A theory of design-by-contract for distributed multiparty interactions*

Laura Bocchi[1], Kohei Honda[2], Emilio Tuosto[1], and Nobuko Yoshida[3]

[1] University of Leicester
[2] Queen Mary, University of London
[3] Imperial College London

Abstract. Reliability is a critical issue in many multi-organizational distributed applications, be they web services, financial protocols, scientific computing infrastructure, and software controlling public services such as transport. Design by contract traditionally addresses reliability by elaborating type signatures for sequential programs centring on asymmetric procedural invocations. In this paper we generalise the notion of Design by Contract to multiparty distributed applications each using one or more complex, and possibly long-running, application-level protocols. Our main contribution is an assertion method for distributed multiparty interactions centring on the notion of *global assertion*, which specifies constraints on a whole interaction scenario by elaborating *multiparty session types* from [5, 6, 27]. The paper establishes the key technical results underpinning the usage of this method for specification, verification and static and dynamic behavioural validation.

Contents

* Last update: October 19, 2009.

1 Introduction

Background. Reliability is a critical issue in many multi-organizational distributed applications such as Web Services and financial protocols. For designing, implementing and managing these applications, it is essential to have a tractable and rigorous description of how interactions should proceed through collaboration among participants, what content messages should carry and in what formats, 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 descriptions of application-level distributed protocols is severely limited. As an example of multi-organizational distributed applications, we consider financial protocols. The International Organization for Standardization (ISO) is currently working on a methodology for specifying and developing financial protocols, called *universal financial industry message scheme* [46]. A typical financial protocol is used for deciding the transaction of a 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 (say auditing) concerns, that each party properly carries out the responsibility associated with their roles in the protocol, in the sense that it sends a message with a valid content in a valid format at a correct timing, responding to valid messages by others. Currently the description of such a protocol is given only informally (except for message formats), describing only a small subset of traces from the whole protocol structure, 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 misleading in key aspects, and are legally unusable.
2. *It is incomplete:* it is not possible to describe the whole structure and essential constraints on the protocol execution. There is a lack of clarity on what has to de described.
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.

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

In this paper we present a logical method for specifying and verifying the structures and constraints of distributed multiparty interactions, based on the notion of Design-by-Contract (DbC). The traditional DbC for sequential programs specifies a contract between a user and a program as a set of pre-conditions, post-conditions and invariants over a type signature. The aim is to improve the reliability (e.g. correctness and robustness) of the produced software. DbC is a basis of a wide range of software engineering practice, from high-level modelling to program design to runtime assertion checking [28, 35, 40]. One of the key ideas underlying the traditional DbC is to elaborate *type signatures* (of say classes and objects) with *logical formulae*. Logical formulae are a general, rigorous and flexible tool to describe constraints [22, 26]. Instead of just saying "the method fooBar of an object of class ABC should be invoked with a string and an integer, and then it will return (if ever) another string", we can give a more precise specification, say "if we invoke the method fooBar of an object of class ABC with a string representing a date between 2007 and 2008 say s and an integer less than 1000, say n, then it will, if ever, return a string which represents the date n days after s".

A type signature describes the basic shape of how a user can interact with a program, stipulating its key *interface* to other components (which may be developed by other programmers). For this reason the type signature is a more stable part of systems design, functioning as a minimal contract among components with a basic safety guarantee. By associating this type signature with logical predicates, DbC enables a highly effective framework for specifying, validating and controlling systems' interfaces and behaviours, usable throughout all phases of systems development. As a modelling and programming practice, it encourages engineers to make explicit and formal the contracts among software modules, and build software on the basis of these contracts [23, 37].

The traditional DbC has focused on the type signatures for procedure invocation, a fundamental building block in sequential programming. Its analogue in distributed interactions, request-reply, is an important micro-protocol but has a basic limitation: in practice, a typical distributed application implements interaction scenarios that are much more complex than a request-reply. In this paper we extend the core idea of DbC — contract-based software development through elaboration of type signa-

tures with logical predicates — to the design and modelling of distributed applications.

We consider distributed applications whose activity is organized into abstraction units called *sessions*. Each session consists of a structured series of message exchanges among multiple participants. Sessions can 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. Each session follows a stipulated protocol defining how participants interact and a conversation would evolve, which tends to be a relatively stable part of design and development of distributed applications.

In previous work, it has been shown that the protocol structures of distributed multiparty sessions can be specified and used as formal type signatures [5, 6, 27], called *multiparty session types*. A session type describes the skeleton of global interaction scenarios through type constructors such as sequencing, branching and recursion, offering the standard benefit of type signature such as type error freedom. To organize a logical specification associated with this type signature, the target of a logical specification is not limited to a single call-return, and must describe how constraints will accumulate as many interactions proceed in a session, governing its future paths. For enabling the precise, complete and formal descriptions of rich behaviours of interactional applications, we need a rigorous semantic foundation which supports diverse engineering concerns such as endpoint programs conformance to a global specification, testing, runtime assertion checking and communication monitoring. One of the key technical elements is the formal association of logical descriptions to the semantics of endpoint programs, and a proof system validating them.

This Work. The present work introduces an assertion method for distributed interactions centring on the notion of *global assertion*, which specifies global constraints on a whole interaction scenario by elaborating, with logical formulae, type signatures for multiparty sessions [5, 27], together with basic results underpinning the usage of the assertion method for diverse engineering concerns.

The key ideas of our framework, illustrated in Figure 1, are presented below.

(0) A specification for a multiparty session is given as a *global assertion* $\mathcal{G}$, namely logical predicates annotating a protocol structure. Each predicate acts as pre/post-conditions depending on the viewpoint of

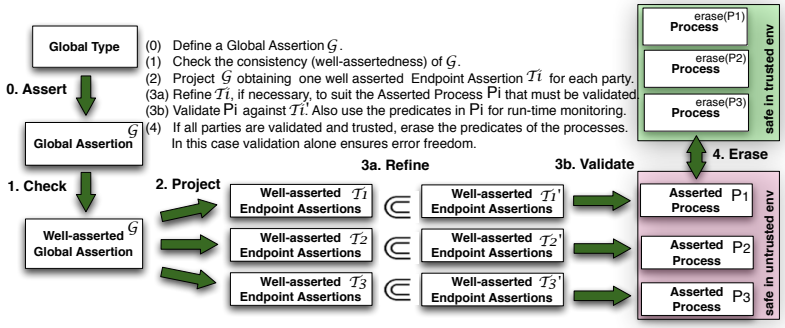

Fig. 1. The assertion method

each participant [4]. Global assertions are informally illustrated through an example in § 2 and formally defined in § 3.

(1) We stipulate a minimal semantic criterion, *well-assertedness of* $\mathcal{G}$ (§ 3.3) characterising consistent specifications with respect to the temporal flow of events, to avoid for instance un-satisfiable constraints.

(2) $\mathcal{G}$ is automatically projected onto endpoints, yielding one *endpoint assertion* ($\mathcal{T}_i$) for each participant, specifying her/his behavioural responsibility (§ 4). Consistency is checked only once at the global level, since the endpoint assertions are consistent by *well-assertedness of projections*.

(3) Communicating processes with local monitors are modelled as π-calculus processes elaborated with predicates (§ 5.1). Each process can effectively check that both incoming and outgoing interactions satisfy the specified constrains. We can also validate processes against an endpoint assertion at a design time, through a compositional proof system (§5.2) and an effective validation algorithm.

(4) In a trusted environment, where all participants can be assumed validated, we can dispense with the run-time monitoring on predicates to ensure the correct execution of the protocol (§ 7.3).

This paper introduces a formal theory underpinning the above framework and establishes its key results, presented through the following technical contributions: a validation algorithm for consistency of global assertions (Proposition 4.6); semantic foundations of global assertions through

[4] Global assertion may be designed from scratch or by elaborating an existing global session type in the sense of [27].

labelled transitions (Propositions 7.5 and 7.8); a compositional proof system for validating processes against assertions, whose larger examples are given in Section 6 and which is sound with respect to transition-based semantics (Theorem 7.10). The soundness leads to *assertion-error freedom* of validated processes (Theorem 7.11) which ensures that the process will meet its obligations with respect to the original specification assuming that the remaining parties do so; a decidable algorithm for static validation of processes against assertions (Theorem 7.13); and properties of the associated endpoint monitoring methods in untrusted and trusted environments (Proposition 7.14 and Theorem 7.15). Section 8 discusses further results and related work.

As far as we know, the present work is the first to introduce a logical theory which enables effective runtime and compiletime assertion-based validations for non-trivial specifications for a typed π-calculus through the use of global type signature, built on a rigorous semantic foundation.

It has been known that specifications of communicating processes are notably challenging, cf. [19, 34]. We believe the proposed framework offers both clear, expressive specifications for non-trivial properties on the one hand, and tractable validations of processes against these specifications on the other. The main reason this has been made possible is by centring on *specifications per type signature*, inheriting the idea from the traditional DbC. The difference from the traditional DbC is that this type signature is for communication behaviour rather than functional behaviour.

From the viewpoint of specifications, the per-session based approach enable a specification of the properties of communication behaviours independent from individual processes: this is important for high-level specifications because an individual communicating program may change how it will use multiple protocols to achieve its task, but the structure of each protocol (which is determined by agreement among multiple parties) would general be more stable. The approach also leads to expressive specifications, allowing us to capture not only the structure of interactions but also the range of message values and the conditions under which each branch and its sub-conversation is selected.

From the viewpoint of validation, the specification-per-type-signature approach enables algorithmic static compositional validation of processes against their specifications even when a process contains recursion and concurrent composition. In spite of the potential interleaving and mutual interactions/interference, a validation algorithm only needs to check reciprocity of specifications per each session, through the rely-guarantee (assumption-commitment) framework [30] (this validation is carried out

by checking how the whole process behaviour across multiple sessions contributes to the satisfiability of given per-session specifications, e.g. an output in a session using a value satisfies its specification because of a value coming from an input in another session). As a result we obtain an algorithmic static validation for expressive behavioural specifications in the context of the π-calculus, which may not have been known in the extant studies.

However the very methodological choice of the present approach, that is to have specifications on a per-session basis, also leads to its limited applicability. One engineering situation where this choice becomes a limitation is when one primarily wishes to specify the behavioural properties of a whole process (for example when we are concerned with termination of the whole process). Another and related occasion is when one wishes to specify the properties of a combination of several protocols globally, assuming they may often be used in combination. Each such scene may as well demand a more complex specification method, allowing richer specifications and possibly requiring more complex, and less tractable, validation methods. The practical challenge of specifying and validating properties for communicating processes can be met only through a well-organised array of different methods, associated with various tool support.

Against the wide spectrum of different specification and verification methods for typed processes, the present "per-type-signature"-based approach is intended to offer a basic stratum (which may be all that is feasibly specifiable as shared behavioural contracts for public or semi-public protocols). Having the proposed method as a stratum will help ease the complexity when one needs to rise to the challenge of more complex specifications and validations, which corresponds to how the traditional DbC eases a verification of the whole program property by enabling abstract treatment of each procedural call in the program.

A further challenge is to combine and integrate different methods catering for different concerns, for which we may need a common logical basis (the use of Hennessy-Milner logic [19, 25] in §8 may suggest such a potential).

The full proofs of the technical results are relegated to Appendix for readablity.

2 DbC for Distributed Multiparty Interactions by Pictures

Preliminaries. This section informally illustrates some of the key ideas of *global assertions*, using two simple examples used throughout the present paper.

A global assertion associates logical formulae to a type signature described as a *global session type* from [27]. We call such logic formulae *interaction predicates* (or often simply *predicates*). Each predicate defines both what one party is obliged to guarantee and, dually, what the other parties can rely on. More specifically:

(1) Each message in a session needs to respect a predicate over variables representing its content (e.g., "the seller will send an invoice to the buyer where the amount of product is the amount previously specified in the order"). The interaction predicate is an obligation for the sender and a guarantee for the receiver.

(2) Each branch is associated to a predicate that constrains the selection of that branch (e.g., "a seller must not proceed with the sale of a product if the size of the order exceeds the size of the stock"). As in (1) the predicate is an obligation for the selector and a guarantee for the other participant.

(3) Each recursive process can be associated to a predicate (invariant) that must be satisfied at each recursive call, where the invariant is both an obligation and a guarantee for all parties.

The aim of this paper is to provide a design-time method for specification and validation of these constraints. Below we illustrates the key idea of this specification method, global assertions, through sequence diagrams annotated by logical formulae.

Protocol *BuyerSeller***.** Figure 2 specifies a multiparty session where the participants, *Buyer* and *Seller*, exchange messages denoted by *interaction variables* (e.g., o and i). The *Buyer* asynchronously sends o of type *Order* to *Seller*, then *Seller* performs a choice and either sends *Buyer* an invoice i of type $Invoice$ or quits the protocol (e.g., if *Seller* is out of stock).

The predicates, attached to interactions in a way similar to guards in sequence diagrams (but with a different emphasis, cf. § 8), express constraints on the values that can be exchanged in distributed interactions and are assigned to interaction variables. The example uses three predicates. By $A1$ the buyer guarantees that the price in the order is smaller than 100 (e.g., the sale is forbidden for orders greater than 100) and, dually, the seller is guaranteed that the order will not contain a price greater

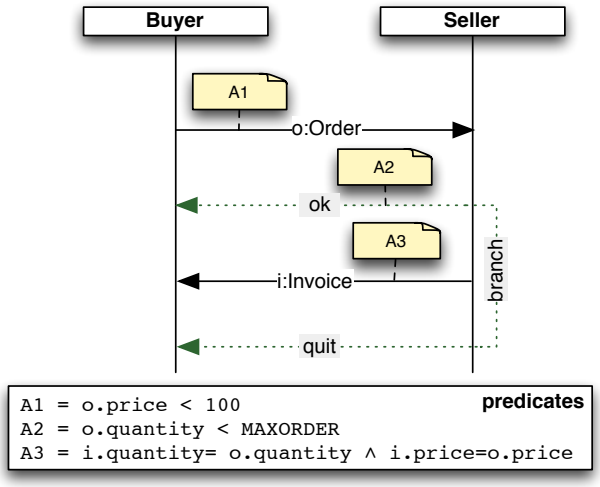

Fig. 2. Global assertion for the protocol *BuyerSeller*

than 100. By $A2$ the seller will select the branch ok only if the quantity requested by the order is less than the constant $MAXORDER$. $A3$ specifies the relationship between the data previously sent in the order and the data being sent by the seller in the invoice.

Protocol *Negotiation***.** The global assertion in Figure 3 models a negotiation between a buyer, a seller and a bank. The protocol starts with *Buyer* proposing a price *BPrice* to *Seller*. *Seller* has the following choices: (1- run) to continue, (2 - end) to terminate the negotiation. In the first case, *Seller* returns a price *SPrice* to *Buyer*. Then *Buyer* has the following choices: (1a - buy) to buy the item. In this case, *Buyer* authorizes *Bank* for the payment through the message *pay*, then *Bank* sends an acknowledgment to *Seller* through the message *paid*. (1b - rec) to execute another cycle of negotiation by recursively invoking *Negotiation* with, as parameters, the current values of *BPrice* and *SPrice*. (1c - stop) to *Buyer* terminate the negotiation.

The declaration *def* of the recursion defines recursion parameters (i.e., p_SPrice, p_BPrice) which are referred to in the recursion body. In this example, p_SPrice and p_BPrice take the value that $SPrice$ and $BPrice$ had in the previous instance of recursion (the one that invoked the current one). In the first instantiation of recursion, p_SPrice and p_BPrice have

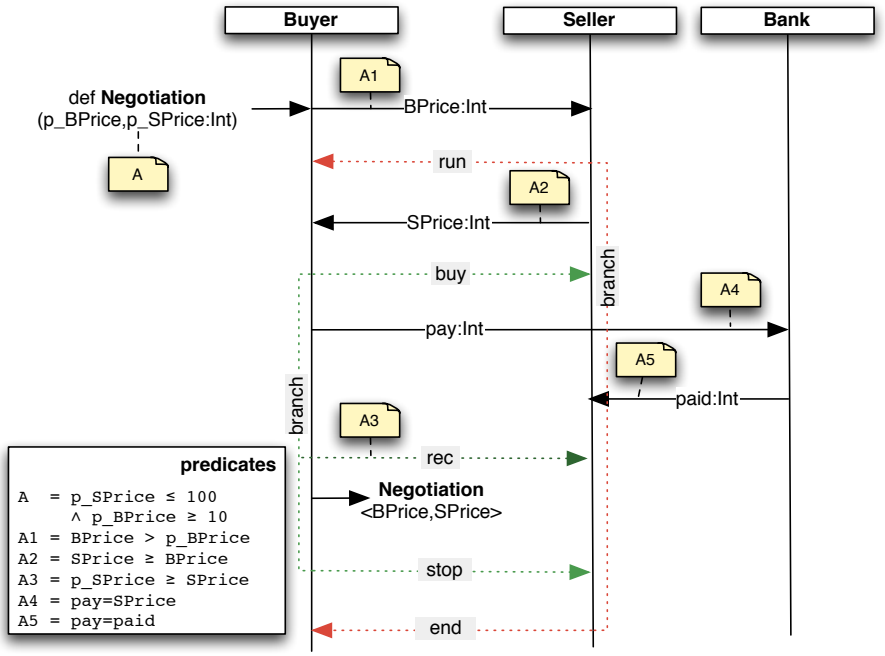

Fig. 3. Global assertion for the protocol *Negotiation*

to be initialized with values (e.g., 1 and 100, respectively). Note this allows us to compare the current interaction variables with those sent in the previous recursion instance (this corresponds to the use of hooked variables in assertion methods, e.g. [30]). For example, $A1$ ensures *Buyer* will increase the price in successive instances of negotiation[5], while $A4$ and $A5$ ensure that the correct amount is paid. The recursion invariant A ensures that the prices remain in the range $1 \ldots 100$.

Using Global Assertions. Global assertions are given a formal syntax with a well-formedness constraint, so that they represent consistent specifications (§ 3). Once specified, they can be projected onto end-point assertions, one for each role in the session (§ 4). A process, which also includes predicates for communication monitoring, can further be vali-

[5] Because of the partial correctness semantics we consider in the present work (cf. Definition 7.2), $A3$ does not ensure eventual termination (e.g. a process may go into an internal infinite loop).

dated against an end-point assertion through compositional proof rules. A validated process can meet its obligations, assuming that the remaining parties also do so. This property, which we call *error-freedom*, is a corollary of the soundness of validation rules with respect to its formal (partial correctness) semantics (Theorems 7.10 and 7.11 in § 7.2). In a trusted environment, where all participants can be assumed validated, there is no need for run-time monitoring on predicates to ensure the correct execution of the protocol (Proposition E.38 in § 7.3). Finally (relative to the decidability of the underlying logic) the proof rules induce a sound decidable validation algorithm against unasserted, but typed, programs which can statically guarantee their correct interactional behaviours (Proposition 7.13 in § 7.3).

$$
\begin{array}{llll}
\text{Global} & \mathcal{G} & ::= & \mathtt{p} \to \mathtt{p}' \colon k\ (\tilde{v} : \tilde{S})\{A\}.\mathcal{G}' & \text{values} \\
& & \mid & \mathtt{p} \to \mathtt{p}' \colon k\ \{\{A_j\} l_j \colon \mathcal{G}_j\}_{j \in J} & \text{branching} \\
& & \mid & \mathcal{G}, \mathcal{G}' & \text{parallel} \\
& & \mid & \mu \mathbf{t} \langle \tilde{e} \rangle (...v_i : S_i \,@\, \mathtt{L}_i ...)\{A\}.\mathcal{G} & \text{recursion} \\
& & \mid & \mathbf{t} \langle \tilde{e} \rangle & \text{variable} \\
& & \mid & \mathsf{end} & \mathsf{end} \\
\text{Sort} & S & ::= & \mathsf{bool} \mid \mathsf{nat} \mid \ldots \mid \mathcal{G} & \\
\text{Location} & \mathtt{L} & ::= & \{\mathtt{p}, \mathtt{p}'\} &
\end{array}
$$

Fig. 4. Syntax of Global Types with Assertions

3 Global Assertions

3.1 Syntax

Global assertions (range over by $\mathcal{G}, \mathcal{G}', \ldots$) elaborate global session types in [27] with logical formulae. The syntax is given in Figure 4. We let $u, v, \ldots$ range over *variables*, $a, b, c, \ldots$ range over *shared names*, and $s, s', \ldots$ range over *session channels*. First-order *expressions* are denoted by $e, e', \ldots$ and are built on *values* (ranged over by $\mathtt{n}, \mathtt{m}, \ldots$) which include *constants* (such as numerals and booleans) and *shared names*. A vector of zero or more variables (resp. basic types called *sorts*) is denoted by $\tilde{v}$ (resp. $\tilde{S}$) and we abbreviate $v_1 : S_1, \ldots, v_n : S_n$ (for a given natural number n) as $\tilde{v} : \tilde{S}$. A *location* $\mathtt{L}$ specifies the participants in an interaction. $A, B, \ldots$ range over logical formulae (Definition 3.2) and, when occurring in a global assertion, they are called *interaction predicate*. We write $\mathtt{p} \in \mathcal{G}$ if $\mathtt{p}$ occurs in $\mathcal{G}$.

$\mathtt{p} \to \mathtt{p}' \colon k\ (\tilde{v} : \tilde{S})\{A\}.\mathcal{G}'$ describes an *interaction* between a sender $\mathtt{p}$ and a receiver $\mathtt{p}'$ via channel k (represented as a natural number), followed by the interactions $\mathcal{G}'$. The logical variables $\tilde{v}$, called *interaction variables*, bind their occurrences in A and $\mathcal{G}'$, denoting potential message values. Intuitively, $\mathtt{p}$ *guarantees* the interaction predicate A (which constrains $\tilde{v}$) to $\mathtt{p}'$, whereas $\mathtt{p}'$ *relies* on A; in the traditional framework of pre/post conditions, A behaves like a precondition for $\mathtt{p}$ (since it is what $\mathtt{p}$ should ensure before sending) while the same A behaves as a post-condition for $\mathtt{p}'$ (since it is guaranteed to $\mathtt{p}'$ as a receiver).

$\mathtt{p} \to \mathtt{p}' \colon k\,\{\{A_j\} l_j \colon \mathcal{G}_j\}_{j \in J}$ says participant $\mathtt{p}$ sends one of the labels to channel k which is then received by participant $\mathtt{p}'$. Each branch j takes

place on the condition A_j, guaranteed by the sender and relied upon by the receiver. If j is chosen, the interactions described in $\mathcal{G}_j$ take place.

$\mathcal{G}, \mathcal{G}'$ represents the concurrent run of the interactions specified by $\mathcal{G}$ and $\mathcal{G}'$. end represents the end of a global session. We identify "$\mathcal{G}, \mathsf{end}$" and "$\mathsf{end}, \mathcal{G}$" with $\mathcal{G}$.

$\mu \mathbf{t} \langle \tilde{e} \rangle (\tilde{v_1} : \tilde{S_1} @ \mathtt{l}_1 \dots \tilde{v_n} : \tilde{S_n} @ \mathtt{l}_n) \{A\}.\mathcal{G}$ is a recursive type with parameters [19], annotated with logical formulae. We assume, for each $i, j \in \{1 \dots n\}$ s.t. $i \neq j$, $\tilde{v}_i$ and $\tilde{v}_j$ are pairwise disjoint and $L_i \neq L_j$. In $\mu \mathbf{t} \langle \tilde{e} \rangle (\tilde{v}_1 : \tilde{S}_1 @ \mathtt{l}_1 \dots \tilde{v}_n : \tilde{S}_n @ \mathtt{l}_n) \{A\}.\mathcal{G}$, we annotate variables with the set of participants that have either sent or received them (the location L_i represents such set of participants). Each $\tilde{v}_i$ groups variables used by L_i and has initial values $\tilde{e}_i$. $\tilde{v}_i$ act as binders over A and $\mathcal{G}$. We assume assertion variables $(\mathbf{t}, \mathbf{t}', \dots)$ are guarded by prefixes, i.e. the underlying recursive types are contractive. We often omit the annotation, writing $\mu \mathbf{t} \langle \tilde{e} \rangle (\tilde{v} : \tilde{S}) \{A\}.\mathcal{G}$ when clear from the context or irrelevant.

A recursive global assertion can be unfolded inductively to an infinite tree, as in the *equi-recursive* view on recursive types [41]. Unlike recursive types, such an unfolding includes parameter instantiation [19], making it less tractable to check equivalence. Intuitively a recursion and its unfolding denote an identical specification, but for simplicity we treat global assertions syntactically.[6]

If we erase interaction variables and predicates from global assertions, we obtain *global types* from [27]. The global type thus obtained from $\mathcal{G}$ is written $\mathsf{erase}(\mathcal{G})$.

Definition 3.1 (Coherence). We say $\mathcal{G}$ is *coherent* if $\mathsf{erase}(\mathcal{G})$ is coherent in the sense of [27, Definition 4.2]. *Hereafter, we assume all global assertions considered to be coherent.*

Coherence in [27, Definition 4.2] ensures that a global type only describes interactions which follow linear channel usage. For example,

$$\mathtt{p} \to \mathtt{q} : k(u)\{A\}.\mathtt{r} \to \mathtt{q} : k(v)\{B\}.\mathsf{end} \tag{3.1}$$

does not use k linearly, since two interactions are causally unordered (because they have two distinct senders, following Lamport's principle [33]: see [27, §3] for details).

