P \xrightarrow{\alpha} Q} \qquad \text{[Str]}$$

Figure 7: Labelled transition for processes: non-error cases (top) - error cases (centre) - τ and context cases (bottom)

The transition rules for processes are shown in Figure 7 and are the standard synchronous ones except that (*i*) they include the reduction semantics given in § 5.1 (i.e., $P \to Q$ induces $P \xrightarrow{\tau} Q$), and (*ii*) predicates are checked at each communication action and if the predicate is

$$\frac{-}{\langle \Gamma, \Delta \rangle \xrightarrow{\tau} \langle \Gamma, \Delta \rangle} \quad \text{[TR-TAU]}$$

$$\frac{-}{\langle (a : \mathcal{G}, \Gamma), \Delta \rangle \xrightarrow{\overline{a}[2..n](\tilde{s})} \langle (a : \mathcal{G}, \Gamma), \Delta, \tilde{s} : \mathcal{G} \upharpoonright 1 \,@\, \mathtt{1} \rangle} \quad \text{[TR-LINKOUT]}$$

$$\frac{-}{\langle (a : \mathcal{G}, \Gamma), \Delta \rangle \xrightarrow{a[i](\tilde{s})} \langle (a : \mathcal{G}, \Gamma), \Delta, \tilde{s} : \mathcal{G} \upharpoonright i \,@\, \mathtt{i} \rangle} \quad \text{[TR-LINKIN]}$$

$$\frac{A[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{true}}{\langle \Gamma, (\Delta, \tilde{s} : k!(\tilde{v} : \tilde{S})\{A\}; \mathcal{T} \,@\, \mathtt{p}) \rangle \xrightarrow{s_k ! \tilde{\mathtt{n}}} \langle \Gamma, (\Delta, \tilde{s} : \mathcal{T}[\tilde{\mathtt{n}}/\tilde{v}] \,@\, \mathtt{p}) \rangle} \quad \text{[TR-SEND]}$$

$$\frac{A[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{true}}{\langle \Gamma, (\Delta, \tilde{s} : k?(\tilde{v} : S)\{A\}; \mathcal{T} \,@\, \mathtt{p}) \rangle \xrightarrow{s_k ? \tilde{\mathtt{n}}} \langle \Gamma, (\Delta, \tilde{s} : \mathcal{T}[\tilde{\mathtt{n}}/\tilde{v}] \,@\, \mathtt{p}) \rangle} \quad \text{[TR-RECV]}$$

$$\frac{A_j \downarrow \mathsf{true}}{\langle \Gamma, (\Delta, \tilde{s} : k \oplus \{\{A_i\} l_i : \mathcal{T}_i\}_{i \in I} \,@\, \mathtt{p}) \rangle \xrightarrow{s_k \lhd l_j} \langle \Gamma, (\Delta, \tilde{s} : \mathcal{T}_j \,@\, \mathtt{p}) \rangle} \quad \text{[TR-SEL]}$$

$$\frac{A_j \downarrow \mathsf{true}}{\langle \Gamma, (\Delta, \tilde{s} : k \& \{\{A_i\} l_i : \mathcal{T}_i\}_{i \in I} \,@\, \mathtt{p}) \rangle \xrightarrow{s_k \rhd l_j} \langle \Gamma, (\Delta, \tilde{s} : \mathcal{T}_j \,@\, \mathtt{p}) \rangle} \quad \text{[TR-BRANCH]}$$

Figure 8: Labelled transition for endpoint assertions

violated then in the case of input/branching action the process moves to errT, and in the case of an output/selection the process moves to errH with τ-action.

The asymmetry between the violations in incoming messages, which generate visible moves, and outgoing messages, which result in silent moves, reflects the fact that we want to ensure that a participant will respect a contract as long as the environment does so. In fact, we model the fact that outgoing violations are not immediately visible and can be addressed and fixed locally (e.g., "blocked" by a runtime monitor). Incoming violations instead are caused by messages that are already in a queue, and whose effects are not isolated; such messages may, for instance, have auditing relevance, or raise issues to be addressed by some control authority, etc.

The semantics of endpoint assertions is defined as another labelled transition relation, of form $\langle \Gamma, \Delta \rangle \xrightarrow{\alpha} \langle \Gamma', \Delta' \rangle$, which reads: *the specification* $\langle \Gamma, \Delta \rangle$ allows *the action* α*, with* $\langle \Gamma', \Delta' \rangle$ *as the specification for its continuation.* In this transition relation, only legitimate (assertion-satisfying) actions are considered. Figure 8 lists the full rules. The transition rules for endpoint assertions are the standard synchronous ones. By rule [TR-TAU] specification $\langle \Gamma, \Delta \rangle$ can always make a silent transition. Rule [TR-LINKOUT] starts a new session for participant 1, specified as the projection on 1 of the global assertion $\mathcal{G}$ associated to a by the global assertion assignment. Rule [TR-LINKIN] is similar, for participant i. Rule [TR-SEND] (resp. [TR-RECEIVE]) sends (resp. receives) a vector of values only if it satisfies the current predicate A. Similarly, [TR-SEL] (resp. [TR-BRANCH]) sends (resp. receives) a branching label only if the associated predicate is satisfied. Noticeably, $\langle \Gamma, \Delta \rangle$ never allows violating actions.

6.2. Behavioural Conformance of Processes

We define the semantic counterpart of $\Gamma \vdash P \triangleright \Delta$ by using a simulation between the transitions of processes and those of assertions. The simulation relation (Definition 6.1), requires an input/branching action to be simulated only for "legal" values/labels, i.e. for actions in which predicates are not violated. Intuitively, we demand that a process conforms to a given contract as long as its environment behaves as prescribed by that contract. Below we use the *predicate erasure* $\mathsf{erase}(P)$ that erases all the predicates from P. Similarly $\mathsf{erase}(\Delta)$ erases predicates from the underlying session types, giving the *typing* environments. We denote with Γ_{typing} the environment that associates free interaction variables to sorts (in the sense of [29]). Hereafter, if there exists an α such that $\langle \Gamma, \Delta \rangle$ allows α then we say that $\langle \Gamma, \Delta \rangle$ *is capable* to move at the subject $\mathsf{sbj}(\alpha)$ (the subject of α).

Definition 6.1 (Conditional Simulation). Let $\mathcal{R}$ be a binary relation whose elements relate a closed process P without errH or errT and a pair of assignments $\langle \Gamma, \Delta \rangle$ such that $\Gamma_{typing} \vdash \mathsf{erase}(P) \triangleright \mathsf{erase}(\Delta)$ in the typing rules in [29, §4]. Then $\mathcal{R}$ is a *conditional simulation* if, for each $(P, \langle \Gamma, \Delta \rangle) \in \mathcal{R}$:

1. for each input/branching/session-accept transition $P \xrightarrow{\alpha} P'$, $\langle \Gamma, \Delta \rangle$ is capable to move at $\mathsf{sbj}(\alpha)$ and, if $\langle \Gamma, \Delta \rangle \xrightarrow{\alpha} \langle \Gamma', \Delta' \rangle$ then $(P', \langle \Gamma', \Delta' \rangle) \in \mathcal{R}$
2. for each output/selection/τ/session-request transition $P \xrightarrow{\alpha} P'$, $\langle \Gamma, \Delta \rangle \xrightarrow{\alpha} \langle \Gamma', \Delta' \rangle$ such that $(P', \langle \Gamma', \Delta' \rangle) \in \mathcal{R}$

We write $P \precsim \langle \Gamma, \Delta \rangle$ when $(P, \langle \Gamma, \Delta \rangle) \in \mathcal{R}$ for some conditional simulation $\mathcal{R}$.

The condition that P must be well-typed against Γ_{typing} and $\mathsf{erase}(\Delta)$ prevents "incomplete" processes (i.e., those that do not perform all the activities required by the respective endpoint assertion) from being considered semantically correct. For instance, without this condition the inaction $\mathbf{0}$ would be conditionally simulated by any Δ.

Example 6.2 (Example of Conditional Simulation). This example illustrates why the definition of conditional simulation distinguishes between input and output actions. We verify $P \precsim \langle \Gamma, \Delta \rangle$ for the process and endpoint assertion assignment defined below:

$$P = s_1!\langle 10 \rangle(v)\{v \geq 10\}; s_2?(u)\{u \geq 10\}; \mathbf{0}$$
$$\Delta = s_1, s_2 : k_1!(v : \mathsf{Int})\{v \geq 10\}; k_2?(u : \mathsf{Int})\{u \geq 10\}; \mathsf{end} \mathbin{@} \mathtt{p}$$

First, P can make a transition for rule [Send]:

$$P \xrightarrow{s_1!10} s_2?(u)\{u \geq 10\}; \mathbf{0}$$

Following Definition 6.1 (2), we observe that, by rule [Tr-Send] (since $10 \geq 10 \downarrow \mathsf{true}$)

$$\langle \Gamma, \Delta \rangle \xrightarrow{s_1!10} \langle \Gamma, s_1, s_2 : k_2?(u : \mathsf{Int})\{u \geq 10\}; \mathsf{end} \mathbin{@} \mathtt{p} \rangle$$

Next, the process can perform an input step, receiving a range of possible values. Assume the process is receiving $u = 5$ which violates the predicate $u \geq 10$:

$$s_2?(u)\{u \geq 10\}; \mathbf{0} \xrightarrow{s_2?5} \mathsf{errT}$$

The process conforms to the assertion by Definition 6.1 (1) since conditional simulation only requires $\langle \Gamma, s_1, s_2 : k_2?(u : \mathsf{Int})\{u \geq 10\}; \mathsf{end} \mathbin{@} \mathtt{p} \rangle$ to be capable to move at subject $s_2?$. Having received a violating value is not P's fault since P does not have an obligation on u; reversely, the guarantees of P have been violated therefore P does not have any further obligation on the computation. On the other hand, if P receives a valid value for u

$$s_2?(u)\{u \geq 10\}; \mathbf{0} \overset{s_2?10}{\rightarrow} \mathbf{0}$$

the usual requirements apply; namely one must check

$$\langle \Gamma, s_1, s_2 : k_2?(u : \mathsf{Int})\{u \geq 10\}; \mathsf{end} \mathbin{@} \mathtt{p} \rangle \overset{s_2!10}{\rightarrow} \langle \Gamma, s_1, s_2 : \mathsf{end} \mathbin{@} \mathtt{p} \rangle$$

and $\mathbf{0} \precsim \langle \Gamma, s_1, s_2 : \mathsf{end} \mathbin{@} \mathtt{p} \rangle$.

We can now define the satisfaction relation.

Definition 6.3 (Satisfaction). Let P be a closed process and Δ an endpoint assertion assignment. If $P \precsim \langle \Gamma, \Delta \rangle$ then we say that P *satisfies* Δ *under* Γ, and write $\Gamma \models P \triangleright \Delta$. The satisfaction is extended to open processes, denoted $\mathcal{C}; \Gamma \models P \triangleright \Delta$, by considering all closing substitutions respecting Γ and $\mathcal{C}$ over Δ and P.

The judgement $\Gamma \models P \triangleright \Delta$ in Definition 6.3 states that (1) P will send valid messages or selection labels; and (2) P will continue to behave well (i.e., without going into error) w.r.t. the continuation specification after each valid action in (1) as well as after receiving each valid message/label (i.e. which satisfies an associated predicate). The satisfaction is about partial correctness since, if P (is well-typed and) has no visible actions then the satisfaction trivially holds.

6.3. Behavioural Conformance and Refinement

Proposition 6.6 says that a process satisfying a stronger specification also satisfies a weaker one. We prove Proposition 6.6 after two auxiliary lemmas.

Lemma 6.4. *Assume* $\Delta \ni \Delta'$. *If* $\langle \Gamma, \Delta \rangle$ *is capable of moving at the subject* $\mathsf{sbj}(\alpha)$ *then* $\langle \Gamma, \Delta' \rangle$ *is also capable to move at the subject* $\mathsf{sbj}(\alpha)$.

PROOF: The proof is straightforward from the definition refinement (Definition 5.4). Up to the unfolding of recursive assertions, an endpoint assertion may differ from its refinement only in (a) the predicates in case of input/output/selection/branching and (b) the sets of possible labels/branches in case of selection/branching. By Definition 5.4, if a refinement $\mathcal{T}_1$ consists of an input/output/selection/branching with subject k then also the refined process $\mathcal{T}_2$ consists of an input/output/selection/branching, respectively, with subject k (for input and branching we use well-assertedness). This property holds recursively for the respective continuations.

Lemma 6.5. *Assume* $\Delta \ni \Delta'$.

1. *If* $\langle \Gamma, \Delta \rangle \overset{\alpha}{\rightarrow} \langle \Gamma_1, \Delta_1 \rangle$ *s.t.* α *is an output/selection, then* $\langle \Gamma, \Delta' \rangle \overset{\alpha}{\rightarrow} \langle \Gamma_1, \Delta_1' \rangle$ *s.t.* $\Delta_1 \ni \Delta_1'$.
2. *If* $\langle \Gamma, \Delta \rangle \overset{\alpha}{\rightarrow} \langle \Gamma_1, \Delta_1 \rangle$ *s.t.* α *is an input/branching action, and if* $\langle \Gamma, \Delta' \rangle$ *allows* α*, then* $\langle \Gamma, \Delta' \rangle \overset{\alpha}{\rightarrow} \langle \Gamma_1, \Delta_1' \rangle$ *s.t.* $\Delta_1 \ni \Delta_1'$.
3. *If* $\langle \Gamma, \Delta \rangle \overset{\tau}{\rightarrow} \langle \Gamma_1, \Delta_1 \rangle$ *then* $\langle \Gamma, \Delta' \rangle \overset{\tau}{\rightarrow} \langle \Gamma_1, \Delta_1' \rangle$ *or* $\langle \Gamma, \Delta' \rangle \overset{\tau}{\rightarrow}\overset{\tau}{\rightarrow} \langle \Gamma_1, \Delta_1' \rangle$ *s.t.* $\Delta_1 \ni \Delta_1'$.

PROOF: The proof is by induction on the structure of Δ, and is relegated in Appendix D.1.

Proposition 6.6 (Refinement). *If* $\Gamma \models P \triangleright \Delta$ *and* $\Delta \ni \Delta'$ *then* $\Gamma \models P \triangleright \Delta'$.

PROOF: The proof is by induction on the transitions of P. We report below a proof sketch. The full proof is in Appendix D.2. We proceed by case analysis.
Case 1. If $P \xrightarrow{\alpha} P'$ by output/selection/τ transition, since $\Gamma \models P \triangleright \Delta$ then $\langle \Gamma, \Delta \rangle \xrightarrow{\alpha} \langle \Gamma, \Delta_1 \rangle$. By Lemma 6.5, $\langle \Gamma, \Delta' \rangle \xrightarrow{\alpha} \langle \Gamma, \Delta'_1 \rangle$ where $\Delta_1 \ni \Delta'_1$.
Case 2. If $P \xrightarrow{\alpha} P'$ by input/branching, since $\Gamma \models P \triangleright \Delta$ then $\langle \Gamma, \Delta \rangle$ has the capability of a move at the subject $\mathsf{sbj}(\alpha)$. By Lemma 6.4 also $\langle \Gamma, \Delta' \rangle$ has the capability of a move at the subject $\mathsf{sbj}(\alpha)$. We have two possible cases:

- Δ' cannot move (because its predicate is more restrictive) but still $\Gamma \models P \triangleright \Delta'$ since $\langle \Gamma, \Delta' \rangle$ is capable of an input/branching step at the subject $\mathsf{sbj}(\alpha)$,
- $\langle \Gamma, \Delta' \rangle \xrightarrow{\alpha} \langle \Gamma, \Delta'_1 \rangle$. In this case also $\langle \Gamma, \Delta \rangle \xrightarrow{\alpha} \langle \Gamma, \Delta_1 \rangle$ since the refinement is less restrictive that the refined endpoint assertion in input/branching moves. By Lemma 6.5, $\Delta_1 \ni \Delta'_1$. The predicate holds by induction.

7. Subject Reduction and Error Freedom

We prove subject reduction after few auxiliary lemmas. The full proofs are in Appendix F and are based on *message assertions* which extend endpoint assertions with runtime queues (see Appendix C). Synchronization of endpoint assertions with message queues corresponds to a τ action.

Lemma 7.1. *Suppose* $\Gamma \vdash P \triangleright \Delta$. *If* $P \xrightarrow{\tau} P'$ *then* $\Gamma \vdash P' \triangleright \Delta$ *or* $\Gamma \vdash P' \triangleright \Delta'$ *s.t.* $\langle \Gamma, \Delta \rangle \xrightarrow{\tau} \langle \Gamma, \Delta' \rangle$.

PROOF: The proof is relegated to Appendix F.1. □

A straightforward corollary of Lemma 7.1 is the following.

Lemma 7.2 (Subject Reduction for Silent Actions). *Suppose* $\Gamma \vdash P \triangleright \Delta$ *and* $P \to P'$. *Then there exists* Δ' *s.t.* $\Gamma \vdash P' \triangleright \Delta'$ *and either* $\Delta' = \Delta$ *or* $\langle \Gamma, \Delta \rangle \xrightarrow{\tau} \langle \Gamma, \Delta' \rangle$.

A similar result holds for non-silent actions. We first give an auxiliary lemma showing that validation is preserved by structural equivalence.

Lemma 7.3 (Subject Congruence). *If* $\Gamma \vdash P_1 \triangleright \Delta$ *and* $P_1 \equiv P_2$ *then* $\Gamma \vdash P_2 \triangleright \Delta$.

PROOF: The proof is relegated to Appendix F.3. □

Proposition 7.4 (Subject Reduction for Visible Transitions).
If $\Gamma \vdash P \triangleright \Delta$ *and* $P \xrightarrow{\alpha} P'$ *with* $\alpha \neq \tau$ *then*

1. *if* α *is an output, a selection or an action at a shared channel, then there exist* Γ', Δ' *s.t.* $\langle \Gamma, \Delta \rangle \xrightarrow{\alpha} \langle \Gamma', \Delta' \rangle$ *and* $\Gamma' \vdash P' \triangleright \Delta'$.

2. *if* α *is an input or a branching and there exist* Γ', Δ' *s.t.* $\langle \Gamma, \Delta \rangle \xrightarrow{\alpha} \langle \Gamma', \Delta' \rangle$ *then* $\Gamma' \vdash P' \triangleright \Delta'$.

PROOF (SKETCH): (The detailed proof is relegated to Appendix F.4). The proof is by induction on the validation proof of $\Gamma \vdash P \triangleright \Delta$. (1) is the standard property for subject reduction. We introduced (2) for input actions since P could receive, despite behaving according to the specification, violating input from the environment. Therefore, in the case of input actions we require subject reduction to hold only for non violating messages form the environment, namely those messages which are allowed by $\langle \Gamma, \Delta \rangle$. The non trivial case is when rule [REC] is used; in this case the proof follows by observing that (i) P can only move after being unfolded, that (ii) by induction the unfolding satisfies the statement, and that (iii) by Lemma 7.3 (the case of structural equivalence of a process with its unfolding) the statement holds also if we fold P'. □

As an immediate consequence of subject reduction, observing that if there is a derivation for P then P does not contain errors (Proposition Appendix C.4 in Appendix Appendix C), we now establish *error freedom*.

Corollary 7.5 (Predicate Error Freedom). *Suppose P is a closed program, $\Gamma \vdash P \triangleright \Delta$ and $P \xrightarrow{\alpha_1..\alpha_n} P'$ such that $\langle \Gamma, \Delta \rangle$ allows $\alpha_1..\alpha_n$. Then P' contains neither* errH *nor* errT.

8. Soundness

The soundness proof relies on basic properties of the validation rules (cf. Figure 6 on page 20) and their extension with runtime queues and message assertions (cf. Figure C.11 in Appendix C).

Lemma 8.1 (Postponement of [REC]). *For each judgment $\Gamma \vdash P \triangleright \Delta$ derivable from the rules in Figure 6 there exists a proof such that all the applications (if any) of* [REC] *are at the root of the proof tree.*

PROOF: The premise of [REC] for Q (the main process) reads:

$$\mathcal{C}; \Gamma, X : (\tilde{v} : \tilde{S}) \mathcal{T}_1 @ \mathtt{p}_1 \ldots \mathcal{T}_n @ \mathtt{p}_n \vdash Q \triangleright \Delta$$

Note the only condition it demands is the assumption contains $X : (\tilde{v} : \tilde{S}) \mathcal{T}_1 @ \mathtt{p}_1 \ldots \mathcal{T}_n @ \mathtt{p}_n$ and the only effect of the application of [REC] is we lose this assumption. Since no other rules use this assumption, we can always permute an application of any rule with [REC] to obtain the same conclusion. □

Definition 8.2. We say $\mathcal{C}; \Gamma \vdash P \triangleright \Delta$ is *well-initiated* if Δ contains only singleton assignments and, moreover, P has no queue at a free session channel.

Lemma 8.3. *The premise of* [REC] *is well-initiated if and only if its conclusion is well-initiated.*

PROOF: Because the assertion assignment does not change and the process does not change except adding the new definition. □

Theorem 8.4 (Soundness for Open Processes).
Let P be a program phrase. Then $\mathcal{C};\Gamma \vdash P \triangleright \Delta$ implies $\mathcal{C};\Gamma \models P \triangleright \Delta$.

PROOF (SKETCH): (The full proof is relegated to Appendix G.2).

The proof is based on the fact that the set $\mathcal{R}$ of pairs made of a program phrase and its validating environment both closed under any substitution (consistent with the validating environment) is a conditional simulation. Let $\mathcal{R}$ be the relation collecting all the pairs of the form $(P_\sigma, \langle \Gamma_\sigma, \Delta_\sigma \rangle)$ such that $\mathcal{C};\Gamma \vdash P \triangleright \Delta$ where: (i) P is a sub-term of a multi-step $\xrightarrow{\alpha}$-derivative of a program phrase, (ii) Δ is an endpoint assertion assignment possibly containing non-singleton assignments, and (iii) σ is a closing substitution consistent with $\mathcal{C}$ and Γ. We show that $\mathcal{R}$ is a conditional simulation, by rule induction on the validation rules in Figures 6 (and their extension to runtime queues and message assertions in Figure C.11 in the Appendix). Most of the cases straightforwardly follow by induction (on the proof of the validation judgment). The case for [CONSEQ] follows by Proposition 6.6. The case for [REC] deserves particular care: by Lemma 8.1 we assume that, in all derivations for processes in $\mathcal{R}$, the application of rule [REC] only occurs in the last steps of a derivation for a transition derivative of a program phrase, without loss of generality. Under this assumption, by Lemma 8.3, we know that the premise and conclusion of an application of [REC] is well-initiated (therefore we only consider singleton assignments). This case is then proved by the standard syntactic approximation of recursion. □
As an immediate corollary we obtain:

Theorem 8.5 (Soundness for Programs).
Let P be a program. Then $\Gamma \vdash P \triangleright \Delta$ implies $\Gamma \models P \triangleright \Delta$.

9. Extensions and Related Work

Extensions to shared and session channel passing and completeness. The theory we have introduced directly extends to shared channel passing and session channel passing, or delegation, carrying over all formal properties. In both cases, we have only to add predicate annotations to channels in assertions as well as in asserted processes. The shape of the judgement and the proof rules do not change, similarly the semantics of the judgement uses a conditional simulation. We obtain the same soundness result as well as completeness of the proof rules for the class of processes whose newly created channels are immediately exported. The proof system is complete relative to the decidability of the underlying logic for processes without hidden shared names [43, 7]. In order to prove completeness, we introduce the *generation rules* for programs and program phrases by which we can derive a "principal" formula. The general intuition is that, for every P, Γ and Δ such that $\Gamma \models P \triangleright \Delta$, we can generate Δ' such that $\Gamma \vdash P \triangleright \Delta'$ and $\Delta' \ni \Delta$ (thus $\Gamma \vdash P \triangleright \Delta$ by rule [CONSEQ]). Since the presentations of these extensions would require detailed notions of refinement and judgements, for simplicity, we relegate them to [43].

Hennessy-Milner logic for the π-calculus. Hennessy-Milner Logic (HML) is an expressive modal logic with an exact semantic characterisation [26]. The presented theory addresses some of the key challenges in practical logical specifications for the π-calculus, unexplored in the context of HML. First, by starting from global assertions, we gain in significant concision of descriptions while enjoying generality within its scope (properties of individual protocols). Previous work [19, 3] show how specifications in HML, while encompassing essentially arbitrary

behavioural properties of processes, tend to be lengthy from the practical viewpoint. In this context, the direct use of HML is tantamount to *reversing* the methodology depicted in Figure 1 of § 1: we start from endpoint specifications and later try to check their mutual consistency, which may not easily yield understandable global specifications.

As another practical aspect, since $\Rightarrow$ is decidable for practically important classes assertions [43], the present theory also offers algorithmic validation methods for key engineering concerns [44] including consistency of specifications (cf. §3.2) and correctness of process behaviours with full recursion against non-trivial specifications, whose analogue may not be known for the general HML formulae on the π-calculus. The use of the underlying type structures plays a crucial role. This is in contrast to validation for the whose analogue may not be known in the context of general HML formulae.

From the viewpoint of logical specifications for name passing, the present theory takes an *extensional* approach: we are concerned with what behaviours will unfold starting from given channels, than their (in)equality [19]. While our approach does reflect recommended practices in application-level distributed programming (where the direct use of network addresses is discouraged), it is an interesting topic to study how we can treat names as data as studied in [19].

Corresponding assertions and refinement/dependent types. The work [9] combines session-types with *correspondence assertions*. The type system can check that an assertion **end** L is matched by the corresponding **begin** effect. Assertions L in the effects of [9] are lists of values, not general logical logical formulae like ours.

The use of session types to describe behavioural properties of objects and components in CORBA is studied in [47]. In another vein, the refinement types for channels (e.g. [6]) specify value dependency with logical constraints. For example, one might write $?(x\colon \mathsf{int}, !\{y : \mathsf{int} \mid y > x\})$ using the notations from [48, 24]. 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.

Contract-based approaches to functions and communications.. Verification using theories of contracts for programming functional languages, with applications to the validation of financial contracts, is studied in [40, 49]. Our theory uses the π-calculus with session types as the underlying formalism to describe contracts for distributed interactions. We observe that a contract-based approach for sequential computing is generally embeddable to the present framework (noting that function types are a special form of binary session types and that the pre/post conditions in sequential contracts are nothing but predicates for interactions resulting from the embedding); it is an interesting subject of study to integrate these and other sequential notions of contracts into the present framework, which would enable a uniform reasoning of sequential and concurrent processes.

In [11, 20] use c-semirings to model constraints that specify a Service Level Agreement. The later work [18] studies a combination of binary session types and the primitives from [11] for processes to be able to check a logical condition when initiating a session and ensure bilinearity of channels (i.e. duality of communication). It would be interesting to consider global assertions where the logical language is replaced with c-semirings. This would allow global assertions to express soft constraints but it could affect the effectiveness of our approach. However c-semirings do not feature negation and the decidability of logics based on c-semirings has not been deeply investigated.

The global consistency checking is used in advanced security formalisms. In [25] a rely-guarantee technique is applied to a trust-management logic. The main technical difference is that users have to directly annotate each participant with assertions because of the the absence of global assertions. In [5] cryptography is used to ensure integrity of sessions but logical contracts are not considered.

Theories of contracts for web services based on advanced behavioural types are proposed, including those using CCS [10], π-calculus [16], and conversation calculus [13]. Recent work [2] extends contracts to formulae of an intuitionistic logic and studies expressiveness and decidability. Some of the authors in this line of study focus on *compliance* of client and services, often defining compliance in terms of deadlock-freedom, e.g., in [1] a type system guaranteeing a progress property of clients is defined. The work [12] investigates a relationship between for 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.

Our approach differs from the preceding works in its use of global assertions for elaborating the underlying type structure, combined with the associated compositional proof system. This permits us to express and enforce fine-grained contracts of choreographic scenarios. Global/end-point assertions can express constraints over message values (including channels), branches and invariants, which cannot be represented by types alone, cf. [29]. The enriched expressiveness of specifications introduces technical challenges: in particular, consistency of specifications becomes non-trivial. The presented consistency condition for global assertions is mechanically checkable relatively to the decidability of the underling logic, and ensures that the end-point assertions are automatically consistent when projected. On this basis a sound and relatively complete proof system is built that guarantees semantic consistency.

As a different DbC-based approach to concurrency, an extension of DbC has been proposed in [37], using contracts for SCOOP [35] in order to reason about liveness properties of concurrent object-oriented systems. The main difference of our approach from [37] is that our framework specifies focuses on systems based on distributed message passing systems while [37] treats shared resources. The notion of pre-/post-conditions and invariants for global assertions centring on communications and the use of projections are not found in [37]. The treatment of liveness in our framework is an interesting topic for further study.

Since the first work [29] was proposed, the multiparty session types have been developed in several contexts, for example, such as distributed object optimisation [45], parallel algorithms [36, 50] and security [5, 14], some of which initiated industrial collaborations [39, 44], see [28]. For a more expressive language extension, we plan to integrate with dynamic natures based on multirole session types [21] and global escape [15].

References

[1] Lucia Acciai and Michele Borale. A type system for client progress in a service-oriented calculus. In *Concurrency, Graphs and Models*, volume 5065 of *LNCS*, pages 625–641. Springer, 2008.

[2] Massimo Bartoletti and Roberto Zunino. A calculus of contracting processes. In *LICS*, pages 332–341. IEEE Computer Society, 2010.

[3] Martin Berger, Kohe Honda, and Nobuko Yoshida. Completeness and logical full abstraction for modal logics for the typed π-calculus. In *ICALP'08*, volume 5126 of *LNCS*, pages 99–111, 2008.

[4] Lorenzo Bettini et al. Global Progress in Dynamically Interfered Multiparty Sessions. In *CONCUR'08*, volume 5201 of *LNCS*, pages 418–433. Springer, 2008.

[5] Karthikeyan Bhargavan, Ricardo Corin, Pierre-Malo Deniélou, Cédric Fournet, and James Leifer. Cryptographic protocol synthesis and verification for multiparty sessions. In *CSF*, pages 124–140, 2009.

[6] Karthikeyan Bhargavan, Cédric Fournet, and Andrew D. Gordon. Modular verification of security protocol code by typing. In *POPL*, pages 445–456, 2010.
[7] Laura Bocchi, Kohei Honda, Emilio Tuosto, and Nobuko Yoshida. A theory of design-by-contract for distributed multiparty interactions. In *CONCUR'10*, volume 6269 of *LNCS*, pages 162–176. Springer, 2010.
[8] Eduardo Bonelli and Adriana Compagnoni. Multipoint Session Types for a Distributed Calculus. In *TGC'07*, volume 4912 of *LNCS*, pages 240–256, 2008.
[9] Eduardo Bonelli, Adriana Compagnoni, and Elsa Gunter. Correspondence assertions for process synchronization in concurrent communications. *JFC*, 15(2):219–247, 2005.
[10] Mario Bravetti and Gianluigi Zavattaro. A foundational theory of contracts for multi-party service composition. *Fundamenta Informaticae*, XX:1–28, 2008.
[11] Maria Grazia Buscemi and Ugo Montanari. CC-PI: A constraint-based language for specifying service level agreements. In *ESOP*, volume 4421 of *LNCS*, pages 18–32, 2007.
[12] Luis Caires and Frank Pfenning. Session types as intuitionistic linear propositions. In *CONCUR'10*, volume 6269 of *LNCS*, pages 222–236. Springer, 2010.
[13] Luís Caires and Hugo Torres Vieira. Conversation types. In *ESOP*, volume 5502 of *LNCS*, pages 285–300, 2009.
[14] Sara Capecchi, Ilaria Castellani, Mariangiola Dezani-Ciancaglini, and Tamara Rezk. Session Types for Access and Information Flow Control. In *CONCUR'10*, volume 6269 of *LNCS*, pages 237–252. Springer, 2010.
[15] Sara Capecchi, Elena Giachino, and Nobuko Yoshida. Global Escape in Multiparty Sessions. In Kamal Lodaya and Meena Mahajan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010)*, volume 8 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 338–351, Dagstuhl, Germany, 2010. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
[16] Giuseppe Castagna and Luca Padovani. Contracts for mobile processes. In *CONCUR*, volume 5710 of *LNCS*, pages 211–228. Springer, 2009.
[17] Tzu-Chun Chen, Laura Bocchi, Pierre-Malo Deniélou, Kohei Honda, and Nobuko Yoshida. Distributed monitoring for multiparty session enforcement. `http://www.eecs.qmul.ac.uk/~tcchen/monitoring_sessions.html`.
[18] Mario Coppo and Mariangiola Dezani-Ciancaglini. Structured communications with concurrent constraints. In *TGC*, volume 5474 of *LNCS*, pages 104–125, 2008.
[19] Mads Dam. Proof systems for pi-calculus logics. In *Logic for Concurrency and Synchronisation*, Trends in Logic, Studia Logica Library, pages 145–212. Kluwer, 2003.
[20] Rocco De Nicola, Gianluigi Ferrari, Ugo Montanari, Rosario Pugliese, and Emilio Tuosto. A basic calculus for modelling Service Level Agreements. In *Coordination*, volume 3454 of *LNCS*, pages 33 – 48. Springer, 2005.
[21] Pierre-Malo Deniélou and Nobuko Yoshida. Dynamic multirole session types. In *POPL*, pages 435–446. ACM, 2011. Full version, Prototype at `http://www.doc.ic.ac.uk/~pmalo/dynamics`.
[22] Robert W. Floyd. Assigning meaning to programs. In *Proc. Symp. in Applied Mathematics*, volume 19, 1967.
[23] David S. Frankel. *Model Driven Architecture: Applying MDA to Enterprise Computing*. Wiley, 2003.
[24] Tim Freeman and Frank Pfenning. Refinement types for ml. *SIGPLAN Not.*, 26(6):268–277, 1991.
[25] Joshua D. Guttman et al. Trust management in strand spaces: A rely-guarantee method. In *ESOP*, volume 2986 of *LNCS*, pages 325–339. Springer, 2004.
[26] Matthew Hennessy and Robin Milner. Algebraic laws for non-determinism and concurrency. *JACM*, 32(1), 1985.
[27] Tony Hoare. An axiomatic basis of computer programming. *CACM*, 12, 1969.
[28] Kohei Honda, Aybek Mukhamedov, Gary Brown, Tzu-Chun Chen, and Nobuko Yoshida. Scribbling interactions with a formal foundation. In *ICDCIT*, volume 6536 of *LNCS*, pages 55–75. Springer, 2011.
[29] Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In *POPL '08*, pages 273–284. ACM, 2008.
[30] *The Java Modeling Language (JML) Home Page*.
[31] Cliff B. Jones. Specification and design of (parallel) programs. In *IFIP Congress*, pages 321–332, 1983.
[32] K. Rustan M. Leino. Verifying object-oriented software: Lessons and challenges. In *TACAS*, volume 4424 of *LNCS*, page 2, 2007.
[33] Elliot Mendelson. *Introduction to Mathematical Logic*. Wadsworth Inc., 1987.
[34] Bertrand Meyer. Applying "Design by Contract". *Computer*, 25(10):40–51, 1992.
[35] Bertrand Meyer. *Object-Oriented Software Construction (Chapter 31)*. Prentice Hall, 1997.
[36] Nicholas Ng, Nobuko Yoshida, Olivier Pernet, Raymond Hu, and Yiannos Kryftis. Safe parallel programming with session java. In *COORDINATION*, volume 6721 of *LNCS*, pages 110–126. Springer, 2011.
[37] Piotr Nienaltowski, Bertrand Meyer, and Jonathan S. Ostroff. Contracts for concurrency. *Form. Asp. Comput.*, 21(4):305–318, 2009.
[38] OMG. Object constraint language version 2.0, may 2006.
[39] Ocean Observatories Initiative (OOI). `http://www.oceanleadership.org/programs-and-partnerships/ocean-observing/ooi/`.
[40] Simon Peyton Jones, Jean-Marc Eber, and Julian Seward. Composing contracts: an adventure in financial engi-

neering. In *ICFP*, pages 281–292. ACM, 2000.
[41] Benjamin C. Pierce. *Types and Programming Languages*. MIT Press, 2002.
[42] William Pugh. The omega test: a fast and practical integer programming algorithm for dependence analysis. In *Supercomputing '91*, pages 4–13, New York, NY, USA, 1991. ACM.
[43] Full version of this paper. http://www.cs.le.ac.uk/people/lb148/fullpaper.html.
[44] Savara JBoss Project. http://www.jboss.org/savara.
[45] K. C. Sivaramakrishnan, Karthik Nagaraj, Lukasz Ziarek, and Patrick Eugster. Efficient session type guided distributed interaction. In *Coordination'10*, volume 6116 of *LNCS*, pages 152–167. Springer, 2010.
[46] UNIFI. ISO 20022 UNIversal Financial Industry message scheme. http://www.iso20022.org, 2002.
[47] Antonio Vallecillo, Vasco T. Vasconcelos, and António Ravara. Typing the behavior of objects and components using session types. *Fundamenta Informaticæ*, 73(4):583–598, 2006.
[48] Hongwei Xi and Frank Pfenning. Dependent types in practical programming. In *POPL*, pages 214–227, 1999.
[49] Dana Xu and Simon Peyton Jones. Static contract checking for Haskell. In *POPL*, pages 41–52, 2009.
[50] Nobuko Yoshida, Pierre-Malo Deniélou, Andi Bejleri, and Raymond Hu. Parameterised multiparty session types. In *FoSSaCs*, volume 6014 of *LNCS*, pages 128–145, 2010.

Appendix A. Algorithm for Annotating Recursion Parameters with Locations

We define an algorithm to determine the locations of recursion parameters in a global assertion $\mathcal{G}$. More precisely, we give a function $\mathsf{L}(\mathcal{G};\mathcal{E};\mathsf{MK};\mathsf{C})$ where MK (after "must know") is a set of pairs $(\mathtt{p},v)$ which reads "participant $\mathtt{p}$ is required to have met interaction variable v", and C assigns interaction parameters and a vector of sets of interaction variables to a assertion variable $\mathbf{t}$ so that the i^{th} element of the vector includes all the variables passed as the i^{th} argument of a recursive invocation of $\mathbf{t}$ in $\mathcal{G}$.

$\mathsf{L}(\mathcal{G};\mathcal{E};\mathsf{MK};\mathsf{C})$ returns a triple $(\mathcal{E}',\mathsf{MK}',\mathsf{C}')$ as follows:

1. if $\mathcal{G}=\mathtt{p}\to\mathtt{p}'\colon k\,(\tilde{v})\{A\}.\mathcal{G}'$, then return $\mathsf{L}(\mathcal{G}';\mathcal{E},\tilde{v}@\{\mathtt{p},\mathtt{p}'\};\mathsf{MK}\setminus\{(\mathtt{p},v),(\mathtt{p}',v)\ :\ v\in\tilde{v}\};\mathsf{C})$
2. if $\mathcal{G}=\mathtt{p}\to\mathtt{p}'\colon k\,\{\{A_j\}l_j\colon\ \mathcal{G}_j\}_{j\in J}$ then return $\Cup_{j\in J}\mathsf{L}(\mathcal{G}_j;\mathcal{E};\mathsf{MK};\mathsf{C})$
3. if $\mathcal{G}=\mathcal{G}_1,\mathcal{G}_2$ then return $\Cup_{i\in\{1,2\}}\mathsf{L}(\mathcal{G}_i;\mathcal{E};\mathsf{MK};\mathsf{C})$
4. if $\mathcal{G}=\mathbf{t}\langle u_1,\ldots,u_n:A_1,...,A_n\rangle$[1], assuming $\mathsf{C}(\mathbf{t})=\tilde{v},V_1,\ldots,V_n$, then return $(\mathcal{E},\mathsf{MK},\mathsf{C}[\mathbf{t}\mapsto\tilde{v},V_1\cup var(A_1)\setminus u_1,\ldots,V_n\cup var(A_n)\setminus u_n])$
5. if $\mathcal{G}=\mathsf{end}$ then return $(\mathcal{E},\mathsf{MK},\mathsf{C})$
6. if $\mathcal{G}=\mu\mathbf{t}\langle u_1,\ldots,u_n:A_1,...,A_n\rangle(\tilde{v})\{A\}.\mathcal{G}'$ then return $\mathsf{L}(\mathcal{G}';\mathcal{E};\mathsf{MK};\mathsf{C}[\mathbf{t}\mapsto\tilde{v},var(A_1)\setminus u_1,\ldots,var(A_n)\setminus u_n])$.

where $\Cup$ returns $\mathsf{BadAssertion}$ if one of its arguments is $\mathsf{BadAssertion}$ otherwise

$$\Cup_{j\in J}(\mathcal{E}_j,\mathsf{MK}_j,\mathsf{C}_j)=(\cup_{j\in J}\mathcal{E}_j,\cup_{j\in J}\mathsf{MK}_j,\cup_{j\in J}\mathsf{C}_j)$$

If $\mathsf{L}(\mathcal{G};\mathcal{E};\mathsf{MK};\mathsf{C})$ returns a triple $(\mathcal{E}',\mathsf{MK}',\mathsf{C}')$ such that, for every $\mathbf{t}\in dom(\mathsf{C}')$, $\mathsf{C}'(\mathbf{t})$ has the form

$$\tilde{v},\{v_1^1,\ldots,v_{n_1}^1\},\ldots,\{v_1^m,\ldots,v_{n_m}^m\}$$

and for each $1\le i\le m$, $v_1^i,\ldots,v_{n_i}^i$ have the same location $\mathtt{L_i}$ in $\mathcal{E}$, such variables are located at $\mathtt{L_i}$. Otherwise the program is badly specified when the previous condition does not hold or $\mathsf{L}(\mathcal{G};\mathcal{E};\mathsf{MK};\mathsf{C})$ returns $\mathsf{BadAssertion}$.

Appendix B. Endpoint Assertions and Projection

Appendix B.1. An Alternative Definition of Projection

Given $\mathcal{G}$ and A, the *projection of $\mathcal{G}$ for a participant* $\mathtt{p}$ *with respect to* A is denoted by $(\mathcal{G})\downarrow^A_{\mathtt{p}}$ and, assuming $\mathtt{p}_1\neq\mathtt{p}_2$, recursively defined as follows.

[1] The algorithm requires the assertion used for assignments to recursion parameters, to be partitioned into a number of independent sub-predicates $A_1,...,A_n$, one for each recursion parameter (see also case 6). This requirement still allows us to model global assertions as described in § 3.

$$(1)\quad (\mathtt{p}_1 \to \mathtt{p}_2\colon k\,(\tilde{v}\colon \tilde{S})\{A\}.\mathcal{G}')\downarrow_{\mathtt{p}}^{A_P} = \begin{cases} k!(\tilde{v}\colon \tilde{S})\{A\}.(\mathcal{G}')\downarrow_{\mathtt{p}}^{A\wedge A_P} & \text{if } \mathtt{p}=\mathtt{p}_1 \\ k?(\tilde{v}\colon \tilde{S})\{OPT(A,A_P)\upharpoonright \mathtt{p}\}.(\mathcal{G}')\downarrow_{\mathtt{p}}^{A\wedge A_P} & \text{if } \mathtt{p}=\mathtt{p}_2 \\ (\mathcal{G}')\downarrow_{\mathtt{p}}^{A\wedge A_P} & \text{otherwise} \end{cases}$$

$$(2)\quad (\mathtt{p}_1 \to \mathtt{p}_2\colon k\,\{\{A_i\}l_i\colon \mathcal{G}_i\}_{i\in I})\downarrow_{\mathtt{p}}^{A_P} = \begin{cases} k\oplus\{\{A_i\}l_i\colon (\mathcal{G}_i)\downarrow_{\mathtt{p}}^{A_i\wedge A_P}\}_{i\in I} & \text{if } \mathtt{p}=\mathtt{p}_1 \\ k\&\{\{OPT(A_i,A_P)\upharpoonright \mathtt{p}\}l_i\colon (\mathcal{G}_i)\downarrow_{\mathtt{p}}^{A_i\wedge A_P}\}_{i\in I} & \text{if } \mathtt{p}=\mathtt{p}_2 \\ (\mathcal{G}_1)\downarrow_{\mathtt{p}}^{A_P\wedge\bigvee_{j\in I}A_j}\ (=(\mathcal{G}_i)\downarrow_{\mathtt{p}}^{A_P\wedge\bigvee_{j\in I}A_j}) & \text{otherwise} \end{cases}$$

$$(3)\quad (\mathcal{G}_1,\mathcal{G}_2)\downarrow_{\mathtt{p}}^{A_P} = \begin{cases} (\mathcal{G}_i)\downarrow_{\mathtt{p}}^{A_P} & \text{if } \mathtt{p}\in\mathcal{G}_i \text{ and } \mathtt{p}\notin\mathcal{G}_j, i\neq j\in\{1,2\} \\ \mathsf{end} & \text{if } \mathtt{p}\notin\mathcal{G}_1 \text{ and } \mathtt{p}\notin\mathcal{G}_2 \end{cases}$$

$$(4)\quad (\mu\mathbf{t}\langle\tilde{u}:A'\rangle(\tilde{v}\colon\tilde{S})\{A\}.\mathcal{G})\downarrow_{\mathtt{p}}^{A_P} = \mu\mathbf{t}\langle\tilde{u}:A'\upharpoonright\mathtt{p}\rangle(\tilde{v}\colon S)\{A\upharpoonright\mathtt{p}\}.(\mathcal{G})\downarrow_{\mathtt{p}}^{A_P}$$

$$(5)\quad (\mathbf{t}\langle\tilde{u}:A'\rangle)\downarrow_{\mathtt{p}}^{A_P} = \mathbf{t}\langle\tilde{u}:A'\upharpoonright\mathtt{p}\rangle$$

$$(6)\quad (\mathsf{end})\downarrow_{\mathtt{p}}^{A_P} = \mathsf{end}$$

Definition 4.2 uses $A\wedge A_P$ in the projection of value and branching on the receiving party. Here we use OPT to eliminate, where possible, some sub-predicates from A_P. More specifically:

1. if $\mathtt{p}$ knows all variables in A then A_P is omitted,
2. otherwise,
   (a) we include all sub-predicates of A_P involving variables of A that $\mathtt{p}$ does not know,
   (b) we include recursively all predicates of A_P involving variables of the predicates added in (a) that $\mathtt{p}$ does not know.

We assume A and A_P are conjunctions of sub-predicates and we denote with $set(A)$ the set of such sub-predicates for A. Set of predicates, are ranged over by S, S', and $var(S) = \cup_{A_i\in S} var(A_i)$. We denote with $set^{-1}(S)$ the predicate obtained as the conjunction of the predicates in S. We define $OPT(A,A_P) = set^{-1}(f_{ext}(set(A),set(A_P))$ where $f_{ext}(S_{to},S_{from})$ is the following recursive function from couples of sets of predicates to sets of predicates (S_{to} is the set of predicates to include in the projection and A_{from} is the set of predicates from which to extract sub-predicates):

f_{ext} **(**S_{to}**,** S_{from}**) =**
if $S_{from} = \emptyset$ **or** $var(S_{to})\setminus I(\mathcal{G})\upharpoonright\mathtt{p}\cap var(S_{from}) = \emptyset$ **then return** S_{to}
else return $S_{to}\wedge OPT(S_B, S_{from}\setminus S_B)$
// with $S_B = \{A : A\in S_{from}$ **and** $v_i\in var(A)\}_{v_i\in var(S_{to})\setminus I(\mathcal{G})\upharpoonright\mathtt{p}}$

Remark. By using the definition above instead of Definition 4.2 we still preserve the properties of Lemma 4.6 and Proposition 4.7. In fact, eliminating some sub-predicates from A_P does not affects the construction, in `LSat`, of the preconditions 'bag'. In fact, the eliminated predicates are just a replication of some predicates that have been met in previous interactions, thus are already included in the predicate 'bag'. For the same reason, also validation is unaffected by the usage of the definition above instead of Definition 4.2.

Appendix B.2. Proof of Lemma 4.6

The proof relies on the following tautologies of first-order logic:

(a) $\forall w(C(w)\supset C') \equiv \exists w(C(w))\supset C'$, if w does not occur free in C'
(b) $\exists w(C(w))\wedge\exists w(C(w)\wedge C'(w)) \equiv \exists w(C(w)\wedge C'(w))$

and on the following lemma

Lemma Appendix B.1. *For any global assertion $\mathcal{G}$, if $A \equiv B$ then $\mathtt{GSat}(\mathcal{G},A)$ holds iff $\mathtt{GSat}(\mathcal{G},B)$ holds and $\mathtt{LSat}(\mathcal{G},A)$ holds iff $\mathtt{LSat}(\mathcal{G},B)$ holds.*
PROOF: Trivial by inspection of the Definitions 3.7 and 4.5.