Finally, we give the syntax of interaction predicates as formulae of predicate calculus with equality [36, §2.8] (which includes equality of shared names).

[6] This does not lose generality for e.g. validation because of our treatment of the semantics of endpoint assertions, cf. § 7.

Definition 3.2 (Logical Language). *The grammar of* logical formulae *or* predicates *$(A, B, \ldots)$ is given as:*

$$\begin{array}{rcl} A & ::= & e_1 = e_2 \mid e_1 > e_2 \mid \phi(e_1, \ldots, e_n) \\ & \mid & A_1 \wedge A_2 \mid \neg A \mid \exists v(A) \end{array}$$

where $e_1, ..$ are expressions and $\phi, \phi', ..$ range over a pre-defined set of predicates with fixed arities and types. We let $var(A)$ denote the set of free variables *of A, similarly for $var(e)$.*

Convention 3.3. Henceforth, we assume a fixed model for logical formulae satisfying: (1) validity of each closed atomic formula including equality and inequality is polynomially decidable; and (2) validity of closed formulae is decidable.

Though different applications may need different logical languages, this criteria or its minor variations are natural in the present engineering context. A concrete example which fits these criteria is number theory without multiplication, Presburger arithmetic. In practice such restricted logic is often sufficiently expressive: for example, many practical uses of multiplication are encodable [24], and formulae with quantifiers may be calculated efficiently [39, 43].

Example 3.4 (Negotiation Protocol in a Global Assertion). The following is the same negotiation protocol in the sequence diagram in § 2 with $\mathtt{L} = \{Buyer, Seller\}$. A branch does not mention the predicate true.

$$\begin{array}{rl} \mathcal{G}_{neg} = & \mu \mathbf{t}\langle 10, 100\rangle(p\_BPrice : Int, p\_SPrice : Int\,@\mathtt{L})\{A\}. \\ & \mathtt{Buyer} \to \mathtt{Seller}\colon s\,(BPrice : Int)\{A1\}. \\ & \mathtt{Seller} \to \mathtt{Buyer}\colon b\{\mathsf{run} : \mathcal{G}_{run}, \mathsf{end} : \mathsf{end}\} \\ \mathcal{G}_{run} = & \mathtt{Seller} \to \mathtt{Buyer}\colon b\,(SPrice : Int)\{A2\}. \\ & \mathtt{Buyer} \to \mathtt{Seller}\colon s \\ & \{\mathsf{buy} : \mathcal{G}_{buy}, \{A3\}\mathsf{rec} : \mathbf{t}\langle BPrice, SPrice\rangle, \mathsf{stop} : \mathsf{end}\} \\ \mathcal{G}_{buy} = & \mathtt{Buyer} \to \mathtt{Bank}\colon n\,(pay : Int)\{A4\}. \\ & \mathtt{Bank} \to \mathtt{Seller}\colon s\,(paid : Int)\{A5\}.\mathsf{end} \end{array}$$

$$\begin{array}{ll} A1 \equiv BPrice > p\_BPrice & A2 \equiv SPrice \geqslant BPrice \\ A3 \equiv p\_SPrice \geqslant SPrice & A4 \equiv pay = SPrice \\ A5 \equiv pay = paid & A \equiv p\_SPrice \leqslant 100 \wedge p\_BPrice \geqslant 10 \end{array}$$

$\mathcal{G}_{neg}$ has recursion parameters p_bPrice and p_sPrice denoting the prices proposed by *Buyer* and *Seller* in the previous recursion instance and they are 10 and 100 in the first instance. The annotation locates p_bPrice and p_sPrice in the participants *Buyer* and *Seller* who know the actual values (i.e., $bPrice$ and $sPrice$).

3.2 Consistency Principles

In Example 3.4, the predicates specify constraints on communication events following the natural temporal sequence of interactions. For example, the final message from *Bank* is constrained by predicate $paid = pay$. *Bank* can surely check the validity of the predicate since it has directly received *pay* in a preceding interaction. The underlying principle is *history-sensitivity*:

> *An interaction predicate guaranteed by a participant is defined only on interaction variables introduced in the preceding interactions in which the participant is involved.*

There is another principle implicit in this simple interaction predicate; it does not *retrospectively* add a constraint to previous messages (such as *pay* should be say 100 euros): if we neglect the newly introduced variable *paid*, no additional constraint is given to the preceding variables such as *pay*. This *locality* principle says:

> *An interaction formula should only add constraints to the variables it introduces.*

As discussed later, this point is related to a third principle, *temporal satisfiability*, stating that a process can always find one valid forward path at each interaction point, until it meets the session end.

Convention 3.5. From now on, we always assume the standard bound name convention for all syntactic entities with bindings.

Definition 3.6. Write $I(\mathcal{G})$ for the set of interaction variables occurring in $\mathcal{G}$. We say a participant $\mathtt{p}$ *knows* an interaction variable $u \in \mathcal{I}(\mathcal{G})$ if $\mathtt{p}$ uses u in an interaction of $\mathcal{G}$ or if u is a parameter of a recursive definition whose location $\mathtt{L}$ contains $\mathtt{p}$; the set of variables that $\mathtt{p} \in \mathcal{G}$ knows is denoted by $\mathcal{I}(\mathcal{G}) \upharpoonright \mathtt{p}$.

We first introduce a simple compositional checker for *history-sensitivity*. We use environments Γ defined by the grammar:

$$\Gamma \ ::= \ \varnothing \ \mid \ \Gamma, u : S \,@\, \mathtt{L} \ \mid \ \Gamma, \mathbf{t} : S_1 \,@\, \mathtt{L}_1 \ldots S_n \,@\, \mathtt{L}_n$$

which assigns to an interaction variable say u its sort along with its location $S \,@\, \mathtt{L}$. We write $\Gamma \vdash u \,@\, \mathtt{p}$ when $\mathtt{p} \in \Gamma(u)$ and $\Gamma \vdash e \,@\, \mathtt{p}$ when $\Gamma \vdash u \,@\, \mathtt{p}$ for all $u \in var(e)$. This environment also maps types variables $\mathbf{t}$ to a sequence of pairs $S_i \,@\, \mathtt{L}_i$ to handle recursive types.

$$\frac{\Gamma, \tilde{v} : \tilde{S} \,@\, \{\mathtt{p}, \mathtt{p}'\} \vdash \mathcal{G} \quad \forall u \in var(A) \text{ not in } \tilde{v}, \Gamma \vdash u \,@\, \mathtt{p}}{\Gamma \vdash \mathtt{p} \to \mathtt{p}' \colon k\,(\tilde{v} : \tilde{S})\{A\}.\mathcal{G}} \quad [\textsc{Val}]$$

$$\frac{\forall j \in J,\ \Gamma \vdash \mathcal{G}_j \quad \forall u \in \bigcup_{j \in J} var(A_j) \text{ not in } \tilde{v}, \Gamma \vdash u \,@\, \mathtt{p}}{\Gamma \vdash \mathtt{p} \to \mathtt{p}' \colon k\,\{\{A_j\} l_j \colon \mathcal{G}_j\}_{j \in J}} \quad [\textsc{Branch}]$$

$$\frac{\Gamma \vdash \mathcal{G} \quad \mathcal{G} \vdash \mathcal{G}'}{\Gamma \vdash \mathcal{G}, \mathcal{G}'} \qquad \frac{}{\Gamma \vdash \mathsf{end}} \quad [\textsc{Par}]/[\textsc{End}]$$

$$\frac{\Gamma \vdash e_1 : S_1 \,@\, \mathtt{L}_1 \cdots \Gamma \vdash e_n : S_n \,@\, \mathtt{L}_n}{\Gamma, \mathbf{t} : S_1 \,@\, \mathtt{L}_1 \ldots S_n \,@\, \mathtt{L}_n \vdash \mathbf{t}\langle \tilde{e} \rangle} \quad [\textsc{Rec}]$$

$$\frac{\begin{array}{c}\Gamma' = \Gamma, \mathbf{t} : S_1 \,@\, \mathtt{L}_1 \ldots S_n \,@\, \mathtt{L}_n \quad \Gamma' \vdash \mathcal{G} \\ dom(\Gamma') \supseteq var(A) \quad \forall i . \Gamma \vdash v_i : S_i \,@\, \mathtt{L}_i, e_i : S_i \,@\, \mathtt{L}_i\end{array}}{\Gamma \vdash \mu \mathbf{t}\langle \tilde{e} \rangle (v_1 : S_1 \,@\, \mathtt{L}_1, \ldots, v_n : S_n \,@\, \mathtt{L}_n)\{A\}.\mathcal{G}} \quad [\textsc{Def}]$$

Fig. 5. Well-formedness for Global Assertions

Well-formedness for global assertions (defined by the rules in Figure 5) disciplines the usage of assertion variables and restricts the set of interaction variables that can be used in each assertion, following the history sensitivity principle. The key rules are [Val] and [Branch] which require that the participant who sends/selects must know all the interaction variables onto which the assertion A predicates. Other rules are straightforward. Note the rules are purely syntactic, hence the validation of $\mathcal{G}$ is a linear time problem.

3.3 Well Asserted Global Assertions

On the basis of well-formedness, we introduce compositional validation rules by which a global assertion now conforms to the remaining two principles, *locality* and *temporal satisfiability*. Since both are about logical satisfiability of formulae, the validation this time should include validity checking.

To clarify the goal of the validation, consider the following global assertion:

$$\begin{array}{l}\mathcal{G} = \mathtt{Alice} \to \mathtt{Bob} \colon b\,(u : Int)\{u < 10\}. \\ \qquad \mathtt{Bob} \to \mathtt{Alice} \colon a\,(v : Int)\{u > v \wedge v > 6\}.\mathsf{end}\end{array}$$

This global assertion *is* satisfiable if the predicates are considered as an ensemble (i.e., $\exists u, v(u < 10 \wedge u > v \wedge v > 6)$), but is indeed problematic from a temporary perspective: a process faithfully following this specification step by step can block. For example if *Alice* sends $u = 6$, which does not violate $u < 10$, then *Bob* will not be able to find a value that satisfies $6 > v \wedge v > 6$. The criteria here can be expanded as:

> *For each possible value that satisfies a predicate A, it is possible, for each interaction predicate A' that appear after A, to find values satisfying A'.*

For a branching point, predicates are taken disjunctively (see Def. 3.7 below). Note this criteria is equivalent to the *temporal satisfiability* discussed above: but it also caters for *locality* since the violation of locality means that a sender can unexpectedly meet an unsatisfiable predicate, as discussed in the example above. Below we pre-annotate each occurrence of a type variable $\mathbf{t}$ with a predicate associated to the recursion that defines it together with its formal parameters $\tilde{v}$, writing e.g. $\mathbf{t}_{A(\tilde{v})}$. Note this annotation is always possible if $\mathbf{t}$ is bound in a whole global type, i.e. if the whole global type is closed.

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

1. $\mathcal{G} = \mathtt{p}_1 \to \mathtt{p}_2 \colon k\,(\tilde{v} : \tilde{S})\{A'\}.\mathcal{G}'$
   $\begin{cases} \text{if } A \supset \exists \tilde{v}(A') \text{ then } GSat(\mathcal{G}, A) = GSat(\mathcal{G}', A \wedge A') \\ \text{otherwise } GSat(\mathcal{G}, A) = \mathsf{false} \end{cases}$
2. $\mathcal{G} = \mathtt{p}_1 \to \mathtt{p}_2 \colon k\,\{\{A_j\}l_j \colon \mathcal{G}_j\}_{j \in J}$ with $J = \{1, \ldots, n\}$
   $\begin{cases} \text{if } A \supset (A_1 \vee \ldots \vee A_n) \text{ then } GSat(\mathcal{G}, A) = \\ \qquad GSat(\mathcal{G}_1, A \wedge A_1) \wedge \ldots \wedge GSat(\mathcal{G}_n, A \wedge A_n) \\ \text{otherwise } GSat(\mathcal{G}, A) = \mathsf{false} \end{cases}$
3. $\mathcal{G} = \mathcal{G}_1, \mathcal{G}_2$ then $GSat(\mathcal{G}, A) = GSat(\mathcal{G}_1, A) \wedge GSat(\mathcal{G}_2, A)$
4. $\mathcal{G} = \mu \mathbf{t} \langle \tilde{e} \rangle (\tilde{v} : \tilde{S})\{A'\}.\mathcal{G}'$
   $\begin{cases} \text{if } A \supset A'[\tilde{e}/\tilde{v}] \text{ then } GSat(\mathcal{G}, A) = GSat(\mathcal{G}', A \wedge A') \\ \text{otherwise } GSat(\mathcal{G}, A) = \mathsf{false} \end{cases}$
5. $\mathcal{G} = \mathbf{t}_{A'(\tilde{v})} \langle \tilde{e} \rangle$ then $GSat(\mathcal{G}, A) = \mathsf{true}$ provided that $A \supset A'[\tilde{e}/\tilde{v}]$
6. $\mathcal{G} = \mathsf{end}$ then $GSat(\mathcal{G}, A) = \mathsf{true}$

$\mathcal{G}$ is *well asserted* if it is well-formed and $GSat(\mathcal{G}, \mathsf{true}) = \mathsf{true}$

Notice that $GSat(\mathcal{G}, \mathsf{false}) = \mathsf{true}$ (i.e., a global assertion is trivially satisfiable if provided with a bad environment). The boolean function incrementally builds the conjunction of all the predicates that precedes 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) we require 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). For example, the following protocol is well-asserted even if the first branch is unsatisfiable $\neg\exists v(v > 0 \wedge v < 0)$ since the second branch is satisfiable:

$$\begin{aligned}\mathcal{G} = {} & \mathtt{Alice} \to \mathtt{Bob}\colon b\,(v : Int)\{v > 0\}. \\ & \mathtt{Bob} \to \mathtt{Alice}\colon a\,\{\{v < 0\}l_1 \colon \mathcal{G}_1, \{v > 0\}l_2 \colon \mathcal{G}_2\}\end{aligned}$$

Notice we do not specify any relationship among the assertions in a branch (such as a XOR relationship in order to enforce a determinism in potential paths) because we wish to allow specifications to be as vague as one wants (as far as consistent), unlike a programming construct [26]. For the same reason, the notion of well-assertedness does not prohibit unreachable branches. The remaining clauses of Definition 3.7 are intuitive: in (4) (resp. (5)), we require A to imply the satisfiability of the invariant with the initialization parameters (resp. with the parameters assigned by the invocation inside the recursion). Finally we observe:

- Under Convention 3.3, well-assertedness is decidable. The fixed shape of the implications (e.g. no nested quantifiers) may suggests a potential for efficient checking [43].
- Definition 3.7 uses syntactic precedence for capturing temporal precedence. As seen in (3.1) before, this is an over-approximation. Under the coherence of $\mathcal{G}$ which gives a clear notion of causality among actions (cf. [27, §3]), we can optimise the calculation in Definition 3.7 by considering all and only truly causally preceding actions (by IO-chains, [27, §3.2]), with exactly the same semantic result. For detailed discussions on semantic characterisations of the well-formedness and well-assertedness, as well as concrete examples of (violation of) these conditions, see Appendix B.

$$
\begin{array}{rcll}
\text{Local } \mathcal{T} & ::= & k!(\tilde{v} : \tilde{S})\{A\}; \mathcal{T} & \text{send} \\
& | & k?(\tilde{v} : \tilde{S})\{A\}; \mathcal{T} & \text{receive} \\
& | & k \oplus \{\{A_j\} l_i : \mathcal{T}_i\}_{i \in I} & \text{selection} \\
& | & k \& \{\{A_i\} l_i : \mathcal{T}_i\}_{i \in I} & \text{branching} \\
& | & \mu \mathbf{t} \langle \tilde{e} \rangle (\tilde{v} : \tilde{S})\{A\}.\mathcal{T} & \text{def} \\
& | & \mathbf{t} \langle \tilde{e} \rangle & \text{recursion} \\
& | & \mathsf{end} & \text{end}
\end{array}
$$

Fig. 6. Syntax of Endpoint Assertions

4 End-point Assertions and Projection

4.1 Syntax

In this section we present *end-point assertions* which specify the behavioural contract from the perspective of a single participant. Just as a global assertion annotates a global session type with logical formulae, an endpoint assertion similarly annotates a local session type [27, §4], allowing for a wide range of specifications. The highlight is the projection of global assertions onto end-point assertions where the projected logical formulae capture indirect constraints for a receiver, which are essential for the projected endpoint assertions to fully capture the original global assertion. We leave the formal semantics of endpoint assertions to §7.

End-point assertions (ranged over by $\mathcal{T}, \mathcal{T}', ..$) specify endpoint behaviour of processes on a *per-session* basis. The grammar is given in Figure 6. All constructs come from local session types in [27], adding annotations on logical formulae.

$k!(\tilde{v} : \tilde{S})\{A\}; \mathcal{T}$ specifies a sender should guarantee that values of type $\tilde{S}$ denoted by logical variables $\tilde{v}$ to be sent via k, should satisfy A, then behaves as specified in $\mathcal{T}$. As before, $\tilde{v}$ and A are called *interaction variables* and *interaction predicate*, respectively. Dually, $k?(\tilde{v} : \tilde{S})\{A\}; \mathcal{T}$ says the receiving party can rely on the arriving values, denoted by $\tilde{v}$, to satisfy the interaction predicate A.

$k \oplus \{l_i : T_i\}_{i \in I}$ specifies for a selection where, for each i, the process should guarantee A_i when selecting l_i via k. Dually $k \& \{l_i : T_i\}_{i \in I}$ describes a branching at k where, if l_i is to be chosen, it can assume A_i to hold.

$\mu \mathbf{t}\langle \tilde{e} \rangle (\tilde{v} : \tilde{S})\{A\}.\mathcal{T}$ specifies constraints for a recursive session using parameters [19] given by $\tilde{v}$ of sort $\tilde{S}$ with the initial values $\tilde{e}$. The use of parameters makes the specification highly expressive (semantically it denotes a maximal fixed point, cf. §7).

Continuing Example 3.4 (the global assertion $\mathcal{G}_{neg}$), the next example illustrates the importance of projecting indirect constraints of interactions between other participants.

Example 4.1 (Seller's Endpoint Assertion). A seller's specification from $\mathcal{G}_{neg}$ but without projecting indirect constraints (i.e., those associated to interactions between third parties) results in the following end-point assertion (with A and $A1$,$A2$,$A3$ from Example 3.4):

$$\begin{aligned}
\mathcal{T}_{negBad} &= \mu \mathbf{t}\langle 10, 100 \rangle (p_BPrice : Int, p_SPrice : Int)\{A\}. \\
&\qquad s?(BPrice : Int)\{A1\}; b \oplus \{\mathsf{run} : \mathcal{T}_{run}, \mathsf{end} : \mathsf{end}\} \\
\mathcal{T}_{runBad} &= b!(SPrice : Int)\{A2\}; s\&\{\{\mathsf{true}\}\mathsf{buy} : \mathcal{T}_{conf}, \\
&\qquad\qquad \{A3\}\mathsf{rec} : \mathbf{t}\langle BPrice, SPrice \rangle, \{\mathsf{true}\}\mathsf{stop} : \mathsf{end}\} \\
\mathcal{T}_{confBad} &= s?(paid : Int)\{\mathsf{true}\}; \mathsf{end}
\end{aligned}$$

In $\mathcal{T}_{confBad}$, *Seller* receives *paid* which is constrained by predicate $paid = pay$, but *Seller* cannot use the predicate since s/he has neither sent nor received *pay*. A method by which we can automatically allows *Seller* to infer a meaningful constraint using this indirect predicate is given in Definition 4.2 below.

4.2 Projection

We now define the projection of a global assertion to end-point assertions at each participant.

Definition 4.2 (Projection). *We inductively define* $Proj(\mathcal{G}, A, \mathtt{p})$ *where* $\mathtt{p}$ *occurs in* $\mathcal{G}$ *assuming* $\mathtt{p}_1 \neq \mathtt{p}_2$ *by:*

1. $Proj(\mathtt{p}_1 \to \mathtt{p}_2 : k\,(\tilde{v} : \tilde{S})\{A\}.\mathcal{G}', A_{Proj}, \mathtt{p}) =$
$$\begin{cases} k!(\tilde{v} : \tilde{S})\{A\}.\mathcal{G}_{Proj} & \text{if } \mathtt{p} = \mathtt{p}_1 \\ k?(\tilde{v} : \tilde{S})\{\exists V_{ext}(A \wedge A_{Proj})\}.\mathcal{G}_{Proj} & \text{if } \mathtt{p} = \mathtt{p}_2 \\ \mathcal{G}_{Proj} & \text{otw} \end{cases}$$
with $\mathcal{G}_{Proj} = Proj(\mathcal{G}', A \wedge A_{Proj}, \mathtt{p})$
and $V_{ext} = var(A_{Proj}) \backslash \mathcal{I}(\mathcal{G}) \upharpoonright \mathtt{p}$.

2. $Proj(\mathtt{p}_1 \to \mathtt{p}_2 \colon k\,\{\{A_i\}l_i \colon \mathcal{G}_i\}_{i\in I}, A_{Proj}, \mathtt{p}) =$
$$\begin{cases} k \oplus \{\{A_i\}l_i \colon \mathcal{G}^i_{Proj}\}_{i\in I} & \text{if } \mathtt{p} = \mathtt{p}_1 \\ k\&\{\{\exists V_{ext}(A_i \wedge A_{Proj})\}l_i \colon \mathcal{G}^i_{Proj}\}_{i\in I} & \text{if } \mathtt{p} = \mathtt{p}_2 \\ \mathcal{G}^1_{Proj} & \text{if } \mathtt{p} \neq \mathtt{p}_2,\ \mathtt{p} \neq \mathtt{p}_1 \text{ and} \\ & \forall i, j \in I, \mathcal{G}^i_{Proj} = \mathcal{G}^j_{Proj} \end{cases}$$
with $\mathcal{G}^i_{Proj} = Proj(\mathcal{G}_i, A_i \wedge A_{Proj}, \mathtt{p})$
and $V_{ext} = var(A_{Proj}) \backslash \mathcal{I}(\mathcal{G}) \upharpoonright \mathtt{p}$.

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

4. $Proj(\mu\mathbf{t}\langle\tilde{e}\rangle(\tilde{v} : \tilde{S})\{A\}.\mathcal{G}, A_{Proj}, \mathtt{p}) =$
$\mu\mathbf{t}\langle\tilde{e}'\rangle(\tilde{v}' : \tilde{S}')\{\exists V_{ext}(A)\}.Proj(\mathcal{G}, A \wedge A_{Proj}, \mathtt{p})$
where $\tilde{v}'$ *are the restriction of* $\tilde{v}$ *to those in* $\mathcal{I}(\mathcal{G}) \upharpoonright \mathtt{p}$,
V_{ext} *is the variables not located at* $\mathtt{p}$, *i.e.* $V_{ext} = var(A) \backslash (\mathcal{I}(\mathcal{G}) \upharpoonright \mathtt{p})$,
and $\tilde{e}'$ *are the restriction of* $\tilde{e}$ *to those located at* $\mathtt{p}$.

5. $Proj(\mathbf{t}\langle\tilde{e}\rangle, A_{Proj}, \mathtt{p}) = \mathbf{t}\langle\tilde{e}'\rangle$
where $\tilde{e}'$ *are all the expressions in* $\tilde{e}$ *such that* $var(e'_i) \subseteq \mathcal{I}(\mathcal{G}) \upharpoonright p$.

6. $Proj(\mathsf{end}, A_{Proj}, \mathtt{p}) = \mathsf{end}$

When a side condition does not hold the $Proj(\mathcal{G}, A, \mathtt{p})$ *is undefined. The* projection of $\mathcal{G}$ onto $\mathtt{p}$, *written* $\mathcal{G} \upharpoonright \mathtt{p}$, *is defined as* $Proj(\mathtt{p}, \mathcal{G}, \mathsf{true})$.

Remark 4.3. In (4) above, by well-formedness of $\mathcal{G}$ we have $var(e'_i) \subseteq \mathcal{I}(\mathcal{G}) \upharpoonright p$, hence all bound names in the original global assertion are kept bound in the resulting projection.

The assertion projection defined above is identical to the type projection in [27] if we erase logical formulae. Below we focus on the treatment of logical formulae, leaving other aspects to [27].

In (1) we project to either a send or receive (or a continuation). For send, the projection of a predicate A consists of A itself. For receive, we consider that the other participants may in general not be trusted. The receiver, by verifying that only the current predicate is not violated, is not able to verify that some violation did not occur during interactions between third parties. Let us consider the following global assertion which is well-asserted:

$$\begin{aligned} \mathcal{G} = {} & \mathtt{Seller} \to \mathtt{Buyer} \colon b\,(price : Int)\{price > 10\}. \\ & \mathtt{Buyer} \to \mathtt{Bank} \colon c\,(pay : Int)\{pay \geqslant price\}.\mathsf{end} \end{aligned}$$

The predicate $pay \geqslant price$ alone is not meaningful to $Bank$ since $Bank$ doesn't know the value of $price$. In other words, $Bank$ cannot verify the value of pay respects the contract only through $pay \geqslant price$. For this reason, we should project, to $Bank$, also the predicate of the past interactions between $Buyer$ and $Seller$:

$$\mathcal{G} \upharpoonright \texttt{Bank} = c?(pay : Int)\{\exists price(price > 10 \wedge pay \geqslant price)\}; \qquad (4.1)$$

In general, we project to $\texttt{p}$ all the past predicates, even if associated to interactions in which $\texttt{p}$ was not involved. [7] The existential quantification binds the variables that the participants do not know (i.e., they are not in $\mathcal{I}(\mathcal{G}) \upharpoonright \texttt{p}$). The aim of projecting the past predicates in input and branching is twofold. First, we want to enable the receiver to detect violations in interactions in which it was not involved. Second, we want to provide each participant with the strongest set of precondition possible, so to avoid the burden of the so-called defensive programming (e.g., in the implementation of $Bank$ a programmer may concentrate on the case $pay \leqslant 10$).

In (2) we project a branch global assertion onto a branching, a selection or the projection of the continuation, following [27]. The ways predicates are handled follow (1).

In (3), as in [27], we require that each participant is contained in at most a single global assertion in a parallel composition to ensure that each global assertion is single threaded.

In (4) the projection of a recursion definition to $\texttt{p}$ consists of the recursion definition itself where we project only the recursion parameters that are known to $\texttt{p}$. As in send and branch, the projected predicate contains also all the predicates that have been collected in the past, even if associated to interactions in which $\texttt{p}$ was not involved. The existential quantification binds the variables that participants do not know (i.e., they are not in $\mathcal{I}(\mathcal{G}) \upharpoonright \texttt{p}$).

In (5) the projection of a recursive call is the recursive call itself where we project only the recursion parameters known to $\texttt{p}$.

4.3 Well-Asserted Endpoint Assertions

Finally we show the projection preserves the basic principles discussed in §3.2, *locality* and *temporal satisfiability* (*history-sensitivity* is vacuous

[7] The resulting formulae become substantially more compact by taking only causally preceding actions. The result is equivalent.

since all actions are now about a single participant). The well-assertedness for end-point assertion is defined inductively using the function:

$$LSat(\mathcal{T},\ A)$$

which says $\mathcal{T}$ is satisfiable under A, which is defined as follows. Apart from the difference in the shape of syntax, the rules exactly follow those of Definition 3.7. We use the annotation $\mathbf{t}_{A(\tilde{v})}$ as before.

Definition 4.4 (Well Asserted End-point Assertions). *We define a boolean function $LSat(\mathcal{T}, A)$ recursively as follows:*

1. $\mathcal{T} = k!(\tilde{v} : \tilde{S})\{A'\}; \mathcal{T}'$ *or* $k?(\tilde{v} : \tilde{S})\{A'\}; \mathcal{T}'$
$$\begin{cases} \textit{if } A \supset \exists \tilde{v}(A') \textit{ then } LSat(\mathcal{T}, A) = LSat(\mathcal{T}', A \wedge A') \\ \textit{otherwise } LSat(\mathcal{T}, A) = \mathsf{false} \end{cases}$$
2. $\mathcal{T} = k \oplus \{\{A_j\}l_j : \mathcal{T}_j\}_{j\in J}$ *or* $\mathcal{T} = k\&\{\{A_j\}l_j : \mathcal{T}_j\}_{j\in J}$ *with* $j = 1, \ldots, n$
$$\begin{cases} \textit{if } A \supset (A_1 \vee \ldots \vee A_n) \textit{ then } LSat(\mathcal{T}, A) = \\ \qquad LSat(\mathcal{T}_1, A \wedge A_1) \wedge \ldots \wedge LSat(\mathcal{T}_n, A \wedge A_n) \\ \textit{otherwise } LSat(\mathcal{T}, A) = \mathsf{false} \end{cases}$$
3. $\mathcal{T} = \mu\mathbf{t}\langle\tilde{e}\rangle(\tilde{v} : \tilde{S})\{A'\}.\mathcal{T}'$
$$\begin{cases} \textit{if } A \supset A'[\tilde{e}/\tilde{v}] \\ \qquad \textit{then } LSat(\mathcal{T}, A) = LSat(\mathcal{T}', A \wedge A') \\ \qquad \textit{otherwise } LSat(\mathcal{T}, A) = \mathsf{false} \end{cases}$$
4. $\mathcal{T} = \mathbf{t}_{A'(\tilde{v})}\langle\tilde{e}\rangle$ *then* $LSat(\mathcal{T}, A) = A \supset A'[\tilde{e}/\tilde{v}]$
5. $\mathcal{T} = \mathsf{end}$ *then* $LSat(\mathcal{T}, A) = \mathsf{true}$

We say $\mathcal{T}$ is well-asserted *if* $LSat(\mathcal{T}, \mathsf{true}) = \mathsf{true}$

When $LSat(\mathcal{T}, \mathsf{true})$ holds, $\mathcal{T}$ is *well-asserted.*

Lemma 4.5. *If $\mathcal{G}$ is well-formed then, for all predicates $A_\mathcal{G}, A_\mathcal{T}$ such that $A_\mathcal{T} \supset A_\mathcal{G}$ and all $\mathtt{p} \in \mathcal{G}$*

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

Proof. By structural induction of $\mathcal{G}$. For example, in the case of recursion, if we project $\mu\mathbf{t}\langle\tilde{e}\rangle(\tilde{v} : \tilde{S})\{A\}.\mathcal{G}'$ to $\mathtt{p}$ we get $\mu\mathbf{t}\langle\tilde{e}'\rangle(\tilde{v}' : \tilde{S}')\{\exists V_{ext}(A)\}.\mathcal{T}'$. By induction hypothesis we have $A_\mathcal{T} \supset A_\mathcal{G}$ and by well-assertedness of $\mathcal{G}$ we have $A_\mathcal{G} \supset A[\tilde{e}/\tilde{v}]$ hence $A_\mathcal{T}$ entails $A[\tilde{e}/\tilde{v}]$. But by the standard axiom for quantification, and writing $\tilde{w}$ to be the restriction of $\tilde{v}$ to V_{ext}, we know $A[\tilde{e}/\tilde{v}]$ entails $(\exists\tilde{w})A[\tilde{e}'/\tilde{v}']$ which immediately entails $(\exists V_{ext})A[\tilde{e}'/\tilde{v}']$, as required. See Appendix C.2 for the full proof. □

Lemma 4.5 shows that *GSat* implies *LSat* assuming that the set of predicates of the global assertion imply those of the end-point assertion (intuitively, the predicates in global assertions are projected by adding existential quantifiers which make them weaker).

Corollary 4.6 (Well-assertedness of Projections). *Let* $\mathcal{G}$ *be a well asserted coherent global assertion then for each* $p \in \mathsf{pid}(\mathcal{G})$*, if* $\mathcal{G} \upharpoonright \mathtt{p}$ *is defined then* $\mathcal{G} \upharpoonright \mathtt{p}$ *is also well-asserted.*

$$
\begin{array}{rll}
P & ::= \overline{a}_{[2..n]}\,(\tilde{s}).P \mid a_{[p]}\,(\tilde{s}).P & \text{session request/acceptance} \\
& \mid s!\langle\tilde{e}\rangle(\tilde{v})\{A\};P & \text{value sending} \\
& \mid s?(\tilde{v})\{A\};P & \text{value reception} \\
& \mid s \lhd \{A\}l;P & \text{label selection} \\
& \mid s \rhd \{\{A_i\}l_i\colon P_i\}_{i\in I} & \text{label branching} \\
& \mid \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q & \text{conditional branch} \\
& \mid \mathsf{errH} \mid \mathsf{errT} & \text{error} \\
& \mid P \mid Q \mid \mathbf{0} \mid s\colon\tilde{h} & \text{parallel/idle/msg queue} \\
& \mid (\nu a)P \mid (\nu\tilde{s})P & \text{name/channel hiding} \\
& \mid \mathsf{def}\ D\ \mathsf{in}\ P \mid X\langle\tilde{e}\tilde{s}\rangle & \text{recursion def/call} \\
e & ::= n \mid e\ \mathsf{and}\ e' \mid \mathsf{not}\ e \quad \ldots & \text{expressions} \\
n & ::= a \mid \mathsf{true} \mid \mathsf{false} & \text{values} \\
h & ::= l \mid \tilde{n} \mid \tilde{s} & \text{messages-in-transit} \\
D & ::= \{\langle X_i(\tilde{v}_i\tilde{s}_i) = P_i\rangle\}_{i\in I} & \text{declaration for recursion}
\end{array}
$$

Fig. 7. Syntax of Asserted Processes

5 Compositional Validation of Processes

5.1 The π-Calculus with Assertions

We use the π-calculus with multiparty session initializations from [27, §2], augmented with predicates associated to communication actions, in correspondence with end-point assertions. A processes with predicates is intended to model the behaviour of a concurrent end-point program combined with a local communication monitor, which checks at runtime whether or not each message in both directions satisfy a stipulated contract.[8]

Asserted processes or often simply *processes* $(P, Q, \ldots)$ are given by the grammar in Figure 7. The only differences from [27] is the addition of

[8] These logical formulae can also be considered as part of the execution of a program, supplied by a designer/programmer for e.g. debugging, cf. §8.

$$\overline{a}_{[2..\mathtt{n}]}\,(\tilde{s}).P \mid a_{[2]}\,(\tilde{s}).P_2 \mid \ldots \mid a_{[\mathtt{n}]}\,(\tilde{s}).P_n \to \quad \text{[R-Link]}$$
$$(\nu \tilde{s})(P_1 \mid P_2 \mid \ldots \mid P_n \mid s_1 : \varnothing \mid \ldots \mid s_n : \varnothing)$$
$$s!\langle \tilde{e} \rangle(\tilde{v})\{A\}; P \mid s : \tilde{h} \to P[\tilde{\mathtt{n}}/\tilde{v}] \mid s_k : \tilde{h} \cdot \mathtt{n} \quad (\tilde{e} \downarrow \tilde{\mathtt{n}} \wedge A[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{true}) \quad \text{[R-Send]}$$
$$s?(\tilde{v})\{A\}; P \mid s : \mathtt{n} \cdot \tilde{h} \to P[\tilde{\mathtt{n}}/\tilde{v}] \mid s : \tilde{h} \quad (A[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{true}) \quad \text{[R-Recv]}$$
$$s \rhd \{\{A_i\} l_i \colon P_i\}_{i \in I} \mid s : l_j \cdot \tilde{h} \to P_j \mid s : \tilde{h} \quad (j \in I \text{ and } A_j \downarrow \mathsf{true}) \quad \text{[R-Branch]}$$
$$s \lhd \{A\} l; P \mid s : \tilde{h} \to P \mid s : \tilde{h} \cdot l \quad (A \downarrow \mathsf{true}) \quad \text{[R-Label]}$$
$$\mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \to P \qquad (e \downarrow \mathsf{true}) \quad \text{[R-IfT]}$$
$$\mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \to Q \qquad (e \downarrow \mathsf{false}) \quad \text{[R-IfF]}$$
$$\mathsf{def}\ D\ \mathsf{in}\ C[X\langle \tilde{e}\tilde{s} \rangle] \to \mathsf{def}\ D\ \mathsf{in}\ Q \quad \text{[R-Def]}$$
$$(\text{where } \langle X\langle \tilde{v}\tilde{s} \rangle = P \rangle \in D \text{ and } C[P[\tilde{e}/\tilde{v}]] \to Q)$$

Fig. 8. Reduction without errors

annotations of logical formulae. $\overline{a}_{[2..\mathtt{n}]}\,(\tilde{s}).P$ and $a_{[\mathtt{p}]}\,(\tilde{s}).P$ model the behaviours of multicasting a request for starting a new session ($\overline{a}[2..\mathtt{n}]$) and to accept a session request by others (each $a[\mathtt{i}]$ with $2 \leqslant i \leqslant n$) through multiparty synchronisation.[9] Send, receive and branching are associated to a predicate A. Label branching associates a predicate to each branch. Conditional, parallel composition, inaction, name hiding, recursive process definition and recursion calls, are standard. (For simplicity we omit delegation, see §8.) Process $s : h_1..h_n$ represent messages in transit going through a channel s in an asynchronous order-preserving message delivery as in TCP, where *each* h_i is either a branching label or a vector of values. The empty queue is written $s : \varnothing$.

Processes errH (for "error here") and errT (for "error there") represent the run-time detection of a violation. Intuitively, errH denotes the violation of a predicate on which the process has an obligation (i.e. send and selection) while errT detects a violation caused by the environment (i.e. receive and branching).

[9] Session initializations are not asserted because the current notion of contracts gives a specification on the per-session basis.

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

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

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

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

Fig. 9. Reduction: error cases

The reduction rules with internal predicate checking are given in Figures 8 and 9, where the latter collects the rules whose internal predicate checking result in errors due to predicate violation. We generate $\to$ by closing it under $\mid$ and ν and taking terms modulo the standard structural equality, cf. [27]. The rules in Figure 8 are identical with the reduction rules for multiparty session types presented in [27] *except* the satisfaction of the predicate is checked successfully at each communication action: *send*, *receive*, *selection* and *branching*. In the rules, 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. $\tilde{\mathtt{n}}$). Recall that in Convention 3.3 we assume decidability of the underlying logic. Initially [Link] establishes a session through multiparty synchronisation, generating queues. Note all session channels are hidden at the initiation. The remaining rules model communications within a session, which are mediated by the message queues, modelling the TCP-like asynchronous order-preserving message delivery.

Example 5.1 (Seller's Process). Continuing from Example 3.4 we present a process which implements $\mathcal{G}_{neg}$, setting *Buyer*, *Seller*, *Bank* to be participants $\mathtt{1}, \mathtt{2}, \mathtt{3}$ respectively and omitting the predicates true:

$$P_{neg} = \overline{a}_{[\mathtt{2},\mathtt{3}]}\,(s, b, n).P_1 \mid a_{[\mathtt{2}]}\,(s, b, n).P_2 \mid a_{[\mathtt{3}]}\,(s, b, n).P_3$$

We focus on the process for *Seller*:

$$
\begin{aligned}
P_2 \quad &= \mathsf{def} X(p\_BPrice, p\_SPrice, s, b, n) = Q_2 \\
&\qquad \mathsf{in} X\langle 10, 100, s, b, n\rangle \\
\\
Q_2 \quad &= s?(BPrice)\{A \wedge A1\}; \\
&\quad \mathsf{if}\ e\ \mathsf{then}\ (b \lhd \mathsf{run}; Q_{run})\ \mathsf{else}\ (b \lhd \mathsf{end}; \mathbf{0}) \\
\\
Q_{run} &= b!(SPrice)\{A2\}; \\
&\quad s \rhd \{\mathsf{buy} : Q_{buy}, \{A3\}\mathsf{rec} : Q_{rec}, \mathsf{stop} : \mathbf{0}\} \\
\\
Q_{buy} &= n?(paid)\{\exists pay(A \wedge A1 \wedge \ldots \wedge A5)\}; \mathbf{0} \\
\\
Q_{rec} &= X\langle BPrice, SPrice, s, b, n\rangle
\end{aligned}
$$

P_2 can be validated against $\mathcal{G}_{neg} \upharpoonright Seller$. Since both the selections in Q_2 are annotated with predicate true, the expression e can be arbitrary. Note the existential is applied to *pay* in Q_{buy} since *Seller* does not know *pay*.

If we get rid of all logical formulae from an asserted process P, we obtain processes from [27]. We write the resulting process $\mathsf{erase}(P)$, called the *erasure* of P. The following convention simplifies our subsequent technical development.

Convention 5.2 (well-typedness). Henceforth we assume, for each asserted process P, its erasure $\mathsf{erase}(P)$ is well-typed by the type discipline in [27] (for reference the typing rules are listed in Appendix D.2.

5.2 Validation Rules

A *program phrase* is a process without queues or ν-hiding of session channels. A *program* is a program phrase which is closed (i.e. without free variables/process variables) and which is without free session channels. Programs are processes which programmers will write, in contrast to *run-time processes* (those possibly with queues and channel hiding) which represent the process of execution. In this section we present compositional validation rules for programs and program phrases against end-point assertions. The purpose is to guarantee that when a process is executed, it will never run into assertion errors as far as incoming messages are valid.

The validation rules closely follow the typing rules in [27], lifting global/local types in [27] to global/end-point assertions. A *sorting* $(\Gamma, \Gamma', ..)$ is a finite function mapping (i) variables to sorts $(\tilde{v} : \tilde{S})$, (ii) shared names

to global assertions ($a : \mathcal{G}$), and (*iii*) process variables to sequences of sorts and end-point assertions ($X : (\tilde{v} : \tilde{S})\mathcal{T}_1 @ \mathtt{p}_1 \ldots \mathcal{T}_n @ \mathtt{p}_n$). When we write e.g. $\Gamma, \tilde{v} : \tilde{S}$, we assume $\mathsf{dom}\Gamma \cap \{\tilde{v}\} = \varnothing$. We also use *located end-point assertions* $\mathcal{T} @ \mathtt{p}$ which indicates an end-point assertion $\mathcal{T}$ for participant the $\mathtt{p}$. *Assertion assignments* specify constraints on end-point behaviour *per each session* and are defined as

$$\Delta \quad ::= \quad \varnothing \quad | \quad \Delta, \tilde{s} : \{\mathcal{T} @ \mathtt{p}\}_{\mathtt{p} \in I}$$

An assertion assignment maps channels for a session to endpoint assertions for multiple participants, e.g. $\tilde{s} : \{\mathcal{T} @ \mathtt{p}\}_{\mathtt{p} \in I}$, though programs and program phrases only need a *singleton assignment* of the form $\tilde{s} : \{\mathcal{T} @ \mathtt{p}\}$, abbreviated as $\tilde{s} : \mathcal{T} @ \mathtt{p}$.

An *assertion environment* ($\mathcal{C}, \mathcal{C}', ...$) records the incremental conjunction of predicates and is defined by

$$\mathcal{C} \quad ::= \quad \mathsf{true} \quad | \quad \mathcal{C} \wedge A$$

An assertion environment can be regarded as a predicate, but the special font emphasises its (in effect) list-like shape.

The validation rules are given in Figure 10. By Convention 5.2 the rules omit typing constraints irrelevant for our purpose (Appendix D.2 lists the typing rules which give these constraints). We assume global and end-point assertions occurring in each rule are always well-asserted. Judgements have the shape:

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

which reads: "under $\mathcal{C}$ and Γ, process P is validated w.r.t. Δ". The validation rules validate the communication behaviour of processes as well as elaborating processes with the predicates from endpoint assertions (by neglecting the latter aspect, the rules can validate the behaviour of unasserted processe against endpoint assertionss, cf. §7.3).

Rule [Send] validates a participant $\mathtt{p}$ sending values $\tilde{e}$ on the (the k-th) session channel, provided that $\tilde{e}$ satisfy the predicate in the current assertion environment ($\mathcal{C} \supset A[\tilde{e}/\tilde{v}]$ ensures the sent values satisfy the specification) and that the continuation is valid, once $\tilde{v}$ gets replaced by $\tilde{e}$ (notice the substitution in P affects both expressions and predicates in it.)

Rule [Recv] validates a value reception against the continuation of the end-point assertion in the extended assertion environment $\mathcal{C} \wedge A$. The

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

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

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

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

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

$$\frac{\Gamma \vdash a : \mathcal{G} \quad \mathcal{C}; \Gamma \vdash P \rhd \Delta, \tilde{s} : (\mathcal{G} \upharpoonright \mathtt{p}) \,@\, \mathtt{p}}{\mathcal{C}; \Gamma \vdash a_{[\mathtt{p}]}\,(\tilde{s}).P \rhd \Delta} \quad [\textsc{Macc}]$$

$$\frac{\mathcal{C}; \Gamma \vdash P \rhd \Delta \quad \mathcal{C}; \Gamma \vdash Q \rhd \Delta' \quad \Delta \asymp \Delta'}{\mathcal{C}; \Gamma \vdash P \mid Q \rhd \Delta \circ \Delta'} \quad [\textsc{Conc}]$$

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

$$\frac{\Delta\ \mathsf{end\ only}}{\mathcal{C}; \Gamma \vdash \mathbf{0} \rhd \Delta} \qquad \frac{\mathcal{C}; \Gamma, a : \mathcal{G} \vdash P \rhd \Delta}{\mathcal{C}; \Gamma \vdash (\boldsymbol{\nu}\, a)P \rhd \Delta} \quad [\textsc{Inact}],[\textsc{NRes}]$$

$$\frac{\Gamma \vdash v_i : S_i \quad LSat(\mathcal{T}_i[\tilde{e}/\tilde{v}], \mathcal{C})\ (1 \leqslant i \leqslant n)}{\begin{array}{l}\mathcal{C}; \Gamma, X : (\tilde{v} : \tilde{S})\mathcal{T}_1 \,@\, \mathtt{p}_1 \ldots \mathcal{T}_n \,@\, \mathtt{p}_\mathtt{n} \vdash X\langle \tilde{e}\tilde{s}_1..\tilde{s}_n \rangle \\ \qquad \rhd \tilde{s}_1 : \mathcal{T}_1[\tilde{e}/\tilde{v}] \,@\, \mathtt{p}_1, .., \tilde{s}_n : \mathcal{T}_n[\tilde{e}/\tilde{v}] \,@\, \mathtt{p}_n\end{array}} \quad [\textsc{Var}]$$

$$\frac{\begin{array}{l}\mathcal{C}; \Gamma, X : (\tilde{v} : \tilde{S})\mathcal{T}_1 \,@\, \mathtt{p}_1 \ldots \mathcal{T}_n \,@\, \mathtt{p}_\mathtt{n}, \tilde{v} : \tilde{S} \vdash P \\ \qquad \rhd \tilde{s_1} : \mathcal{T}_1 \,@\, \mathtt{p}_1 \ldots \tilde{s_n} : \mathcal{T}_n \,@\, \mathtt{p}_n \\ \mathcal{C}; \Gamma, X : (\tilde{v} : \tilde{S})\mathcal{T}_1 \,@\, \mathtt{p1} \ldots \mathcal{T}_n \,@\, \mathtt{p}_\mathtt{n} \vdash Q \rhd \Delta\end{array}}{\mathcal{C}; \Gamma \vdash \mathsf{def}\ X(\tilde{v}\tilde{s}_1 \ldots \tilde{s}_n) = P\ \mathsf{in}\ Q\ \rhd\ \Delta} \quad [\textsc{Def}]$$

$$\frac{\mathcal{C}'; \Gamma \vdash P \rhd \Delta' \quad \mathcal{C} \supset \mathcal{C}' \quad \Delta' \ni \Delta}{\mathcal{C}; \Gamma \vdash P \rhd \Delta} \quad [\textsc{Conseq}]$$

Fig. 10. Validation and Elaboration Rules for Program Phrases

conjunction $\mathcal{C} \wedge A$ indicates the process can rely on A for the received values after the input.

Rules [Sel] and [Branch] are understood in the same way ($\mathtt{n}$ is the number of participants in $\mathcal{G}$ and $\mathtt{p}$ is one of the participants).

Rules [Mcast] and [Macc] validate session request and acceptance for a participant by validating the continuation against the projection of the global assertion on that participant. Notice that the assertion environment $\mathcal{C}$ does not involve any interaction variable used in the assertions of P.

Rule [Conc] validates parallel composition. Following [27], $\Delta \asymp \Delta'$ denotes linear compatibility, meaning that: *If we have* $\tilde{s}_1 : \{\mathcal{T}_p \, @_{\mathtt{p}}\}_{p \in I}$ *in* Δ *and, further,* $\tilde{s}_2 : \{\mathcal{T}_q \, @_{\mathtt{q}}\}_{q \in J}$ *in* Δ'*, then either* $\tilde{s}_1 \cap \tilde{s}_2 = \varnothing$ *or, if not,* $\tilde{s}_1 = \tilde{s}_2$ *and* $I \cap J = \varnothing$. If this holds, the composition $\Delta \circ \Delta'$ denotes the pointwise (per-session) union of assertion assignments.

Rule [If] validates a conditional against Δ if each branch is validated against the same Δ, under the extended environment $\mathcal{C} \wedge e$ or $\mathcal{C} \wedge \neg e$, as in the corresponding rule in Hoare logic [26]. In [Inact], as in the underlying type discipline, we demand Δ only contains end as end-point assertions.

Rule [Var] validates a call for X against the substitution of the end-point assertion associated to X with the substitution of $\tilde{e}$ to the recursion parameters, if the result of substitutions for each end-point assertion $\mathcal{T}_i$ is consistent using *LSat* from §4. Observe $\mathcal{T}_i$ typically takes the form of a recursive end-point assertion with initialization coming from the variables $\tilde{v}$. In this case *LSat* checks this initialization satisfies the recursive invariant.

In [Def], the definition-call is validated if (1) the defined process is validated against the given end-point assertion and (2) the predicate associated to the process, with substitution of the recursion parameters with $\tilde{e}$, is satisfied in the current environment. The validity of this rule hinges on the partial correctness nature of the semantics of the judgement.