We now proceed with the proof of Lemma 4.6.

Let $\mathcal{G}_{root}$ be a well-asserted global assertion and $\mathtt{p}$ be a participant of $\mathcal{G}_{root}$. For every subterm $\mathcal{G}$ of $\mathcal{G}_{root}$ and for every predicate $A_{\mathcal{G}}$ such that

$$var(A_{\mathcal{G}}) \subseteq I(\mathcal{G}_{root}) \setminus I(\mathcal{G}) \tag{B.1}$$

$$\mathtt{GSat}(\mathcal{G},A_{\mathcal{G}}) \supset \mathtt{LSat}(\mathcal{G}\upharpoonright \mathtt{p}, \exists \tilde{v}_{ext}(A_{\mathcal{G}}))$$

where

$$\tilde{v}_{ext} = var(A_{\mathcal{G}}) \setminus I(\mathcal{G}_{root}\upharpoonright p) \tag{B.2}$$

PROOF: The proof is by induction on on the structure of $\mathcal{G}$.

Value sending/receiving. $\mathcal{G} = \mathtt{p}_1 \to \mathtt{p}_2 : k\,(\tilde{v} : \tilde{S})\{A\}.\mathcal{G}'$
Case $\mathtt{p} = \mathtt{p}_1$**.** By Definition 4.2, $\mathcal{G}\upharpoonright \mathtt{p} = k!(\tilde{v} : \tilde{S})\{A\};(\mathcal{G}'\upharpoonright \mathtt{p})$ and by (B.2) and history sensitivity of $\mathcal{G}$

$$\tilde{v}_{ext} \cap var(A) = \emptyset \tag{B.3}$$

By temporal satisfiability of $\mathcal{G}, A_{\mathcal{G}} \supset \exists \tilde{v}(A)$ which, by (B.3) and (a), is equivalent to

$$\exists \tilde{v}_{ext}(A_{\mathcal{G}}) \supset \exists \tilde{v}(A) \tag{B.4}$$

By (B.4), Lemma Appendix B.1, and Definition 4.5, $\mathtt{LSat}(\mathcal{G}\upharpoonright \mathtt{p}, \exists \tilde{v}_{ext}(A_{\mathcal{G}}))$ returns true iff $\mathtt{LSat}(\mathcal{G}'\upharpoonright \mathtt{p}, \exists \tilde{v}_{ext}(A_{\mathcal{G}}) \wedge A)$ does. By (B.3) $\mathtt{LSat}(\mathcal{G}'\upharpoonright \mathtt{p}, \exists \tilde{v}_{ext}(A_{\mathcal{G}}) \wedge A) \equiv \mathtt{LSat}(\mathcal{G}'\upharpoonright \mathtt{p}, \exists \tilde{v}_{ext}(A_{\mathcal{G}} \wedge A))$. The thesis follows by induction on $\mathtt{GSat}(\mathcal{G}', A_{\mathcal{G}} \wedge A)$ and $\mathtt{LSat}(\mathcal{G}'\upharpoonright \mathtt{p}, \exists \tilde{v}_{ext}(A_{\mathcal{G}} \wedge A))$, observing that by history sensitivity of $\mathcal{G}$, $\tilde{v}_{ext} = var(A_{\mathcal{G}} \wedge A) \setminus I(\mathcal{G}_{root}\upharpoonright p)$, and $var(A_{\mathcal{G}} \wedge A) \subseteq I(\mathcal{G}_{root}) \setminus I(\mathcal{G})$.
Case $\mathtt{p} = \mathtt{p}_2$**.** By Definition 4.2, $\mathcal{G}\upharpoonright \mathtt{p} = k?(\tilde{v} : \tilde{S})\{\exists \tilde{v}_{ext}\tilde{v}'_{ext}(A_{\mathcal{T}} \wedge A)\};\mathcal{T}'$, where $\tilde{v}'_{ext}$ is the (possibly empty) vector of variables of A, which are not in $\tilde{v}_{ext}$ and which are not known by $\mathtt{p}$. By temporal satisfiability of $\mathcal{G}, A_{\mathcal{G}} \supset \exists \tilde{v}(A)$, which is equivalent to

$$A_{\mathcal{G}} \supset A_{\mathcal{G}} \wedge \exists \tilde{v}(A) \tag{B.5}$$

From (B.5) and (B.1)

$$A_{\mathcal{G}} \supset \exists \tilde{v}(A_{\mathcal{G}} \wedge A) \tag{B.6}$$

By weakening the conclusion of (B.6)

$$A_{\mathcal{G}} \supset \exists \tilde{v} \exists \tilde{v}_{ext}\tilde{v}'_{ext}(A_{\mathcal{G}} \wedge A) \tag{B.7}$$

Since $\tilde{v}_{ext}$ does not appear free in the conclusion of (B.7) then, by (a), (B.7) is equivalent to

$$\exists \tilde{v}_{ext}(A_{\mathcal{G}}) \supset \exists \tilde{v} \exists \tilde{v}_{ext}\tilde{v}'_{ext}(A_{\mathcal{G}} \wedge A) \tag{B.8}$$

Let $\mathcal{T}' = \mathcal{G}'\upharpoonright \mathtt{p}$; by (B.8) and Definition 4.5

$$\mathtt{LSat}(\mathcal{G}\upharpoonright \mathtt{p}, \exists \tilde{v}_{ext}(A_{\mathcal{G}})) \text{ is equivalent } \mathtt{LSat}(\mathcal{T}', \exists \tilde{v}_{ext}(A_{\mathcal{G}}) \wedge \exists \tilde{v}_{ext}\tilde{v}'_{ext}(A_{\mathcal{G}} \wedge A))$$

and the latter is equivalent to $\mathtt{LSat}(\mathcal{T}', \exists \tilde{v}_{ext}\tilde{v}'_{ext}(A_{\mathcal{G}} \wedge A))$ by (b). The lemma holds for this case by induction on $\mathtt{GSat}(\mathcal{G}', A_{\mathcal{G}} \wedge A)$ and $\mathtt{LSat}(\mathcal{T}', \exists \tilde{v}_{ext}\tilde{v}'_{ext}(A_{\mathcal{G}} \wedge A))$.

Like in the previous case, we obtain the thesis by induction.

Branching. $\mathcal{G} = \mathtt{p}_1 \to \mathtt{p}_2 \colon k\{\{A_j\}l_j \colon \mathcal{G}_j\}_{j\in J}$.
Case $\mathtt{p} = \mathtt{p}_1$. By Definition 4.2, $\mathcal{G}\upharpoonright\mathtt{p} = k \oplus \{\{A_j\}l_j : \mathcal{T}_j\}_{j\in J}$ where, by (B.2) and history sensitivity

$$\tilde{v}_{ext} \cap \bigcup_{j\in J} var(A_j) = \emptyset \tag{B.9}$$

By temporal satisfiability of $\mathcal{G}$,

$$A_{\mathcal{G}} \supset A_1 \vee \ldots \vee A_n \tag{B.10}$$

By (B.9) and (a), (B.10) is equivalent to

$$\exists\tilde{v}_{ext}(A_{\mathcal{G}}) \supset A_1 \vee \ldots \vee A_n \tag{B.11}$$

By (B.11) and Definition 4.5

$$\mathtt{LSat}(\mathcal{G}\upharpoonright\mathtt{p}, \exists\tilde{v}_{ext}(A_{\mathcal{G}})) = \wedge_{j\in J}\mathtt{LSat}(\mathcal{T}_j, \exists\tilde{v}_{ext}(A_{\mathcal{G}}) \wedge A_j)$$

where for each $j \in J$, $\mathcal{T}_j = \mathcal{G}_j\upharpoonright\mathtt{p}$. By (B.9) $\mathtt{LSat}(\mathcal{T}_j, \exists\tilde{v}_{ext}(A_{\mathcal{G}}) \wedge A_j) \equiv \mathtt{LSat}(\mathcal{T}_j, \exists\tilde{v}_{ext}(A_{\mathcal{G}} \wedge A_j))$. The thesis follows by applying the inductive hypothesis on all $\mathtt{GSat}(\mathcal{G}_j, A_{\mathcal{G}} \wedge A_j)$ and $\mathtt{LSat}(\mathcal{T}_j, \exists\tilde{v}_{ext}(A_{\mathcal{G}} \wedge A_j))$ with $j \in J$.

Like in the previous cases, we obtain the thesis by induction.
Case $\mathtt{p} = \mathtt{p}_2$. By Definition 4.2, $\mathcal{G}\upharpoonright\mathtt{p} = k\&\{\{\exists\tilde{v}_{ext}\tilde{v}^j_{ext}(A_{\mathcal{T}} \wedge A_j)\}l_j : \mathcal{T}_j\}_{j\in J}$ where for all $j \in J$, $\tilde{v}^j_{ext}$ are the (possibly empty) vectors of variables of A_j, which are not in $\tilde{v}_{ext}$ and which p does not know. By temporal satisfiability of $\mathcal{G}$, $A_{\mathcal{G}} \supset A_1 \vee \ldots \vee A_n$ which is equivalent to

$$A_{\mathcal{G}} \supset A_{\mathcal{G}} \wedge (A_1 \vee \ldots \vee A_n) \tag{B.12}$$

By weakening the conclusion of (B.12)

$$A_{\mathcal{G}} \supset \exists\tilde{v}_{ext}\tilde{v}^j_{ext}(A_{\mathcal{G}} \wedge (A_1 \vee \ldots \vee A_j)) \tag{B.13}$$

Since $\tilde{v}_{ext}$ does not appear free in the conclusion of (B.13) then, by (a), (B.13) is equivalent to

$$\exists\tilde{v}_{ext}(A_{\mathcal{G}}) \supset \exists\tilde{v}_{ext}\tilde{v}^j_{ext}(A_{\mathcal{G}} \wedge (A_1 \vee \ldots \vee A_j)) \tag{B.14}$$

By (B.14) and Definition 4.5, $\mathtt{LSat}(\mathcal{G}\upharpoonright\mathtt{p}, \exists\tilde{v}_{ext}(A_{\mathcal{G}})) = \wedge_{j\in J}\mathtt{LSat}(\mathcal{T}_j, \exists\tilde{v}_{ext}(A_{\mathcal{T}} \wedge A_j))$. The thesis follows by the inductive hypothesis, for all $j \in J$, on $\mathtt{GSat}(\mathcal{G}_j, A_{\mathcal{G}} \wedge A_j)$ and $\mathtt{LSat}(\mathcal{T}_j, \exists\tilde{v}_{ext}\tilde{v}^j_{ext}(A_{\mathcal{G}} \wedge A_j))$. Observe $\mathcal{T}_j = \mathcal{G}_j\upharpoonright\mathtt{p}$ by definition of projection, (B.2) holds for $\tilde{v}_{ext}\tilde{v}^j_{ext}$ on $A_{\mathcal{G}} \wedge A_j$, and (B.1) holds for $A_{\mathcal{G}} \wedge A_j$ by history sensitivity of $\mathcal{G}$.

Recursion. Given $\mathcal{G} = \mu\mathbf{t}\langle\tilde{u} : A\rangle(\tilde{v} : \tilde{S})\{B\}.\mathcal{G}'$, let

$$\begin{array}{rcll}
\tilde{v}^A_{ext} & = & var(A) \setminus I(\mathcal{G}_{root}\upharpoonright\mathtt{p}) & \\
\tilde{v}^B_{ext} & = & var(B) \setminus I(\mathcal{G}_{root}\upharpoonright\mathtt{p}) & \\
\mathcal{G}\upharpoonright\mathtt{p} & = & \mu\mathbf{t}\langle\tilde{u} : \exists\tilde{v}^A_{ext}(A)\rangle(\tilde{v} : \tilde{S})\{\exists\tilde{v}^B_{ext}(B)\}.\mathcal{T}' & \text{where } \mathcal{T}' = \mathcal{G}'\upharpoonright\mathtt{p}
\end{array}$$

By well-assertedness of $\mathcal{G}$

$$(A_{\mathcal{G}} \supset \exists\tilde{u}(A)) \qquad \text{and} \qquad (A_{\mathcal{G}} \wedge A \supset B[\tilde{u}/\tilde{v}]) \tag{B.15}$$

By weakening the conclusions of the two predicates in (B.15)

$$(A_{\mathcal{T}} \supset \exists \tilde{u}\tilde{v}^{A}_{ext}(A)) \qquad \text{and} \qquad (A_{\mathcal{T}} \wedge A \supset \tilde{v}^{B}_{ext}(B[\tilde{u}/\tilde{v}])) \tag{B.16}$$

By (B.1) and (a), (B.16) is equivalent to

$$(\exists \tilde{v}_{ext}(A_{\mathcal{T}}) \supset \exists \tilde{u}\tilde{v}^{A}_{ext}(A)) \qquad \text{and} \qquad (\exists \tilde{v}_{ext}(A_{\mathcal{T}} \wedge A) \supset \tilde{v}^{B}_{ext}(B[\tilde{u}/\tilde{v}])) \tag{B.17}$$

By (B.17) and Definition 4.5, $\mathtt{LSat}(\mathcal{G} \upharpoonright \mathtt{p}, \exists \tilde{v}_{ext}(A_{\mathcal{G}})) = \mathtt{LSat}(\mathcal{T}', \exists \tilde{v}_{ext}(A_{\mathcal{T}} \wedge B))$ which holds by induction.

Composition. $\mathcal{G} = \mathcal{G}_1, \mathcal{G}_2$. The projection is either $\mathcal{G}_1$ or $\mathcal{G}_2$, hence the result is immediate by induction hypothesis.

Assertion Variable. $\mathcal{G} = t_{B(\tilde{v})}\langle \tilde{u} : A \rangle$. The proof proceeds like in the case of recursion definition.

End - end. Immediate.

Appendix C. Runtime Processes and 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* (corresponding to *message types* in [4]), which abstract messages in queues.

Definition Appendix C.1 (Message Assertions). *The syntax of endpoint assertions (cf. Section 4) are extended as follows:*

$$\mathcal{T} ::= \ldots \mid \mathcal{M} \mid \mathcal{M}; \mathcal{T} \qquad\qquad \mathcal{M} ::= k!\langle \tilde{\mathtt{n}} \rangle \mid k \oplus l \mid \mathcal{M}; \mathcal{M}'$$

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

In Definition Appendix C.1, $k!\langle \tilde{\mathtt{n}} \rangle$ represents a value $\tilde{\mathtt{n}}$ in a queue, $k \oplus l$ represents a label in a queue, and $\mathcal{M}; \mathcal{M}'$ represents a queue with multiple elements. We use a context for endpoint assertion assignments $\mathcal{H}[\,\cdot\,]$; such context extends endpoint assignments to non-singleton assignments, and groups queues and assignments related to the same session. Namely, a vector of session channels is assigned to a set $\{\mathcal{T}_{\mathtt{p}} \,@\, \mathtt{p}\}_{\mathtt{p} \in I}$ of located endpoint assertions, indexed by participants.

$$\mathcal{H}[\,\cdot\,] ::= [\,\cdot\,] \mid \mathcal{H}[\,\cdot\,],\ \mathcal{T}_{\mathtt{p}} @ \mathtt{p} \mid \mathcal{T}_{\mathtt{p}} @ \mathtt{p},\ \mathcal{H}[\,\cdot\,]$$

We introduce the composition operator $\circ$ of endpoint assignments:

$$(\Delta_1, \tilde{s} : \emptyset) \circ \Delta_2 = \Delta_1 \circ \Delta_2$$

$$(\Delta_1, \tilde{s} : \mathcal{H}_1[\mathcal{M} @ \mathtt{p}]) \circ (\Delta_2, \tilde{s} : \mathcal{H}_2[\mathcal{T}_{\mathtt{p}} @ \mathtt{p}]) = (\Delta_1, \tilde{s} : \mathcal{H}_1[\mathcal{M}' @ \mathtt{p}]) \circ (\Delta_2, \tilde{s} : \mathcal{H}_2[\mathcal{T}'_{\mathtt{p}} @ \mathtt{p}])$$

where, in the second rule, we add a prefix of a message assertion to an endpoint assertion from the head of a queue ($\mathcal{M} * \mathcal{T}_{\mathtt{q}} = \mathcal{M}' * \mathcal{T}'_{\mathtt{p}}$), and $*$ is a commutative and associative operator s.t.

$$(k!\langle \tilde{\mathtt{n}} \rangle; \mathcal{M}) * \mathcal{T} = \mathcal{M} * k!\langle \tilde{\mathtt{n}} \rangle; \mathcal{T} \qquad\qquad (k \oplus l; \mathcal{M}) * \mathcal{T} = \mathcal{M} * k \oplus l; \mathcal{T}$$

$$\frac{A[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{true}}{\tilde{s}\colon \mathcal{H}[k!(\tilde{v}:\tilde{S})\{A\};\mathcal{T}] \to \tilde{s}\colon \mathcal{H}[k!\langle\tilde{\mathtt{n}}\rangle;\mathcal{T}[\tilde{\mathtt{n}}/\tilde{v}]]} \quad \text{[MA-SEND]}$$

$$\frac{A_j \downarrow \mathsf{true},\ j \in I}{\tilde{s}\colon \mathcal{H}[k \oplus \{\{A_i\}l_i : \mathcal{T}_i\}_{i\in I}] \to \tilde{s}\colon \mathcal{H}[k \oplus l_j;\mathcal{T}_j]} \quad \text{[MA-SEL]}$$

$$\frac{A[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{true}}{\tilde{s}\colon \mathcal{H}[k!\langle\tilde{\mathtt{n}}\rangle;\mathcal{T} \mathbin{@} \mathtt{p}, k?(\tilde{v}:\tilde{S})\{A\};\mathcal{T}' \mathbin{@} \mathtt{q}] \to \tilde{s}\colon \mathcal{H}[\mathcal{T} \mathbin{@} \mathtt{p}, \mathcal{T}'[\tilde{\mathtt{n}}/\tilde{v}] \mathbin{@} \mathtt{q}]} \quad \text{[MA-SR]}$$

$$\frac{A_j \downarrow \mathsf{true},\ j \in I}{\tilde{s}\colon \mathcal{H}[k \oplus l_j;\mathcal{T} \mathbin{@} \mathtt{p}, k\&\{\{A_i\}l_i : \mathcal{T}_i\}_{i\in I} \mathbin{@} \mathtt{q}] \to \tilde{s}\colon \mathcal{H}[\mathcal{T} \mathbin{@} \mathtt{p}, \mathcal{T}' \mathbin{@} \mathtt{q}]} \quad \text{[MA-SB]}$$

$$\frac{\Delta_1 \to \Delta_1'}{\Delta_1,\Delta_2 \to \Delta_1',\Delta_2} \qquad \frac{\Delta_2 \to \Delta_2'}{\Delta_1,\Delta_2 \to \Delta_1,\Delta_2'} \quad \text{[MA-CON1]/[MA-CON2]}$$

Figure C.9: Additional reduction rules for message assertions

Figure C.9 introduces the reduction rules for message assertions, which extend the rules in Definition 5, and play a key role in the proof of Subject Reduction (Lemma 7.2).

Rule [MA-SEND] non-deterministically instantiates an endpoint assertion for sending a value under predicate A to a corresponding message assertion whose carried values satisfies A. Rule [MA-SEL] non-deterministically instantiates an assertion for selecting a branch under the predicates $\{A_i\}_{i\in I}$ to a specific label (message assertion) l_j when A_j (with $j \in I$) evaluates to true. Rule [MA-SR] depicts how a sending message assertion interacts with its dual, the assertion for receiving. Rule [MA-SB] depict how a selection message assertion interacts with the assertion for branching. Rules [MA-CON1] and [MA-CON2] close the reduction under contexts. The motivation for having the non-deterministic instantiation rules for value sending and selection is to enable the assertion reduction to follow the process reduction: an assertion assignment has more reductions than the corresponding process, which serves the purpose since we only demand that the assertion *can* follow the process in reduction (as long as the predicate is satisfied).

Next we define transitions involving message assertions, where a non-singleton assignment may have a transition which represents a reduction at a free session channel; such transitions model communications from/to endpoint assertions to/from queues belonging to the same (non-singleton) assignment. The transition rules for message assertions, both the visible and the invisible ones, are given in Figure C.10. Rules [TR-M-SEND] and [TR-M-SEL] are straightforward and are used only over singleton assertions, just as in Figure 8. Other τ-transition relations follow the reduction rules defined in Figure C.9.

Proposition Appendix C.2 (Extended Transitions).

1. (coincidence with reduction) *If* $\langle\Gamma,\Delta\rangle \xrightarrow{\tau} \langle\Gamma,\Delta'\rangle$ *by the rules in Figure C.10, then* $\Delta \to \Delta'$.
2. (coincidence with reduction) $P \to Q$ *iff* $P \xrightarrow{\tau} Q$.
3. (determinism of non-free session actions) *Suppose* $\langle\Gamma,\Delta\rangle \xrightarrow{\alpha} \langle\Gamma',\Delta'\rangle$ *such that* α *is not derived from the rules in Figure C.10. Then* $\langle\Gamma,\Delta\rangle \xrightarrow{\alpha} \langle\Gamma'',\Delta''\rangle$ *implies* $\Gamma' = \Gamma''$ *and* $\Delta' = \Delta''$.

PROOF: (1) and (3) are straightforward. (2) is also direct from the definitions.

$$\frac{-}{\langle \Gamma, (\Delta, \tilde{s}:k!\langle \tilde{\mathtt{n}}\rangle; \mathcal{T} \,@\, \mathtt{p})\rangle \xrightarrow{s_k!\tilde{\mathtt{n}}} \langle \Gamma, (\Delta, \tilde{s}:\mathcal{T} \,@\, \mathtt{p})\rangle} \quad \text{[TR-M-SEND]}$$

$$\frac{-}{\langle \Gamma, (\Delta, \tilde{s}:k \oplus l_j; \mathcal{T} \,@\, \mathtt{p})\rangle \xrightarrow{s_k \triangleright l_j} \langle \Gamma, (\Delta, \tilde{s}:\mathcal{T} \,@\, \mathtt{p})\rangle} \quad \text{[TR-M-SEL]}$$

$$\frac{A[\tilde{\mathtt{n}}/\tilde{v}]\downarrow \mathsf{true}}{\tilde{s}:\mathcal{H}[k!(\tilde{v}:\tilde{S})\{A\};\mathcal{T}] \xrightarrow{\tau} \tilde{s}:\mathcal{H}[k!\langle \tilde{\mathtt{n}}\rangle;\mathcal{T}[\tilde{\mathtt{n}}/\tilde{v}]]} \quad \text{[TR-TAU-SEND]}$$

$$\frac{A_j\downarrow \mathsf{true} \quad j\in I}{\tilde{s}:\mathcal{H}[k\oplus\{\{A_i\}l_i:\mathcal{T}_i\}_{i\in I}] \xrightarrow{\tau} \tilde{s}:\mathcal{H}[k\oplus l_j;\mathcal{T}_j]} \quad \text{[TR-TAU-SEL]}$$

$$\frac{A[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{true}}{\tilde{s}:\mathcal{H}[k!\langle \tilde{\mathtt{n}}\rangle;\mathcal{T}@\mathtt{p}, k?(\tilde{v}:\tilde{S})\{A\};\mathcal{T}'\,@\,\mathtt{q}] \xrightarrow{\tau} \tilde{s}:\mathcal{H}[\mathcal{T}\,@\,\mathtt{p}, \mathcal{T}'[\tilde{\mathtt{n}}/\tilde{v}]\,@\,\mathtt{q}]} \quad \text{[TR-TAU-VAL]}$$

$$\frac{A_j \downarrow \mathsf{true} \quad j\in I}{\tilde{s}:\mathcal{H}[k\oplus l_j;\mathcal{T}@\mathtt{p}, k\&\{\{A_i\}l_i:\mathcal{T}_i\}_{i\in I}@\mathtt{q}] \xrightarrow{\tau} \tilde{s}:\mathcal{H}[\mathcal{T}@\mathtt{p}, \mathcal{T}'@\mathtt{q}]} \quad \text{[TR-TAU-BRA]}$$

$$\frac{\Delta_1 \to \Delta_1'}{\Delta_1,\Delta_2 \xrightarrow{\tau} \Delta_1',\Delta_2} \qquad \frac{\Delta_2 \to \Delta_2'}{\Delta_1,\Delta_2 \xrightarrow{\tau} \Delta_1,\Delta_2'} \quad \text{[TR-CON1]/[TR-CON2]}$$

Figure C.10: Labelled transition for message assertions

First, we extend $\Subset$ to message assertions by extending Definition 5.4 with the clauses about message assertions.

Definition Appendix C.3 (Refinement for Message Assertions). A binary relation $\mathcal{R}$ over closed endpoint assertions is a *refinement relation* if $\mathcal{T}_1\mathcal{R}\mathcal{T}_2$ implies one of the conditions in Definition 5.4 or one of the following conditions holds.

- $\mathcal{T}_1 = k!\langle \tilde{\mathtt{n}}\rangle;\mathcal{T}_1'$ and $\mathcal{T}_2 = k!\langle \tilde{\mathtt{n}}\rangle;\mathcal{T}_2'$ such that $\mathcal{T}_1'\mathcal{R}\mathcal{T}_2'$.
- $\mathcal{T}_1 = k\oplus l_j;\mathcal{T}_1'$ and $\mathcal{T}_2 = k\oplus l_j;\mathcal{T}_2'$ such that $\mathcal{T}_1'\mathcal{R}\mathcal{T}_2'$.
- $\mathcal{T}_1 = k!\langle \tilde{\mathtt{n}}\rangle;\mathcal{T}_1'$ and $\mathcal{T}_2 = k!(\tilde{v}:\tilde{S})\{A\};\mathcal{T}_2'$ such that $A[\tilde{\mathtt{n}}/\tilde{v}]\downarrow \mathsf{true}$ and $\mathcal{T}_1'\mathcal{R}\mathcal{T}_2'[\tilde{\mathtt{n}}/\tilde{v}]$.
- $\mathcal{T}_1 = k\oplus l_j;\mathcal{T}'$ and $\mathcal{T}_2 = k\oplus\{\{A_i\}l_i:\mathcal{T}_i'\}_{i\in I}$ with $(j\in I)$ such that $A_j\downarrow \mathsf{true}$ and $\mathcal{T}'\mathcal{R}\mathcal{T}_j'$.

If $\mathcal{T}_1\mathcal{R}\mathcal{T}_2$ for some refinement relation $\mathcal{R}$, then we say $\mathcal{T}_1$ is a *refinement* of $\mathcal{T}_2$, denoted $\mathcal{T}_1 \Subset \mathcal{T}_2$.

Assertion assignments used for refinements now include non-singleton assignments; in spite of this, the non-trivial refinement of endpoint assertions is only applied to singleton assignments.

Finally, we list the validation rules for queues and channel hiding in Figure C.11 (from [4]).

The first four rules concern the extended sets of processes and assertion assignments (accordingly [CONC] now uses the extended $\circ$ defined above). Henceforth we write $\mathcal{C};\Gamma \vdash P \triangleright \Delta$ for

$$\frac{-}{\mathcal{C};\Gamma \vdash s_k : \emptyset \triangleright \tilde{s} : \{\emptyset \mathbin{@} \mathtt{p}\}_{\mathtt{p}\in I}} \quad [\textsc{Qnil}]$$

$$\frac{\mathcal{C};\Gamma \vdash s_k : \tilde{h} \triangleright \Delta, \tilde{s} : \mathcal{H}[\mathcal{T} \mathbin{@} \mathtt{p}]}{\mathcal{C};\Gamma \vdash s_k : \tilde{h} \cdot \tilde{\mathtt{n}} \triangleright \Delta, \tilde{s} : \mathcal{H}[k!\langle \tilde{\mathtt{n}} \rangle; \mathcal{T} \mathbin{@} \mathtt{p}]} \quad [\textsc{Qval}]$$

$$\frac{\mathcal{C};\Gamma \vdash s_k : \tilde{h} \triangleright \Delta, \tilde{s} : \mathcal{H}[\mathcal{T} \mathbin{@} \mathtt{p}]}{\mathcal{C};\Gamma \vdash s_k : \tilde{h} \cdot l \triangleright \Delta, \tilde{s} : \mathcal{H}[k \oplus l; \mathcal{T} \mathbin{@} \mathtt{p}]} \quad [\textsc{Qsel}]$$

$$\frac{\mathcal{C};\Gamma \vdash P \triangleright \Delta, \tilde{s} : \{\mathcal{T}_\mathtt{p} \mathbin{@} \mathtt{p}\}_{\mathtt{p}\in I} \quad \{\mathcal{T}_\mathtt{p} \mathbin{@} \mathtt{p}\}_{\mathtt{p}\in I} \text{ coherent}}{\mathcal{C};\Gamma \vdash (\nu \tilde{s}) P \triangleright \Delta} \quad [\textsc{Cres}]$$

Figure C.11: Validation rules for runtime processes

a runtime process P when it is derived by combining the rules of Figure 6 and those of Figure C.11.
Note that the validation of the composability of multiple processes is relegated to the session hiding rule [CRes] rather than to the parallel composition rule [Conc]. By the shape of these rules we immediately observe:

Proposition Appendix C.4. *Suppose* $\Gamma \vdash P \triangleright \Delta$*, then* P *contains no* errH *nor* errT.

Appendix D. Refinement (Proofs of Section 6.3)

Appendix D.1. Proof of Lemma 6.5

Assume $\Delta \Subset \Delta'$ below.

1. If $\langle \Gamma, \Delta \rangle \xrightarrow{\alpha} \langle \Gamma_1, \Delta_1 \rangle$ such that α is an output or selection, then $\langle \Gamma, \Delta' \rangle \xrightarrow{\alpha} \langle \Gamma_1, \Delta'_1 \rangle$ such that $\Delta_1 \Subset \Delta'_1$ again.
2. If $\langle \Gamma, \Delta \rangle \xrightarrow{\alpha} \langle \Gamma_1, \Delta_1 \rangle$ such that α is an input or branching action, and if $\langle \Gamma, \Delta' \rangle$ allows α, then $\langle \Gamma, \Delta' \rangle \xrightarrow{\alpha} \langle \Gamma_1, \Delta'_1 \rangle$ such that $\Delta_1 \Subset \Delta'_1$ again.
3. If $\langle \Gamma, \Delta \rangle \xrightarrow{\tau} \langle \Gamma, \Delta_1 \rangle$ then $\langle \Gamma, \Delta' \rangle \xrightarrow{\tau} \langle \Gamma, \Delta'_1 \rangle$ or $\langle \Gamma, \Delta' \rangle \xrightarrow{\tau}\xrightarrow{\tau} \langle \Gamma, \Delta'_1 \rangle$ such that $\Delta_1 \Subset \Delta'_1$ again.

Proof: We first proof (1) and (2) by induction on the structure of Δ. We assume Δ (resp. Δ') to have the structure $\Delta_{side}, \Delta_{ref}$ (resp. $\Delta'_{side}, \Delta'_{ref}$) where Δ_{ref} has the form $\tilde{s} : \mathcal{T} \mathbin{@} \mathtt{p}$. We assume without loss of generality that the transition is from Δ_{ref}. We do not consider [TR-LinkOut] and [TR-LinkIn] since they just add the same new element to the assertion assignment. For the same reason, in the proofs below for value input and value output, we do not consider the cases of new name import and export, since they only add to Γ the same new elements.

Values sending. If $\Delta = \Delta_{side}, \tilde{s} : k!(\tilde{v} : \tilde{S})\{A_1\}; \mathcal{T}' \mathbin{@} \mathtt{p}$ then $\langle \Gamma, \Delta \rangle \xrightarrow{s_k!\tilde{\mathtt{n}}} \langle \Gamma, \Delta_{side}, \tilde{s} : \mathcal{T}' \mathbin{@} \mathtt{p} \rangle$ by [TR-Send]. By hypothesis we have $\Delta \Subset \Delta'$, so we can set $\Delta' = \Delta'_{side}, \tilde{s} : k!(\tilde{v} : \tilde{S})\{A_2\}; \mathcal{T}'' \mathbin{@} \mathtt{p}$ by Definition 5.4, with $\Delta_{side} \Subset \Delta'_{side}$, $\mathcal{T}' \Subset \mathcal{T}''$ and $A_1 \supset A_2$. Since $A_1 \supset A_2$ then

$$A_1[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{true} \supset A_2[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{true}$$

It follows that also the following transition is possible: $\langle \Gamma, \Delta' \rangle \xrightarrow{s_k!\tilde{\mathtt{n}}} \langle \Gamma, \Delta'_{side}, \tilde{s} : \mathcal{T}'' \mathbin{@} \mathtt{p} \rangle$. The lemma hold by induction for this case since $\Delta_{side}, \tilde{s} : \mathcal{T}' \mathbin{@} \mathtt{p} \Subset \Delta'_{side}, \tilde{s} : \mathcal{T}'' \mathbin{@} \mathtt{p}$.

Value receiving. If $\Delta = \Delta_{side}, \tilde{s}:\ k?(\tilde{v}:\tilde{S})\{A_1\};\mathcal{T}' \,@\, \mathtt{p}$ then $\langle \Gamma, \Delta\rangle \stackrel{s_k?\tilde{\mathtt{n}}}{\to} \langle \Gamma, \Delta_{side}, \tilde{s}:\mathcal{T}' \,@\, \mathtt{p}\rangle$ by [Tr-Rec]. Assume further we have

$$\langle \Gamma, \Delta'\rangle \text{ allows } s_k?\tilde{\mathtt{n}} \tag{D.1}$$

As before, by hypothesis and by Definition 5.4 we can set: $\Delta' = \Delta'_{side}, \tilde{s}:k?(\tilde{v}:\tilde{S})\{A_2\};\mathcal{T}'' \,@\, \mathtt{p}$ such that $\Delta_{side} \ni \Delta'_{side}$, $\mathcal{T}' \ni \mathcal{T}''$ and $A_2 \supset A_1$. By (D.1), however, we also have $A_2[\tilde{\mathtt{n}}/\tilde{v}]\downarrow \mathsf{true}$. It follows that the following transition is possible: $\langle \Gamma, \Delta'\rangle \stackrel{s_k?\tilde{\mathtt{n}}}{\to} \langle \Gamma, \Delta'_{side}, \tilde{s}:\mathcal{T}'' \,@\, \mathtt{p}\rangle$. The statement hold since $\Delta_{side}, \tilde{s}:\mathcal{T}' \,@\, \mathtt{p} \ni \Delta'_{side}, \tilde{s}:\mathcal{T}'' \,@\, \mathtt{p}$.

Selection. If $\Delta = \Delta_{side}, \tilde{s}:k \oplus \{\{A_{1i}\}l_i:\mathcal{T}_{1i}\}_{i\in I} \,@\, \mathtt{p}$ then by [Tr-Sel] $\langle \Gamma, \Delta\rangle \stackrel{s_k \lhd l_j}{\to} \langle \Gamma, \Delta_{side}, \tilde{s}:\mathcal{T}_{1j} \,@\, \mathtt{p}\rangle$. By hypothesis and by Definition 5.4, we can set $\Delta' = \Delta'_{side}, \tilde{s}:k \oplus \{\{A_{1i}\}l_i:\mathcal{T}_{1i}\}_{i\in J} \,@\, \mathtt{p}$ with $\Delta_{side} \ni \Delta'_{side}$, and there exists $i \in J$ such that $l_j = l_i$, $A_{1i} \supset A_{2j}$ and $\mathcal{T}_{1i} \ni \mathcal{T}_{2j}$. It follows that also the following transition is possible: $\langle \Gamma, \Delta'\rangle \stackrel{s_k \lhd l_j}{\to} \langle \Gamma, \Delta'_{side}, \tilde{s}:\mathcal{T}_{2j} \,@\, \mathtt{p}\rangle$. The lemma hold since $\Delta_{side}, \tilde{s}:\mathcal{T}_{1i} \,@\, \mathtt{p} \ni \Delta'_{side}, \tilde{s}:\mathcal{T}_{2j} \,@\, \mathtt{p}$.

Branching. If $\Delta = \Delta_{side}, \tilde{s}:k\&\{\{A_{1i}\}l_i:\mathcal{T}_{1i}\}_{i\in I} \,@\, \mathtt{p}$ then $\langle \Gamma, \Delta\rangle \stackrel{s_k \rhd l_j}{\to} \langle \Gamma, \Delta_{side}, \tilde{s}:\mathcal{T}_{1j} \,@\, \mathtt{p}\rangle$. Assume further we have

$$\langle \Gamma, \Delta'\rangle \text{ allows } s_k \rhd l_j \tag{D.2}$$

By hypothesis and by Definition 5.4, we can set $\Delta = \Delta'_{side}, \tilde{s}:k\&\{\{A_{1i}\}l_i:\mathcal{T}_{1i}\}_{i\in J} \,@\, \mathtt{p}$ with $\Delta_{side} \ni \Delta'_{side}$. By (D.2) we know $j \in J$ and $A_{1j}\downarrow \mathsf{true}$. It follows that also the following transition is possible: $\langle \Gamma, \Delta'\rangle \stackrel{s_k \rhd l_j}{\to} \langle \Gamma, \Delta'_{side}, \tilde{s}:\mathcal{T}_{2j} \,@\, \mathtt{p}\rangle$. The lemma hold since $\Delta_{side}, \tilde{s}:\mathcal{T}_{1i} \,@\, \mathtt{p} \ni \Delta'_{side}, \tilde{s}:\mathcal{T}_{2j} \,@\, \mathtt{p}$.

In the proof for (3), we denote with τ the (silent) transition in Figure 8 and with $\tau_{\mathtt{s}}$ the transitions in Figure C.10 (reductions at a free session channel). The case for τ is immediate.

We now prove (3) for actions $\tau_{\mathtt{s}}$, more precisely we prove that if $\langle \Gamma, \Delta\rangle \stackrel{\tau_{\mathtt{s}}}{\to} \langle \Gamma, \Delta_1\rangle$ then $\langle \Gamma, \Delta'\rangle \stackrel{\tau_{\mathtt{s}}}{\to} \langle \Gamma, \Delta'_1\rangle$ or $\langle \Gamma, \Delta'\rangle \stackrel{\tau_{\mathtt{s}}}{\to}\stackrel{\tau_{\mathtt{s}}}{\to} \langle \Gamma, \Delta'_1\rangle$ such that $\Delta_1 \ni \Delta'_1$ again.

The proof is identical to the proof of Lemma 6.5 except the pairs introduced in Definition Appendix C.3. The case for identical pairs is immediate. For the remaining two cases, we treat the case of send. The case of selection is by the same argument. First we consider the case of a visible action. Using the same notations as in the proof of Lemma 6.5:

$$\begin{array}{lcl} \Delta & = & \Delta_{side}, \tilde{s}:k!\langle \tilde{\mathtt{n}}\rangle;\mathcal{T}_1 \,@\, \mathtt{p} \\ \Delta' & = & \Delta'_{side}, \tilde{s}:k!(\tilde{v}:\tilde{S})\{A\};\mathcal{T}'_1 \,@\, \mathtt{p} \end{array}$$

such that $\Delta_{side} \ni \Delta'_{side}$, $\mathcal{T}_1 \ni \mathcal{T}'_1$ and $A[\tilde{\mathtt{n}}/\tilde{v}]\downarrow \mathsf{true}$. Now consider the following labelled transition:

$$\langle \Gamma, \Delta\rangle \stackrel{s_k!\tilde{\mathtt{n}}}{\to} \langle \Gamma, \Delta_{side}, \tilde{s}:\mathcal{T}' \,@\, \mathtt{p}\rangle$$

By $A[\tilde{\mathtt{n}}/\tilde{v}]\downarrow \mathsf{true}$ we can derive:

$$\langle \Gamma, \Delta'\rangle \stackrel{s_k!\tilde{\mathtt{n}}}{\to} \langle \Gamma, \Delta'_{side}, \tilde{s}:\mathcal{T}'_1 \,@\, \mathtt{p}\rangle.$$

as required. Next we consider the $\tau_{\mathtt{s}}$-action. Suppose

$$\langle \Gamma, \Delta\rangle \stackrel{\tau_{\mathtt{s}}}{\to} \langle \Gamma, \Delta_1\rangle \tag{D.3}$$

First assume in (D.3) that this action is induced by the reduction from $\tilde{s} : \mathcal{H}[k!(\tilde{v} : \tilde{S})\{A\}; \mathcal{T}]$ in Δ to its instantiation $\tilde{s} : \mathcal{H}[k!\langle \tilde{\mathtt{n}} \rangle; \mathcal{T}[\tilde{\mathtt{n}}/\tilde{v}] \mathbin{@} \mathtt{p}]$ in Δ_1 such that

$$A[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{true}. \tag{D.4}$$

Then Δ' will have the corresponding reduction from

$$\tilde{s} : \mathcal{H}[k!(\tilde{v} : \tilde{S})\{A'\}; \mathcal{T}']$$

in Δ', because, by the definition of refinement, we have $A \supset A'$, hence by (D.4) we obtain $A'[\tilde{\mathtt{n}}/\tilde{v}] \downarrow$ true too, so that we obtain the corresponding instantiation:

$$\tilde{s} : \mathcal{H}[k!\langle \tilde{\mathtt{n}} \rangle; \mathcal{T}'[\tilde{\mathtt{n}}/\tilde{v}] \mathbin{@} \mathtt{p}]$$

for which we have, by definition, $k!\langle \tilde{\mathtt{n}} \rangle; \mathcal{T}[\tilde{\mathtt{n}}/\tilde{v}] \ \ni \ k!\langle \tilde{\mathtt{n}} \rangle; \mathcal{T}'[\tilde{\mathtt{n}}/\tilde{v}]$ as required. On the other hand if the transition in (D.3) is induced by the following redex in Δ

$$\tilde{s} : \mathcal{H}[k!\langle \tilde{\mathtt{n}} \rangle; \mathcal{T}_a \mathbin{@} \mathtt{p}, k?(\tilde{v})\{A_b\}; \mathcal{T}_b \mathbin{@} \mathtt{q}]$$

and, under $A[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{true}$, this has the reduction into:

$$\tilde{s} : \mathcal{H}[\mathcal{T}_a \mathbin{@} \mathtt{p}, \mathcal{T}_b \mathbin{@} [\tilde{\mathtt{n}}/\tilde{v}] \mathbin{@} \mathtt{q}]$$

First assume the corresponding assertions in Δ' have the isomorphic shape:

$$\tilde{s} : \mathcal{H}[k!\langle \tilde{\mathtt{n}} \rangle; \mathcal{T}'_a \mathbin{@} \mathtt{p}, k?(\tilde{v})\{A'_b\}; \mathcal{T}'_b \mathbin{@} \mathtt{q}] \tag{D.5}$$

such that $\mathcal{T}_a \supset \mathcal{T}'_a$, $A'_b \supset A_b$, and $\mathcal{T}_b[\tilde{\mathtt{m}}/\tilde{v}] \ni \mathcal{T}'_b[\tilde{\mathtt{m}}/\tilde{v}]$ (if $A'_b[\tilde{\mathtt{m}}/\tilde{v}] \downarrow \mathsf{true}$). Thus (D.5) can have the corresponding reduction, hence $\langle \Gamma, \Delta' \rangle$ can have the corresponding $\tau_{\mathtt{s}}$-action, and the result is again in the closure, as required. Second when the corresponding assertions in Δ' do *not* have the isomorphic shape, we can set:

$$\tilde{s} : \mathcal{H}[k!(\tilde{v} : \tilde{S})\{A'_a\}; \mathcal{T}'_a \mathbin{@} \mathtt{p}, k?(\tilde{v})\{A'_b\}; \mathcal{T}'_b \mathbin{@} \mathtt{q}] \tag{D.6}$$

s.t. $\mathcal{T}_a[\tilde{\mathtt{m}}/\tilde{v}] \ni \mathcal{T}'_b[\tilde{\mathtt{m}}/\tilde{v}]$ (if $A_a[\tilde{\mathtt{m}}/\tilde{v}] \downarrow \mathsf{true}$), $\mathcal{T}_b[\tilde{\mathtt{m}}/\tilde{v}]\mathcal{T}'_b[\tilde{\mathtt{m}}/\tilde{v}]$ (if $A'_b[\tilde{\mathtt{m}}/\tilde{v}] \downarrow \mathsf{true}$), and the following:

$$A_a \supset A'_a[\tilde{\mathtt{n}}/\tilde{v}] \tag{D.7}$$

$$A'_b \supset A_b. \tag{D.8}$$

By (D.7) we know (D.6) has the reduction into:

$$\tilde{s} : \mathcal{H}[k!\langle \tilde{\mathtt{n}} \rangle; \mathcal{T}'_a[\tilde{\mathtt{m}}/\tilde{v}] \mathbin{@} \mathtt{p}, k?(\tilde{v})\{A'_b\}; \mathcal{T}'_b \mathbin{@} \mathtt{q}] \tag{D.9}$$

From (D.8) we obtain the reduction from (D.9) into $\tilde{s} : \mathcal{H}[\mathcal{T}'_a[\tilde{\mathtt{m}}/\tilde{v}] \mathbin{@} \mathtt{p}, \mathcal{T}'_b \mathbin{@} [\tilde{\mathtt{n}}/\tilde{v}] \mathbin{@} \mathtt{q}]$ as required.

Appendix D.2. Proof of Proposition 6.6

If $\Gamma \models P \triangleright \Delta$ and $\Delta \ni \Delta'$ then $\Gamma \models P \triangleright \Delta'$.

PROOF: The proof is by induction on the transitions of P. We proceed by case analysis. We consider only the cases of Definition 5.4 (i.e., we ignore the cases where the end-point assertion differs from its refinement only in the continuation since they are straightforward by induction). In this proof we refer to the transition rules for asserted processes in Figure 7 and the transition rules for end-point assertions in Figure 8.

Value sending. If $P = s_k!\langle\vec{e}\rangle(\vec{v})\{A_1\};P'$ then $\Delta = \Delta_{side}, \tilde{s}:k!(\tilde{v}:\tilde{S})\{A_1\};\mathcal{T}'\,@\,\mathtt{p}$. Let $\tilde{e}\downarrow\tilde{\mathtt{n}}$, $P \stackrel{s_k!\tilde{\mathtt{n}}}{\rightarrow} P'$ by either [SEND] or [SENDERR] according to whether $A_1[\tilde{\mathtt{n}}/\tilde{v}]\downarrow\mathsf{true}$ or $A_1[\tilde{\mathtt{n}}/\tilde{v}]\downarrow\mathsf{false}$. Since $\Gamma \models P \triangleright \Delta$ (by hypothesis) then:

Case $A_1[\tilde{\mathtt{n}}/\tilde{v}]\downarrow$ **true.** $\langle\Gamma,\Delta\rangle \stackrel{s_k!\tilde{\mathtt{n}}}{\rightarrow} \langle\Gamma,\Delta_{side},\tilde{s}:\mathcal{T}'\,@\,\mathtt{p}\rangle$ by [TR-SEND]. By Lemma 6.5 $\langle\Gamma,\Delta'\rangle \stackrel{s_k!\tilde{\mathtt{n}}}{\rightarrow} \langle\Gamma,\Delta'_{side},\tilde{s}:\mathcal{T}''\,@\,\mathtt{p}\rangle$. By induction, if $\Gamma \models P' \triangleright \Delta_{side},\tilde{s}:\mathcal{T}'\,@\,\mathtt{p}$ then $\Gamma \models P' \triangleright \Delta'_{side},\tilde{s}:\mathcal{T}''\,@\,\mathtt{p}$.

Case $A_1[\tilde{\mathtt{n}}/\tilde{v}]\downarrow$ **false.** $\langle\Gamma,\Delta\rangle$ cannot make any transition and the lemma holds trivially.