Rule [Conseq] uses the relation $\ni$. This is a semantic notion we later define in Definition 7.4, defining the natural notion of *refinement*: if $\mathcal{T} \ni \mathcal{T}'$, then $\mathcal{T}$ is more refined (more restrictive) than $\mathcal{T}'$ as a specification. Thus $\ni$ can be considered as a form of entailment, motivating the rule. As a practical consequence, the rule has the following instance, where $\mathcal{T} \approx_{\mathsf{tree}} \mathcal{T}'$ indicates $\mathcal{T}$ and $\mathcal{T}'$ are identical when we unfold all recursions in them to infinite trees.

$$\frac{\mathcal{C}; \Gamma \vdash P \rhd \Delta, \ \tilde{s} : \mathcal{T}' @ p \quad \mathcal{T} \approx_{\mathsf{tree}} \mathcal{T}'}{\mathcal{C}; \Gamma \vdash P \rhd \Delta, \ \tilde{s} : \mathcal{T} @ p} \tag{5.1}$$

This instance is admissible since two recursive assertions whose tree representations are identical have the same meaning, in terms of the transition semantics we shall define in §7. Note also this rule allows a program to be internally asserted with a more restrictive (stronger) predicates than given in a given endpoint assertion.

The purpose of validation is to validate a well-typed program against a given endpoint assertion assignment Δ and a sorting Γ under the trivial assertion environment true, as well as elaborating the program with interaction predicates in the specification. In the rest of the paper we write $\Gamma \vdash P \triangleright \Delta$ for $\mathsf{true}; \Gamma \vdash P \triangleright \Delta$.

Observe that, since each interaction predicate is directly copied from endpoint assertions to processes in session prefix rules, the proof system in effect validates unasserted processes, i.e. typed processes without elaboration by logical formulae, against given specifications. This aspect is discussed in more detail in §7.3 later.

Example 5.3 (Validating Seller Process). We can check the process P_{neg} from Example 5.1 can be validated under $a : \mathcal{G}_{neg}$ from Example 3.4 in §3, using the properly projected version of $\mathcal{G}_{neg}$.

Closing this section, we present the following enhanced version of the standard substitution lemma for the π-calculus. Note the lemma is vacuous when $\mathcal{C}[\tilde{\mathtt{n}}/\tilde{u}]$ is unsatisfiable.

Lemma 5.4 (Substitution). *Let $\mathcal{C}; \Gamma, \tilde{u} : \tilde{S} \vdash P \triangleright \Delta$ with Δ well-asserted. If $\tilde{u}$ is free in P and $\tilde{\mathtt{n}}$ have sorts $\tilde{S}$ then $\mathcal{C}[\tilde{\mathtt{n}}/\tilde{u}]; \Gamma \vdash P[\tilde{\mathtt{n}}/\tilde{u}] \triangleright \Delta[\tilde{\mathtt{n}}/\tilde{u}]$ and $\Delta[\tilde{\mathtt{n}}/\tilde{u}]$ is well-asserted.*

Proof. By rule induction, see Appendix D.3 for details. □

6 A Larger Example of Validation: Negotiation Protocol

In this section we illustrate a comprehensive example of a recursive three-party negotiation protocol. In § 6.1 we present the global assertion and its projections into the participants. § 6.2 presents the processes implementing each end-point assertion. In § 6.3 we show that each of the processes is conform to one of the end-point assertions. For readability we use, instead of numbers, the mnemonics `S`, `B`, `N`, `ChS`, `ChB` and `ChN` to denote participants (i.e., seller, buyer and bank) and their channels.

6.1 Global assertion and projections

The following global assertions models a negotiation between a seller and a buyer (participants `S` and `B`, respectively) which possibly terminates with a purchase which is notified to a bank (participant `N`).The negotiation is modelled as a recursion where the parameter p_{old} represents the value proposed in the previous instance of negotiation and is initialized to 10 in the first instance. The buyer proposes a price p to the seller, then the seller decides whether to sell (i.e., label ok), quit (i.e., label stop) or run another instance of negotiation (i.e., label neg) where the current value of p is assigned to p_{old}. Notice that p_{old} is located only in `S` and `B` since `N` does not know p thus he does not know p_{old} that takes the value of p in the recursion invocation.

$$
\begin{array}{l}
\mathcal{G} = \mu \mathbf{t}\langle 10\rangle(p_{old} : Int \,@\, \{\mathtt{S},\mathtt{B}\})\{p_{old} \geqslant 10\}. \\
\quad \mathtt{B} \to \mathtt{S}\colon chS\,(p : Int)\{p \geqslant 10\}. \\
\quad \mathtt{S} \to \mathtt{B}\colon chB\{\{\mathsf{true}\}\mathsf{ok} : \mathsf{end}, \\
\qquad\qquad \{\mathsf{true}\}\mathsf{stop} : \mathtt{B} \to \mathtt{N}\colon chN\,(c : Int)\{c = p\}.\mathsf{end}, \\
\qquad\qquad \{p > p_{old}\}\mathsf{neg} : \mathbf{t}\langle p\rangle\}
\end{array}
$$

We present below the three end-point assertions obtained by projecting $\mathcal{G}$ on the three participants `S`, `B` and `N`. The projection for Seller becomes:

$$
\begin{array}{lll}
\mathcal{G} \upharpoonright \mathtt{S} & = \mu \mathbf{t}\langle 10\rangle(p_{old} : Int)\{p_{old} \geqslant 10\}.\mathcal{T}_{S1} \\
\mathcal{T}_{S1} & = chS?(p : Int)\{p_{old} \geqslant 10 \wedge p \geqslant 10\}; \mathcal{T}_{S2} \\
\mathcal{T}_{S2} & = chB \oplus \{\ \{\mathsf{true}\}\mathsf{ok} : \mathsf{end}, \\
& \qquad\qquad \{\mathsf{true}\}\mathsf{stop} : \mathsf{end}, \\
& \qquad\qquad \{p > p_{old}\}\mathsf{neg} : \mathbf{t}\langle p\rangle\ \}
\end{array}
$$

Next we consider the projection to Buyer:

$$\begin{aligned}
\mathcal{G} \upharpoonright \mathtt{B} &= \mu \mathbf{t}\langle 10\rangle(p_{old} : Int)\{p_{old} \geqslant 10\}.\mathcal{T}_{B1} \\
\mathcal{T}_{B1} &= chS!(p : Int)\{p \geqslant 10\}; \mathcal{T}_{B2} \\
\mathcal{T}_{B2} &= chB\&\{\ \{p_{old} \geqslant 10 \wedge p \geqslant 10\}\mathsf{ok} : \mathcal{T}_{Bbuy}, \\
&\qquad\qquad \{p_{old} \geqslant 10 \wedge p \geqslant 10\}\mathsf{stop} : \mathsf{end}, \\
&\qquad\qquad \{p_{old} \geqslant 10 \wedge p \geqslant 10 \wedge p > p_{old}\}\mathsf{neg} : \mathbf{t}\langle p\rangle\ \} \\
\mathcal{T}_{Bbuy} &= chN!(c : Int)\{c = p\}; \mathsf{end}
\end{aligned}$$

Finally we list the projection to Bank.

$$\begin{aligned}
\mathcal{G} \upharpoonright \mathtt{N} &= \mu \mathbf{t}\langle\rangle()\{\exists p_{old}(p_{old} \geqslant 10)\}.\mathcal{T}_{Nbuy} \\
\mathcal{T}_{NBuy} &= chN?(c : Int)\{\exists p, p_{old}.(p_{old} \geqslant 10 \wedge p \geqslant 10 \wedge p > p_{old} \wedge c = p)\}; \mathsf{end}
\end{aligned}$$

In these projections, the predicates associated to output or selection consist of the corresponding predicate of the global assertion, for example the predicates of $\mathcal{T}_{S2}$ and $\mathcal{T}_{B1}$. According to the rules in the definition of projection the predicates associated to input or branching are defined as the logical conjunction of the corresponding predicates of the global assertion and the previous predicates. For example, the predicate of $\mathcal{T}_{S1}$ includes the the recursion invariant and the predicates of $\mathcal{T}_{B2}$ include both the invariant and the predicate in $\mathcal{T}_{B1}$. Notice that the existential quantifier applies only to the interaction predicates of `N` since the seller and the buyer know both p and p_{old}. Also in the recursion invariant of $\mathcal{G} \upharpoonright \mathtt{N}$, since the variable p_{old} is not located in `N`, then it is bound by the existential quantifier.

The predicates of $\mathcal{T}_{Nbuy}$ include all the previous predicates, also those of interactions between the seller and the buyer.

6.2 Processes

We model a negotiation protocol as a process P. P consists of the parallel composition of three processes, one for each participant. In P, the seller starts a session and the other participants accept the session request. The session is modelled by the three processes P_{Seller}, P_{Buyer} and P_{Bank}. P_{Seller} defines a recursion and invokes the first instance with $p_{old} = 10$. In the recursion, the seller receives a proposal and, according to the condition e decides whether to reiterate the negotiation or accept the price. P_{Buyer} also defines a recursion and P_{B1} always increments the price proposed by the buyer of one unit, with respect to the previous instance (i.e., $p = p_{old} + 1$). Notice that the whereas the buyer offers, in the branching, al the possible branches of the end-point assertions in § 6.1, the seller can

in the conditional and selection only admit some of the offered branches. If the seller accept the price, process P_{B1} sends a notification to the bank. P_{Bank} receives the notification.

$$
\begin{array}{ll}
P & = \overline{a}_{[\mathtt{B},\mathtt{N}]}\,(ChS, ChB, ChN).P_{Seller} \mid a_{[\mathtt{B}]}\,(ChS, ChB, ChN).P_{Buyer} \mid \\
 & \quad a_{[\mathtt{N}]}\,(ChS, ChB, ChN).P_{Bank} \\
 & \\
P_{Seller} & = def\ X(p_{old}, ChS, ChB, ChN) = P_{S1}\ \mathsf{in}\ X\langle 10, ChS, ChB, ChN\rangle \\
P_{S1} & = ChS?(p)\{p_{old} \geqslant 10 \wedge p \geqslant 10\}; P_{S2} \\
P_{S2} & = \mathsf{if}\ (p > 10)\ \mathsf{then}\ P_{ST}\ \mathsf{else}\ P_{SF} \\
P_{ST} & = ChB \lhd \{p > p_{old}\}\mathsf{neg}; X\langle p, ChS, ChB, ChN\rangle \\
P_{SF} & = ChB \lhd \{\mathsf{true}\}\mathsf{ok}; \mathbf{0} \\
 & \\
P_{Buyer} & = def\ X(p_{old}, ChS, ChB, ChN) = P_{B1}\ \mathsf{in}\ X\langle p, ChS, ChB, ChN\rangle \\
P_{B1} & = ChS!\langle p_{old} + 1\rangle(p)\{p \geqslant 10\}; \\
 & \quad ChB \rhd \{\{p_{old} \geqslant 10 \wedge p \geqslant 10\}\mathsf{ok}\colon ChN!\langle p\rangle(c)\{c = p\}; \mathbf{0}, \\
 & \qquad\quad \{p_{old} \geqslant 10 \wedge p \geqslant 10\}\mathsf{stop}\colon \mathbf{0}, \\
 & \qquad\quad \{p_{old} \geqslant 10 \wedge p \geqslant 10 \wedge p_{old} > p\}\mathsf{neg}\colon X\langle p, ChS, ChB, ChN\rangle\} \\
 & \\
P_{Bank} & = def\ X(ChS, ChB, ChN) = P_{N1}\ \mathsf{in}\ X\langle ChS, ChB, ChN\rangle \\
P_{N1} & = ChN?(c)\{\exists p_{old}, p(p_{old} \geqslant 10 \wedge p \geqslant 10 \wedge c = p)\}; \mathbf{0}
\end{array}
$$

6.3 Validation

We want to validate P against

$$\Delta = \tilde{s} : \mathcal{G} \upharpoonright \mathtt{S}, \tilde{s} : \mathcal{G} \upharpoonright \mathtt{B}, \tilde{s} : \mathcal{G} \upharpoonright \mathtt{N}$$

where $\tilde{s} = ChS, ChB, ChN$. First we apply rule [CONC] twice and decompose the validation into the validation of:

1. $\overline{a}_{[2,3]}\,(\tilde{s}).P_{Seller}$,
2. $a_{[2]}\,(\tilde{s}).P_{Buyer}$,
3. $a_{[3]}\,(\tilde{s}).P_{Bank}$.

6.3.1 (1) Seller process. By rule [MCAST], in order to show the validity of $\overline{a}_{[2,3]}\,(\tilde{s}).P_{Seller}$ we have to prove

$$\mathsf{true}, \Gamma \vdash def\ X(p_{old}, \tilde{s}) = P_{S1}\ \mathsf{in}\ X\langle 10, \tilde{s}\rangle \rhd \tilde{s} : \mathcal{G} \upharpoonright \mathtt{S} \mathbin{@} \mathtt{S}.$$

In order to proceed with the validation we have to consider the following points (cf. Definition 7.6 and Proposition 7.7 in § 7).

– we say $\mathcal{T} \approx_{\mathsf{tree}} \mathcal{T}'$ when $\mathcal{T}$ and $\mathcal{T}'$ are identical as inifinite trees, when their recursions are unfolded inductively;
– we say $\Delta \approx \Delta'$ if Δ and Δ' only differ on the fact that Δ includes $\mathcal{T}$ and Δ' includes $\mathcal{T}'$ such that $\mathcal{T} \approx_{\mathsf{tree}} \mathcal{T}'$.
– $\Delta \approx \Delta'$ implies $\Delta \ni \Delta'$
– $\mathcal{G} \upharpoonright \mathtt{S} = \mu\mathbf{t}\langle 10\rangle(p_{old} : Int)\{p_{old} \geqslant 10\}.\mathcal{T}_{S1} \approx_{\mathsf{tree}} \mathcal{T}'_{S1}[10/p_{old}]$, hence $\mathcal{T}'_{S1} = \mathcal{T}_{S1}[\mu\mathbf{t}.../\mathbf{t}]$ implies $\mathcal{T}'_{S1}[10/p_{old}] \ni \mathcal{G} \upharpoonright \mathtt{S}$

It follows that, by rule [Conc], we have only to prove

$$\mathsf{true}, \Gamma \vdash def\ X(p_{old}, \tilde{s}) = P_{S1}\ \mathsf{in}\ X\langle 10, \tilde{s}\rangle \rhd \tilde{s} : \mathcal{T}'_{S1}[10/p_{old}]\,@\,\mathtt{S}$$

where $\mathcal{T}'_{S1} = \mathcal{T}_{S1}[\mu\mathbf{t}\langle 10\rangle(p_{old} : Int)\{p_{old} \geqslant 10\}.\mathcal{T}_{S1}/\mathbf{t}]$.
By rule [Def], where $\Gamma' = \Gamma, X : (p_{old} : Int)\tilde{s} : \mathcal{T}'_{S1}\,@\,\mathtt{S}$

$$\frac{\begin{array}{c}(1)\mathsf{true}, \Gamma' \vdash P_{S1} \rhd \tilde{s} : \mathcal{T}_{S1}\,@\,\mathtt{S} \\ (2)\mathsf{true}, \Gamma' \vdash X\langle 10, \tilde{s}\rangle \rhd \tilde{s} : \mathcal{T}'_{S1}[10/p_{old}]\,@\,\mathtt{S}\end{array}}{\mathsf{true}, \Gamma \vdash def\ X(p_{old}, \tilde{s}) = P_{S1}\ \mathsf{in}\ X\langle 10, \tilde{s}\rangle \rhd \tilde{s} : \mathcal{T}'_{S1}[10/p_{old}]\,@\,\mathtt{S}}$$

Premise (2) follows by rule [Var] applied to $X\langle 10, \tilde{s}\rangle$ against $\mathcal{T}'_{S1}[10/p_{old}]$.

$$\frac{LSat(\mathcal{T}'_{S1}[10/p_{old}], \mathsf{true})}{\mathsf{true}, \Gamma' \vdash X\langle 10, ChS, ChB, ChN\rangle \rhd \tilde{s} : \mathcal{T}'_{S1}[10/p_{old}]\,@\,\mathtt{S}.}$$

$$\begin{array}{l} LSat(\mathcal{G} \upharpoonright \mathtt{S}, \mathsf{true}) \\ = LSat(\mathcal{T}_{S2}[10/p_{old}], A[10/p_{old}]) \qquad (\star) \\ = LSat(\mathsf{end}, A[10/p_{old}] \wedge \mathsf{true}) \wedge LSat(\mu\mathbf{t}..., A[10/p_{old}] \wedge p > 10) \qquad (\dagger) \\ = \mathsf{true} \qquad (\ddagger) \end{array}$$

where:

- $(\star)$ is by $\mathsf{true} \supset A[10/p_{old}]$.
- $(\dagger)$ is by $A[10/p_{old}] \supset \mathsf{true} \vee p > 10$.
- $(\ddagger)$ is by:

  $LSat(\mathsf{end}, A \wedge \mathsf{true}) = \mathsf{true}$,
  $LSat(\mathbf{t}\langle p\rangle, A[10/p_{old}] \wedge p > 10) = \mathsf{true}\ (with\ A[10/p_{old}] \wedge p > 10 \supset p \geqslant 10)$.

It follows the proof for (2) where we denote predicate $p_{old} \geqslant 10 \wedge p \geqslant 10$ by A and $\mathcal{T}'_{S1} = chS?(p : Int)\{A\}; chB \oplus \{\{\mathsf{true}\}\mathsf{ok} : \mathsf{end}, \{\mathsf{true}\}\mathsf{stop} : \mathsf{end}, \{p > 10\}\mathsf{neg} : \mu\mathbf{t}...\}$:

$$\dfrac{\dfrac{\dfrac{LSAT(\mathcal{T}'_{S1}[p/p_{old}], A\wedge p>10)}{(A\wedge p>10)\supset p>10 \;\; A\wedge p>10, \Gamma' \vdash X\langle p,\tilde{s}\rangle \rhd \tilde{s} : \mathcal{T}'_{S1}@\mathtt{S}}\,[\textsc{Var}]}{A\wedge p>10, \Gamma' \vdash ChB \lhd \{p>10\}\mathsf{neg}; X\langle p,\tilde{s}\rangle \rhd \tilde{s} : \mathcal{T}_{S2}@\mathtt{S}}\,[\textsc{Sel}] \quad A\wedge\neg(p>10), \Gamma' \vdash ChB \lhd \{\mathsf{true}\}\mathsf{ok}; \mathbf{0} \rhd \tilde{s} : \mathcal{T}_{S2}@\mathtt{S}}{A, \Gamma' \vdash \mathsf{if}\ (p>10)\ \mathsf{then}\ P_{ST}\ \mathsf{else}\ P_{SF} \rhd \tilde{s} : \mathcal{T}_{S2}@\mathtt{S}}\,[\textsc{If}]$$

$$\dfrac{\dfrac{\Delta\ \mathsf{end\ only}}{A\wedge\neg(p>10), \Gamma' \vdash \mathbf{0} \rhd \tilde{s} : \mathsf{end}@\mathtt{S}}\,[\textsc{Inact}]}{\mathsf{true}, \Gamma' \vdash ChS?(p)\{A\}; P_{S2} \rhd \tilde{s} : ChS?(p:Int)\{A\}; \mathcal{T}_{S2}@\mathtt{S}}\,[\textsc{Sel}]\,[\textsc{Rcv}]$$

In the rule [Var] we apply LSat to the unfolding of $\mathcal{G} \upharpoonright \mathtt{S}$ where the parameter p_{old} has been substituted with the actual parameter p. We denote $(p \geqslant 10)$ with $A[p/p_{old}]$. We simplify some predicate with a shorter equivalent one for readability:

$\mathcal{T}'_{S1}[p/p_{old}] = chS?(p:Int)\{A[p/p_{old}]\}; ChB \oplus \{\{\mathsf{true}\}\mathsf{ok} : \mathsf{end}, \{\mathsf{true}\}\mathsf{stop} : \mathsf{end}, \{p>10\}\mathsf{neg} : \mu\mathbf{t}...\}$

and

$LSat(\mathcal{T}'_{S1}[p/p_{old}], A\wedge p>10) =$ $\quad (with\ A\wedge p>10 \supset A[p/p_{old}])$
$LSat(\mathcal{T}'_{S2}[P/p_{old}], A\wedge p>10) =$ $\quad (with\ A\wedge p>10 \supset \mathsf{true}\vee p>10)$
$LSat(\mathsf{end}, A\wedge p>10) \wedge LSat(\mathcal{T}_{S1}[p/p_{old}], A\wedge p>10) = \mathsf{true}$
since:
$LSat(\mathsf{end}, A\wedge p>10) = \mathsf{true}$, and
$LSat(\mathcal{T}_{S1}[p/p_{old}], A\wedge p>10) =$
...
$LSat(\mathbf{t}\langle p\rangle, A\wedge p>10) = \mathsf{true}$ $\quad (with\ A\wedge p>10 \supset p>10).$

6.3.2 (2) Buyer process. By rule [Mcast], in order to show the validity of $a_{[\mathtt{B}]}\,(ChS, ChB, ChN).P_{Buyer}$ we have to prove

$$\mathsf{true}, \Gamma \vdash def\ X(p_{old}, \tilde{s}) = P_{B1}\ \mathsf{in}\ X\langle 10, \tilde{s}\rangle \rhd \tilde{s} : \mathcal{G} \upharpoonright \mathtt{B}\,@\,\mathtt{B}.$$

Notice that $\mathcal{T}'_{B1}[10/p_{old}] \ni \mathcal{G} \upharpoonright \mathtt{B}$. It follows that, by rule [Conc], we have to prove

$$\mathsf{true}, \Gamma \vdash def\ X(p_{old}, \tilde{s}) = P_{B1}\ \mathsf{in}\ X\langle 10, \tilde{s}\rangle \rhd \tilde{s} : \mathcal{T}'_{B1}[10/p_{old}]\,@\,\mathtt{B}$$

where $\mathcal{T}'_{B1} = \mathcal{T}_{B1}[\mu\mathbf{t}\langle 10\rangle(p_{old}:Int)\{p_{old}\geqslant 10\}.\mathcal{T}_{B1}/\mathbf{t}]$.

By rule [Def], where $\Gamma' = \Gamma, X : (p_{old}:Int)\tilde{s} : \mathcal{T}'_{B1}\,@\,\mathtt{B}$

$$\dfrac{\begin{array}{c}(1)\mathsf{true}, \Gamma' \vdash P_{B1} \rhd \tilde{s} : \mathcal{T}_{B1}\,@\,\mathtt{B} \\ (2)\mathsf{true}, \Gamma' \vdash X\langle 10, \tilde{s}\rangle \rhd \tilde{s} : \mathcal{T}'_{B1}[10/p_{old}]\,@\,\mathtt{B}\end{array}}{\mathsf{true}, \Gamma \vdash def\ X(p_{old}, \tilde{s}) = P_{B1}\ \mathsf{in}\ X\langle 10, \tilde{s}\rangle \rhd \tilde{s} : \mathcal{T}'_{S1}[10/p_{old}]\,@\,\mathtt{B}}$$

Premise (2) follows by rule [Var] applied to $X\langle 10, \tilde{s}\rangle$ against the unfolding of $\mathcal{G} \upharpoonright \mathtt{B}$

$$\dfrac{LSat(\mathcal{T}'_{B1}[10/p_{old}], \mathsf{true})}{\mathsf{true}, \Gamma' \vdash X\langle 10, ChS, ChB, ChN\rangle \rhd \tilde{s} : \mathcal{T}'_{B1}[10/p_{old}]\,@\,\mathtt{B}.}$$

The proof $LSat(\mathcal{T}'_{B1}[10/p_{old}]) = \mathsf{true}$ is similar to the one for the seller (i.e., $LSat(\mathcal{T}'_{S1}[10/p_{old}]) = \mathsf{true}$).