Value receiving. If $P = s_k?(\vec{v})\{A_1\};P'$ then $\Delta = \Delta_{side},\tilde{s}:\ k?(\tilde{v}:\tilde{S})\{A_1\};\mathcal{T}'\,@\,\mathtt{p}$. $P \stackrel{s_k?\tilde{\mathtt{n}}}{\rightarrow} P'$ by either [RECV] or [RECVERR] according to whether $A_1[\tilde{\mathtt{n}}/\tilde{v}]\downarrow\mathsf{true}$ or $A_1[\tilde{\mathtt{n}}/\tilde{v}]\downarrow\mathsf{false}$. Since $\Gamma \models P \triangleright \Delta$ (by hypothesis) then:

Case $A_1[\tilde{\mathtt{n}}/\tilde{v}]\downarrow$ **true.** $\langle\Gamma,\Delta\rangle \stackrel{s_k?\tilde{\mathtt{n}}}{\rightarrow} \langle\Gamma,\Delta_{side},\tilde{s}:\mathcal{T}'\,@\,\mathtt{p}\rangle$ by [TR-REC]. By Lemma 6.5 $\langle\Gamma,\Delta'\rangle \stackrel{s_k?\tilde{\mathtt{n}}}{\rightarrow} \langle\Gamma,\Delta'_{side},\tilde{s}:\mathcal{T}''\,@\,\mathtt{p}\rangle$. By induction, if $\Gamma \models P' \triangleright \Delta_{side},\tilde{s}:\mathcal{T}'\,@\,\mathtt{p}$ then $\Gamma \models P' \triangleright \Delta'_{side},\tilde{s}:\mathcal{T}''\,@\,\mathtt{p}$.

Case $A_1[\tilde{\mathtt{n}}/\tilde{v}]\downarrow$ **false.** if $A_1[\tilde{\mathtt{n}}/\tilde{v}]\downarrow\mathsf{false}$ then $\langle\Gamma,\Delta\rangle$ cannot make any transition and the lemma holds trivially.

Select. If $P = s_k \lhd \{A_{1j}\}l_j : P'$ then $\Delta = \Delta_{side},\tilde{s}:k\oplus\{\{A_{1i}\}l_i:\mathcal{T}_{1i}\}_{i\in I}\,@\,\mathtt{p}$. $P \stackrel{s_k\lhd l_j}{\rightarrow} P'$ by either [LABEL] or [LABELERR] according to whether $A_{1j}\downarrow\mathsf{true}$ or $A_{1j}\downarrow\mathsf{false}$. Since $\Gamma \models P \triangleright \Delta$ (by hypothesis) then:

Case $A_{1j}\downarrow$ **true.** $\langle\Gamma,\Delta\rangle \stackrel{s_k\lhd l_j}{\rightarrow} \langle\Gamma,\Delta_{side},\tilde{s}:\mathcal{T}_{1j}\,@\,\mathtt{p}\rangle$ by [TR-SEL]. By Lemma 6.5 $\langle\Gamma,\Delta'\rangle \stackrel{s_k\lhd l_j}{\rightarrow} \langle\Gamma,\Delta'_{side},\tilde{s}:\mathcal{T}_{2j}\,@\,\mathtt{p}\rangle$. By induction, if $\Gamma \models P' \triangleright \Delta_{side},\tilde{s}:\mathcal{T}_{1j}\,@\,\mathtt{p}$ then $\Gamma \models P' \triangleright \Delta'_{side},\tilde{s}:\mathcal{T}_{2j}\,@\,\mathtt{p}$.

Case $A_{1j}\downarrow$ **false.** $\langle\Gamma,\Delta\rangle$ cannot make any transition and the lemma holds trivially.

Branching. If $P = s_k \rhd \{\{A_{1j}\}l_j:\ P_j\}_{i\in I}$ then $\Delta = \Delta_{side},\tilde{s}:k\&\{\{A_{1i}\}l_i:\mathcal{T}_{1i}\}_{i\in I}\,@\,\mathtt{p}$. $P \stackrel{s_k\rhd l_j}{\rightarrow} P'$ by either [BRANCH] or [BRANCHERR] according to whether $A_{1j}\downarrow\mathsf{true}$ or $A_{1j}\downarrow\mathsf{false}$. Since $\Gamma \models P \triangleright \Delta$ (by hypothesis) then:

Case $A_{1j}\downarrow$ **true.** $\langle\Gamma,\Delta\rangle \stackrel{s_k\rhd l_j}{\rightarrow} \langle\Gamma,\Delta_{side},\tilde{s}:\mathcal{T}_{1j}\,@\,\mathtt{p}\rangle$. By Lemma 6.5 $\langle\Gamma,\Delta'\rangle \stackrel{s_k\rhd l_j}{\rightarrow} \langle\Gamma,\Delta'_{side},\tilde{s}:\mathcal{T}_{2j}\,@\,\mathtt{p}\rangle$. By induction, if $\Gamma \models P' \triangleright \Delta_{side},\tilde{s}:\mathcal{T}_{1j}\,@\,\mathtt{p}$ then $\Gamma \models P' \triangleright \Delta'_{side},\tilde{s}:\mathcal{T}_{2j}\,@\,\mathtt{p}$.

Case $A_{1j}\downarrow$ **false.** $\langle\Gamma,\Delta\rangle$ cannot make any transition and the lemma holds trivially.

Parallel. If $P = Q \mid R$ then $\Delta = \Delta_Q,\Delta_R$, where $\mathcal{C};\Gamma \vdash Q \triangleright \Delta_Q$ and $\mathcal{C};\Gamma \vdash R \triangleright \Delta_R$. Let us consider the case in which $Q \mid R \stackrel{\alpha}{\rightarrow} Q' \mid R$ (by [PAR]) and for some rule $\langle\Gamma,\Delta_R,\Delta_Q\rangle \stackrel{\alpha}{\rightarrow} \langle\Gamma,\Delta_R,\Delta_{Q1}\rangle$. Since $\Delta \ni \Delta'$ (by hypothesis), then $\Delta' = \Delta'_Q,\Delta'_R$ where $\Delta_R \ni \Delta'_R$ and $\Delta_Q \ni \Delta'_Q$ (by rule [CONC]). By Lemma 6.5, $\langle\Gamma,\Delta'\rangle \stackrel{\alpha}{\rightarrow} \langle\Gamma,\Delta'_R,\Delta'_{Q1}\rangle$. By induction, if $\Gamma \models P' \triangleright \Delta_R,\Delta_{Q1}$ then $\Gamma \models P' \triangleright \Delta'_R,\Delta'_{Q1}$.

Appendix E. Substitution and Evaluation Lemmas

Both subject reduction for visible transitions and soundness hinge on the substitution lemma and on the evaluation lemma.

The substitution lemma uses the following lemma saying that any substitution of a free variable with a constant in an endpoint assertion preserves well-assertedness.

Lemma Appendix E.1. *If $\mathcal{T}$ is well-typed and well-asserted, $\tilde{u} : \tilde{S}$ and $\tilde{\mathtt{n}} : \tilde{S}$, then $\mathcal{T}[\tilde{\mathtt{n}}/\tilde{u}]$ is well-asserted.*

PROOF: Straightforward from the fact that `LSat` universally quantifies the free variables of $\mathcal{T}$. □

Lemma Appendix E.2 (Substitution Lemma). *Let $\mathcal{C} \vdash P \triangleright \Delta$ with Δ well-asserted and $u : S$ in the typing environment* $\mathsf{erase}(\Gamma)$ *(i.e., [29]). If $u \in \mathrm{fn}(P)$ and $\mathtt{n}$ has sort S then $\mathcal{C}[\mathtt{n}/u]; \Gamma \vdash P[\mathtt{n}/u] \triangleright \Delta[\mathtt{n}/u]$ and $\Delta[\mathtt{n}/u]$ is well-asserted.*

PROOF: The substitution lemma uses the following lemma saying that any substitution of a free variable with a constant in an endpoint assertion preserves well-assertedness.

Lemma Appendix E.3. *If $\mathcal{T}$ is well-asserted and well-typed under* $\mathsf{erase}(\Gamma), \tilde{u} : \tilde{S}$ *and $\tilde{\mathtt{n}} : \tilde{S}$ then $\mathcal{T}[\tilde{\mathtt{n}}/\tilde{u}]$ is well-asserted.*

PROOF: The proof trivially follows from the fact that `LSat` universally quantifies the free variables of $\mathcal{T}$. □

PROOF: The proof is by rule induction on validation rules. We proceed by case analysis o the rules in Figure 6 and Figure C.11. Assume $u : S'$.

Values sending. If $P = s_k!\langle\tilde{e}\rangle(\tilde{v})\{A\}; P'$ then by [SND]

$$\frac{\mathcal{C} \supset A[\tilde{e}/\tilde{v}] \quad \mathcal{C}; \Gamma \vdash P'[\tilde{e}/\tilde{v}] \triangleright \Delta, \tilde{s} \colon \mathcal{T} \mathbin{@} \mathtt{p}}{\mathcal{C}; \Gamma \vdash s_k!\langle\tilde{e}\rangle(\tilde{v})\{A\}; P' \triangleright \Delta, \tilde{s} \colon k!(\tilde{v} : \tilde{S})\{A\}; \mathcal{T} \mathbin{@} \mathtt{p}}$$

Without loss of generality we assume $u \notin \tilde{v}$. By definition, $(\mathcal{C} \supset A[\tilde{e}/\tilde{v}])[\mathtt{n}/u]$ is equivalent to $\mathcal{C}[\mathtt{n}/u] \supset A[\tilde{e}/\tilde{v}][\mathtt{n}/u]$ and, since $\mathcal{C} \supset A[\tilde{e}/\tilde{v}]$ is supposed to be universally quantified on the free variable u, thus

$$\mathcal{C}[\mathtt{n}/u] \supset A[\tilde{e}/\tilde{v}][\mathtt{n}/u] \tag{E.1}$$

Moreover, by inductive hypothesis, we have

$$\mathcal{C}[\mathtt{n}/u]; \Gamma \vdash P'[\tilde{e}/\tilde{v}][\mathtt{n}/u] \triangleright (\Delta, \tilde{s} \colon \mathcal{T} \mathbin{@} \mathtt{p})[\mathtt{n}/u] \tag{E.2}$$

By applying [SND] with premises (E.1) and (E.2) we obtain

$$\mathcal{C}[\mathtt{n}/u]; \Gamma \vdash (s_k!\langle\tilde{e}\rangle(\tilde{v})\{A\}; P')[\mathtt{n}/u] \triangleright (\Delta, \tilde{s} : k!(\tilde{v} : \tilde{S})\{A\}; \mathcal{T} \mathbin{@} \mathtt{p})[\mathtt{n}/u]$$

The substituted endpoint assertion is well-asserted by Lemma Appendix E.3.

Values receiving. If $P = s_k?(\tilde{v})\{A\}; P'$ then by [RCV]

$$\frac{\mathcal{C} \wedge A, \Gamma \vdash P' \triangleright \Delta, \tilde{s} \colon \mathcal{T} \mathbin{@} \mathtt{p}}{\mathcal{C}; \Gamma \vdash s_k?(\tilde{v})\{A\}; P' \triangleright \Delta, \tilde{s} : k?(\tilde{v} : \tilde{S})\{A\}; \mathcal{T} \mathbin{@} \mathtt{p}}$$

Without loss of generality we assume $u \notin \tilde{v}$. By inductive hypothesis

$$(\mathcal{C} \wedge A)[\mathtt{n}/u]; \Gamma \vdash P'[\mathtt{n}/u] \triangleright (\Delta, \tilde{s} \colon \mathcal{T} \mathbin{@} \mathtt{p})[\mathtt{n}/u]. \tag{E.3}$$

By applying (E.3) as a premise for [RCV] we obtain

$$\mathcal{C}[\mathtt{n}/u]; \Gamma \vdash (s_k?(\tilde{v})\{A\}; P')[\mathtt{n}/u] \triangleright (\Delta, \tilde{s} \colon \ k?(\tilde{v} : \tilde{S})\{A\}; \mathcal{T} \mathbin{@} \mathtt{p})[\mathtt{n}/u]$$

The substituted endpoint assertion is well-asserted by Lemma Appendix E.3.

Selection. If $P = s_k \triangleleft \{A_j\} l_j : P_j$ then by [SEL]

$$\frac{\Gamma \vdash \mathcal{C} \supset A_j \quad \mathcal{C}; \Gamma, u : S' \vdash P_j \triangleright \Delta, \tilde{s} \colon \mathcal{T}_j \,@\, \mathtt{p} \quad j \in I}{\mathcal{C}; \Gamma, u : S' \vdash s_k \triangleleft \{A_j\} l_j : P_j \triangleright \Delta, \tilde{s} : k \oplus \{\{A_i\} l_i : \mathcal{T}_i\}_{i \in I} \,@\, \mathtt{p}}$$

By inductive hypothesis

$$\mathcal{C}[\mathtt{n}/u]; \Gamma \vdash P_j[\mathtt{n}/u] \triangleright (\Delta, \tilde{s} \colon \mathcal{T} \,@\, \mathtt{p})[\mathtt{n}/u] \tag{E.4}$$

Since $(\mathcal{C} \supset A_j)$ then also $(\mathcal{C} \supset A_j)[\mathtt{n}/u]$ holds, i.e.

$$\Gamma \vdash \mathcal{C}[\mathtt{n}/u] \supset A_j[\mathtt{n}/u] \tag{E.5}$$

By applying (E.4) and (E.5) as a premise for [SEL] we obtain

$$\mathcal{C}[\mathtt{n}/u]; \Gamma \vdash (s_k \triangleleft \{A_j\} l_j : P_j)[\mathtt{n}/u] \triangleright (\Delta, \tilde{s} : k \oplus \{\{A_i\} l_i : \mathcal{T}_i\}_{i \in I} \,@\, \mathtt{p})[\mathtt{n}/u]$$

where the substituted endpoint assertion is well-asserted by Lemma Appendix E.3

Branching. The case for [BRA] is similar to the case of [SEL].

Conditional. If $P = \mathsf{if}\ e\ \mathsf{then}\ Q\ \mathsf{else}\ R$ then by [IF]

$$\frac{\mathcal{C} \wedge e; \Gamma \vdash Q \triangleright \Delta \quad \mathcal{C} \wedge \neg e; \Gamma \vdash R \triangleright \Delta}{\mathcal{C}; \Gamma \vdash \mathsf{if}\ e\ \mathsf{then}\ Q\ \mathsf{else}\ R \triangleright \Delta}$$

By inductive hypothesis

$$\begin{array}{l} (\mathcal{C} \wedge e)[\mathtt{n}/u]; \Gamma \vdash Q[\mathtt{n}/u] \triangleright \Delta[\mathtt{n}/u] \text{ and} \\ (\mathcal{C} \wedge \neg e)[\mathtt{n}/u]; \Gamma \vdash R[\mathtt{n}/u] \triangleright \Delta[\mathtt{n}/u] \end{array} \tag{E.6}$$

By applying (E.6) as a premise for [IF] we obtain

$$\mathcal{C}[\mathtt{n}/u]; \Gamma \vdash \mathsf{if}\ e[\mathtt{n}/u]\ \mathsf{then}\ Q[\mathtt{n}/u]\ \mathsf{else}\ R[\mathtt{n}/u] \triangleright \Delta[\mathtt{n}/u]$$

where the substituted endpoint assertion is well-asserted by inductive hypothesis.

Session Request. If $P = \overline{a}_{[2..\mathtt{n}]}(\tilde{s}).P'$ by [MCAST]

$$\frac{\Gamma \vdash a \colon \mathcal{G} \quad \mathcal{C}; \Gamma \vdash P \triangleright \Delta, \tilde{s} \colon (\mathcal{G} \upharpoonright \mathtt{1}) \,@\, \mathtt{1}}{\mathcal{C}; \Gamma \vdash \overline{a}_{[2..\mathtt{n}]}(\tilde{s}).P' \triangleright \Delta}$$

By inductive hypothesis

$$\mathcal{C}[\mathtt{n}/u]; \Gamma \vdash P'[\mathtt{n}/u] \triangleright \Delta[\mathtt{n}/u], \tilde{s} \colon (\mathcal{G} \upharpoonright \mathtt{1}) \,@\, \mathtt{1}[\mathtt{n}/u] \tag{E.7}$$

Also $\Gamma \vdash a \colon \mathcal{G}[\mathtt{n}/u]$ (trivially since $\mathcal{G}$ does not have free variables). By applying (E.7) as a premise for [MCAST] we obtain

$$\Gamma \vdash \overline{a}_{[2..\mathtt{n}]}(\tilde{s}).P'[\mathtt{n}/u] \triangleright \Delta[\mathtt{n}/u]$$

where the substituted endpoint assertion is well-asserted by inductive hypothesis.

Session Acceptance. The case for [MACC] is similar to the case for [MCAST].

Recursive Invocation. If $P = X\langle \tilde{e}\tilde{s}_1..\tilde{s}_n\rangle$ by [VAR]

$$\frac{\mathcal{T}_1[\tilde{e}/\tilde{v}]...\mathcal{T}_n[\tilde{e}/\tilde{v}] \text{ well-asserted}}{\begin{array}{c}\mathcal{C};\Gamma, u : S', X : (\tilde{v} : \tilde{S})\mathcal{T}_1 @ \mathtt{p}_1..\mathcal{T}_n @ \mathtt{p}_\mathtt{n} \vdash X\langle \tilde{e}\tilde{s}_1..\tilde{s}_n\rangle \\ \triangleright \Delta, \tilde{s}_1 : \mathcal{T}_1[\tilde{e}/\tilde{v}] @ \mathtt{p}_1, .., \tilde{s}_n : \mathcal{T}_n[\tilde{e}/\tilde{v}] @ \mathtt{p}_\mathtt{n}\end{array}}$$

Without loss of generality we assume $u \notin \tilde{v}$. Since $\mathcal{T}_1[\tilde{e}/\tilde{v}]...\mathcal{T}_n[\tilde{e}/\tilde{v}]$ are well-typed under $\mathsf{erase}(\Gamma), u : S', \tilde{v} : \tilde{S}$ and $\mathtt{n} : S'$ then $\mathcal{T}_1[\tilde{e}/\tilde{v}][\mathtt{n}/u]...\mathcal{T}_n[\tilde{e}/\tilde{v}][\mathtt{n}/u]$ are also well-typed. $\mathcal{T}_1[\tilde{e}/\tilde{v}][\mathtt{n}/u]...\mathcal{T}_n[\tilde{e}/\tilde{v}][\mathtt{n}/u]$ are well-asserted by Lemma Appendix E.3.

By applying $\mathcal{T}_1[\tilde{e}/\tilde{v}][\mathtt{n}/u]...\mathcal{T}_n[\tilde{e}/\tilde{v}][\mathtt{n}/u]$ as a premise of [VAR] we obtain

$$\begin{array}{c}\mathcal{C}[\mathtt{n}/u];\Gamma, X : (\tilde{v} : \tilde{S})\mathcal{T}_1[\mathtt{n}/u] @ \mathtt{p}_1 \ldots \mathcal{T}_n[\mathtt{n}/u] @ \mathtt{p}_\mathtt{n} \vdash X\langle \tilde{e}\tilde{s}_1..\tilde{s}_n\rangle[\mathtt{n}/u] \\ \triangleright \tilde{s}_1 : \mathcal{T}_1[\tilde{e}/\tilde{v}][\mathtt{n}/u] @ \mathtt{p}_1..\tilde{s}_n : \mathcal{T}_n[\tilde{e}/\tilde{v}][\mathtt{n}/u] @ \mathtt{p}_\mathtt{n}\end{array}$$

Remaining Cases. The cases for [CONC], [IDLE], [HIDE], [CONSEQ] and [REC] and [CRES] are straightforward. Also [QNIL], [QVAL] and [QSEL] are immediate since there are no free variables in P.

Lemma Appendix E.4 (Evaluation Lemma). *If $\mathcal{C};\Gamma \vdash P(\tilde{e}) \triangleright \Delta(\tilde{e})$ and $\tilde{e} \downarrow \tilde{\mathtt{n}}$ then we have $\mathcal{C};\Gamma \vdash P[\tilde{\mathtt{n}}/\tilde{e}] \triangleright \Delta[\tilde{\mathtt{n}}/\tilde{e}]$.*

PROOF:
PROOF: The proof is by rule induction on the validation rules (Figures 6 and C.11). We proceed by case analysis. By decidability of underlying logic, we can write $A[\tilde{e}/\tilde{v}] \downarrow \mathsf{true}$ when a closed formula $A[\tilde{e}/\tilde{v}]$ evaluates to true. Note that if we further have $\tilde{e} \downarrow \tilde{\mathtt{n}}$ then we have $A[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{true}$.

Values sending. If $P(\tilde{e}) = s_k!\langle\tilde{e}\rangle(\tilde{v})\{A\};P'$ then $P(\tilde{\mathtt{n}}) = s_k!\langle\tilde{\mathtt{n}}\rangle(\tilde{v})\{A\};P'$ and

$$\Delta(\tilde{e}) = \Delta', \tilde{s} \colon k!(\tilde{v} : \tilde{S})\{A\};\mathcal{T} @ \mathtt{p}$$

with and $\mathcal{C} \supset A[\tilde{e}/\tilde{v}]$. Notice that $\mathcal{C} \supset A[\tilde{e}/\tilde{v}]$ is equivalent to

$$\mathcal{C} \supset A[\tilde{\mathtt{n}}/\tilde{v}] \tag{E.8}$$

By inductive hypothesis

$$\mathcal{C};\Gamma \vdash P'[\tilde{\mathtt{n}}/\tilde{e}] \triangleright \Delta'[\tilde{\mathtt{n}}/\tilde{e}], \tilde{s} \colon \mathcal{T}[\tilde{\mathtt{n}}/\tilde{e}] @ \mathtt{p} \tag{E.9}$$

By applying (E.8) and (E.9) to the validation rule [SEND] the lemma holds for this case.

Recursion Invocation. If $P(\tilde{e}) = X\langle \tilde{e}\tilde{s}_1..\tilde{s}_n\rangle$ (since $P(\tilde{e})$ is well-formed against Δ by hypothesis) then $P(\tilde{\mathtt{n}}) = X\langle \tilde{\mathtt{n}}\tilde{s}_1..\tilde{s}_n\rangle$. Since $\mathcal{C} \supset A[\tilde{e}/\tilde{v}]$ is equivalent to $\mathcal{C} \supset A[\tilde{\mathtt{n}}/\tilde{v}]$ then $P(\tilde{\mathtt{n}})$ is well-formed against $\Delta[\tilde{\mathtt{n}}/\tilde{v}]$ by rule [VAR].

Conditional. $P(e) = \mathsf{if}\ e\ \mathsf{then}\ Q\ \mathsf{else}\ R$ the property holds by induction since $\mathcal{C} \wedge e \downarrow \mathsf{true}$ is equivalent to $\mathcal{C} \wedge n \downarrow \mathsf{true}$.

Remaining cases. The remaining cases are straightforward by induction.

Appendix F. Subject Reduction (Proofs of Section 7)

Subject reduction relies on the following auxiliary lemmas.

Lemma Appendix F.1. *Suppose $\mathcal{C};\Gamma \vdash P|Q \triangleright \Delta$ (i.e., there is a derivation). There exists another derivation with the same conclusion and with the same or lesser length than the original derivation such that the last rule applied is* [Conc].

PROOF: By the shape of the validation rules, the last rule applied to derive this judgement can only be either [Conc] or [Conseq]. In the latter case, since $\ni$ is only applied point-wise, and this does not affect the composability by $\circ$ we can first apply the same refinement [Conseq] point-wise then finally apply [Conc], to obtain the same final conclusion. □

In this subsection, for a convenience of the case analysis, we explicitly write $P \xrightarrow{\tau_{\mathtt{s}}} P'$ if $P \xrightarrow{\tau} P'$ is derived by the reduction rules at free session channels. The current definition of τ-action as well as $\tau_{\mathtt{s}}$-action is not based on compatible visible actions but is defined from reduction. The following lemma shows that, in spite of this, the τ-action is indeed derivable from complementary visible actions except for initiation and conditionals. Below, $C[\_]$ denotes a reduction context. Notice that, in cases (4) and (5), P' could be errT. Assume below all transitions are typed under the implicit typing.

Lemma Appendix F.2. *If $P \to P'$ then one of the following cases hold:*

1. $P \equiv C[\mathsf{if}\ e\ \mathsf{then}\ Q_1\ \mathsf{else}\ Q_2]$ *s.t.* $P' \equiv C[Q_1]$ *(if* $e \downarrow \mathsf{true}$*) or* $P' \equiv C[Q_2]$ *(if* $e \downarrow \mathsf{false}$*)*
2. $P \equiv C[P_1 \mid \ldots \mid P_n]$ *s.t.* $P_1 \xrightarrow{a[2..n](\tilde{s})} P_1'$ *and* $(P_i \xrightarrow{a[i](\tilde{s})} P_i')_{2\le i\le n}$ *with* $P' \equiv C[(\nu \tilde{s})(P_1' \mid \ldots \mid P_n')]$
3. $P \equiv C[Q|s:\tilde{h}]$ *s.t.* $Q \xrightarrow{s!\tilde{\mathtt{n}}} Q'$ *and* $P' \equiv C[Q'|s:\tilde{h}\cdot\tilde{\mathtt{n}}]$
4. $P \equiv C[Q|s:\tilde{h}\cdot\tilde{\mathtt{n}}]$ *s.t.* $Q \xrightarrow{s?\tilde{\mathtt{n}}} Q'$ *and* $P' \equiv C[Q'|s:\tilde{h}]$
5. $P \equiv C[Q|s:\tilde{h}]$ *s.t.* $Q \xrightarrow{s\lhd l} Q'$ *and* $P' \equiv C[Q'|s:\tilde{h}\cdot l]$
6. $P \equiv C[Q|s:\tilde{h}\cdot l]$ *s.t.* $Q \xrightarrow{s\rhd l} Q'$ *and* $P' \equiv C[Q'|s:\tilde{h}]$
7. $P \equiv C[Q \mid s:\tilde{h}]$ *s.t.* $Q \xrightarrow{\tau} \mathsf{errH}$ *and* $P' \equiv C[\mathsf{errH} \mid s:\tilde{h}]$

PROOF: Immediate from the corresponding reduction rules. □

By Lemma Appendix F.2 we can reduce the reasoning on each communication-induced reduction to the corresponding visible action combined with the accompanying transformation of a queue. The difference cases are analysed below, after an auxiliary lemma that rules out case (7) in case of validated processes.

Lemma Appendix F.3. *Suppose $\mathcal{C};\Gamma \vdash P \triangleright \Delta$. Then $P \not\xrightarrow{\tau}$ errH.*

PROOF: Assume that both $\mathcal{C};\Gamma \vdash P \triangleright \Delta$ and $P \xrightarrow{\tau} \mathsf{errH}$. By inspection of the transition rules (cf. Figure 7), P can move to errH only when the last rule applied is either [SendErr] or [LabelErr]. In case of [SendErr], we let $P = C[s_k!\langle\tilde{\mathtt{n}}\rangle(v)\{A\}.P']$ and $A[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{false}$ for some $C[\_]$, s_k, $\tilde{v}$, $\tilde{\mathtt{n}}$, A, and P'. Then by inspecting the validation rules, the derivation of $C[s_k!\langle\tilde{\mathtt{n}}\rangle(v)\{A\}.P']$ is done applying rule [Snd] to the following (after zero or applications of [Conc], [Hide], and [Cres])

$$\mathcal{C};\Gamma' \vdash s_k!\langle\tilde{e}\rangle(v)\{A\}.P' \triangleright \Delta', \tilde{s}:k!(\tilde{v})\{A\}.\mathcal{T}'@\mathtt{p}$$

for some Γ', Δ', $\mathtt{p}$ and $\mathcal{T}'$ with $\mathcal{C}' \supset A[\tilde{e}/\tilde{v}]$. Notice that the assertion environment $\mathcal{C}$ does not change after the application of rules [Conc], [Hide], and [Cres]. Since by Proposition Appendix C.4, P does not contain errT nor errH, then $[\tilde{\mathtt{n}}/\tilde{v}]$ is a substitution consistent with $\mathcal{C}$ thus

$$\mathcal{C} \supset A[\tilde{e}/\tilde{v}]$$

which contradicts $A[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{false}$. The case of [LabelErr] is similar.

Lemma Appendix F.4. *Assume all transitions are well-typed, and Δ be the derivation via reduction of a singleton assignment.*

1. *If $P \xrightarrow{\overline{a}[2..n](\tilde{s})} P'$ and $\Gamma \vdash P \triangleright \Delta$ such that $\Gamma(a) = \mathcal{G}$ then $\Gamma \vdash P' \triangleright \Delta, \tilde{s} : \mathcal{G} \upharpoonright 1 @ 1$*
2. *If $P \xrightarrow{a[\mathtt{p}](\tilde{s})} P'$ and $\Gamma \vdash P \triangleright \Delta$ such that $\Gamma(a) = \mathcal{G}$ then $\Gamma \vdash P' \triangleright \Delta, \tilde{s} : \mathcal{G} \upharpoonright \mathtt{p} @ \mathtt{p}$*
3. *If $P \xrightarrow{s!\tilde{\mathtt{n}}} P'$ and $\Gamma \vdash P | s : \tilde{h} \triangleright \Delta$ then $\Gamma \vdash P' | s : \tilde{h} \cdot \tilde{\mathtt{n}} \triangleright \Delta'$ such that $\Delta \to \Delta'$*
4. *If $P \xrightarrow{s \lhd l} P'$ and $\Gamma \vdash P | s : \tilde{h} \triangleright \Delta$ then $\Gamma \vdash P' | s : \tilde{h} \cdot l \triangleright \Delta'$ such that $\Delta \to \Delta'$*
5. *If $P \xrightarrow{s?\tilde{\mathtt{n}}} P'$ and $\Gamma \vdash P | s : \tilde{h} \cdot \tilde{\mathtt{n}} \triangleright \Delta$ then $\Gamma \vdash P' | s : \tilde{h} \triangleright \Delta'$ such that $\Delta \to \Delta'$*
6. *If $P \xrightarrow{s \rhd l} P'$ and $\Gamma \vdash P | s : \tilde{h} \cdot l \triangleright \Delta$ then $\Gamma \vdash P' | s : \tilde{h} \triangleright \Delta'$ such that $\Delta \to \Delta'$*

Further in the cases of (3,4,5,6) above, Δ and Δ' only differ in the assignment at $\tilde{s}$ such that $s \in \tilde{s}$.

Remark. In clause (3) we do not include the case of bound outputs since we do not need them in Lemma Appendix F.2 (due to the use of contexts).

Proof: (1) and (2) are immediate. Below we show the cases (3) and (5) since (4) (resp. (6)) is an easy version of (3) (resp. (5)).

Case (3) Suppose we have $\Gamma \vdash P | s : \tilde{h} \triangleright \Delta$. By Lemma Appendix F.1, we safely assume the last rule applied is [Conc]. Thus we can assume for some Δ_0 and Δ_1 that $\Gamma \vdash s : \tilde{h} \triangleright \Delta_1$ with $\Delta_0 \circ \Delta_1 = \Delta$, and

$$\Gamma \vdash P \triangleright \Delta_0 \tag{F.1}$$

Now consider the transition $P \xrightarrow{s!\tilde{\mathtt{n}}} P'$, by (F.1) we observe Δ_0 has the shape

$$\Delta_0 = \tilde{s} : \mathcal{H}[k!(\tilde{v})\{A\}; \mathcal{T} @ \mathtt{p}], \Delta_{00}$$

for some $\mathtt{p}$ and with $s = s_k$; and that P' can be typed by Δ_0' such that:

$$\Delta_0' = \tilde{s} : \mathcal{H}[\mathcal{T}[\tilde{\mathtt{n}}/\tilde{v}] @ \mathtt{p}], \Delta_{00} \tag{F.2}$$

Now the assertion Δ_1 for the queue has the shape, omitting the vacuous "end": $\Delta_1 = \tilde{s} : \mathcal{H}[\mathcal{M} @ \mathtt{p}]$ hence the addition of the values to this queue, $s : \tilde{h} \cdot \tilde{\mathtt{n}}$, must have the endpoint assertion:

$$\Delta_1' = \tilde{s} : \mathcal{H}[k!\langle \tilde{\mathtt{n}} \rangle; \mathcal{M} @ \mathtt{p}] \tag{F.3}$$

Setting $\Delta' = \Delta_0' \circ \Delta_1'$, we know $\Gamma \vdash P' | s : \tilde{h} \cdot \tilde{\mathtt{n}} \triangleright \Delta'$. By (F.2) and (F.3) and the type composition $\circ$ and the reduction $\to$, we obtain

$$\begin{array}{rcl} \Delta_0' \circ \Delta_1' & = & \tilde{s} : \mathcal{H}[k!\langle \tilde{\mathtt{n}} \rangle; \mathcal{T}[\tilde{\mathtt{n}}/\tilde{v}] @ \mathtt{p}],\ \Delta_{00},\ \Delta_1 \\ & \leftarrow & \Delta_0, \Delta_1 \end{array}$$

That is we have $\Delta \to \Delta'$, and the only change is at the assertion assignment at s, as required.
Case (5). Suppose we have $\Gamma \vdash P|s:\tilde{h}\cdot\tilde{\mathtt{n}} \triangleright \Delta$. Again by Lemma Appendix F.1, we safely assume the last rule applied is [Conc]. Thus we can assume, for some Δ_0 and Δ_1: $\Gamma \vdash s:\tilde{h}\cdot\tilde{\mathtt{n}} \triangleright \Delta_1$, $\Delta_0 \circ \Delta_1 = \Delta$, and

$$\Gamma \vdash P \triangleright \Delta_0 \tag{F.4}$$

Now consider the transition

$$P \xrightarrow{s?\tilde{\mathtt{n}}} P' \tag{F.5}$$

As before, we can infer, from (F.4) and (F.5) the shape of Δ_0 as follows, with $s = s_k$:

$$\Delta_0 = \tilde{s}\colon \mathcal{H}[k?(\tilde{v})\{A\}; \mathcal{T}@\mathtt{p}], \Delta_{00}$$

for some $\mathtt{p}$; and that P' can be validated against Δ'_0 given as

$$\Delta'_0 = \tilde{s}\colon \mathcal{H}[\mathcal{T}[\tilde{\mathtt{n}}/\tilde{v}]@\mathtt{p}], \Delta_{00} \tag{F.6}$$

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

$$\Delta_1 = \tilde{s}\colon \mathcal{H}[k!\langle\tilde{\mathtt{n}}\rangle; \mathcal{M}@\mathtt{p}] \tag{F.7}$$

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

$$\Delta'_1 = \tilde{s}\colon \mathcal{H}[\mathcal{M}@\mathtt{p}] \tag{F.8}$$

Note this is symmetric to the case (1) above. As before, setting $\Delta' = \Delta'_0 \circ \Delta'_1$, we know: $\Gamma \vdash P'|s:\tilde{h} \triangleright \Delta'$. By (F.6) and (F.8) and the type composition $\circ$ and the type reduction $\to$, we obtain

$$\begin{array}{rcl} \Delta'_0 \circ \Delta'_1 & = & \tilde{s}\colon \mathcal{H}[\mathcal{T}[\tilde{\mathtt{n}}/\tilde{v}]@\mathtt{p}],\ \Delta_{00},\ \Delta_1 \\ & \leftarrow & \Delta_0, \Delta_1 \end{array}$$

That is we have $\Delta \to \Delta'$, and the only change from Δ to Δ' is at the type assignment at s, as required.

Appendix F.1. Proof of Lemma 7.1

Suppose $\Gamma \vdash P \triangleright \Delta$.

1. If $P \xrightarrow{\tau} P'$ then $\Gamma \vdash P' \triangleright \Delta$ again.
2. If $P \xrightarrow{\tau_s} P'$ then $\Gamma \vdash P' \triangleright \Delta'$ such that $\Delta \to \Delta'$.

Proof: Assume

$$\mathsf{true};\Gamma_0 \vdash P \triangleright \Delta_0 \quad \text{and} \quad P \xrightarrow{\tau} P' \tag{F.9}$$

Each of the six cases in Lemmas Appendix F.2 are possible, which we inspect one by one. Below let $C[\_]$ be an appropriate reduction context.
Case (1): Conditional. By Lemmas Appendix F.2 (1) assume $P = C[R]$ where $R = \mathsf{if}\ e\ \mathsf{then}\ Q_1\ \mathsf{else}\ Q_2$ such that if $e \downarrow \mathsf{true}$ then $P' = Q_1$. Since $C[\_]$ is a reduction context we know R is closed. Therefore we can safely set $\mathsf{true};\Gamma \vdash R \triangleright \Delta$. By Lemma Appendix F.1 we postpone refinements. We can assume R is inferred by [If] of Figure 6. Hence we have $\mathsf{true} \wedge e;\Gamma \vdash Q_1 \triangleright \Delta$. By [Conseq] we get $\mathsf{true};\Gamma \vdash Q_1 \triangleright \Delta$ as required. Dually for the case of $e \downarrow \mathsf{false}$.

Case (2): Link. By Lemma Appendix F.2 (2) we set $P = C[R]$ where $R = P_1 \mid \ldots \mid P_n$ with their initialization actions compensating each other, as given in Lemma Appendix F.2 (2), i.e.

$$P_1 \overset{\overline{a}[2..n](\tilde{s})}{\rightarrow} P_1' \quad \text{and} \quad P_i \overset{a[i](\tilde{s})}{\rightarrow} P_i' \quad (2 \leq i \leq n)$$

As before, we can safely set $\mathsf{true};\Gamma \vdash R \triangleright \Delta$. By Lemma Appendix F.1 we can assume R is inferred by [MACC] or [MCAST] of Figure 6 by consecutive applications of [CONC], hence we safely assume: $\mathsf{true};\Gamma \vdash P_i \triangleright \Delta_i$ such that $\Delta_1 \circ \ldots \circ \Delta_n = \Delta$. By Lemma Appendix F.4 (1) and (2), we have, with $\Gamma(a) = \mathcal{G}$,

$$\mathsf{true};\Gamma \vdash P_i' \triangleright \Delta_i, \tilde{s} : (\mathcal{G} \upharpoonright i)@i$$

hence

$$\mathsf{true};\Gamma \vdash P_1 | .. | P_n \triangleright \Delta, \tilde{s} : \{(\mathcal{G} \upharpoonright i)@i\}_{1 \leq i \leq n}$$

Since $\{(\mathcal{G} \upharpoonright i)@i\}_{1 \leq i \leq n}$ is obviously coherent, we have $\mathsf{true};\Gamma \vdash (\nu \tilde{s})(P_1 | .. | P_n) \triangleright \Delta$ as required.
Case (3): Send. By Lemmas Appendix F.2 (3) we set $P = C[Q | s : \tilde{h}]$ with

$$Q \overset{s!\tilde{\mathfrak{n}}}{\rightarrow} Q' \tag{F.10}$$

As above we can safely set

$$\mathsf{true};\Gamma \vdash Q | s : \tilde{h} \triangleright \Delta \tag{F.11}$$

By Lemmas Appendix F.4 (3), (F.10) and (F.11), we infer: $\mathsf{true};\Gamma \vdash Q' | s : \tilde{h} \cdot \tilde{\mathfrak{n}} \triangleright \Delta'$ such that $\Delta \rightarrow \Delta'$ where the only change is at $\tilde{s}$ which contains s. Since P reduces to P' by τ-transition rather than $\tau_{\mathtt{s}}$-transition, we know that this $\tilde{s}$ in R are hidden in P. Assume therefore, without loss of generality:

$$\begin{gathered} P \equiv C'[(\nu \tilde{s})(Q | s : \tilde{h} | R)] \\ \Gamma_1 \vdash Q | s : \tilde{h} | R \triangleright \Delta_1 \\ \Delta' \text{ coherent and } \Delta_1 = \Delta \circ \Delta_{01} \end{gathered}$$

By Lemma Appendix G.1 and noting $\Delta_1 \rightarrow \Delta' \circ \Delta_{01}$ we know $\Delta_1 = \Delta' \circ \Delta_{01}$ is also coherent, hence as required.
Case (4): Receive. By Lemmas Appendix F.2 (4) we set $P = C[Q | s : \tilde{h} \cdot \tilde{\mathfrak{n}}]$ with

$$Q \overset{s?\tilde{\mathfrak{n}}}{\rightarrow} Q' \tag{F.12}$$

As above we can safely set

$$\mathsf{true};\Gamma \vdash Q | s : \tilde{h} \cdot \tilde{\mathfrak{n}} \triangleright \Delta \tag{F.13}$$

As before, by Lemmas Appendix F.4 (4), (F.12) and (F.13), we get: $\mathsf{true};\Gamma \vdash Q' | s : \tilde{h} \triangleright \Delta'$ such that $\Delta \rightarrow \Delta'$ where the only change is at $\tilde{s}$ which contains s. Then we can set, without loss of generality:

$$\begin{gathered} P \equiv C'[(\nu \tilde{s})(Q | s : \tilde{h} | R)] \\ \Gamma_1 \vdash Q | s : \tilde{h} | R \triangleright \Delta_1 \\ \Delta' \text{ coherent and } \Delta_1 = \Delta \circ \Delta_{01} \end{gathered}$$

As before, by Lemma Appendix G.1 and $\Delta_1 \rightarrow \Delta' \circ \Delta_{01}$ we know $\Delta_1 = \Delta' \circ \Delta_{01}$ is also coherent, hence done.

Case (5): Select. The argument exactly follows case (3) above using Lemma Appendix F.2 (5) and Lemma Appendix F.4 (5) instead of Lemma Appendix F.2 (3) and Lemma Appendix F.4 (3), respectively.
Case (6): Branch. The argument exactly follows case (4) above except using Lemma Appendix F.2 (6) and Lemma Appendix F.4 (6) instead of Lemma Appendix F.2 (4) and Lemma Appendix F.4 (4), respectively.

Finally the τ_s-reduction,

$$\mathsf{true};\Gamma_0 \vdash P \triangleright \Delta_0 \quad \text{and} \quad P \xrightarrow{\tau_s} P'$$

rather than (F.9), precisely follow the same reasoning as given in the cases of 3,4,5,6 above, excepting we do not have to hide $\tilde{s}$.

Appendix F.2. Proof of Lemma Appendix F.5

Subject congruence relies on the following auxiliary lemma to handle the case structural equivalence between a process and its unfolding.

Lemma Appendix F.5. *Let* $\Gamma' = \Gamma, X : (\tilde{v} : \tilde{S})\mathcal{T}_1 @ \mathtt{p}_1..\mathcal{T}_n @ \mathtt{p}_n$ *and assume*

$$\mathcal{C};\Gamma' \vdash P' \triangleright \Delta' \text{ and } \mathcal{C};\Gamma' \vdash P \triangleright \tilde{s}_1 : \mathcal{T}_1 @ \mathtt{p}_1..\tilde{s}_n : \mathcal{T}_n @ \mathtt{p}_n$$

Then $\mathcal{C};\Gamma \vdash P'[\mu X(\tilde{v}\tilde{s}).P/X] \triangleright \Delta'$.

PROOF: Recall that $\Gamma' = \Gamma, X : (\tilde{v} : \tilde{S})\mathcal{T}_1 @ \mathtt{p}_1..\mathcal{T}_n @ \mathtt{p}_n$. Furthermore,

$$\mathcal{C};\Gamma' \vdash P' \triangleright \Delta' \tag{F.14}$$

where we let $\Delta' = \tilde{s}_1 : \mathcal{T}_1' @ \mathtt{p}_1..\tilde{s}_n : \mathcal{T}_n' @ \mathtt{p}_n$, and

$$\mathcal{C};\Gamma' \vdash P \triangleright \tilde{s}_1 : \mathcal{T}_1 @ \mathtt{p}_1..\tilde{s}_n : \mathcal{T}_n @ \mathtt{p}_n \tag{F.15}$$

where we denote $\tilde{s}_1 : \mathcal{T}_1 @ \mathtt{p}_1..\tilde{s}_n : \mathcal{T}_n @ \mathtt{p}_n$ with Δ. The thesis is

$$\mathcal{C};\Gamma \vdash P'[\mu X(\tilde{v}\tilde{s}).P/X] \triangleright \Delta' \tag{F.16}$$

The proof is by induction on the depth of the derivation of (F.14).

Base Cases. The validation terminates by applying one rule: either [IDLE] or [VAR].
***Case** [IDLE].* In this case $P' = \mathbf{0}$ and Δ' is end only. Since $P'[\mu X(\tilde{v}\tilde{s}).P/X] = \mathbf{0}$ we can validate it against Δ' via [IDLE], obtaining (F.16).
***Case** [VAR].* In this case either $P' = X'\langle\tilde{e}\tilde{s}\rangle$ or $P' = X\langle\tilde{e}\tilde{s}\rangle$ for some $\tilde{e}$. In the former case, $P'[\mu X(\tilde{v}\tilde{s}).P/X] = P'$ and (F.16) follows straightforwardly from (F.14). In the latter case, by (F.14) (applying [VAR], with $\tilde{s} = \tilde{s}_1,..,\tilde{s}_n$):

$$\frac{\mathcal{T}_1[\tilde{e}/\tilde{v}],..,\mathcal{T}_n[\tilde{e}/\tilde{v}] \text{ well-asserted and well-typed under } \Gamma, \tilde{v} : \tilde{S}}{\mathcal{C};\Gamma' \vdash X\langle\tilde{e}\tilde{s}_1..\tilde{s}_n\rangle \triangleright \tilde{s}_1 : \mathcal{T}_1[\tilde{e}/\tilde{v}] @ \mathtt{p}_1,..,\tilde{s}_n : \mathcal{T}_n[\tilde{e}/\tilde{v}] @ \mathtt{p}_n}$$

thus

$$\mathcal{T}_i' = \mathcal{T}_i[\tilde{e}/\tilde{v}] \quad i = 1,..,n \tag{F.17}$$

Substituting, $P'[\mu X(\tilde{v}\tilde{s}).P/X] = \mu X\langle\tilde{e}\tilde{s}\rangle(\tilde{v}\tilde{s}).P$. By applying (F.15) as a premise for rule [REC] we obtain

$$\frac{\mathcal{C};\Gamma' \vdash P \triangleright \tilde{s_1} : \mathcal{T}_1 @ \mathtt{p}_1..\tilde{s_n} : \mathcal{T}_n @ \mathtt{p}_n}{\mathcal{C};\Gamma \vdash \mu X\langle\tilde{e}\tilde{s}_1..\tilde{s}_n\rangle(\tilde{v}\tilde{s}_1..\tilde{s}_n).P\tilde{s_1} : \mathcal{T}_1[\tilde{e}/\tilde{v}] @ \mathtt{p}_1..\tilde{s_n} : \mathcal{T}_n[\tilde{e}/\tilde{v}] @ \mathtt{p}_n}$$

from which we obtain (F.16) observing that by (F.17) $\tilde{s_1} : \mathcal{T}_1[\tilde{e}/\tilde{v}] @ \mathtt{p}_1..\tilde{s_n} : \mathcal{T}_n[\tilde{e}/\tilde{v}] @ \mathtt{p}_n = \Delta'$.

Inductive Cases. We show only the most relevant cases, i.e., those where the last rule applied is [Send], and [Rec]. The case for [Rcv] is similar to [Send] considering that the substitution of X does not affect the assertion environment $\mathcal{C}$. The cases for [Sel] and [Bra] are similar to [Send] and [Rcv], respectively.