It follows the proof for (2) where we denote predicate $p_{old} \geqslant 10 \wedge p \geqslant 10$ by A and $\mathcal{T}'_{B1} = chS!(p:Int)\{p \geqslant 10\}; chB\&\{\{A\}\mathsf{ok} : \mathsf{end}, \{A\}\mathsf{stop} : \mathsf{end}, \{A \wedge p > 10\}\mathsf{neg} : \mu\mathbf{t}...\}$:

$$\dfrac{\begin{array}{l}(1)\, A, \Gamma' \vdash \mathbf{0} \rhd \tilde{s} : \mathcal{T}_{B2}@\mathtt{B} \\ (2)\, A, \Gamma' \vdash ChN!\langle p\rangle(c)\{c = p\}; \mathbf{0} \rhd \tilde{s} : \mathcal{T}_{B2}@\mathtt{B} \\ (3)\, A \wedge p > 10, \Gamma' \vdash X\langle p, \tilde{s}\rangle \rhd \tilde{s} : \mathcal{T}_{B2}@\mathtt{B}\end{array}}{\dfrac{\mathsf{true} \supset 11 \geqslant 10 \;\; \mathsf{true}, \Gamma' \vdash ChB \rhd \{\{A\}\mathsf{ok}\colon \mathcal{T}_{Bbuy},\ \{A\}\mathsf{stop}\colon \mathbf{0},\ \{A \wedge p > 10\}\mathsf{neg}\colon X\langle 11, \tilde{s}\rangle\} \rhd \tilde{s} : \mathcal{T}_{B2}@\mathtt{B}}{\mathsf{true}, \Gamma' \vdash ChS!\langle 10+1\rangle(p)\{p \geqslant 10\}; P_{B2} \rhd \tilde{s} : chS!(p:Int)\{p \geqslant 10\}; \mathcal{T}_{B2}@\mathtt{B}}\,[\textsc{Snd}]}\,[\textsc{Branch}]$$

We split then the proof in three part, one for each branch:

$$(1)\quad \dfrac{\Delta \text{ } \mathsf{end} \text{ only}}{A, \Gamma' \vdash \mathbf{0} \rhd \tilde{s} : \mathsf{end}@\mathtt{B}}\,[\textsc{Inact}]$$

$$\dfrac{\dfrac{(2)}{\dfrac{\Delta \text{ } \mathsf{end} \text{ only}}{A \supset p = p \;\; A, \Gamma' \vdash \mathbf{0} \rhd \tilde{s} : \mathsf{end}@\mathtt{B}}\,[\textsc{Inact}]}{A, \Gamma' \vdash ChN!\langle p\rangle(c)\{c = p\}; \mathbf{0} \rhd \tilde{s} : chN!(c:Int)\{c = p\}; \mathsf{end}@\mathtt{B}}\,[\textsc{Snd}]$$

$$(3)\quad \dfrac{LSAT(\mathcal{T}'_{B1}[p/p_{old}], A \wedge p > 10)}{A \wedge p > 10, \Gamma' \vdash X\langle p, \tilde{s}\rangle \rhd \tilde{s} : \mathcal{T}'_{B1}@\mathtt{B}}\,[\textsc{Var}]$$

The proof for $LSat(\mathcal{T}'_{B1}[p/p_{old}]) = \mathsf{true}$ is similar to the one of the seller ($LSat(\mathcal{T}'_{S1}[p/p_{old}]) = \mathsf{true}$).

6.3.3 (3) Bank process. Since $\mathcal{T}_{Nbuy} \ni \mathcal{G} \upharpoonright \mathtt{N}$, let $\Gamma' = \Gamma, X : (p_{old} : Int)\tilde{s} : \mathcal{T}_{Nbuy}\,@\,\mathtt{N}$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\Delta \text{ } \mathsf{end} \text{ only}}{\exists p, p_{old}(p \geqslant 10 \wedge p_{old} \geqslant 10 \wedge c = p), \Gamma' \vdash \mathbf{0} \rhd \tilde{s} : \mathsf{end}\,@\,\mathtt{N}}\,[\textsc{Inact}]}{\mathsf{true}, \Gamma' \vdash P_{N1} \rhd \tilde{s} : \mathcal{T}_{Nbuy}\,@\,\mathtt{N}}\,[\textsc{Rcv}] \quad \dfrac{LSAT(\mathcal{T}_{Nbuy}, \mathsf{true})}{\mathsf{true}, \Gamma' \vdash X\langle\tilde{s}\rangle \rhd \tilde{s} : \mathcal{T}_{Nbuy}\,@\,\mathtt{N}}\,[\textsc{Var}]}{\mathsf{true}, \Gamma \vdash def\ X(ChS, ChB, ChN) = P_{N1}\ \mathsf{in}\ X\langle ChS, ChB, ChN\rangle \rhd \tilde{s} : \mathcal{T}_{Nbuy}\,@\,\mathtt{N} \;\; \mathcal{T}_{Nbuy} \ni \mathcal{G} \upharpoonright \mathtt{N}}\,[\textsc{Def}]}{\mathsf{true}, \Gamma \vdash def\ X(ChS, ChB, ChN) = P_{N1}\ \mathsf{in}\ X\langle ChS, ChB, ChN\rangle \rhd \tilde{s} : \mathcal{G} \upharpoonright \mathtt{N}\,@\,\mathtt{N}}\,[\textsc{Conseq}]$$

where it is straightforward to derive $LSat(\mathcal{T}_{Nbuy}, \exists p, p_{old}(p \geqslant 10 \wedge p_{old} \geqslant 10 \wedge c = p)) = \mathsf{true}$.

7 Soundness, Algorithmic Validation and Monitoring

7.1 Semantics of Assertions

In this section we first define the semantics of end-point assertions using labelled transition relations, then establish several basic semantic results underpinning the proposed method, including soundness of the validation rules in § 5.2, assertion-error freedom and the erasure theorem. These results offer a firm basis for the applications of the assertion method for static and dynamic validation, discussed at the end of the section.

Convention 7.1. Throughout we assume each ν-bound name is annotated by a global assertion, written e.g. $(\nu a : \mathcal{G})P$.

The labelled transition for asserted processes is the standard one except assertions in session communications are checked. Transitions use the following labels:

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

where in $(\boldsymbol{\nu}\tilde{a}{:}\tilde{\mathcal{G}})s!\tilde{\mathtt{n}}$ names in $\tilde{a}$ should occur among $\tilde{\mathtt{n}}$. Note when a name is exported we annotate it with a global assertion. The rules are standard except that The τ-action is induced by the reduction, i.e. $P \xrightarrow{\tau} Q$ iff $P \to Q$, and are given in Appendix E.2, Figure 15, writing $P \xrightarrow{\alpha} Q$ when P has a one-step transition α and becomes Q.

The transition of assertions has the form

$$\langle \Gamma, \Delta \rangle \xrightarrow{\alpha} \langle \Gamma', \Delta' \rangle$$

which reads: *the specification* $\langle \Gamma, \Delta \rangle$ allows *the action* α*, with the specification* $\langle \Gamma', \Delta' \rangle$ *for the continuation.* Above we restrict Δ to a collection of *singleton assignments*, i.e. of shape $\tilde{s} : \mathcal{T}@\mathtt{p}$. Figure 11 defines the labelled transition rules, assuming all occurring assertions are well-asserted and equating recursive assertions with their unfoldings [19], e.g.

$$\mu\mathbf{t}\langle 1\rangle(v).k!(x)\{x{\geqslant}v\}; \mathbf{t}\langle v{+}1\rangle \tag{7.1}$$

is equated with

$$k!(x)\{x{\geqslant}1\}; \mu\mathbf{t}\langle 2\rangle(v).k!(x)\{x{\geqslant}v\}; \mathbf{t}\langle v{+}1\rangle \tag{7.2}$$

(note this unfolding makes an output action possible). The τ-action does not change the environment and the assertion assignment.

$$\langle a:\mathcal{G}\cdot\Gamma,\Delta\rangle \xrightarrow{\overline{a}[2..n](\tilde{s})} \langle a:\mathcal{G}\cdot\Gamma,\Delta,\tilde{s}:\mathcal{G}\upharpoonright 1@\mathtt{p}_1\rangle \qquad \text{[TR-LinkOut]}$$

$$\langle a:\mathcal{G}\cdot\Gamma,\Delta\rangle \xrightarrow{a[i](\tilde{s})} \langle a:\mathcal{G}\cdot\Gamma,\Delta,\tilde{s}:\mathcal{G}\upharpoonright i@\mathtt{p}_i\rangle \qquad \text{[TR-LinkIn]}$$

$$\frac{\tilde{\mathtt{n}}:\tilde{S}\quad A[\tilde{\mathtt{n}}/\tilde{v}]\downarrow\mathsf{true}\quad \Gamma'=\Gamma,\tilde{a}:\tilde{\mathcal{G}}\quad \tilde{a}:\tilde{\mathcal{G}}\text{ occur in }\tilde{\mathtt{n}}:\tilde{S}}{\langle\Gamma,(\Delta,\tilde{s}:k!(\tilde{v}:\tilde{S})\{A\};\mathcal{T}@\mathtt{p})\rangle \xrightarrow{(\nu\tilde{a}:\tilde{\mathcal{G}})\,s_k!\tilde{\mathtt{n}}} \langle\Gamma',(\Delta,\tilde{s}:\mathcal{T}[\tilde{\mathtt{n}}/\tilde{v}]@\mathtt{p})\rangle} \qquad \text{[TR-Send]}$$

$$\frac{\tilde{\mathtt{n}}:\tilde{S}\quad A[\tilde{\mathtt{n}}/\tilde{v}]\downarrow\mathsf{true}\quad \Gamma'=\Gamma,\tilde{a}:\tilde{\mathcal{G}}\quad \tilde{a}:\tilde{\mathcal{G}}\text{ occur in }\tilde{\mathtt{n}}:\tilde{S}}{\langle\Gamma,(\Delta,\tilde{s}:k?(\tilde{v}:\tilde{S})\{A\};\mathcal{T}@\mathtt{p})\rangle \xrightarrow{s_k?\tilde{\mathtt{n}}} \langle\Gamma',(\Delta,\tilde{s}:\mathcal{T}[\tilde{\mathtt{n}}/\tilde{v}]@\mathtt{p})\rangle} \qquad \text{[TR-Rec]}$$

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

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

$$\langle\Gamma,\Delta\rangle \xrightarrow{\tau} \langle\Gamma,\Delta\rangle \qquad \text{[TR-Tau]}$$

Fig. 11. Labelled Transition for End-point assertions

Based on these transition relations, we define two basic relations: the *satisfaction* $\models$, written

$$P \models (\Gamma,\Delta) \tag{7.3}$$

which says when a process P satisfies a transition behaviour specified by a pair of global and endpoint assertion environmens (Γ,Δ); and the *refinement* $\ni$, written

$$\Delta_1 \ni \Delta_2 \tag{7.4}$$

which says Δ_1 refines Δ_2. The relation $\models$ is defined using a conditional simulation which extends the standard typed (bi)simulation in π-calculi [42] in that, in the case of input/branching, simulation is only necessary for the action for each "valid" value or branching label, i.e. that is type-

correct *and* that does not violate the associated assertion, demanding a validated process behaves correctly only if the environment behaves correctly.

Definition 7.2 (Conditional Simulation). *Let $\mathcal{R}$ be a binary relation whose element is of the form $(P, \langle \Gamma, \Delta \rangle)$ where P is a closed process without* errH *or* errT*, Γ a sorting and Δ a collection of singleton assignments, both using only well-asserted global/endpoint assertions. Then $\mathcal{R}$ is a* conditional simulation *if, for each $(P, \langle \Gamma, \Delta \rangle) \in \mathcal{R}$, the following holds:*

- *for each input (resp. branching) move $P \xrightarrow{\alpha} P'$, $\langle \Gamma, \Delta \rangle$ has an input (resp. branching) move at the subject $\mathsf{sbj}(\alpha)$ and, if $\langle \Gamma, \Delta \rangle \xrightarrow{\alpha} \langle \Gamma', \Delta' \rangle$ then $(P', \langle \Gamma', \Delta' \rangle) \in R$.*
- *for each output/selection/τ move $P \xrightarrow{\alpha} P'$, we have $\langle \Gamma, \Delta \rangle \xrightarrow{\alpha} \langle \Gamma', \Delta' \rangle$ such that $(P', \langle \Gamma', \Delta' \rangle) \in R$.*

If $\mathcal{R}$ is a conditional simulation we write $P \lesssim \langle \Gamma, \Delta \rangle$ for $(P, \langle \Gamma, \Delta \rangle) \in \mathcal{R}$.

Definition 7.3 (Satisfaction). *Let P be a closed program phrase and Δ an endpoint assertion assignment. If $P \lesssim \langle \Gamma, \Delta \rangle$ then we say that P* satisfies *Δ* under *Γ, and write $\Gamma \models P \rhd \Delta$. The satisfaction is extended to open processes, denoted $\mathcal{C}; \Gamma \models P \rhd \Delta$ by considering all closing substitutions respecting Γ and $\mathcal{C}$ over Δ and P.*

Thus $\Gamma \models P \rhd \Delta$ says that P always sends valid messages or selection labels and does so under valid conditions; and that P can continue to behave well if it receives messages/labels satisfying the corresponding interaction predicates, without ever going into error.

Note this satisfaction is about partial correctness since if P has no visible actions, then the satisfaction trivially holds (in fact the satisfaction has a direct embedding into a partial correctness fragment of the Hennessy-Milner logic, cf. §8).

In the validation rules, we use the refinement relation $\Delta \ni \Delta'$. This is given as the point-wise extension of the relation over end-point assertions, denoted $\mathcal{T} \ni \mathcal{T}'$. Below recursive assertions are again equated with their unfoldings. An endpoint assertion is *closed* (resp. *open*) if it does not (resp. it may) contain free variables.

Definition 7.4 (Refinement). A binary relation $\mathcal{R}$ over closed well-asserted end-point assertions is a *refinement relation* if $\mathcal{T}_1 \mathcal{R} \mathcal{T}_2$ implies one of the following 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 $A_2\sigma \downarrow \mathsf{true}$.
- $\mathcal{T}_1 = k \oplus \{\{A_i\} l_i : \mathcal{T}_i\}_{i \in I}$ and $\mathcal{T}_2 = k \oplus \{\{A_j\} l_j : \mathcal{T}_j\}_{j \in J}$ where for each $i \in I$, we have $l_i = l_j$, $A_i \supset A_j$ and $\mathcal{T}_i \mathcal{R} \mathcal{T}_j$ for some $j \in J$.
- $\mathcal{T}_1 = k \& \{\{A_i\} l_i : \mathcal{T}_i\}_{i \in I}$ and $\mathcal{T}_2 = k \& \{\{A_j\} l_j : \mathcal{T}_j\}_{j \in J}$ where for each $j \in J$, we have $l_i = l_j$, $A_j \supset A_i$ and $\mathcal{T}_i \mathcal{R} \mathcal{T}_j$ for some $j \in J$.

If $\mathcal{T}_1 \mathcal{R} \mathcal{T}_2$ for some refinement relation $\mathcal{R}$, we say $\mathcal{T}_1$ is a *refinement* of $\mathcal{T}_2$, denoted $\mathcal{T}_1 \ni \mathcal{T}_2$. The relation $\ni$ extends to open endpoint assertions in the standard way.

If $\mathcal{T}_1 \ni \mathcal{T}_2$, then $\mathcal{T}_1$ specifies a *more restricted (more gentle; more well-behaved; more generous) behaviour* than $\mathcal{T}_2$, in the sense that $\mathcal{T}_1$ tolerates less outputs and selections and allows for more branches and more input values. In other words, a simple way to refine an end-point-assertion is to strengthen its interaction predicates for send/selection actions; and/or to weaken those for receive/branching actions. The following says:

> *if P satisfies a specification which is more refined than the other then P also satisfies this less refined specification.*

Proposition 7.5 (Refinement). *If $\Gamma \models P \rhd \Delta$ and $\Delta \ni \Delta'$ then $\Gamma \models P \rhd \Delta'$.*

Proof. By constructing a suitable conditional simulation by the original one and the refinement relation, see Appendix E.3. □

We also note:

Definition 7.6. Let $\mathcal{T}$ be a closed endpoint assertion. Define *Tree*$(\mathcal{T})$ to be the tree generated by applying the following the following unfolding:

$$\mathsf{unfold}(\mu \mathbf{t}\langle \tilde{e} \rangle (\tilde{v} : \tilde{S})\{A\}.\mathcal{T}) \stackrel{\text{def}}{=} \mathcal{T}[\mu \mathbf{t}(\tilde{v} : \tilde{S})\{A\}.\mathcal{T}/\mathbf{t}][\tilde{e}/\tilde{v}]$$

where $\mathcal{T}[\mu \mathbf{t}(\tilde{v} : \tilde{S})\{A\}.\mathcal{T}/\mathbf{t}]$ substitutes $\mu \mathbf{t}\langle \tilde{e}_i \rangle (\tilde{v} : \tilde{S})\{A\}.\mathcal{T}$ for each call $\mathbf{t}\langle \tilde{e}_i \rangle$ in $\mathcal{T}$. Then we write $\mathcal{T} \approx_{\mathsf{tree}} \mathcal{T}'$ when *Tree*$(\mathcal{T})$ = *Tree*$(\mathcal{T}')$.

Proposition 7.7. *$\mathcal{T} \approx_{\mathsf{tree}} \mathcal{T}'$ implies $\mathcal{T} \ni \mathcal{T}'$. Hence, writing $\Delta \approx \Delta'$ when Δ and Δ' only differ in $\approx_{\mathsf{tree}}$-related endpoint assertions, $\Gamma \models P \rhd \Delta$ and $\Delta \approx \Delta'$ imply $\Gamma \models P \rhd \Delta'$ again.*

Proof. Immediate since $\mathcal{T} \approx_{\mathsf{tree}} \mathcal{T}'$ means $\mathcal{T}$ and $\mathcal{T}'$ have precisely the same transitions. □

Note Proposition 7.7 justifies the rule in (5.1), page 30. We can further show $\approx_{\mathsf{tree}}$ and $\ni \cap \ni^{-1}$ coincide.

7.2 Soundness

The proof of soundness is done in the following three steps:

- (Step 1) We show that the visible transitions of a process preserve provability.
- (Step 2) Using Step 1 we show the same for the tau-action.
- (Step 3) We establish a conditional simulation through the rule induction on validation rules, using the results from the previous steps.

Step 2 demands analysis of the interplay between the τ-action (reduction) and provability, and for that purpose we need validation rules for runtime processes, which add to those in Figure 10 the validation rules of queues on the basis of the typing technique in [5] and the following rule for hiding session channels [5, 27]:

$$\frac{\Gamma \vdash P \rhd \Delta, \tilde{s}\!:\!\{\mathcal{T}_{\mathtt{p}} \mathbin{@} \mathtt{p}\}_{\mathtt{p}\in I} \quad \{\mathcal{T}_{\mathtt{p}} \mathbin{@} \mathtt{p}\}_{\mathtt{p}\in I} \text{ coherent}}{\Gamma \vdash \ (\boldsymbol{\nu}\, \tilde{s})P \rhd \Delta} \qquad [\textsc{CRes}]$$

where we say $\{\mathcal{T}_{\mathtt{p}} \mathbin{@} \mathtt{p}\}_{\mathtt{p}\in I}$ is *coherent* if for some $\mathcal{G}$ s.t. $I = \mathsf{pid}(\mathcal{G})$, we have $\mathcal{T}_{\mathtt{p}} \ni \mathcal{G} \upharpoonright \mathtt{p}$ for each $\mathtt{p} \in I$. The underlying type discipline also demands, when hiding session channels, that there is a queue at each of these channels. The following is proved for processes under the extended validation using Lemma 5.4.

Proposition 7.8 (Subject Transition for Visible Actions). *Let $\Gamma \vdash P \rhd \Delta$ be a closed process and $\langle \Gamma, \Delta\rangle \xrightarrow{\alpha} \langle \Gamma', \Delta'\rangle$ with $\alpha \neq \tau$. Then $P \xrightarrow{\alpha} P'$ implies $\Gamma' \vdash P' \rhd \Delta'$.*

Proof. By case analysis, see Appendix E.5. □

For the τ-action, since a reduction at free session channels changes the shape of linear typing as discussed in [5, 27], it also affects the shape of end-point assertions. [10] We solve this problem following the method in [5], using a non-deterministic reduction of assertion assignments, written $\Delta \to \Delta'$, which is given in Appendix E.6.

By Proposition 7.8 and by a technical lemma on reduction (which attributes each kind of reduction at a session channel to the corresponding visible action together with a change in the shape of a queue content), we obtain:

[10] This does not affect the statement of Theorem 7.10 since derivatives of program phrases never have reduction at free session channels.

Lemma 7.9 (Subject Reduction). *Suppose* $\Gamma \vdash P \rhd \Delta$ *and* $P \xrightarrow{\tau} P'$. *Then* $\Gamma \vdash P' \rhd \Delta'$ *such that either* $\Delta' = \Delta$ *or* $\Delta \to \Delta'$

Proof. By Prop. 7.8 and Lemma 5.4, see Appendix E.6. □

The main result of this section follows.

Theorem 7.10 (Soundness of Validation Rules). *Let* P *be a program phrase. Then* $\mathcal{C}; \Gamma \vdash P \rhd \Delta$ *implies* $\mathcal{C}; \Gamma \models P \rhd \Delta$.

Proof. (outline) We let $\mathcal{R}$ be the set of pairs such that each has the form $(P_\sigma, \langle \Gamma\sigma, \Delta_\sigma \rangle)$ such that $\mathcal{C}; \Gamma \vdash P \rhd \Delta$, σ is a Γ-$\mathcal{C}$ respecting closing substitution, P is a runtime process, and prove $\mathcal{R}$ is a conditional simulation extended to runtime processes and non-singleton assertion assignments, using induction on the validation rules. The rules for prefixes ([Send], [Rcv], [Sel] [Branch], [Mcast], and [Macc]) results in processes with only visible actions, for which we use Prop. 7.8 and induction hypothesis. The non-trivial case for [Conc] is when the τ-action takes place, for which and for [If], we use Lemma 7.9. [Inact] is vacuous. [NRes] uses induction hypothesis. [Var] is direct from the assumption. [Def] uses the standard syntactic approximation of a recursive process, starting from $\mathbf{0}$ (which satisfies any specification). Finally soundness [Conseq] is direct from Proposition 7.5. See Appendix E.7 for the full proofs. □

As an immediate corollary we obtain the error freedom. Below we say $\langle \Gamma, \Delta \rangle$ *allows* a sequence of actions $\tilde{\alpha}$ if for some $\langle \Gamma', \Delta' \rangle$ we have $\langle \Gamma, \Delta \rangle \xrightarrow{\tilde{\alpha}} \langle \Gamma', \Delta' \rangle$.

Theorem 7.11 (Error Freedom w.r.t. Arbitrary Transition). *Suppose* $\Gamma \vdash P \rhd \Delta$ *and* $P \xrightarrow{\tilde{\alpha}} P'$ *such that* $\langle \Gamma, \Delta \rangle$ *allows* $\tilde{\alpha}$. *Then* P' *contains neither* errH *nor* errT.

7.3 Effective Validation and Runtime Monitoring

Convention 7.12. In the rest of the present section we assume each interaction predicate in a well-asserted global assertion only uses a quantifier-free logical formula.

Effective Validation. We first discuss an algorithm for design-time (static) validation of programs. First we observe that the validation rules in Figure 10 in effect give the compositional proof rules for typed, but unasserted, processes. To clarity, Figure 12 takes off the elaboration of logical formulae

from the rules for the prefixes for session communication from Figure 10 (note the remaining rules in Figure 10 do not mention logical formulae inside a process).

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

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

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

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

Fig. 12. Validation for Unasserted Session Actions

Note however the rules do not directly induce an algorithm, due to the shape of [Def] (which introduces a new hypothesis when going from the conclusion to the premise) and because [Conseq] is needed for equating recursive endpoint assertions and their unfolding. The following rule, catering for a simple recursive definition (easily extensible to the full syntax of process definition) mitigates the former by demanding a parallel search and avoids the latter issue by assuming the structural correspondence between session recursions and process recursions, which may often be found in practice.

$$\frac{\mathcal{C}; \Gamma, \tilde{v} : \tilde{S}, \; X : (\tilde{v} : \tilde{S}) \mathcal{T} \, @ \, \mathtt{p} \vdash P \rhd \tilde{s_1} : \mathsf{unfold}(\mathcal{T}) \, @ \, \mathtt{p} \quad LSat(\mathcal{T}', \mathcal{C}) \quad \mathcal{T}[\tilde{e}/\tilde{v}] = \mathcal{T}'}{\mathcal{C}; \Gamma \vdash \mathsf{def} \; X(\tilde{v} \tilde{s}_1 .. \tilde{s}_n) = P \; \mathsf{in} \; X \langle \tilde{e} \tilde{s} \rangle \; \rhd \; \tilde{s} : \mathcal{T}' \, @ \, \mathtt{p}}$$

Note there can be multiple candidates for $\mathcal{T}$ such that $\mathcal{T}[\tilde{e}/\tilde{v}] = \mathcal{T}'$.

Write $\Gamma \vdash^{\star} P \rhd \Delta$ for the validation rules which dispenses with [Def] and [Conseq] while adding this rule. By reading each rule from the bottom to top we obtain a decidable algorithm for validation (which demands parallel/non-deterministic search for multiple candidates for recursion noted above as noted above, see Appendix E.9).

Theorem 7.13 (Effective Validation). *Given Γ, P and Δ, the provability of $\Gamma \vdash^{\star} P \triangleright \Delta$ is decidable.*

Proof. By the obvious recursive algorithms and decidability of validity of the underlying logical language, see Appendix E.9. □

Communication Monitoring. Because of e.g. internal loops between communication actions, a complete algorithmic validation of programs may not always be feasible. A flexible and modular method for ensuring dynamic safety of communicating programs is *runtime local communication monitoring* at each endpoint, formalised as reduction/transition with predicate checking in the presented calculus (cf. Figure 8). Using this formalisation, we clarify potential ways the proposed assertion method may be used for monitoring.

Firstly, a basic role of such a monitor is, in an untrusted environment, to filter bad incoming messages and to ensure proper outgoing messages. Recall, in our model, monitors notify violations through reduction to errH or errT. Note also $\Gamma \vdash^{\mathsf{elab}} P \triangleright \Delta$ implies P has neither errH nor errT.