Case *[Send].* In this case $P' = s_k!\langle\tilde{e}'\rangle(\tilde{v}')\{A\};P''$, $P = s_k!\langle\tilde{e}'\rangle(\tilde{v}')\{A\};P''[\mu X(\tilde{v}\tilde{s}).P/X]$, and $\Delta' = \Delta'', \tilde{s}_k : k!(\tilde{v}' : \tilde{S}')\{A\};\mathcal{T} \mathbin{@} \mathtt{p}_k$. From (F.14), which is obtained using rule [Send] we have premises

$$\mathcal{C} \supset A[\tilde{e}'/\tilde{v}'] \quad \text{and} \quad \mathcal{C};\Gamma' \vdash P''[\tilde{e}'/\tilde{v}'] \triangleright (\Delta'', \tilde{s}_k : \mathcal{T} \mathbin{@} \mathtt{p}_k)[\tilde{e}'/\tilde{v}'] \tag{F.18}$$

From (F.18), by induction, follows

$$\mathcal{C};\Gamma' \vdash P''[\tilde{e}'/\tilde{v}'][\mu X(\tilde{v}\tilde{s}).P/X]] \triangleright (\Delta'', \tilde{s}_k : \mathcal{T} \mathbin{@} \mathtt{p}_k)[\tilde{e}'/\tilde{v}'] \tag{F.19}$$

By using (F.19) and $\mathcal{C} \supset A[\tilde{e}'/\tilde{v}']$ (from F.18) as premises for [Send] we obtain (F.16) for this case:

$$\mathcal{C};\Gamma' \vdash s_k!\langle\tilde{e}'\rangle(\tilde{v}')\{A\};P''[\mu X(\tilde{v}\tilde{s}).P/X] \triangleright \Delta'$$

Case *[Rec].* Since we use Barendregts convention and $X \in dom(\Gamma')$, if rule [Rec] is used then $P' = \mu X'\langle\tilde{e}\tilde{s}\rangle(\tilde{v}'\tilde{s}).P'''$ with $X' \neq X$ and $P'[\mu X(\tilde{v}\tilde{s}).P/X] = \mu X'\langle\tilde{e}\tilde{s}\rangle(\tilde{v}'\tilde{s}).P'''[\mu X(\tilde{v}\tilde{s}).P/X]$. By (F.14)

$$\frac{\mathcal{C};\Gamma, X' : (\tilde{v}' : \tilde{S}')\mathcal{T}_1 \mathbin{@} \mathtt{p}_1..\mathcal{T}_n \mathbin{@} \mathtt{p}_n \vdash P''' \triangleright \tilde{s_1} : \mathcal{T}_1 \mathbin{@} \mathtt{p}_1..\tilde{s_n} : \mathcal{T}_n \mathbin{@} \mathtt{p}_n}{\mathcal{C};\Gamma \vdash \mu X'\langle\tilde{e}\tilde{s}\rangle(\tilde{v}'\tilde{s}).P''' \triangleright \tilde{s_1} : \mathcal{T}_1[\tilde{e}'/\tilde{v}'] \mathbin{@} \mathtt{p}_1..\tilde{s_n} : \mathcal{T}_n[\tilde{e}'/\tilde{v}'] \mathbin{@} \mathtt{p}_n} \tag{F.20}$$

As in the previous cases, we can use induction on the premise of F.20 obtaining

$$\mathcal{C};\Gamma, X' : (\tilde{v}' : \tilde{S}')\mathcal{T}_1 \mathbin{@} \mathtt{p}_1..\mathcal{T}_n \mathbin{@} \mathtt{p}_n \vdash P'''[\mu X(\tilde{v}\tilde{s}).P/X] \triangleright \tilde{s_1} : \mathcal{T}_1 \mathbin{@} \mathtt{p}_1..\tilde{s_n} : \mathcal{T}_n \mathbin{@} \mathtt{p}_n$$

which can be then used as a premise for [Rec] to obtain F.16 for this case:

$$\mathcal{C};\Gamma \vdash \mu X'\langle\tilde{e}\tilde{s}\rangle(\tilde{v}'\tilde{s}).P'''[\mu X(\tilde{v}\tilde{s}).P/X] \triangleright \tilde{s_1} : \mathcal{T}_1[\tilde{e}'/\tilde{v}'] \mathbin{@} \mathtt{p}_1..\tilde{s_n} : \mathcal{T}_n[\tilde{e}'/\tilde{v}'] \mathbin{@} \mathtt{p}_n$$

Appendix F.3. Proof of Lemma 7.3 (Subject Congruence)

If $\Gamma \vdash P_1 \triangleright \Delta$ and $P_1 \equiv P_2$ then $\Gamma \vdash P_2 \triangleright \Delta$

Proof: We proceed by case analysis considering, for each rule for structural equivalence, the two symmetric cases:

Case $P \equiv P \mid \mathbf{0}$. If $P_2 = P_1 \mid \mathbf{0}$, by hypothesis

$$\Gamma \vdash P_1 \triangleright \Delta \tag{F.21}$$

Assume $\Delta' = t : \mathsf{end}$ with $t \notin \mathrm{dom}(\Delta)$, then $\Gamma \vdash \mathbf{0} \triangleright \Delta'$ by rule [Idle]. By [Conc] with premises (F.21) and $\mathcal{C};\Gamma \vdash \mathbf{0} \triangleright \Delta'$:

$$\Gamma \vdash P_1 \mid \mathbf{0} \triangleright \Delta, \Delta'$$

By rule [Conseq] and since $\Delta, \Delta' \ni \Delta$ then $\Gamma \vdash P_1 \mid \mathbf{0} \triangleright \Delta$ as required.

If $P_1 = P_2 \mid \mathbf{0}$ then $\mathcal{C};\Gamma \vdash P_1 \triangleright \Delta$ follows from the first premise of [conc] applied to P_2.

Case $P \mid Q \equiv Q \mid P$. Immediate, observing that the order of the premises of [Conc] is immaterial.

Case $(P \mid Q) \mid R \equiv P \mid (Q \mid R)$**.** This case follows from the fact that the both the derivation for $(P \mid Q) \mid R$ and the one for $P \mid (Q \mid R)$ depend, after two applications of [Conc], to premises $\Gamma \vdash P \triangleright \Delta_P$, $\mathcal{C}; \Gamma \vdash Q \triangleright \Delta_Q$ and $\Gamma \vdash R \triangleright \Delta_R$ for some Δ_P, Δ_Q and Δ_R.
Case $(\nu a)\mathbf{0} \equiv \mathbf{0}$**.** This case is straightforward from the application of [Hide] and [End].
Case $(\nu \tilde{s})\mathbf{0} \equiv \mathbf{0}$**.** This case is similar to the previous case and follows from the application of [Cres] in Figure C.11 and [End].
Case $(\nu a)(\nu a')P \equiv (\nu a')(\nu a)P$ ***(resp.*** $(\nu \tilde{s})(\nu \tilde{s}')P \equiv (\nu \tilde{s}')(\nu \tilde{s})P$***).*** Straightforward from the fact that the order of elements in Γ (resp. Δ) is immaterial.
Case $(\nu a)(P) \mid Q \equiv (\nu a)(P \mid Q)$ ***if*** $a \notin \mathrm{fn}(Q)$**.** If $P_1 = (\nu a : \mathcal{G})(P) \mid Q$ and $P_2 = (\nu a : \mathcal{G})(P \mid Q)$ is straightforward by [Consec] and [Conc]. The case for $P_1 = (\nu a : \mathcal{G})(P \mid Q)$ and $P_2 = (\nu a : \mathcal{G})(P) \mid Q$ and follows from the fact that name a is only used by rules [Macc] and [Mcast] which are not used in the validation tree of Q since $a \notin \mathrm{fn}(Q)$.
Case $(\nu \tilde{s})(P) \mid Q \equiv (\nu \tilde{s})(P \mid Q)$ ***if*** $\tilde{s} \notin \mathrm{fn}(Q)$**.** If $P_1 = (\nu \tilde{s})(P) \mid Q$ and $P_2 = (\nu \tilde{s})(P \mid Q)$, from the validation tree of P_1 it follows (applying [Conc], and in the first case also [Cres])

$$\Gamma \models P \triangleright \Delta, \tilde{s} : \{\mathcal{T}_\mathtt{p} \mathbin{@} \mathtt{p}\}_{\mathtt{p} \in I} \qquad \text{and} \qquad \mathcal{C}; \Gamma \models Q \triangleright \Delta \tag{F.22}$$

The validation of $\Gamma \models P_2 \triangleright \Delta$, after applying [Cres], is reduced into $\Gamma \models P \mid Q \triangleright \Delta, \tilde{s} : \{\mathcal{T}_\mathtt{p} \mathbin{@} \mathtt{p}\}_{\mathtt{p} \in I}$ which holds by [Conc] with premises (F.22).

The case for $P_1 = (\nu \tilde{s})(P \mid Q)$, and $P_2 = (\nu \tilde{s})(P) \mid Q$ proceeds similarly.
Case $\mu X\langle \tilde{e}\tilde{t}\rangle(\tilde{v}\tilde{s}).P \equiv P[\mu X(\tilde{v}\tilde{s}).P/X][\tilde{e}\tilde{t}/\tilde{v}\tilde{s}]$

If $P_1 = \mu X\langle \tilde{e}\tilde{s}\rangle(\tilde{v}\tilde{s}).P$ (note that for P_1 to be validated it must be $\tilde{t} = \tilde{s}$.) and $P_2 = P[\mu X(\tilde{v}\tilde{s}).P/X][\tilde{e}\tilde{s}/\tilde{v}\tilde{s}]$ then by hypothesis

$$\Gamma \vdash \mu X\langle \tilde{e}\tilde{s}\rangle(\tilde{v}\tilde{s}).P \triangleright \tilde{s}_1 : \mathcal{T}_1[\tilde{e}/\tilde{v}] \mathbin{@} \mathtt{p}_1 .. \tilde{s}_n : \mathcal{T}_n[\tilde{e}/\tilde{v}] \mathbin{@} \mathtt{p}_n \tag{F.23}$$

By [Rec], (F.23) has premise

$$\Gamma, X : (\tilde{v} : \tilde{S})\mathcal{T}_1 \mathbin{@} \mathtt{p}_1 .. \mathcal{T}_n \mathbin{@} \mathtt{p}_n \vdash P \triangleright \tilde{s}_1 : \mathcal{T}_1 \mathbin{@} \mathtt{p}_1 .. \tilde{s}_n : \mathcal{T}_n \mathbin{@} \mathtt{p}_n \tag{F.24}$$

We proceed by induction on the size of the derivation tree of P_1. The base cases are for trees obtained using first [Rec] and then either [Idle] or [Var]. The case for [Idle] is straightforward. The case of [Var] is also straightforward considering that if $P = X\langle \tilde{e}'\tilde{s}\rangle$ then $P_1 = \mu X\langle \tilde{e}\tilde{t}\rangle(\tilde{v}\tilde{s}).X\langle \tilde{e}'\tilde{s}\rangle$ and $P_2 = X\langle \tilde{e}'\tilde{s}\rangle[\mu X(\tilde{v}\tilde{s}).P/X][\tilde{e}\tilde{t}/\tilde{v}\tilde{s}] = \mu X\langle \tilde{e}\tilde{t}\rangle(\tilde{v}\tilde{s}).X\langle \tilde{e}'\tilde{s}\rangle$ thus $P_1 = P_2$.

In all other cases, observe that by (F.24) and the substitution lemma

$$\Gamma, X : (\tilde{v} : \tilde{S})\mathcal{T}_1 \mathbin{@} \mathtt{p}_1 .. \mathcal{T}_n \mathbin{@} \mathtt{p}_n \vdash P[\tilde{e}\tilde{s}/\tilde{v}\tilde{s}] \triangleright \tilde{s}_1 : \mathcal{T}_1[\tilde{e}/\tilde{v}] \mathbin{@} \mathtt{p}_1 .. \tilde{s}_n : \mathcal{T}_n[\tilde{e}/\tilde{v}] \mathbin{@} \mathtt{p}_n \tag{F.25}$$

Since P is not $X\langle \tilde{e}'\tilde{t}\rangle$ (which is considered in the base case), whatever rule is applied to validate (F.25) can be applied also to

$$\Gamma, X : (\tilde{v} : \tilde{S})\mathcal{T}_1 \mathbin{@} \mathtt{p}_1 .. \mathcal{T}_n \mathbin{@} \mathtt{p}_n \vdash P[\mu X(\tilde{v}\tilde{s}).P/X][\tilde{e}\tilde{s}/\tilde{v}\tilde{s}] \triangleright \tilde{s}_1 : \mathcal{T}_1[\tilde{e}/\tilde{v}] \mathbin{@} \mathtt{p}_1 .. \tilde{s}_n : \mathcal{T}_n[\tilde{e}/\tilde{v}] \mathbin{@} \mathtt{p}_n \tag{F.26}$$

and, assuming that the validation tree of (F.25) has premise $\mathcal{C}, \Gamma' \vdash P' \triangleright \Delta'$ we can observe by inspecting the validation rules (except [Idle] and [Var]) that the validation tree of (F.26) has premise $\mathcal{C}, \Gamma' \vdash P'[\mu X(\tilde{v}\tilde{s}).P/X] \triangleright \Delta'$. The thesis follows from Lemma Appendix F.5.

If $P_1 = P[\mu X(\tilde{v}\tilde{s}).P/X][\tilde{e}\tilde{t}/\tilde{v}\tilde{s}]$ and $P_2 = \mu X\langle \tilde{e}\tilde{t}\rangle(\tilde{v}\tilde{s}).P$ then by hypothesis

$$\Gamma \vdash P[\mu X(\tilde{v}\tilde{s}).P/X][\tilde{e}\tilde{t}/\tilde{v}\tilde{s}] \triangleright \Delta \tag{F.27}$$

for some Δ, and since $X \in \mathrm{fn}(P)$ the validation tree includes (with $\Gamma \subseteq \Gamma'$)

$$\frac{A;\Gamma', X : (\tilde{v} : \tilde{S})\mathcal{T}_1 @ \mathtt{p}_1..\mathcal{T}_n @ \mathtt{p}_n \vdash P \triangleright \tilde{s}_1 : \mathcal{T}_1 @ \mathtt{p}_1, .., \tilde{s}_n : \mathcal{T}_n @ \mathtt{p}_n}{A;\Gamma' \vdash \mu X\langle \tilde{e}'\tilde{s}\rangle(\tilde{v}\tilde{s}).P \triangleright \tilde{s}_1 : \mathcal{T}_1[\tilde{e}'/\tilde{v}] @ \mathtt{p}_1, .., \tilde{s}_n : \mathcal{T}_n[\tilde{e}'/\tilde{v}] @ \mathtt{p}_n} \tag{F.28}$$

From the premise of (F.28) and [Rec] we obtain

$$A;\Gamma' \vdash \mu X\langle \tilde{e}\tilde{s}\rangle(\tilde{v}\tilde{s}).P \triangleright \tilde{s}_1 : \mathcal{T}_1[\tilde{e}/\tilde{v}] @ \mathtt{p}_1, .., \tilde{s}_n : \mathcal{T}_n[\tilde{e}/\tilde{v}] @ \mathtt{p}_n$$

from which it follows

$$\Gamma \vdash \mu X\langle \tilde{e}\tilde{s}\rangle(\tilde{v}\tilde{s}).P \triangleright \tilde{s}_1 : \mathcal{T}_1[\tilde{e}/\tilde{v}] @ \mathtt{p}_1, .., \tilde{s}_n : \mathcal{T}_n[\tilde{e}/\tilde{v}] @ \mathtt{p}_n$$

since:

1. Since P is closed, A express constraints only on $\tilde{v}$ or on variables that are introduced in P. Such variables however do not appear in the unfolding, i.e., they are not in $\mathrm{fn}(\mu X\langle \tilde{e}\tilde{s}\rangle(\tilde{v}\tilde{s}).P)$.
2. Γ' does not add to Γ any definition on free process variables in $\mu X\langle \tilde{e}\tilde{s}\rangle(\tilde{v}\tilde{s}).P$.

From the premise of (F.28) we also know that Δ in (F.27) is $\tilde{t}_1 : \mathcal{T}_1[\tilde{e}/\tilde{v}] @ \mathtt{p}_1..\tilde{t}_n : \mathcal{T}_n[\tilde{e}/\tilde{v}] @ \mathtt{p}_n$ thus

$$\Gamma \vdash \mu X\langle \tilde{e}\tilde{s}\rangle(\tilde{v}\tilde{s}).P \triangleright \Delta$$

Case $(\nu s_1..s_n)(s_1 : \emptyset \mid \ldots \mid s_n : \emptyset) \equiv \mathbf{0}$. The case is straightforward by application of [Cres] and [Qnil] in Figure C.11.

Appendix F.4. Proof of Proposition 7.4 (Subject Reduction - Visible Transitions)

If $\Gamma \vdash P \triangleright \Delta$, $P \xrightarrow{\alpha} P'$, and $\langle \Gamma, \Delta\rangle \xrightarrow{\alpha} \langle \Gamma', \Delta'\rangle$ where $\alpha \neq \tau$, then we have $\Gamma' \vdash P' \triangleright \Delta'$.

Proof: The proof is by rule induction on the validation rules in Figures 6 and C.11, showing a stronger result which adds to the statement:

> If $P \xrightarrow{\alpha} P'$ and $\Gamma \vdash P \triangleright \Delta$ with α being an output, a selection, or an action at a shared channel (accept and request), then $\langle \Gamma, \Delta\rangle$ allows α.

In the following proof we refer to both the transition rules for asserted processes in Figure 7 and the transition rules for endpoint assertions in Figure 8. Assume we have:

1. $\Gamma \vdash P \triangleright \Delta$ (which stands for $\mathsf{true}; \Gamma \vdash P \triangleright \Delta$)
2. $P \xrightarrow{\alpha} P'$ and
3. $\langle \Gamma, \Delta\rangle \xrightarrow{\alpha} \langle \Gamma', \Delta'\rangle$

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 [Snd] (Figure 6). In this case, we derive $\mathsf{true}; \Gamma \vdash P \triangleright \Delta$ with:

$$P = s_k!\langle \tilde{e}\rangle(\tilde{v})\{A\}; Q \quad \text{and} \quad \Delta = \Delta_0, \tilde{s} : k!(\tilde{v} : \tilde{S})\{A\}; \mathcal{T} @ \mathtt{p}$$

By the first premise of [Send] we have:

$$\mathsf{true} \supset A[\tilde{e}/\tilde{v}] \tag{F.29}$$

Since P is closed, we can set $\tilde{e}\downarrow\tilde{\mathtt{n}}$. By (F.29) we infer $A[\tilde{\mathtt{n}}/\tilde{v}]\downarrow\mathsf{true}$. It follows that P can move only by [Send] (i.e., not [SendErr]), hence, setting $\alpha = s_k!\tilde{\mathtt{n}}$, $P \xrightarrow{\alpha} Q[\tilde{\mathtt{n}}/\tilde{v}] \stackrel{\text{def}}{=} P'$. Now Δ can move by [TR-Send]: $\langle\Gamma,\Delta\rangle \xrightarrow{\alpha} \langle\Gamma,\ (\Delta_0, \tilde{s}\colon \mathcal{T}[\tilde{\mathtt{n}}/\tilde{v}]\,@\,\mathtt{p})\rangle$. By the second premise of [Send] in Figure 6, we have

$$\mathsf{true};\Gamma \vdash Q[\tilde{e}/\tilde{v}] \triangleright \Delta_0, \tilde{s}\colon \mathcal{T}[\tilde{e}/\tilde{v}]\,@\,\mathtt{p} \tag{F.30}$$

By Lemma Appendix E.4 (Evaluation Lemma), (F.30) immediately gives $\mathsf{true};\Gamma \vdash Q[\tilde{\mathtt{n}}/\tilde{v}] \triangleright \Delta_0, \tilde{s}\colon \mathcal{T}[\tilde{\mathtt{n}}/\tilde{v}]\,@\,\mathtt{p}$ as required.

Case [Rcv] (Figure 6). In this case the conclusion is $\mathsf{true};\Gamma \vdash P \triangleright \Delta$ with:

$$P = s_k?(\tilde{v})\{A\};Q \quad \text{and} \quad \Delta = \Delta_0, \tilde{s}\colon k?(\tilde{v}:\tilde{S})\{A\};\mathcal{T}\,@\,\mathtt{p}$$

By the shape of P we can set $\alpha = s_k?\tilde{\mathtt{n}}$. By [TR-Rec] $\langle\Gamma,\Delta\rangle \xrightarrow{\alpha} \langle\Gamma,\Delta_0,\tilde{s}\colon \mathcal{T}[\tilde{\mathtt{n}}/\tilde{v}]\,@\,\mathtt{p}\rangle$ for which $A[\tilde{\mathtt{n}}/\tilde{v}]\downarrow\mathsf{true}$. Thus P can move only by [RecV] (not by [RecvErr]), obtaining $P \xrightarrow{\alpha} Q[\tilde{\mathtt{n}}/\tilde{v}]$. Now the premise of [Rcv] in Figure 6 says:

$$\mathsf{true}\wedge A;\Gamma \vdash Q \triangleright \Delta_0, \tilde{s}\colon \mathcal{T}\,@\,\mathtt{p}$$

By Lemma Appendix E.2 (Substitution Lemma) we obtain

$$\mathsf{true}\wedge A[\tilde{\mathtt{n}}/\tilde{v}];\Gamma \vdash Q[\tilde{\mathtt{n}}/\tilde{v}] \triangleright \Delta_0, \tilde{s}\colon \mathcal{T}[\tilde{\mathtt{n}}/\tilde{v}]\,@\,\mathtt{p}$$

By $A[\tilde{\mathtt{n}}/\tilde{v}]\downarrow\mathsf{true}$ and by [Conseq] we obtain $\mathsf{true};\Gamma \vdash Q[\tilde{\mathtt{n}}/\tilde{v}] \triangleright \Delta_0, \tilde{s}\colon \mathcal{T}[\tilde{\mathtt{n}}/\tilde{v}]\,@\,\mathtt{p}$ as required.

Case [Sel] (Figure 6). We can set $\mathsf{true};\Gamma \vdash P \triangleright \Delta$ such that:

$$P = s_k \lhd \{A_j\}l_j : P_j \quad \text{and} \quad \Delta = \Delta_0, \tilde{s}\colon k \oplus \{\{A_i\}l_i : \mathcal{T}_i\}_{i\in I}\,@\,\mathtt{p}$$

By the premise of the rule we have $\mathsf{true} \supset A_j$ hence $A_j \downarrow \mathsf{true}$, therefore P can move only by [Label] (i.e., not [LabelErr]). Thus we set $\alpha = s_k \lhd l_j$ and we have $P \xrightarrow{\alpha} P_j$. The following assertion transition is also possible by [TR-Select]: $\langle\Gamma,\Delta\rangle \xrightarrow{\alpha} \langle\Gamma,\Delta_0,\tilde{s}\colon \mathcal{T}_j\,@\,\mathtt{p}\rangle$. By the second premise of [Label] in Figure 6 we get $\mathsf{true};\Gamma \vdash P_j \triangleright \Delta_0, \tilde{s}\colon \mathcal{T}_j\,@\,\mathtt{p}$ as required.

Case [Branch] (Figure 6). In this case we have $\mathsf{true};\Gamma \vdash P \triangleright \Delta$ such that

$$P = s_k \rhd \{\{A_i\}l_i\colon P_i\}_{i\in I} \quad \text{and} \quad \Delta = \Delta_0, \tilde{s}\colon k\&\{\{A_i\}l_i : \mathcal{T}_i\}_{i\in I}\,@\,\mathtt{p}$$

By the shape of P we can set $\alpha = s_k \rhd l_j$ for which we have, by [TR-Choice]: $A_j \downarrow \mathsf{true}$ $\langle\Gamma,\Delta\rangle \xrightarrow{\alpha} \langle\Gamma;\Delta_0,\tilde{s}\colon\ \mathcal{T}_j\,@\,\mathtt{p}\rangle$. Thus P can move only by [Branch] (not by [BranchErr]), obtaining: $P \xrightarrow{\alpha} P_j$. Now the premise of [Branch] in Figure 6 says:

$$\mathsf{true}\wedge A_j;\Gamma \vdash P_j \triangleright \Delta_0, \tilde{s}\colon\ \mathcal{T}_j\,@\,\mathtt{p}$$

By $A_j \downarrow \mathsf{true}$ and [Conseq] we obtain: $\mathsf{true};\Gamma \vdash Q[\tilde{\mathtt{n}}/\tilde{v}] \triangleright \Delta_0, \tilde{s}\colon\ \mathcal{T}[\tilde{\mathtt{n}}/\tilde{v}]\,@\,\mathtt{p}$ as required.

Case [Mcast] (Figure 6). In this case we have $\mathsf{true};\Gamma \vdash P \triangleright \Delta$ such that, combining with the premises of the rule: $P = \overline{a}_{[2..\mathtt{n}]}(\tilde{s}).Q$, $\mathsf{true};\Gamma \vdash Q \triangleright \Delta, \tilde{s}\colon (\mathcal{G}\upharpoonright\mathtt{1})\,@\,\mathtt{1}$, and

$$\Gamma \vdash a\colon\ \mathcal{G} \tag{F.31}$$

By the shape of P we can set $\alpha = \overline{a}[2..n](\tilde{s})$ and $\overline{a}_{[2..\mathtt{n}]}(\tilde{s}).Q \xrightarrow{\alpha} Q$. By (F.31) the following transition is possible using [TR-LinkOut] we obtain $\langle\Gamma,\Delta\rangle \xrightarrow{\alpha} \langle\Gamma,\Delta,\tilde{s}\colon (\mathcal{G}\upharpoonright\mathtt{1})\,@\,\mathtt{1}\rangle$ as required.