Proposition 7.14 (Monitoring, 1). *Assume P is a closed process without errT nor errH such that $\Gamma \vdash^{\mathsf{elab}} P \triangleright \Delta$, where Δ consists of singleton assignments.* (1) *If $P \xrightarrow{\alpha} P'$ such that α is output/selection/shared action, then $\Gamma' \vdash^{\mathsf{elab}} P' \triangleright \Delta'$ with $\langle \Gamma, \Delta \rangle \xrightarrow{\alpha} \langle \Gamma', \Delta' \rangle$.* (2) *If $P \xrightarrow{\alpha} P'$ such that α is either input or branching, and if $\langle \Gamma, \Delta \rangle \xrightarrow{\alpha} \langle \Gamma', \Delta' \rangle$, then P' is without errT and $\Gamma' \vdash^{\mathsf{elab}} P' \triangleright \Delta'$.* (3) *If $P \xrightarrow{\tau} P'$ and P' is without errH, then $\Gamma \vdash^{\mathsf{elab}} P' \triangleright \Delta'$ with $\langle \Gamma, \Delta \rangle \xrightarrow{\alpha} \langle \Gamma, \Delta' \rangle$.*

Proof. All are immediate from the definition of elaboration rules. □

Above (1) says that a monitor never allows a wrong output/selection (i.e., where α is violating); (2) says a monitor can always detect an erroneous input/branching.

The monitoring can also be used in a trusted environment where we can assume all components are either elaborated or validated, as in e.g. debugging. Write $\Gamma \vdash P$ (resp. $\Gamma \vdash^{\mathsf{elab}} P$, resp. $\Gamma \models P$) when $\Gamma \vdash P \triangleright \varnothing$ (resp. $\Gamma \vdash^{\mathsf{elab}} P \triangleright \varnothing$, resp. $\Gamma \models P \triangleright \varnothing$). We write $\Gamma \models P \sim Q$ when $\Gamma \models P$, $\Gamma \models Q$ and P and Q are strongly bisimilar by their typed transitions. Further $\mathsf{erase}(P)$ (resp. $\mathsf{eraseO}(P)$) denotes the result of turning all predicates annotating communications (resp. annotating output/selection) to be true.

Theorem 7.15 (Monitoring, 2). *Let P be a closed program and $\Gamma \vdash^{\mathsf{elab}} P$.*

1. *(safety assurance) $P \to^+ Q$ implies Q does not contain* errT.
2. *(consistency) If $\mathsf{eraseO}(P) \to^+ Q$ such that Q contains* errT *then $P \to^+ Q'$ such that Q' contains* errH.
3. *(monitor freedom) If $\Gamma \vdash P$ then $P \to^+ Q$ implies Q contains neither* errT *nor* errH.
4. *(erasure, 1) If $\Gamma \vdash P$ then $P \to^+ Q$ implies $\mathsf{erase}(P) \to^+ \mathsf{erase}(Q)$; and $\mathsf{erase}(P) \to^+ R$ implies $R \equiv \mathsf{erase}(Q)$ such that $P \to^+ Q$.*
5. *(erasure, 2) If $\Gamma \vdash P$ then we have $\Gamma \models \mathsf{erase}(P)$ and $\Gamma \models P \sim \mathsf{erase}(P)$.*

Proof. See Appendix E.8. □

Above (1) says that if all participants are elaborated, a bad incoming message never arrives, as it would generate a errH in the sending party. (2) says that an input/branching violation can always be attributed to a send/selection violation. (3) is an immediate corollary of Theorem 7.11, and indicates that we can dispense with monitors if we know all participants are validated. (4) is the standard erasure, while (5) strengthens (4) with visible behaviour.

In practice, a local communication monitor may be automatically generated from assertions as a stub code, integrating a finite state automaton [45] with predicate checking. Such a monitor is relatively efficient if working under Convention 3.3 and assuming global assertions with quantifier-free predicates. Because of the shape of projected formulae (Definition 4.2), monitoring output/selection is polynomial time w.r.t. the size of a message and the predicate. Via Theorem 7.15 (1), this suggests an efficient debugging framework through output/selection monitoring. For input/branching, the need of existential quantifiers for indirect predicates (cf. page 21) makes a good theoretical bound hard to obtain, though the fixed shape of quantifications and logical simplifications before deployment may improve practical efficiency.

8 Further Topics, Related Work and Conclusion

This work introduced an assertion method to specify behavioural constraints for multiparty sessions starting from global descriptions, and presented basic technical results underpinning its practical usage. Our aim is to generalise the merit of the widely practiced traditional DbC to the context of distributed communications, enabling clear and precise descriptions and their effective use for practical concerns [46, 48]. In the following we discuss related works and further technical topics.

Hennessy-Milner Logic. Hennessy-Milner Logic (HML) is an expressive modal logic with a strong semantic characterisation [25]. HML can specify essentially arbitrary behavioural properties of processes, and may offer a common basis for different specification/verification methods as suggested in the preceding works [4, 19] (in fact it is not hard to see that endpoint assertions can be embedded into the Hennessy-Milner logic [25] for the π-calculus with parametrised fixed point operators [19] based on weak modality [4]).

The present work introduces two technical elements that address the key challenges for logical specifications of processes, unexplored in the context of HML. The first is global assertions, resulting in significant concision of descriptions while enjoying clarity as well as generality within its scope (description of individual protocols). The preceding works [4, 19] show practical specifications in HML tend to be lengthy and complex. In fact, the direct use of HML is tantamount to *reversing* the proposed specification process depicted in Figure 1 in Introduction: we start from endpoint specifications and later try to check their mutual consistency. This process would more than double the size of specifications a user needs to write. It lacks clarity as to the target interaction structures, and mutual consistency checking may not be generally effective. For tractable specifications, recent studies and industry experiences strongly suggest the paramount merit of global descriptions for the specifications of communicating processes.

The second novel element is the provision of effective methods for some of the key engineering concerns in modelling and building communications. For endpoint communication monitoring, Proposition 7.14 and Theorem 7.15 illustrate different ways the proposed assertion method may be employed. Communication monitoring plays a fundamental role in distributed applications, including runtime management. No similar results are known in the context of HML. Further Theorem 7.13 offers a basis for statically verifying π-calculus processes with full recursion, starting

from global assertions. This is made possible through the use of underlying type structures. Because of its expressiveness, effective validation of general π-calculus processes against general HML formulae is impossible and the only known results are for finite state processes [19].

State and Cross-Session Assertions In the DbC for sequential languages, a state is often treated in association with the notion of *attributes* (instance variables) of an object. These attributes have a special status: they are public, but the state is encapsulated. This framework is particularly relevant in some domains of distributed interactions such as business protocols, where some state is audited in conjunction with external interactions. In this case, type signature should contain these variables, and a global assertion can specify interaction predicates referring to the present and preceding ("hooked") state, stipulating the state change a sender (or a selector) should honour when that action takes place, just as we have done in the present work for stateless interaction variables.

Another use of state is when it is not declared in a global contract: state can be entirely local to a process. Since [Conseq] allows local refinement, using state specifications locally to a process over multiple sessions does not contradict with the presented framework (note that our proof rules do check the correctness of the behaviour across multiple sessions: the main point is its *specificaitons* are done for individual sessions).

Correspondence Assertions. The authors of [7] combine session-types with *correspondence assertions*, for checking the correspondence of "effects" depending on some values. Critical code is annotated with **begin** L and **end** L assertions and the type system checks that each **begin** is matched by the corresponding **end** effect. Assertions L in the effects of [7] are lists of values, not general formulae. It is an interesting topic to relate how global assertions can represent the notions related to correspondence assertions.

Integration of Types and Logics As we observed in Introduction, Design-by-Contract integrates type signatures and logical formulae for enriched contracts, used throughout the software engineering process. There are other works which integrate types and logical formulae. In X10 [18] and several dependently typed formalisms [44, 49], we use formulae to flexibly extend the specifications given by types. In X10 [18], this is based on extending types with notations for constraints (there are other novel types such as types for distributed locations). In dependently typed functional languages [44, 49], dependent types combined with constraints on value

range contribute to more exact safety guarantee. In these languages these specifications are mainly used for constraint on data structures and functions, not for the behaviour of communicating processes as in the present work. Combination of these methods and the present work will be an interesting topic for further study.

Design and Analysis of Service-Oriented Computing. DbC-like approaches have recently been promoted in the design and analysis of SOC applications [8, 15, 16]. A theory of contracts for Web Services based on *filters* (namely, behavioural types equipped with a preorder) is proposed in [16]; filters allow one to formally define the compatibility between a client and a service in order to statically check that interactions successfully complete. An approach similar to filters has been taken in [8]; however, the latter work introduces the possibility of refining contracts for multiparty protocols by means of a must-testing preorder. In our work we consider multiparty protocols, global assertions predicate about values, and our theory is based on the π-calculus while the theories in [16] and [8] have been defined for CCS-based contracts.

A few works on SOC focus on *compliance* of client and services. Very roughly, the idea is to define compliance in terms of deadlock-freedom. For instance, notions of compliance based on must-testing are proposed in [8] and [17]. In [17] (which extends the approach introduced in [16] to multiparty protocols modelled by mobile processes) contracts are formalized by types of the form $T_1 | \dots | T_n$ used to check compliance (T_i are session types that use filters): the contract is well formed if each T_i is compliant with $T_1 | \dots T_{i-1} | T_{i+1} | \dots T_n$, namely the latter *must*-pass the test T_i. Similarly, in [1] a type system guaranteeing a progress property of clients is defined on CaSPiS, a inspired by the π-calculus and featuring pipeline communications; the type system ensures that in (non-diverging) well-typed (two parties) systems clients' invocations do not stuck because of inadequate interactions with the service. A limit of those approaches wrt global assertions is that they abstract away from values and focus only on protocols interactions.

In [2] a functional calculus (called λreq) features a "call-by-contract" mechanism for selecting and invoking services that respect given behavioural requirements. Services are modelled as functions with side effects representing the access to resources and logged into *histories*. Code is decorated with *framings* which enforce safety properties on execution histories by means of a type and effect system which over-approximates the actual run-time interactions. The same authors propose a methodol-

ogy for designing and composing services [3] enforcing safety properties. Static analysis and model checking techniques enable the verification and monitoring of systems. A limit of the approaches in [2, 3] is that they can only deal with finite-state systems.

Rely-Guarantee Reasoning for Processes. Several authors apply the rely-guarantee method [29, 30] to communicating processes, starting from one of the earliest works on the rely-guarantee method by Misra and Chandy, cf. [13, 21, 31, 32, 38, 50]. In particular, Misra and Chandy already present a compositional proof system for their reasoning method. In these works we can already find assertions based on the distinction of obligations and guarantees for senders and receivers. Some of the central differences of the proposed work from these preceding works is, apart from the use of the π-calculus, the use of type signature for interactions as a basis of an assertion method and the results on the effectiveness of the dynamic and static validation methods.

In a context closer to the present work, Broy and Krueger [11] and later Broy, Krueger and Meisinger [12] present a formal notion of interaction interface based on input/output streams and predicates over them (their stream-based model follows [9]), modelling Message Sequence Charts (MSCs). MSCs are often used for high-level modelling for communicating systems. They define a relation (predicate) over input and output streams, which, due to its relational nature, can express obligations and guarantees of each participant. The use of streams can naturally model systems whose structure is defined statically including embedded systems. The use of the π-calculus in the present work leads to a natural accommodation of dynamic features in target processes, including generation of new channels and processes. Since they directly predicate over streams, their framework lacks a compositional proof system, which is the central element of the present work. Aspects of realizability and computability of predicate over streams are studied in [10].

Process Calculi with Constraints. In another vein, a number of works combine process calculi with constraints (e.g., [47]). In [14, 20] the combination of process calculi and concurrent constraint programming has been applied to model constraints that specify a Service Level Agreement on Quality of Service parameters. Our aim is to provide a framework, through the use of types, that ensures the satisfaction of the required constraints through static validation or run-time monitoring.

Sequence Diagrams. Sequence diagrams allows assertion annotations called *guards*. Their standard usage follows the semantics of method invocation in sequential objects, specifying a legal state of an invoker. The present work may offer a framework to use such diagrams for specifying distributed asynchronous interactions with a formal basis.

References

1. L. Acciai and M. 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. M. Bartoletti, P. Degano, G. Ferrari, and R. Zunino. Secure service orchestration. In *FOSAD*, pages 24–74, 2007.
3. M. Bartoletti, P. Degano, G. Ferrari, and R. Zunino. Semantics-based design for secure web services. *IEEE Transactions on Software Engineering*, 34(1):33–49, 2008.
4. M. Berger, K. Honda, and N. Yoshida. Completeness and logical full abstraction for modal logics for the typed π-calculus. In *ICALP*, 2008.
5. L. Bettini et al. Global Progress in Dynamically Interfered Multiparty Sessions. In *CONCUR'08*, volume 5201 of *LNCS*, pages 418–433. Springer, 2008.
6. E. Bonelli and A. Compagnoni. Multipoint Session Types for a Distributed Calculus. In *TGC'07*, volume 4912 of *LNCS*, pages 240–256, 2008.
7. E. Bonelli, A. Compagnoni, and E. Gunter. Correspondence assertions for process synchronization in concurrent communications. *J. of Functional Programming*, 15(2):219–247, 2005.
8. M. Bravetti and G. Zavattaro. A foundational theory of contracts for multi-party service composition. *Fundamenta Informaticae*, XX:1–28, 2008.
9. M. Broy. Advanced component interface specification. In *TPPP '94: Proceedings of the International Workshop on Theory and Practice of Parallel Programming*, pages 369–392, London, UK, 1995. Springer-Verlag.
10. M. Broy. Interaction and realizability. In *SOFSEM '07: Proceedings of the 33rd conference on Current Trends in Theory and Practice of Computer Science*, pages 29–50, Berlin, Heidelberg, 2007. Springer-Verlag.
11. M. Broy and I. Krüger. Interaction interfaces - towards a scientific foundation of a methodological usage of message sequence charts. In *ICFEM '98: Proceedings of the Second IEEE International Conference on Formal Engineering Methods*, page 2, Washington, DC, USA, 1998. IEEE Computer Society.
12. M. Broy, I. Krüger, and M. Meisinger. A formal model of services. *ACM Trans. Softw. Eng. Methodol.*, 16(1):5, 2007.
13. M. Broy and K. Stolen. *Specification and development of interactive systems: focus on streams, interfaces, and refinement*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2001.
14. M. Buscemi and U. Montanari. Cc-pi: A constraint-based language for specifying service level agreements. In *ESOP*, volume 4421 of *LNCS*, pages 18–32. Springer, 2007.
15. L. Caires and H. Vieira. Conversation types. In G. Castagna, editor, *ESOP08*, volume 5502 of *LNCS*, pages 285–300. Springer, 2008.
16. G. Castagna, N. Gesbert, and L. Padovani. A theory of contracts for web services. In *POPL'08*, pages 261–272. ACM, 2008.
17. G. Castagna and L. Padovani. Contracts for mobile processes. In *CONCUR*, 2009. To appear.
18. P. Charles et al. X10: an object-oriented approach to non-uniform cluster computing. In *OOPSLA*, pages 519–538, 2005.
19. M. Dam. Proof systems for pi-calculus logics. In *Logic for Concurrency and Synchronisation, R. de Queiroz (ed.)*, Trends in Logic, Studia Logica Library, pages 145–212. Kluwer, 2003.

20. R. De Nicola, G. Ferrari, U. Montanari, R. Pugliese, and E. Tuosto. A Basic Calculus for Modelling Service Level Agreements. In *Coordination*, 2005.
21. W.-P. de Roever, F. de Boer, U. Hannemann, J. Hooman, Y. Lakhnech, M. Poel, and J. Zwiers. *Concurrency Verification: Introduction to Compositional and Noncompositional Methods*. CUP, 2001.
22. R. W. Floyd. Assigning meaning to programs. In *Proc. Symp. in Applied Mathematics*, volume 19, 1967.
23. D. S. Frankel. *Model Driven Architecture: Applying MDA to Enterprise Computing*. Wiley, January 2003.
24. M. K. Ganai. Efficient decision procedure for bounded integer non-linear operations. In *HVC '08*, pages 68–83. LNCS, 2009.
25. M. Hennessy and R. Milner. Algebraic Laws for Non-Determinism and Concurrency. *JACM*, 32(1), 1985.
26. T. Hoare. An axiomatic basis of computer programming. *CACM*, 12, 1969.
27. K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. In *POPL '08*, pages 273–284. ACM, 2008.
28. *The Java Modeling Language (JML) Home Page*.
29. C. B. Jones. Development methods for computer programs including a notion of interference, June 1981. Printed as: Programming Research Group, Technical Monograph 25.
30. C. B. Jones. Specification and design of (parallel) programs. In *IFIP Congress*, pages 321–332, 1983.
31. C. B. Jones. Accommodating interference in the formal design of concurrent object-based programs. *Formal Methods in System Design*, 8(2):105–122, 1996.
32. A. Kay and J. N. Reed. A rely and guarantee method for timed csp: A specification and design of a telephone exchange. *IEEE Trans. Softw. Eng.*, 19(6):625–639, 1993.
33. L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–564, July 1978.
34. L. Lamport and F. B. Schneider. The "hoare logic" of csp, and all that. *ACM Trans. Program. Lang. Syst.*, 6(2):281–296, 1984.
35. K. R. M. Leino. Verifying object-oriented software: Lessons and challenges. In *TACAS*, page 2, 2007.
36. E. Mendelson. *Introduction to Mathematical Logic*. Wadsworth Inc., 1987.
37. B. Meyer. Applying "design by contract". *Computer*, 25(10):40–51, 1992.
38. J. Misra and K. M. Chandy. Proofs of networks of processes. *IEEE Trans. Software Eng.*, 7(4):417–426, 1981.
39. G. Nelson and D. C. Oppen. A simplifier based on efficient decision algorithms. In *POPL'78*, pages 141–150. ACM, 1978.
40. OMG. Object constraint language version 2.0, may 2006.
41. B. C. Pierce. *Types and Programming Languages*. MIT Press, 2002.
42. B. C. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. In *LICS*, 1993.
43. W. 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.
44. P. M. Rondon, M. Kawaguchi, and R. Jhala. Liquid types. In R. Gupta and S. P. Amarasinghe, editors, *PLDI*, pages 159–169. ACM, 2008.
45. F. B. Schneider. Enforceable security policies. *ACM Trans. Inf. Syst. Secur.*, 3(1):30–50, 2000.

46. UNIFI. ISO 20022 UNIversal Financial Industry message scheme. http://www.iso20022.org, 2002.
47. B. Victor and J. Parrow. Constraints as processes. In *CONCUR*, pages 389–405. Springer, 1996.
48. W3C. Web Services Choreography Description Language. `http://www.w3.org/2002/ws/chor/`.
49. H. Xi and F. Pfenning. Dependent types in practical programming. In *POPL*, pages 214–227, 1999.
50. J. Zwiers. *Compositionality, Concurrency and Partial Correctness - Proof Theories for Networks of Processes, and Their Relationship*, volume 321 of *Lecture Notes in Computer Science*. Springer, 1989.

A Additional Notes for § 1

A.1 Motivations and Contributions: A Further Note

We summarised motivations for the introduction of the proposed framework in Introduction (§ 1), as well as how the proposed approach answers to these motivations. In more detail, we consider the following elements are fundamental for having a sound framework of specifications and validations for application-level distributed systems.

- A **specification method** for assertions so that we can easily and consistently describe what is the *correct* or *desired* form of interactions, which should be associated with an **effective consistency checking**.
- A **semantics of specifications (assertions)** which define the relationship between a specification and the behaviour of processes, together with **compositional proof rules for processes** by which we can compositionally validate whether or not a given process satisfies a given specification; and a **dynamic message monitoring** which can test communication behaviours of each process against a specification efficiently.
- If possible, a **static process validation** which is justified by the proof rules above and by which we can effectively check the correctness of processes against specifications at a compile time.

The present framework is intended to offer potential answers to all of these challenges (and other potential ones). A fundamental idea is to consider only *per-session (or per-type-signature) specifications* of process behaviours and their validation, inheriting the idea from the traditional DbC.

This aspect of the proposed approach contributes to, among others, the static validation of processes. Because of the expressiveness of the π-calculus, it is generally impossible to have a decidable method for validating a process against a non-trivial specification. In the proposed theory, this last point, as well as the preceding two, is made possible through the use of a global signature for multiparty distributed interaction: since each specification is given by the unit of a type signature for interaction, and a process is effectively checkable against this type signature, we can first extract the structure of interactions for *each* session. A global assertion (hence each endpoint assertion projected from it) annotates this type signature, stipulating in what ways a conversation should proceed, including:

– *What values will be carried* at each sending/receiving action;

– *What actions will be chosen* at each branching/selection action, hence how different actions lead to different sub-conversations with different session structures (thus specifying potential structures a session may have depending on accumulated constraints of interactions).

These specifications are given by elaborating each session type signature. A key technical consequence of this specification-per-type-signature approach is the following point:

> *We do not have to consider the effects of interaction among processes; we only have to demand that, at each communication point, the process guarantees specified properties for its own actions (send and select) as far as it can rely on specified properties for its peers' actions (receive and branch).*

There is no need for calculating the effects of interactions and interference among processes one by one because logical specifications are given in the unit of session type signatures and because the rely-guarantee method [30] in each specification enables composability by simple reciprocity checking.

The use of per-session specifications also allows us to specify the properties of communication behaviours independent from individual processes, which is important for the modelling and programming process since a program may often interleave multiple sessions and may change their usage over time.

One technical underpinning of the proposed framework as a specification method is the introduction of **recursive invariants** (in both global and endpoint assertions), discussed in §8. As discussed there, the aim of a recursive invariant is to facilitate a consistent specification of recursive behaviours, especially when different instances of the same recursion/loop may change their values and branching behaviours. Such *state changing repetition (recursion) of interactions* can be easily specified once we have a recursive invariant. We observe:

1. Syntactically it enables the simple validation algorithm for consistency of global assertions (well-assertedness, Definition 3.7, page 16). Without such algorithmic checking, we even are not sure whether a given specification stipulates the behaviour we do demand or not.
2. Semantically its meaning is taken extensionally: the general proof/-validation rules (Figure 10) do not mention recursive invariants but use them through the induced **semantic** constraints on interactions, i.e. what communication behaviours they impose, by taking recursive assertions up to their unfolding (through [Conseq]).

As discussed in §8 in the main sections, the proposed framework easily extends to specifications involving states: in such contexts, the use of recursive invariants can specify invariants to hold between e.g. communicated values and its local state.

A.2 Limitations of the Proposed Approach

A limitation of the proposed framework comes from its deliberate choice to limit the specification of properties of processes to individual sessions. We *can* link multiple predicates inside an asserted process — for example, we may specify a value which a process sends in one session will be the double of a value it receives in another interleaving session. However a global assertion itself is still limited to the unit of a session.

A consequence of this design choice is that this framework is *not* about specifying the whole of a program behaviour: rather the specification is limited to the program's behaviour with respect to each protocol it would use. Limited to this specification, it allows not only efficient monitoring but also decidable process validation (decidable relative to the decidability of the underlying logic). Thus this choice has merits, but in some situations, it leads to limitations.

One engineering situation where this choice may turn out to be a limitation is when one primarily wishes to specify and validate the properties of a process or processes across sessions (for example when one is concerned with termination of the whole process). The second and related scene is when one wishes to assert for, and validate for, a combination of several sessions globally, observing some protocols may often be used in combination. The need for such a combination indeed arises in our exploration of existing business protocols.

In this way a more complex situation naturally demands a more complex method, leading to full functional specifications and validations of a communicating program. In this spectrum of different specification and verification methods, the present "per-type-signature"-based method is intended to offer a basic stratum which is often all that is feasibly specifiable as shared behavioural contracts for public or semi-public protocols. Having the proposed method as a stratum will also help ease the complexity when one needs to rise to the challenge of more complex and wholesome specifications and validations (which corresponds to how the traditional DbC eases a verification of the whole program property by enabling abstract treatment of each procedural call in the program).

A further challenge is to combine and integrate different methods catering for different concerns, for which we may need a common logical

basis (the use of Hennessy-Milner logic in §8 may suggest such a potential). Another question which the present inquiry starts to pose is how we may combine types and assertions for specifications and validations of properties of communicating processes. In the present work assertions are directly given on the basis of types. Here types offer fundamental hooks where assertions can be associated and which enables efficient validations. Further ramifications and consequences of this approach as well as others are worth exploring in both theory and practice.

B Appendix for § 3

B.1 Further Examples

History Sensitivity Principle (HSP) We present some motivating examples for the consistency principles, introduced in § 3.2. We start from history sensitivity. As interactions take place, processes concretize values specified in assertions and accumulate constraints constraining future paths. At each point when a sender sends a value or a selection label, a local monitor may check whether or not the assertion is satisfied; in turn a receiver may also check if e.g. the received value is within the range specified in the contract. The temporal flow plays a fundamental role, since the validation of the interaction predicate on the coming interaction will use values determined (instantiated) in the preceding interactions.

When one sends a message, or chooses one of the potential selection actions, the constraint on its values or on the chosen label should be checked effectively and efficiently. For this purpose, the formula (the interaction predicate) should only use interaction variables which are already determined by the preceding interactions at that local site, in addition to the interaction variable newly introduced in that interaction predicate.