Case [MACC] (Figure 6). Similar to the case [MCAST] above.
Case [PAR] (Figure 6). 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.
Cases [NRES], [CRES] and [BOUT] (Figure C.11). In each case, direct from the induction hypothesis.
Case [CONSEQ] (Figure 6). Suppose the conclusion is $\mathsf{true};\Gamma \vdash P \triangleright \Delta$ which is derived from

$$\mathsf{true};\Gamma \vdash P \triangleright \Delta_0 \tag{F.32}$$

with $\Delta_0 \ni \Delta$. Now first suppose the concerned visible action α is neither a receive action nor a branching. Now suppose $P \xrightarrow{\alpha} P'$ and

$$\langle \Gamma, \Delta \rangle \xrightarrow{\alpha} \langle \Gamma', \Delta' \rangle \tag{F.33}$$

By induction hypothesis and by (F.32), (F.33) gives us: $\langle \Gamma, \Delta_0 \rangle \xrightarrow{\alpha} \langle \Gamma', \Delta'_0 \rangle$ for some Δ'_0 for which we have, by induction hypothesis

$$\mathsf{true};\Gamma' \vdash P' \triangleright \Delta'_0 \tag{F.34}$$

Since the assertion transition is deterministic and by Lemma 6.4 we know $\Delta'_0 \ni \Delta'$, by (F.34) we can use [CONSEQ] to reach $\mathsf{true};\Gamma' \vdash P' \triangleright \Delta'$ as required.
Case [REC] (Figure 6). Suppose the conclusion is

$$\mathsf{true};\Gamma \vdash \mu X\langle \tilde{e}\tilde{s}_1..\tilde{s}_n\rangle(\tilde{v}\tilde{s}_1..\tilde{s}_n).P_{body} \triangleright \Delta$$

By Lemma 7.3 also the unfolding of P can be validated against Δ

$$\mathsf{true};\Gamma \vdash P_{body}[\mu X../X][\tilde{e}/\tilde{v}] \triangleright \Delta$$

and by induction $P_{body}[\mu X../X][\tilde{e}/\tilde{v}] \to P'_{body}[\mu X../X][\tilde{e}/\tilde{v}]$ and $\Delta \to \Delta'$ with

$$\mathsf{true};\Gamma \vdash P'_{body}[\mu X../X][\tilde{e}/\tilde{v}] \triangleright \Delta' \tag{F.35}$$

By (F.35) and again by Lemma 7.3 (by folding the process) we obtain

$$\mathsf{true};\Gamma \vdash \mu X\langle \tilde{e}\rangle(\tilde{v})P'_{body} \triangleright \Delta'$$

Case [VAR] (Figure 6). Immediate since in this case there is no reduction from P.

Appendix G. Soundness (Proofs of Section 8)

Appendix G.1. Assertion Reduction and Coherence

Lemma Appendix G.1 (Assertion Reduction and Coherence). *If Δ is coherent and $\Delta \to \Delta'$ or equivalently $\langle \Gamma, \Delta \rangle \xrightarrow{\tau} \langle \Gamma, \Delta' \rangle$, then Δ' is again coherent.*

PROOF: We only consider the two cases for the send message assertion. The cases for the select message assertions are treated in the same way. We start from a simpler case. Consider the following redex:

$$\tilde{s} : \mathcal{H}[k!(\tilde{v} : \tilde{S})\{A\}; \mathcal{T}@\mathtt{p}] \tag{G.1}$$

For this being coherent, there is some $\mathcal{G}$ such that

$$k!(\tilde{v}:\tilde{S})\{A\};\mathcal{T} \Subset \mathcal{G}\upharpoonright \mathtt{p} \tag{G.2}$$

similarly for other endpoint assertions under $\tilde{s}$. Now consider we have a reduction from (G.1) by the first rule in Figure C.9 into: $\tilde{s}:\mathcal{H}[k!\langle\tilde{\mathtt{n}}\rangle;\mathcal{T}[\tilde{\mathtt{n}}/\tilde{v}]@\mathtt{p}]$ where we have

$$A[\tilde{\mathtt{n}}/\tilde{v}]\downarrow \mathsf{true} \tag{G.3}$$

By (G.2) and (G.3) and because $\Subset$ is transitive we obtain: $k!\langle\tilde{\mathtt{n}}\rangle;\mathcal{T}[\tilde{\mathtt{n}}/\tilde{v}] \Subset \mathcal{G}\upharpoonright \mathtt{p}$ as required. For the other case, the reduction involves a pair. Assume Δ has a redex

$$\tilde{s}:\mathcal{H}[k!\langle\tilde{\mathtt{n}}\rangle\mathcal{T}_a, k?(\tilde{v})\{A_b\};\mathcal{T}_b] \tag{G.4}$$

As before, by coherence we can set: $k!\langle\tilde{\mathtt{n}}\rangle;\mathcal{T}_a \Subset \mathcal{G}\upharpoonright \mathtt{p}$ and $k?(\tilde{v})\{A_b\};\mathcal{T}_b \Subset \mathcal{G}\upharpoonright \mathtt{q}$. Note we can safely assume $\mathcal{G}$ has the shape (up to permutation of utterly unordered actions):

$$\mathcal{G} = \mathtt{p}\to\mathtt{q}:(\tilde{v}:\tilde{S})\{A'\}.\mathcal{G}'$$

hence we can assume: $\mathcal{G}\upharpoonright\mathtt{p} = k!(\tilde{v}:\tilde{S})\{A'\};(\mathcal{G}'\upharpoonright\mathtt{p})$ and $\mathcal{G}\upharpoonright\mathtt{q} = k?(\tilde{v})\{A'\};(\mathcal{G}'\upharpoonright\mathtt{q})$ such that, by Definition Appendix C.3, $A'\supset A_b$ and $A'[\tilde{\mathtt{n}}/\tilde{v}]\downarrow\mathsf{true}$ and hence

$$\mathcal{T}_a \Subset \mathcal{G}'\upharpoonright\mathtt{p}[\tilde{\mathtt{n}}/\tilde{v}] \tag{G.5}$$

$$\mathcal{G}'\upharpoonright\mathtt{q}[\tilde{\mathtt{n}}/\tilde{v}] \supset \mathcal{T}_b[\tilde{\mathtt{n}}/\tilde{v}] \tag{G.6}$$

Now consider the reduction from (G.4) into $\tilde{s}:\mathcal{H}[\mathcal{T}_a,\mathcal{T}_b[\tilde{\mathtt{n}}/\tilde{v}]]$.

By (G.5) and (G.6) we obtain $\mathcal{T}_a \Subset \mathcal{G}'[\tilde{\mathtt{n}}/\tilde{v}]\upharpoonright\mathtt{p}$, and $\mathcal{T}_b[\tilde{\mathtt{n}}/\tilde{v}] \Subset \mathcal{G}'[\tilde{\mathtt{n}}/\tilde{v}]\upharpoonright\mathtt{q}$. Since for each $\mathtt{r}\notin\{\mathtt{p},\mathtt{q}\}$, and because by HSP the variables in $\tilde{v}$ only occur in assertions/actions involving either $\mathtt{p}$ of $\mathtt{q}$, we know $\mathcal{G}'[\tilde{\mathtt{n}}/\tilde{v}]\upharpoonright\mathtt{r} = \mathcal{G}\upharpoonright\mathtt{r}$ hence as required.

Appendix G.2. Proof of Theorem 8.4 (Soundness for Open Processes)

Let P be a program phrase. Then $\mathcal{C};\Gamma\vdash P\triangleright\Delta$ implies $\mathcal{C};\Gamma\models P\triangleright\Delta$.

PROOF: Let $\mathcal{R}$ be the relation collecting all the pairs of the form $(P_\sigma,\langle\Gamma_\sigma,\Delta_\sigma\rangle)$ such that $\mathcal{C};\Gamma\vdash P\triangleright\Delta$ where: (i) P is a sub-term of a multi-step $\xrightarrow{\alpha}$-derivative of a program phrase, (ii) Δ is an assertion assignment possibly containing non-singleton assignments, and (iii) σ is a closing substitution consistent with $\mathcal{C}$ and Γ. We show that $\mathcal{R}$ is a conditional simulation (in the extended sense defined in Definition 6.1), by rule induction on the validation rules (in Figures 6 and C.11). Assume

$$\mathcal{C};\Gamma\vdash P\triangleright\Delta \tag{G.7}$$

is derived and σ is a closing substitution conforming to $\mathcal{C}$ and Γ. We carry out case analysis depending on the last rule used.

Case [SND](Figure 6)**.** We can set $P = s_k!\langle\tilde{e}\rangle(\tilde{v})\{A\};P'$ where, in (G.7), $\Delta=\Delta',\tilde{s}:k!(\tilde{v}:\tilde{S})\{A\};\mathcal{T}@\mathtt{p}$ By the definition of closure

$$P_\sigma = s_k!\langle\tilde{e}_\sigma\rangle(\tilde{v})\{A_\sigma\};P'_\sigma$$

where $\tilde{e}_\sigma\downarrow\tilde{\mathtt{n}}$. Process P_σ can only move because of rule [SEND] (Figure 7) with label is $s_k!\tilde{\mathtt{n}}$. Since $A[\tilde{\mathtt{n}}/\tilde{v}]\downarrow\mathsf{true}$ (by premise of [SEND]) and $A[\tilde{\mathtt{n}}/\tilde{v}]\supset A_\sigma[\tilde{\mathtt{n}}/\tilde{v}]$, then

$$s_k!\langle\tilde{e}_\sigma\rangle(\tilde{v})\{A_\sigma\};P'_\sigma \xrightarrow{s_k!\tilde{\mathtt{n}}} P'_\sigma$$

We observe

$$\Delta_\sigma = \Delta'_\sigma, \tilde{s} : k!(\tilde{v} : \tilde{S})\{A_\sigma\}; \mathcal{T}_\sigma \mathbin{@} \mathtt{p}$$

By [TR-SEND] in Figure 8, since $A_\sigma[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{true}$,

$$\langle \Gamma, \Delta'_\sigma, \tilde{s} : k!(\tilde{v} : \tilde{S})\{A_\sigma\}; \mathcal{T}_\sigma \mathbin{@} \mathtt{p} \rangle \xrightarrow{s_k!\tilde{\mathtt{n}}} \langle \Gamma, \Delta'_\sigma, \tilde{s} : \mathcal{T}_\sigma \mathbin{@} \mathtt{p} \rangle$$

It follows by induction $(P'_\sigma, \langle \Gamma, \Delta'_\sigma, \tilde{s} : \mathcal{T}_\sigma \mathbin{@} \mathtt{p} \rangle) \in R$.
Case [RCV] (Figure 6)**.** We set $P = s_k?(\tilde{v})\{A\}; P'$ and, in (G.7), $\Delta = \Delta', \tilde{s} : k?(\tilde{v} : \tilde{S})\{A\}; \mathcal{T} \mathbin{@} \mathtt{p}$. Let $P_\sigma = s_k?(v)\{A_\sigma\}; P'_\sigma$. Observe

$$\Delta_\sigma = \Delta'_\sigma, \tilde{s} : k?(\tilde{v} : \tilde{S})\{A_\sigma\}; \mathcal{T}_\sigma \mathbin{@} \mathtt{p}$$

Process P_σ can only move because of rule [RCV] in Figure 7 with label $s_k?\tilde{\mathtt{n}}$. By definition of conditional simulation, we only consider the case in which Δ_σ is able to move (i.e., $\tilde{\mathtt{n}} : \tilde{S}$ and $A_\sigma[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{true}$). In such case, by [RCV] in Figure 7, and by [TR-REC] in Figure 8, respectively:

$$s_k?(\tilde{v})\{A_\sigma\}; P'_\sigma \xrightarrow{s_k?\tilde{\mathtt{n}}} P'_\sigma \quad \text{and} \quad \langle \Gamma, \Delta'_\sigma, \tilde{s} : k?(\tilde{v} : \tilde{S})\{A_\sigma\}; \mathcal{T}_\sigma \mathbin{@} \mathtt{p} \rangle \xrightarrow{s_k?\tilde{\mathtt{n}}} \langle \Gamma, \Delta'_\sigma, \tilde{s} : \mathcal{T}_\sigma \mathbin{@} \mathtt{p} \rangle$$

It follows by induction $(P'_\sigma, \langle \Gamma_1, \Delta_\sigma, \tilde{s} : \mathcal{T}_\sigma \mathbin{@} \mathtt{p} \rangle) \in R$.
Case [SEL] (Figure 6). Let $P = s_k \lhd \{A_j\} l_j : P_j$, by [SEL] $\mathcal{C}; \Gamma \vdash s_k \lhd \{A_j\} l_j : P_j \rhd \Delta$ with $\Delta = \Delta', \tilde{s} : k \oplus \{\{A_i\} l_i : \mathcal{T}_i\}_{i \in I} \mathbin{@} \mathtt{p}$ and $j \in I$. Observe that $\Delta_\sigma = \Delta'_\sigma, \tilde{s} : k \oplus \{\{A_{i\sigma}\} l_i : \mathcal{T}_{i\sigma}\}_{i \in I} \mathbin{@} \mathtt{p}$. Also $P_\sigma = s_k \lhd \{A_{j\sigma}\} l_j : P_{j\sigma}$ Process P_σ can only move because of rule [LABEL] in Figure 7 with label $s_k \lhd l_j$ and, since $A_j \downarrow \mathsf{true}$ by well formedness of P and $A_j \implies A_{j\sigma}$ then

$$s_k \lhd \{A_{j\sigma}\} l_j : P_{j\sigma} \xrightarrow{s_k \lhd l_j} P_{j\sigma}$$

By [TR-SEL] in Figure 8, since $A_j \downarrow \mathsf{true}$

$$\langle \Gamma, \Delta'_\sigma, \tilde{s} : k \oplus \{\{A_{i\sigma}\} l_i : \mathcal{T}_{i\sigma}\}_{i \in I} \mathbin{@} \mathtt{p} \rangle \xrightarrow{s_k \lhd l_j} \langle \Gamma, \Delta'_\sigma, \tilde{s} : \mathcal{T}_{j\sigma} \mathbin{@} \mathtt{p} \rangle$$

By induction, $(P_{j\sigma}, \langle \Gamma, \Delta'_\sigma, \tilde{s} : \mathcal{T}_{j\sigma} \mathbin{@} \mathtt{p} \rangle) \in R$.
Case [BRANCH] (Figure 6). We can set $P = s_k \rhd \{\{A_i\} l_i : P_i\}_{i \in I}$ then $P_\sigma = s_k \rhd \{\{A_{i\sigma}\} l_i : P_{i\sigma}\}_{i \in I}$. By [BRANCH] in Figure 6, $\mathcal{C}; \Gamma \vdash s_k \rhd \{\{A_i\} l_i : P_i\}_{i \in I} \rhd \Delta$ with $\Delta = \Delta', \tilde{s} : k \& \{\{A_i\} l_i : \mathcal{T}_i\}_{i \in I} \mathbin{@} \mathtt{p}$. We observe that

$$\Delta_\sigma = \Delta'_\sigma, \tilde{s} : k \& \{\{A_{i\sigma}\} l_i : \mathcal{T}_{i\sigma}\}_{i \in I} \mathbin{@} \mathtt{p}$$

Process P_σ can only move because of rule [BRANCH] with label $s_k \rhd l_j$. By definition of conditional simulation we only consider the case in which $\langle \Gamma, \Delta_\sigma \rangle$ is able to perform a branching move with label $s_k \rhd l_j$, that is when $A_{j\sigma} \downarrow \mathsf{true}$. Assuming $A_{j\sigma} \downarrow \mathsf{true}$, by [BRANCH] in Figure 7:

$$s_k \lhd \{A_{j\sigma}\} l_j : P_{j\sigma} \xrightarrow{s_k \rhd l_j} P_{j\sigma}$$

and by [TR-CHOICE] in Figure 8:

$$\langle \Gamma, \Delta'_\sigma, \tilde{s} : k \& \{\{A_{i\sigma}\} l_i : \mathcal{T}_{i\sigma}\}_{i \in I} \mathbin{@} \mathtt{p} \rangle \xrightarrow{s_k \rhd l_j} \langle \Gamma, \Delta'_\sigma, \tilde{s} : \mathcal{T}_{j\sigma} \mathbin{@} \mathtt{p} \rangle$$

By induction, $(P_{j\sigma}, \langle \Gamma, \Delta'_\sigma, \tilde{s} : \mathcal{T}_{j\sigma} \mathbin{@} \mathtt{p} \rangle) \in R$.

Case [MCAST] (Figure 6). In this case $P = \overline{a}_{[2..n]}(\tilde{s}).P'$ and we can set $\Gamma \vdash \overline{a}_{[2..n]}(\tilde{s}).P \triangleright \Delta$. Let $P_\sigma = \overline{a}_{[2..n]}(\tilde{s}).P'_\sigma$. Process P_σ can only move because of rule [LINKOUT] in Figure 7:

$$\overline{a}_{[2..n]}(\tilde{s}).P'_\sigma \xrightarrow{\overline{a}[2..n](\tilde{s})} P'_\sigma$$

By [TR-LINKOUT] in Figure 8,

$$\langle \Gamma, \Delta_\sigma \rangle \xrightarrow{\overline{a}[2..n](\tilde{s})} \langle \Gamma_1, \Delta_\sigma, \tilde{s} : (\mathcal{G} \upharpoonright 1)_\sigma \mathbin{@} 1 \rangle$$

By induction we have $(P'_\sigma, \langle \Gamma_1, \Delta_\sigma, \tilde{s} : (\mathcal{G} \upharpoonright 1)_\sigma \mathbin{@} 1 \rangle) \in R$.
Case [MACC] (Figure 6). This case is essentially identical to the case [MCAST] above.
Case [CONC] (Figure 6). The cases of independent actions are direct from the induction hypothesis. If the reduction takes place by interaction, then we use Lemma 7.1.
Case [IF] (Figure 6). Let $P = \mathsf{if}\ e\ \mathsf{then}\ Q_1\ \mathsf{else}\ Q_2$. By [IF] in Figure 6, since (G.7) holds, then $\mathcal{C} \wedge e; \Gamma \vdash Q_1 \triangleright \Delta$. We note $P_\sigma = \mathsf{if}\ e_\sigma\ \mathsf{then}\ Q_{1\sigma}\ \mathsf{else}\ Q_{2\sigma}$. Process P_σ can move because of either [IFT] or [IFF] of Figure 7. Let us consider the case in which the transition happens by rule [IFT] (the case with [IFF] is symmetric):

$$\mathsf{if}\ e_\sigma\ \mathsf{then}\ Q_{1\sigma}\ \mathsf{else}\ Q_{2\sigma} \xrightarrow{\tau} Q_{1\sigma} \quad \text{with } e_\sigma \downarrow \mathsf{true}$$

By induction, since $e_\sigma \downarrow \mathsf{true}$ and moreover e_σ does not have free variables $(Q_{1\sigma}, \langle \Gamma, \Delta_\sigma \rangle) \in R$ as required.
Case [IDLE] (Figure 6). We can set $P = \mathbf{0}$; the property holds since there are no transitions.
Case [HIDE] (Figure 6). Immediate from induction hypothesis.
Case [VAR] (Figure 6). We set $P = X\langle \tilde{e}, \tilde{s_1}, ..., \tilde{s_n} \rangle$ with $\Gamma(X) = (\tilde{v} : \tilde{S})\mathcal{T}_1 \mathbin{@} \mathtt{p}_1 \ldots \mathcal{T}_n \mathbin{@} \mathtt{p}_n$. P_σ is a process such that $\mathcal{C}_\sigma, \Gamma_\sigma \vdash P_\sigma[\tilde{e}/\tilde{v}] \triangleright \Delta_\sigma$ where $\Delta_\sigma = \Delta'_\sigma, \tilde{s}_1 : \mathcal{T}_{1\sigma}[\tilde{e}/\tilde{v}] \mathbin{@} \mathtt{p}_1, .., \tilde{s}_n : \mathcal{T}_{n\sigma}[\tilde{e}/\tilde{v}] \mathbin{@} \mathtt{p}_n$ is the closure of the endpoint assertion of P. The property holds straightforwardly by the cases for the other process types.
Case [NRES] (Figure 6). Immediate from induction hypothesis.
Case [REC] (Figure 6). This case is proved by the standard syntactic approximation of a recursion. By Lemma 8.1, we can assume, in all derivations for processes in $\mathcal{R}$, the application of Rule [REC] only occurs in (the last steps of) a derivation for a transition derivative of a program phrase, without loss of generality. Under this assumption, by Lemma 8.3, we know the premise and conclusion of an application of [REC] is well-initiated in the sense of Definition 8.2. Assume that we have

$$\mathcal{C}; \Gamma, X : (\tilde{v} : \tilde{S})\mathcal{T}_1 \mathbin{@} \mathtt{p}_1 \ldots \mathcal{T}_n \mathbin{@} \mathtt{p}_n \models P \triangleright \tilde{s}_1 : \mathcal{T}_1 \mathbin{@} \mathtt{p}_1 \ldots \tilde{s}_n : \mathcal{T}_n \mathbin{@} \mathtt{p}_n \tag{G.8}$$

Further we also assume

$$\mathcal{C}; \Gamma, X : (\tilde{v} : \tilde{S})\mathcal{T}_1 \mathbin{@} \mathtt{p}_1 \ldots \mathcal{T}_n \mathbin{@} \mathtt{p}_n \models Q \triangleright \Delta \tag{G.9}$$

Let x range over interaction names and session channels. In the following we often use the notation for the substitution $Q[(\tilde{x})R/X]$ which replaces each occurrence of $X\langle \tilde{e} \rangle$ with $R[\tilde{e}/\tilde{x}]$. Using well-guardedness of process variables [29, §2], we first approximate the recursion by the following hierarchy:

$$P^0 \stackrel{\text{def}}{=} P' \approx \mathbf{0} \qquad P^1 \stackrel{\text{def}}{=} P[(\tilde{x})P^0/X] \qquad \cdots \qquad P^{n+1} \stackrel{\text{def}}{=} P[(\tilde{x})P^n/X]$$

Above P' 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, $P' \stackrel{\text{def}}{=} (\nu a)(a[2](s).P)$ then $P' \approx \mathbf{0}$. We also set $P^{\omega} = \mu X\langle\tilde{x}\rangle(\tilde{v}).P$ 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 $(\tilde{x})P^0$, $(\tilde{x})P^1$, ... $(\tilde{x})P^n$, and finally $(\tilde{x})P^{\omega}$. We call the result Q^0, Q^1, ... 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 $\mathcal{C};\Gamma \vdash S \triangleright \Delta$ is derivable and that $\mathcal{C}_0;\Gamma_0 \vdash S_0 \triangleright \Delta_0$ occurs in its derivation, hence S_0 occurs in S. Suppose that also $\mathcal{C}_0;\Gamma_0 \vdash S_0' \triangleright \Delta_0$ where S_0' and S_0 have the identical typing (wrt [29]). We can replace the occurrence of S_0 in S by S_0', with the result written S', such that $\mathcal{C};\Gamma \vdash S' \triangleright \Delta$ is derivable.

Using property, we first note that, for any $\langle\Gamma,\Delta\rangle$ and $\mathcal{C}$, we have $\mathcal{C};\Gamma \models P^0 \triangleright \Delta$. Thus we apply this to (G.8) and replace X in P by $(\tilde{v}\tilde{s}_1..\tilde{s}_n)P^0$:

$$\mathcal{C};\Gamma \models P^1 \triangleright \tilde{s}_1 : \mathcal{T}_1 @ \mathtt{p}_1 \ldots \tilde{s}_n : \mathcal{T}_n @ \mathtt{p}_n$$

This can again be used for (G.8) (noting the environment Γ can always be taken as widely as possible in [Var]): $\mathcal{C};\Gamma \models P^2 \triangleright \tilde{s}_1 : \mathcal{T}_1 @ \mathtt{p}_1 \ldots \tilde{s}_n : \mathcal{T}_n @ \mathtt{p}_n$. In this way we know that for an arbitrary n: $\mathcal{C};\Gamma \models P^n \triangleright \tilde{s}_1 : \mathcal{T}_1 @ \mathtt{p}_1 \ldots \tilde{s}_n : \mathcal{T}_n @ \mathtt{p}_n$. By applying this to (G.9), we obtain:

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

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 $\langle\Gamma,\Delta\rangle$, 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 Definition 6.1 hold at each step, hence $\mathcal{R}$ is a conditional simulation.

Case [Conseq] (Figure 6). By Proposition 6.5.

Cases [Qnil], [Qval] and [Qsel] (Figure C.11). These processes (queues) do not have transitions. The behaviours of queues are taken into account as part of $\tau_{\mathtt{s}}$-actions in the case for [Conc], above.

Case [CRes] (Figure 6). By Lemma Appendix G.1. This exhausts all cases.