The history-sensitivity principle (HSP) stipulates precisely this condition, saying that an interaction predicate can only contain those variables which its *sender* previously experienced. Consider:

$$\begin{array}{l}\mathtt{p} \to \mathtt{q} : (v : \mathsf{int})\{v \geqslant 1\}.\\ \mathtt{q} \to \mathtt{r} : (w : \mathsf{int})\{w \geqslant 2\}.\\ \mathtt{r} \to \mathtt{p} : (u : \mathsf{int})\{u = w + 5\}.\\ \mathtt{p} \to \mathtt{r} : (v' : \mathsf{int})\{v' = v + u + 3\}.\\ \mathsf{end}\end{array}$$

does respect HSP, because:

- In the first interaction, $\mathtt{p}$ is the active party. Since there is no previous interactions, it can only use v, its newly introduced variable.
- In the second interaction, $\mathtt{q}$ is an active party: the use of v (beside w which is newly introduced) is justified hence $\mathtt{q}$ does indeed know the value of v in the first interaction. The third interaction is similar.
- In the final interaction, $\mathtt{p}$ is an active party, which knows the values of v and u: hence, besides v', we can use these variables to specify the interaction predicate.

As an example violating HSP, consider the following three-party interaction:

$$\begin{array}{l}\mathtt{p} \to \mathtt{r} : (v : \mathsf{int})\{v \geqslant 3\}.\\ \mathtt{q} \to \mathtt{r} : (w : \mathsf{int})\{w = v + 3\}.\mathsf{end}\end{array}$$

It looks the assertion looks all right, but in fact not quite, as far as we follow the interpretation of the underlying type specification: here the two interactions, one from $\mathtt{p}$ and the other from $\mathtt{q}$ *should be considered as unorderd*: there is no temporal preceding between them. In that case it is absurd for $\mathtt{q}$ to try to check its sending value denoted by w by the value which has been sent somewhere remote asynchronously (it may not have taken place yet).

The following situation also violates the HSP and is indeed undesirable for the purpose of monitoring:

$$\begin{array}{l}\mathtt{p} \to \mathtt{q} : (v : \mathsf{int})\{v \geqslant 3\}.\\ \mathtt{q} \to \mathtt{r} : (w : \mathsf{int})\{w \geqslant v + 3\}.\\ \mathtt{r} \to \mathtt{s} : (u : \mathsf{int})\{u \geqslant v + w\}.\mathsf{end}\end{array}$$

In the third line, the interaction predicate specifies for both v and w: but the value of v cannot be known to $\mathtt{r}$ given the value passing for $\mathtt{r}$ took place between $\mathtt{p}$ and $\mathtt{q}$, without involving $\mathtt{r}$. Thus, for deciding the validity of the predicate, even if the sender decides on the value of u, it cannot check if it is valid or not for sure (note all that the sender can do in this situation is to search for a value of w satisfying the predicate, but not only this is relatively costly but also satisfiability does not mean u is a right value, since, in this example, w can be as big as can be so that whether the given u is right or not can never be known until we can determine the value of w).

Thus it is necessary for an interaction predicate whose sender is say $\mathtt{p}$ to use only variables introduded before in those interactions directly involving $\mathtt{p}$.

Locality Principle (LP) Consider the following example:

$$\begin{array}{l}\mathtt{p} \to \mathtt{q} : (v : \mathsf{int})\{v \geqslant 3\}.\\ \mathtt{q} \to \mathtt{r} : (w : \mathsf{int})\{v = w + w\}.\mathsf{end}\end{array}$$

Suppose in the first interaction, $\mathtt{p}$ sent 5 which satisfies the predicate $v \geqslant 3$. Then in the next step, we find $\mathtt{q}$ cannot choose any w satisfying the predicate $v = w + w$ with v now standing for 5 since the assertion implies v should be even. The second interaction predicate introduces a new constraint on v *after* its value is determined. Thus this global assertion violates the temporal flow. The locality principle says that each interaction predicate should give a constraint locally focusing on the variables introduced at the time, specifying constraints towards future.

B.2 Semantic Characterisation of Consistency Principles

Introduction In §3.2 in the main sections, we informally introduced three consistency principles for global assertions, namely *history-sensitivity*, *locality* and *temporal satisfiability*. The calculation of well-assertedness through the algorithm *GSat* given in Definition 3.7 together with the well-formedness condition in Figure 5 can guarantee, intuitively speaking:

- *History-sensitivity* since well-formedness allows only the directly "known" variables occur in each predicate.
- *Locality* since well-assertedness demands each interaction predicate to be entailed by the preceding interactions except for the newly introduced variables, if any.
- *Temporal satisfiability* since well-assertedness demands that at each interaction point, its predicate (disjunction of the predicates for branching) is satisfiable: that is, the active party can always find one way forward.[11]

This appendix formalises this intuition, showing that the syntactic well-assertedness condition combined with well-formedness coincide with a semantic characterisation of these principles. For this purpose we first reformulate well-assertedness in terms of causal precedence rather than syntactic precedence.

Remark B.1. The technical development in this appendix is independent from the technical results in the main sections.

Causal Precedence for GSat The calculation of well-assertedness (the algorithm *GSat*) presented in § 3 is defined based on the syntactic precedence between interactions. Considering syntactic precedence is an over-approximation as it relates interactions that may not be causally related (later we show that it nevertheless gives the semantically equivalent condition, albeit needing to deal with less compact formulae). As an example, consider the following global assertion:

$$\mathcal{G} \stackrel{\text{def}}{=} \begin{array}{l} \mathtt{p} \to \mathtt{p}' \colon s\,(v : Int)\{v \geqslant 10\}. \\ \mathtt{q} \to \mathtt{q}' \colon t\,(z : Int)\{z \geqslant 20\}.\mathsf{end} \end{array}$$

In $\mathcal{G}$ above, the second interaction is not causally dependent on the first one (notice that the predicates are defined on disjoint sets of variables).

[11] For the family of interaction predicates for branching, since we do not have interaction variables in this case, the temporal satisfiability means one of the branch options should become `true` when interactions reach that point.

Hence it is meaningless (and demands more space) to take into consideration the first predicate ($v \geqslant 10$) when we validate the well-assertedness of the second ($z \geqslant 20$). Rather it suffices to check $\mathsf{true} \supset \exists z.z \geqslant 20$, which indeed trivially holds. In this way the calculation of *GSat* can be simplified by considering only causally preceding actions. Later we use this more accurate (but equivalent) calculation when we consider it in correspondence to locality and temporal satisfiability.

To formalise this observation, we define the causal order among interactions (IO-chains in the sense of [27, §3]). We first define the notion of unfolding of recursive assertions.

Convention B.2 (bound name/variable convention). Hereafter we always assume the standard bound name convention, i.e. all names/variables introduced in binders are pairwise distinct and disjoint from free names/variables.

In the rest of this section we use the shortcut $\tilde{v} : \tilde{S} \,@\, \tilde{\mathtt{L}}$ to denote $\tilde{v}_1 : \tilde{S}_1 \,@\, \tilde{\mathtt{L}}_1, .., \tilde{v}_n : \tilde{S}_n \,@\, \tilde{\mathtt{L}}_n$.

Definition B.3 (unfolding). Given $\mathcal{G} \stackrel{\text{def}}{=} \mu \mathbf{t}\langle \tilde{e} \rangle (\tilde{v} : \tilde{S} \,@\, \tilde{\mathtt{L}})\{A\}.\mathcal{G}'$, write $\mathcal{G}'[(\tilde{v})\mathcal{G}/\mathbf{t}]$ for the result of replacing each occurrence of the form $\mathbf{t}\langle \tilde{e}_i \rangle$ in $\mathcal{G}'$ with $\mathcal{G}[\tilde{e}/\tilde{v}]$, with an appropriate renaming of bound interaction variables. Then we define:

$$\mathsf{unfold}(\mathcal{G}) \stackrel{\text{def}}{=} \mathcal{G}'[(\tilde{v})\mathcal{G}/\mathbf{t}]$$

which we call the *one-time unfolding of* $\mathcal{G}$ or simply the *unfolding of* $\mathcal{G}$.

Example B.4 (unfolding). Given

$$\begin{array}{l} \mathcal{G} \stackrel{\text{def}}{=} \mu \mathbf{t}\langle 2 \rangle (v : S \,@\, \mathtt{L})\{v \gtrsim 1\}. \\ \qquad p \to q : \{l_1 : \mathbf{t}\langle v+1 \rangle, l_2 : \mathbf{t}\langle v+3 \rangle, l_3 : \mathsf{end}\} \end{array}$$

assuming $L = \{p, q\}$ and considering all interaction predicates for $l_{1,2,3}$ are true. Then we have

$$\mathsf{unfold}(\mathcal{G}) = p \to q : \{l_1 : \mathcal{G}'_1, l_2 : \mathcal{G}'_2, l_3 : \mathsf{end}\}$$

where we set:

$$\begin{array}{rl} \mathcal{G}'_1 = & \mu \mathbf{t}\langle 3 \rangle (v : S \,@\, \mathtt{L})\{v \gtrsim 1\}. \\ & \qquad p \to q : \{l_1 : \mathbf{t}\langle v+1 \rangle, l_2 : \mathbf{t}\langle v+3 \rangle, l_3 : \mathsf{end}\} \\ \mathcal{G}'_2 = & \mu \mathbf{t}\langle 5 \rangle (v : S \,@\, \mathtt{L})\{v \gtrsim 1\}. \\ & \qquad p \to q : \{l_1 : \mathbf{t}\langle v+1 \rangle, l_2 : \mathbf{t}\langle v+3 \rangle, l_3 : \mathsf{end}\} \end{array}$$

Note the unfolding involves instantiation (substitution) of variables. Also note that the condition $v \gneqq 1$ is never used in an instantiation: this is because the role of recursive invariants is to facilitate consistent specifications and their validations for consistency. For example, even if v is instantiated with 0, if one of l_i were predicated with $v \geqslant 0$, this specification is indeed consistent (under all principles we shall stipulate soon). However by having $v \gneqq 1$ and by demanding this condition for each instantiation, we automatically know this branch, in any later instantiation of this recursion, is satisfiable.

By unfolding recursions (if any) in $\mathcal{G}$ inductively, we can regard $\mathcal{G}$ as a finite or infinite tree (it is finite whenever it does not contain a recursion: if it is infinite then it may not be regular), called the *tree unfolding of* $\mathcal{G}$, for which we still stipulate Convention B.2. Note the tree unfolding of $\mathcal{G}$ can be finite if $\mathcal{G}$ does not contain any recursion.

Convention B.5.

- We often consider a global assertion say $\mathcal{G}$ as its tree unfolding seen as an inverted tree, unless otherwise specified. We let $\mathtt{I}, \mathtt{I}', \ldots$ range over interactions in such $\mathcal{G}$, i.e. nodes which denote either value sending or branching.
- Given $\mathcal{G}$, we write $(\tilde{v})A$ to denote an interaction predicate introducing interaction variables $\tilde{v}$ (written simply A if it introduces none). Further we often let the notation $(\tilde{v})A$ or A stand for its occurrence in $\mathcal{G}$, as far as no confusion arises.
- Given an interaction $\mathtt{I}$ in $\mathcal{G}$, we say the *sender of* $\mathtt{I}$ including the participant which chooses a branch, and *receiver of* $\mathtt{I}$ including the participant one of whose branches is chosen.

The following causal precedence is the same as the existence of an IO-chain in [27].

Definition B.6 (precedence and causal precedence). Below we fix $\mathcal{G}$ seen as the tree unfolding and only consider interactions in $\mathcal{G}$.

1. An interaction $\mathtt{I}$ *positionally precedes* another $\mathtt{I}'$ if $\mathtt{I}$ occurs above $\mathtt{I}'$, seeing $\mathcal{G}$ as an inverted tree. Similarly we say a recursion *positionally precedes* $\mathtt{I}$.
2. An interaction $\mathtt{I}$ *directly causally precedes* another $\mathtt{I}'$ if $\mathtt{I}$ positionally precedes $\mathtt{I}'$ in the above sense and, moreover, the sender of $\mathtt{I}'$ occurs as either a sender or a receiver of $\mathtt{I}$. In this case we also say there is a *direct causal chain* from $\mathtt{I}$ to $\mathtt{I}'$.

3. An interaction $\mathtt{I}$ *causally precedes* another $\mathtt{I}'$ if there is a sequence of direct causal chains from $\mathtt{I}$ to $\mathtt{I}'$.

Using these conventions and definitions we refine Definition 3.6 (which defines the set of variables a participant is involved in $\mathcal{G}$ as a whole) so that we can talk about whether or not a participant at some interaction point knows about some variable. Below note $\mathcal{G}$, by Convention B.5 (1), is regarded as the tree unfolding of its syntactic representation, which does not include a recursion.

Definition B.7 ("knows", support). Below we fix an interaction $\mathtt{I}$ in $\mathcal{G}$, whose sender is $\mathtt{p}$.

1. We say $\mathtt{p}$ *at* $\mathtt{I}$ *directly knows* $(\tilde{v})A$ (seeing $(\tilde{v})A$ as an occurrence) if $(\tilde{v})A$ is associated with another interaction $\mathtt{I}'$ which directly causally precedes $\mathtt{I}$.
2. We say $\mathtt{p}$ *at* $\mathtt{I}$ *indirectly knows* $(\tilde{v})A$ if the same condition as above (1) except we replace "directly causally precedes" with "causally precedes".
3. The support of A, $\mathsf{Sup}(A)$, is the minimum set of variables whose existential closure on A makes it either equivalent to true or false.

Remark B.8 (knows, support).

- (directly/indirectly knows, 1) Intuitively, if a participant at some interaction indirectly knows a predicate $(\tilde{v})A$, then its action may as well be (directly or indirectly) affected by the value instantiating that variable, since even if the participant has not participated in that interaction, the value may have influenced a sequence of subsequent actions (and instantiation of other variables) leading to the interaction concerned.
- (directly/indirectly knows, 2) The "directly know" relation corresponds to the existence of a (possibly non-minimal) OO-edge or a (possibly non-minimal) IO-edge in [27, § 3.3]; while "indirectly know" corresponds to the notion of IO-chain [27, § 3.3].
- (support) As an example of support, if A is given as "$x = x \wedge y = 1$" then $\mathsf{Sup}(A) = \{y\}$, where x is neglected because it does not affect the validity.

We now define the consistency criteria semantically, using the causal dependency and the support set. Below observe the set of predicates which $\mathtt{p}$ directly or indirectly knows at some $\mathtt{I}$ is always finite, even if the tree unfolding gives an infinite tree.

Definition B.9. Fix $\mathcal{G}$ and an interaction $\mathtt{I}$ occurring in $\mathcal{G}$ such that the sender of $\mathtt{I}$ is $\mathtt{p}$ and the associated predicate of $\mathtt{I}$ is $(\tilde{v})A$ (when $\mathtt{I}$ is a value sending) or is $\{A_h\}_h$ (when $\mathtt{I}$ is a branching). In the latter case we set $A \stackrel{\text{def}}{=} \bigvee_h A_h$ and $\tilde{u}$ to be the empty string. Assume the set of predicates which $\mathtt{p}$ at $\mathtt{I}$ directly knows is $\{(\tilde{u}_i)B_i\}_i$, similarly the set of predicates which $\mathtt{p}$ at $\mathtt{I}$ indirectly knows is $\{(\tilde{z}_j)C_j\}_j$. We then say

1. (history sensitivity, HSP) $\mathtt{I}$ *respects history-sensitivity principle (HSP)* if $\mathsf{Sup}(A) \subseteq (\bigcup\{\tilde{u}_i\}_i) \cup \{\tilde{v}\}$.
2. (locality, LP) $\mathtt{I}$ *respects locality principle (LP)* if $(\bigwedge_j C_j) \supset \exists\tilde{v}.A$.
3. (temporal satisfiability, TSP) $(\tilde{v})A$ *respects temporal satisfiability principle (TSP)* if whenever a valuation σ satisfies $(\bigwedge_j C_j)$, $A\sigma$ is satisfiable.

If (1) (resp. (2), resp. (3)) holds for each interaction occurring in $\mathcal{G}$ then $\mathcal{G}$ respects HSP (resp. LP, resp. TSP).

Above HSP says that, at each interaction, variables that matter in the associated predicate(s) should be introduced in those interactions directly known to the sender. LP says that, at each interaction, the associated predicate(s) should only introduces a constraint on its newly introduced variables. TSP says that, at each interaction, the associated predicate(s) should become satisfiable once all preceding predicates get satisfied, i.e. a sender can always find a way forward, either choosing a value or a selection a label under a valid condition.

While LP and TSP are differently oriented, they are logically equivalent.

Proposition B.10. *$\mathcal{G}$ respects LP if and only if it respects TSP.*

Proof. By the standard definition of satisfiability. □

Intuitively, Proposition B.10 is because if one does not introduce a new constraint to the existing variables then one also preserves the potential that the newly introduced variable can be satisfiable (in particular, for branching, LP says that one of the options should become true, which is also what TSP demands).

By Proposition B.10 we hereafter only consider LP without loss of generality. We further note:

Lemma B.11. *Fix $\mathcal{G}$ respecting HSP. Suppose further, in $\mathcal{G}$, $(\tilde{u})B$ is above $(\tilde{v})A$ and the participant for the interaction for $(\tilde{v})A$ does not indirectly know the interaction for $(\tilde{u})B$. Then no variable from $\tilde{u}$ occurs in $\mathsf{Sup}(A)$.*

Proof. Direct from the definition of "indirectly knows". □

The consistency principles in Definition B.9 are given for the tree unfolding. Since global assertions in fact use invariants to enable finite syntax to capture properties of recurring interactions, we relativise these principles under the use of invariant.

Below and henceforth we often omit types annotating variables in a recursion for brevity.

Convention B.12. For simplicity but without loss of generality we henceforth assume that, in each $\mu \mathbf{t}\langle \tilde{e} \rangle (\tilde{v} : \tilde{S} @ \mathtt{\tilde{L}}) \{A\}.\mathcal{G}'$, the free interaction variables in $\mathcal{G}'$ are a subset of $\{\tilde{v}\}$.

Note that this convention does not lose generality since we can always expand the binding ($\tilde{v}$ above) to cover all variables, while instantiate them with the original free ones. Further observe that, under Convention B.12, the rule for *GSat* of recursion (Definition 3.7)

$$\begin{cases} \text{if } A \supset A'[\tilde{e}/\tilde{v}] \text{ then } \mathit{GSat}(\mathcal{G}, A) = \mathit{GSat}(\mathcal{G}', A') \\ \qquad\qquad \text{otherwise } \mathit{GSat}(\mathcal{G}, A) = \mathsf{false} \end{cases}$$

Thus we only have to check (1) the invariant is true for this (initial) instantiation and that the body is validated under this invariant. Notice that: (a) free variables can be used inside a recursion in effect by passing them as recursion parameters and (b) in that case we can also pass the constraint on these variables as part of the invariant by including the constraint on them.

Definition B.13. Fix $\mathcal{G}$ seen as a syntax (without unfolding) and fix any node in $\mathcal{G}$. If the node occurs inside an immediately enclosing recursion, say $\mu \mathbf{t}\langle \tilde{e} \rangle (\tilde{v} : \tilde{S} @ \mathtt{\tilde{L}}) \{A\}.\mathcal{G}'$, then its *enclosing context* is $\mathcal{G}'$, its *enclosing predicate* is A, and its *enclosing variables* are $\tilde{v}$ located at $\tilde{\mathtt{L}}$. If the node is not enclosed by any recursion, then its enclosing context is the whole $\mathcal{G}$, its enclosing predicate is true, and its enclosing variables are empty.

Definition B.14 (well-enclosed assertion). We say $\mathcal{G}$ is *well-enclosed* if the following conditions holds for each recursive assertion say $\mu \mathbf{t}\langle \tilde{e} \rangle (\tilde{v} : \tilde{S} @ \mathtt{\tilde{L}}) \{A\}.\mathcal{G}'$ occurring in $\mathcal{G}$ seen as a syntax:

1. Each variable in e_i if any is introduced by an interaction involving $\mathtt{L_i}$ or an enclosing variable located at $\mathtt{L_i}$.

2. Let C be the conjunction of (a) the predicates in its enclosing context that have introduced variables in $\tilde{e}$ and (b) the enclosing predicate. Then C should entail $A[\tilde{e}/\tilde{v}]$.
3. If say $\mathbf{t}\langle\tilde{e}'\rangle$ occurs in $\mathcal{G}'$, then (a) each variable in e'_i, if any, should be located at L_i and (b) $A[\tilde{e}'/\tilde{v}]$ should be entailed by the conjunction of A and the predicates preceding this $\mathbf{t}\langle\tilde{e}'\rangle$ which introduce variables in $\tilde{e}'$.

These conditions together stipulate the consistent usage of recursive invariants. In the second condition (b), note $\exists\tilde{e}(A)$ entails $\exists\tilde{v}(A)$, which, like HSP. Thus (b) prevents this newly introduced invariant from imposing a further constraint on existing variables.

We now define HSP and LP which incorporate the notion of recursive invariants.

Definition B.15 (HSP and LP under Invariant). Suppose $\mathcal{G}$ is well-enclosed. Then we say, regarding $\mathcal{G}$ as a finite syntax:

1. $\mathcal{G}$ *satisfies HSP under invariant* if each interaction predicate $(\tilde{v})A$ in an interaction `I` whose sender is `p` satisfies:

$$\mathsf{Sup}(A) \subseteq (\{\tilde{u}\} \cup \{\tilde{v}\})$$

   where $\{\tilde{u}\}$ is the union of variables introduced for interactions involving `p` preceding `I` in the enclosing context of `I`, and the enclosing variables located at `p`.
2. $\mathcal{G}$ *satisfies LP under invariant* if, for each interaction `I`, its associated predicate $(\tilde{v})A$ (given as in Definition B.9), occurring in $\mathcal{G}$ seen as a finite syntax, satisfies:

$$C_1 \wedge C_2 \supset \exists\tilde{v}(A)$$

   where C_1 is the conjunction of all predicates which indirectly causally precedes `I` in the enclosing context and C_2 is the enclosing predicate.

Proposition B.16. *Let $\mathcal{G}$ be well-enclosed below.*

1. *If $\mathcal{G}$ satisfies HSP under invariant then $\mathcal{G}$ satisfies HSP.*
2. *If $\mathcal{G}$ satisfies HSP under invariant and satisfies LP under invariant then $\mathcal{G}$ satisfies LP (hence TSP).*

Proof. For (1), consider $\mathcal{G}$ satisfies HSP under invariant and fix `I` its associated predicate A. Consider $A[\tilde{e}/\tilde{v}]$ in the one-time unfolding of $\mathcal{G}$,

whose substitution is by instantiation. Since $\mathcal{G}$ is well-enclosed, $\mathsf{Sup}(A)$ is always within $\tilde{v}$. By the given condition, all variables in A belong to those originating in p. Since the corresponding condition for locations holds for variables in each e_i, the substitution does not change the locations. The same reasoning holds after an arbitrary unfolding.

For (2) fix $\mathtt{I}$ and $(\tilde{v})A$ and again consider $A' = A\sigma$ with

$$\sigma \stackrel{\text{def}}{=} [\tilde{e}/\tilde{u}] \tag{B.1}$$

in the one-time unfolding of $\mathcal{G}$, with $\tilde{u}$ being the enclosing variables. Note $\tilde{v}$ are still freshly introduced in A'. Now we show $\exists\tilde{v}(A')$ is entailed by the preceding predicates causally indirectly preceding $I\sigma$. Let B be the enclosing predicate (before the unfolding) and C_0 be the conjunction of the indirectly causally preceding predicates in the enclosing context (again before the unfolding). We know, by LP under invariant:

$$B \wedge C_0 \supset \exists\tilde{v}(A) \tag{B.2}$$

By substitution we obtain

$$(B\sigma \wedge C_0\sigma) \supset \exists\tilde{v}(A\sigma) \tag{B.3}$$

Now let C be the conjunction of the indirectly causally preceding predicates (after the unfolding). Then we have

$$C \supset C_0\sigma \tag{B.4}$$

because the same indirectly causally preceding predicates are chosen. Note also $\tilde{e}$ does not contain any variable from $\tilde{u}$. By (1) above, this means the free variables in $A\sigma$ except $\tilde{v}$ come from the directly causally preceding predicates (after the unfolding). We let $\tilde{u}_0$ be these variables. All these variables occur in $B\sigma$: let the remaining variables in $B\sigma$ to be $\tilde{w}_0$. Since C includes all predicates which introduced $\tilde{v}_0$, by being well-enclosed we obtain:

$$C \supset \exists\tilde{w}_0(B\sigma) \tag{B.5}$$

By (B.3) and noting $\tilde{w}_0$ do not occur in $A\sigma$ we know

$$(\exists\tilde{w}_0(B\sigma \wedge C_0\sigma)) \supset \exists\tilde{v}(A\sigma) \tag{B.6}$$

Combined with (B.4) we obtain:

$$C \supset \exists\tilde{v}(A\sigma) \tag{B.7}$$

as required. The case when we apply the n-times unfolding is by exactly the same argument using the third condition in Definition B.14. □

Theorem B.17 (semantic characterisation of well-assertedness).

1. *If $\mathcal{G}$ is well-enclosed and well-formed then $\mathcal{G}$ respects HSP under invariant. Conversely, if $\mathcal{G}$ is well-enclosed and respects HSP under invariant, then there exists $\mathcal{G}'$ such that $\mathcal{G} \equiv \mathcal{G}'$ and $\mathcal{G}'$ is well-formed.*
2. *Let $\mathcal{G}$ be well-formed. If $\mathcal{G}$ is further well-asserted then $\mathcal{G}$ is well-enclosed and moreover respects LP and TSP. Conversely, if $\mathcal{G}$ is well-enclosed and respects LP and TSP, then there exists $\mathcal{G}'$ such that $\mathcal{G} \equiv \mathcal{G}'$ and $\mathcal{G}'$ such that $\mathcal{G}' \equiv \mathcal{G}$ and $\mathcal{G}'$ is well-asserted.*

Proof. (outline) Below we fix `I` occurring in $\mathcal{G}$ whose sender is `p` and whose predicate is $(\tilde{v})A$ with $\tilde{v}$ possibly empty, given following Definition B.9. We argue focusing on `I`.

For (1), if $\mathcal{G}$ is well-formed, then the syntactic support (free variables) of A comes from those introduced in interactions `p` is involved, or from the enclosing variables located at `p`. Since they are both in the semantic support of A as given in Defintion B.15 (1) we are done. Conversely, we can always existentially close spurious variables in A to make its semantic support and syntactic support coincide.

For (2), the calculation of *GSat* uses all syntactically preceding predicates within the enclosing context, say C, together with the enclosing predicate, say B (see the rule under Convention B.12). Let their conjunction be C. Further, following Defintion B.15 (2), let C_0 is the conjunction of all predicates which indirectly causally precedes `I` in the enclosing context. Now let us write C_{00} for the conjunction of all predicates which directly causally precedes `I` in the enclosing context. Let $\tilde{w}$ be the free variables in C which are neither in A nor in B (i.e. those which are introduced by the preceding predicates unrelated to A). Then by Lemma B.11 and by noting that the constraints on the variables in A from the preceding predicates solely come from those which have introduced them (i.e. the predicates which constitute C), we know $\exists\tilde{w}(C) \equiv \exists\tilde{w}(C_0) \equiv C_{00}$. Thus

$$\begin{aligned}(B \wedge C \supset \exists\tilde{v}(A)) &\equiv (\exists\tilde{w}(B \wedge C) \supset \exists\tilde{v}(A)) \\ &\equiv (B \wedge \exists\tilde{w}(C) \supset \exists\tilde{v}(A)) \\ &\equiv (B \wedge \exists\tilde{w}(C_0) \supset \exists\tilde{v}(A)) \\ &\equiv (\exists\tilde{w}(B \wedge C_0) \supset \exists\tilde{v}(A)) \\ &\equiv (B \wedge C_0 \supset \exists\tilde{v}(A))\end{aligned}$$

as required. □

Remark B.18. The proof of Theorem B.17 shows that *GSat* can be optimised so that the entailment at each step can be calculated solely in terms of those predicates which have introduced free variables of the interaction predicate under validation.

C Appendix for § 4

C.1 Examples of Projection

Consider the following well-formed and well-asserted global assertion for a three-party protocol:

$$\begin{aligned}\mathcal{G}_{tP} = \ & \texttt{Alice} \to \texttt{Bob}\colon b\,(v : Int)\{v < 7\}.\\ & \texttt{Bob} \to \texttt{Carol}\colon c\,(u : Int)\{u < v \wedge u < 10\}.\\ & \texttt{Carol} \to \texttt{Alice}\colon a\,(z : Int)\{z < u \wedge z > 0\}.\mathsf{end}\end{aligned}$$

The projection on `Carol` is

$$\begin{aligned}\mathcal{G}_{tP} \upharpoonright \texttt{Carol} = \ & c?(u : Int)\{\exists v(v < 7 \wedge u < v \wedge u < 10)\};\\ & a!(u : Int)\{z < u \wedge z > 0\};\mathsf{end}\end{aligned}$$

$\mathcal{G}_{tP} \upharpoonright \texttt{Carol}$ is calculated straightforwardly from the rules in Definition 4.2. The projection on `Carol` of the first interaction is defined as the projection of the continuation of $\mathcal{G}_{tP}$, called recursively by keeping the predicate $v < 7$ in the collection. When projecting continuation (i.e., the second interaction of $\mathcal{G}_{tP}$) which is an input for `Carol`, we merge the predicate $v < 7$ to the predicate associated to the incoming interaction $u < v \wedge u < 10$, adding the existential quantifier for v since v is not known to `Carol`. By projecting the predicates of third parties into `Carol`, we provide her with a stronger set of assumptions. For example, by considering predicate $u < v \wedge u < 10$ and predicate $v < 7$, `Carol` can infer $u < 6$. In this way, `Carol` can to detect, at run-time, that a messages is surely a violation of the global assertion (i.e., `Alice` may send a value that violate $v < 7$ and `Bob` may fail to check the violation). Despite not knowing which value has been given to v, `Carol` will know that a violation surely occurred if, for example, she receives $y = 8$. We do not project the assumption $v < 7$ in the interaction where `Carol` is the sender. In this case, it is enough for `Carol` to satisfy the current predicate $z < u \wedge z > 0$ for the protocol to run correctly. Notice that, since `Carol` has an obligation on the predicate for the outgoing message she must know all the variables involved (by well-formedness of $\mathcal{G}_{tP}$).

C.2 Proof of Lemma 4.5 (Projection Preserves Well-Assertedness)

Corollary 4.6 follows immediately from Lemma C.1.

Lemma C.1. *Let $\mathcal{G}$ be a well-formed global assertion then, for all predicates $A_\mathcal{G}, A_\mathcal{T}$ such that $A_\mathcal{T} \supset A_\mathcal{G}$ and all $\mathtt{p} \in \mathsf{pig}(\mathcal{G})$,*

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

Proof. The proof is by induction on the projection rules, proceeding by case analysis on $\mathcal{G}$.

If $\mathcal{G} = \mathtt{p_1} \to \mathtt{p_2}\colon k\ (\tilde{v} : \tilde{S})\{A\}.\mathcal{G}'$ then we have two cases.

Case $\mathtt{p} = \mathtt{p_1}$./
By Definition 4.2, $\mathcal{G} \upharpoonright \mathtt{p} = k!(\tilde{v} : \tilde{S})\{A\}; \mathcal{T}'$. From $A_\mathcal{T} \supset A_\mathcal{G}$ (by hypothesis) and $A_\mathcal{G} \supset \exists \tilde{v}(A)$ (by well-assertedness of $\mathcal{G}$) it follows

$$A_\mathcal{T} \supset \exists \tilde{v}(A). \tag{C.1}$$

By (C.1) and 4.4, $LSat(\mathcal{G} \upharpoonright \mathtt{p}, A_\mathcal{T}) = LSat(\mathcal{T}, A_\mathcal{T} \wedge A)$. The lemma holds for this case by induction on $GSat(\mathcal{G}', A_\mathcal{G} \wedge A)$ and $LSat(\mathcal{T}, A_\mathcal{T} \wedge A)$ since $A_\mathcal{T} \wedge A \supset A_\mathcal{G} \wedge A$.

Case $\mathtt{p} = \mathtt{p_2}$./
Then (by 4.2) $\mathcal{G} \upharpoonright \mathtt{p} = k?(\tilde{v} : \tilde{S})\{\exists V_{ext}(A_\mathcal{T} \wedge A)\}; \mathcal{T}'$. From $A_\mathcal{T} \supset A_\mathcal{G}$ (by hypothesis) and $A_\mathcal{G} \supset \exists \tilde{v}(A)$ (by well-formedness of $\mathcal{G}$) it follows $A_\mathcal{T} \supset \exists \tilde{v}(A)$ which is equivalent to

$$A_\mathcal{T} \supset A_\mathcal{T} \wedge \exists \tilde{v}(A). \tag{C.2}$$

Since the consequence of C.2 implies $\exists V_{ext}(A_\mathcal{T} \wedge \exists \tilde{v}(A))$ and $v_1 \dots v_n \notin var(A_\mathcal{T})$,

$$A_\mathcal{T} \supset \exists \tilde{v}(\exists V_{ext}(A_\mathcal{T} \wedge A)). \tag{C.3}$$

By (C.3) and 4.4, $LSat(\mathcal{G} \upharpoonright \mathtt{p}, A_\mathcal{T}) = LSat(\mathcal{T}, A_\mathcal{T} \wedge A)$. The lemma holds for this case by induction on $GSat(\mathcal{G}', A_\mathcal{G} \wedge A)$ and $LSat(\mathcal{T}', A_\mathcal{T} \wedge A)$ since $A_\mathcal{T} \wedge A \supset A_\mathcal{G} \wedge A$.

Branching. If $\mathcal{G} = \mathtt{p_1} \to \mathtt{p_2}\colon k\ \{\{A_j\}l_j\colon \mathcal{G}_j\}_{j \in J}$ then again we have two cases.

Case $\mathtt{p} = \mathtt{p_1}$./ By Definition 4.2, $\mathcal{G} \upharpoonright \mathtt{p} = k \oplus \{\{A_j\}l_j : \mathcal{T}_j\}_{j \in J}$. From $A_\mathcal{T} \supset A_\mathcal{G}$ (by hypothesis) and $A_\mathcal{G} \supset A_1 \vee \ldots \vee A_n$ (by well-formedness of $\mathcal{G}$) it follows

$$A_\mathcal{T} \supset A_1 \vee \ldots \vee A_n. \tag{C.4}$$

By (C.4) and 4.4, $LSat(\mathcal{G} \upharpoonright \mathtt{p}, A_{\mathcal{T}}) = \wedge_{j \in J} LSat(\mathcal{T}_j, A_{\mathcal{T}} \wedge A_j)$. The property holds by induction on $GSat(\mathcal{G}_j, A_{\mathcal{G}} \wedge A_j)$ and $LSat(\mathcal{T}_j, A_{\mathcal{T}} \wedge A_j)$ for all $j \in J$, since $A_{\mathcal{T}} \wedge A_j \supset A_{\mathcal{G}} \wedge A_j$.

Case $\mathtt{p} = \mathtt{p}_2$***.*** By Definition 4.2, $\mathcal{G} \upharpoonright \mathtt{p} = k\&\{\{\exists V_{ext}(A_{\mathcal{T}} \wedge A_j)\} l_j : \mathcal{T}_j\}_{j \in J}$. From $A_{\mathcal{T}} \supset A_{\mathcal{G}}$ (by hypothesis) and $A_{\mathcal{G}} \supset A_1 \vee \ldots \vee A_n$ (by well-formedness of $\mathcal{G}$) it follows $A_{\mathcal{T}} \supset A_1 \vee \ldots \vee A_n$ which is equivalent to

$$A_{\mathcal{T}} \supset (A_{\mathcal{T}} \wedge A_1) \vee \ldots \vee (A_{\mathcal{T}} \wedge A_n). \tag{C.5}$$

By weakening the consequence of C.5 (since for all $j \in J$, $(A_{\mathcal{T}} \wedge A_j) \supset \exists V_{ext}(A_{\mathcal{T}} \wedge A_j)$) we obtain

$$A_{\mathcal{T}} \supset \exists V_{ext}(A_1) \cup \ldots \cup \exists V_{ext}(A_n). \tag{C.6}$$

By (C.6) and 4.4, $LSat(\mathcal{G} \upharpoonright \mathtt{p}, A_{\mathcal{T}}) = \wedge_{j \in J} LSat(\mathcal{T}_j, A_{\mathcal{T}} \wedge A_j)$. The property holds by induction on $GSat(\mathcal{G}_j, A_{\mathcal{G}} \wedge A_j)$ and $LSat(\mathcal{T}_j, A_{\mathcal{T}} \wedge A_j)$ for all $j \in J$, since $A_{\mathcal{T}} \wedge A_j \supset A_{\mathcal{G}} \wedge A_j$.

Recursion. If $\mathcal{G} = \mu \mathbf{t} \langle \tilde{e} \rangle (\tilde{v} : \tilde{S})\{A\}.\mathcal{G}'$ then

$$\mathcal{G} \upharpoonright \mathtt{p} = \mu \mathbf{t} \langle \tilde{e}' \rangle (\tilde{v}' : \tilde{S}')\{\exists V_{ext}(A)\}.\mathcal{T}'.$$

From $A_{\mathcal{T}} \supset A_{\mathcal{G}}$ (by hypothesis) and $A_{\mathcal{G}} \supset A[\tilde{e}/\tilde{v}]$ (by well-formedness of $\mathcal{G}$) we reason, writing $\tilde{w}$ for $\tilde{v}$ minus $\tilde{v}'$:

$$\begin{aligned} A_{\mathcal{T}} &\supset A_{\mathcal{G}} \\ &\supset A[\tilde{e}/\tilde{v}] \\ &\supset (\exists \tilde{w}) A[\tilde{e}'/\tilde{v}'] \\ &\supset (\exists V_{ext}) A[\tilde{e}'/\tilde{v}'] \end{aligned}$$

as required.

Type Variable. If $\mathcal{G} = t_{A(\tilde{v})}\langle \tilde{e} \rangle$, then the projection is $t_{B(\tilde{v}')}\langle \tilde{e'} \rangle$ where $B = (\exists V_{ext})A$, with V_{ext} from the enclosing recursion. Suppose $A_{\mathcal{T}} \supset A_{\mathcal{G}}$. By the well-formedness of $\mathcal{G}$, we have $A_{\mathcal{G}} \supset A[\tilde{e}/\tilde{v}]$. By the rasoning as in the case of recursion we know $A[\tilde{e}/\tilde{v}]$ entails $(\exists V_{ext})A[\tilde{e}'/\tilde{v}']$, as required.

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

End. Immediate. □

P	::=	$\overline{a}[2..\mathtt{n}]\,(\tilde{s}).P$	session request
	$\mid$	$a[\mathtt{p}]\,(\tilde{s}).P$	session acceptance
	$\mid$	$s!\langle\tilde{e}\rangle(\tilde{v})\{A\};P$	value sending
	$\mid$	$s?(\tilde{v})\{A\};P$	value reception
	$\mid$	$s \lhd \{A\}l;P$	label selection
	$\mid$	$s \rhd \{\{A_i\}l_i : P_i\}_{i\in I}$	label branching
	$\mid$	$\mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q$	conditional branch
	$\mid$	$P \mid Q \quad \mid \quad \mathbf{0}$	parallel / idle
	$\mid$	$(\nu a)P$	name hiding
	$\mid$	$\mathsf{def}\ D\ \mathsf{in}\ P \mid X\langle\tilde{e}\tilde{s}\rangle$	recursion def/call
	$\mid$	$\mathsf{errH} \mid \mathsf{errT}$	error
	$\mid$	$s : \tilde{h}$	msg queue
	$\mid$	$(\nu\tilde{s})P$	channel hiding

Fig. 13. Syntax of Asserted Processes

D Appendix for § 5

D.1 Programs and Program Phrases

This section gives a precise grammar of *programs*, *program phrases* and *runtime processes*. This distinction follows [27, Def 4.4] and distinguish, through syntactic shapes, those processes which programmers may write, hence without queues nor session hiding (*program phrases*), complete programs which programmers may write, hence without free variables nor free session channels (*programs*) and processes which include all processes which may include the runtime elements such as queues and session hiding (*processes* or, emphasising the possible presence of these runtime elements *runtime processes*). Note queues and session hiding are generated when session initiation takes place (Rule [R-Link] in Figure 8, page 8). When programs are written, they cannot have started session initiation (linking), which is a runtime phenomenon. When they are put to execution, session initiation will take place and runtime entities such as queues and session hiding are generated.

For convenience, Figure 13 presents a re-ordered version of the productions for asserted processes (for the grammar for expressions, values, messages, process variable declarations, see Figure 7). We remind that the

binders of names are $\boldsymbol{\nu}$ and session request/acceptance prefixes, input and output prefixes bind variables in their continuations and recursive definitions bind process variables. We can now formally introduce the three syntactic categories.

Definition D.1 (programs, program phrases and runtime processes).

1. *A* program phrase *is a process derived using any rules in Figure 13 except the last three productions.*
2. *A* program *is a closed (i.e., with no free (process) variables or session channels) program phrase.*
3. *A* runtime process *or often simply a* process *is simply a process generated from any rules in Figure 13.*

In other words, syntactically correct programs are (the parallel composition of possibly recursive) processes

- with no queues or errors,
- where session names occur on in the scope of session request/acceptance prefixes, and
- where only shared names may occur free.

The remaining productions of processes are used only at *run-time*. In fact, the reduction semantics (Figure 8) of asserted processes guarantees that in programs (*i*) a synchronisation on the session request/accept actions is executed, so that (*ii*) session names are restricted and (*iii*) the corresponding queues for communication among participants are created. Note also, error processes are generated at run-time (Figure 9) when some assertions are violated.

An important technical remark for the proofs of our results is that not only programs but also their *transition derivatives* do not allow queues at free session channels, hence input and output transitions may only happen on restricted channels. This justifies the definition of labelled transition for end-point assertions in § 7.1 ([TR-Tau] in Figure 11).

D.2 Full Typing Rules

For reference, this Appendix presents all of the typing rules for processes, from [27], with minor changes to make the correspondence with the presented validation rules (Figure 10) exact. We use the grammar of (global

$$\underline{\Gamma}, a : S \vdash a : S \quad \underline{\Gamma} \vdash \mathsf{true}, \mathsf{false} : \mathsf{bool} \quad \frac{\underline{\Gamma} \vdash e_i \rhd \mathsf{bool}}{\underline{\Gamma} \vdash e_1 \mathsf{or}\ e_2 : \mathsf{bool}}$$

[Name], [Bool], [Or]

$$\frac{\forall j. \underline{\Gamma} \vdash e_j : S_j \quad \underline{\Gamma} \vdash P[\tilde{e}/\tilde{v}] \rhd \underline{\Delta}, \tilde{s} : T \,@\, \mathtt{p}}{\underline{\Gamma} \vdash s_k!\langle \tilde{e} \rangle(\tilde{v})\{A\}; P \rhd \underline{\Delta}, \tilde{s} : k!\langle \tilde{S} \rangle; T \,@\, \mathtt{p}} \quad [\textsc{Send}]$$

$$\frac{\underline{\Gamma}, \tilde{v} : \tilde{S} \vdash P \rhd \underline{\Delta}, \tilde{s} : T \,@\, \mathtt{p}}{\underline{\Gamma} \vdash s_k?(\tilde{v})\{A\}; P \rhd \underline{\Delta}, \tilde{s} : \ k?\langle \tilde{S} \rangle; T \,@\, \mathtt{p}} \quad [\textsc{Rcv}]$$

$$\frac{\underline{\Gamma} \vdash P \rhd \underline{\Delta}, \tilde{s} : T_j \,@\, \mathtt{p} \quad j \in I}{\underline{\Gamma} \vdash s_k \lhd \{A_j\} l_j; P \rhd \underline{\Delta}, \tilde{s} : k \oplus \{l_i : T_i\}_{i \in I} \,@\, \mathtt{p}} \quad [\textsc{Sel}]$$

$$\frac{\underline{\Gamma} \vdash P_i \rhd \underline{\Delta}, \tilde{s} : T_i \,@\, \mathtt{p} \quad \forall i \in I}{\underline{\Gamma} \vdash \ s_k \rhd \{\{A_i\} l_i : P_i\}_{i \in I} \rhd \underline{\Delta}, \tilde{s} : k \& \{l_i : T_i\}_{i \in I} \,@\, \mathtt{p}} \quad [\textsc{Branch}]$$

$$\frac{\underline{\Gamma} \vdash a : G \quad \underline{\Gamma} \vdash P \rhd \underline{\Delta}, \tilde{s} : (G \upharpoonright \mathtt{1}) \,@\, \mathtt{1} \quad |\tilde{s}| = \max(\mathsf{sid}(G))}{\underline{\Gamma} \vdash \overline{a}_{[2..\mathtt{n}]}(\tilde{s}).P \rhd \underline{\Delta}} \quad [\textsc{Mcast}]$$

$$\frac{\underline{\Gamma} \vdash a : G \quad \underline{\Gamma} \vdash P \rhd \underline{\Delta}, \tilde{s} : (G \upharpoonright \mathtt{p}) \,@\, \mathtt{p} \quad |\tilde{s}| = \max(\mathsf{sid}(G))}{\underline{\Gamma} \vdash a_{[\mathtt{p}]}(\tilde{s}).P \rhd \underline{\Delta}} \quad [\textsc{Macc}]$$

$$\frac{\underline{\Gamma} \vdash P \rhd \underline{\Delta} \quad \underline{\Gamma} \vdash Q \rhd \underline{\Delta}' \quad \underline{\Delta} \asymp \underline{\Delta}'}{\underline{\Gamma} \vdash P \mid Q \rhd \underline{\Delta} \circ \underline{\Delta}'} \quad [\textsc{Conc}]$$

$$\frac{\underline{\Gamma} \vdash e \rhd \mathsf{bool}}{\underline{\Gamma} \vdash \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \rhd \underline{\Delta}} \quad [\textsc{If}]$$

$$\frac{\underline{\Delta}\ \mathsf{end\ only}}{\underline{\Gamma} \vdash \mathbf{0} \rhd \underline{\Delta}} \qquad \frac{\underline{\Gamma}, a : G \vdash P \rhd \underline{\Delta}}{\underline{\Gamma} \vdash (\boldsymbol{\nu}\, a)P \rhd \underline{\Delta}} \quad [\textsc{Inact}],[\textsc{NRes}]$$

$$\frac{\underline{\Gamma} \vdash \tilde{e} : \tilde{S} (1 \leqslant i \leqslant n)}{\underline{\Gamma}, X : \tilde{S}\, T_1 @ \mathtt{p}_1 \ldots T_n @ \mathtt{p}_\mathtt{n} \vdash X\langle \tilde{e} \tilde{s}_1 .. \tilde{s}_n \rangle \rhd \tilde{s}_1 : T_1 \,@\, \mathtt{p}_1, .., \tilde{s}_n : T_n \,@\, \mathtt{p}_n} \quad [\textsc{Var}]$$

$$\frac{\underline{\Gamma}, X : \tilde{S}\, T_1 @ \mathtt{p}_1 \ldots T_n @ \mathtt{p}_\mathtt{n}, \tilde{v} : \tilde{S} \vdash P \rhd \tilde{s_1} : T_1 \,@\, \mathtt{p}_1 \ldots \tilde{s_n} : T_n \,@\, \mathtt{p}_n \quad \underline{\Gamma}, X : \tilde{S}\, T_1 @ \mathtt{p1} \ldots T_n @ \mathtt{p}_\mathtt{n} \vdash Q \rhd \underline{\Delta}}{\underline{\Gamma} \vdash \mathsf{def}\ X(\tilde{v}\tilde{s}_1 \ldots \tilde{s}_n) = P\ \mathsf{in}\ Q \ \rhd\ \underline{\Delta}} \quad [\textsc{Def}]$$

Fig. 14. Full Typing Rules

and local) types from [27], which we reproduce below.

$$
\begin{array}{lrll}
\text{Global } G & ::= & \mathtt{p} \to \mathtt{p}' \colon k\,\langle U \rangle.G' & \text{values} \\
& \mid & \mathtt{p} \to \mathtt{p}' \colon k\,\{l_j \colon G_j\}_{j \in J} & \text{branching} \\
& \mid & G, G' & \text{parallel} \\
& \mid & \mu \mathbf{t}.G & \text{recursive} \\
& \mid & \mathbf{t} & \text{variable} \\
& \mid & \mathsf{end} & \text{end} \\
\text{Sort} \quad S & ::= & \mathsf{bool} \mid \ldots \mid \langle G \rangle & \\
\text{Value} \quad U & ::= & \tilde{S} & \\
\text{Local} \quad T & ::= & k!\langle U \rangle; T & \text{send} \\
& \mid & k?\langle U \rangle; T & \text{receive} \\
& \mid & k \oplus \{l_i \colon T_i\}_{i \in I} & \text{selection} \\
& \mid & k \& \{l_i \colon T_i\}_{i \in I} & \text{branching} \\
& \mid & \mu \mathbf{t}.T & \text{recursive} \\
& \mid & \mathbf{t} & \text{variable} \\
& \mid & \mathsf{end} & \text{end}
\end{array}
$$

Note the difference of fonts used for global and local types from those used for global and endpoint assertions. Global types ($G, G', \ldots$) underlie global assertions ($\mathcal{G}, \mathcal{G}', \ldots$), while local types ($T, T', \ldots$) underlie endpoint assertions ($\mathcal{T}, \mathcal{T}', \ldots$). *Located types* are of the form $T \,@\, \mathtt{p}$. We use two typing environments (to differentiate from assertion environments/assignments we underline the symbols):

1. $\underline{\Gamma}$ assigns variables to value types and names to sorts, as well as process variables to the vector of value types combined with a vector of located types.
2. $\underline{\Delta}$ assigns each vector $\tilde{s}$ of session channels to located types.

The typing rules are given in Figure 14. In the figure, $\mathsf{sid}(\mathcal{G})$ (resp. $\mathsf{pid}(\mathcal{G})$) stands for the set of session channel numbers (resp. participant numbers) in $\mathcal{G}$.

The typing rules still use asserted processes, but logical formulae are given arbitrarily without any constraint. When predicate annotations from processes are erased, these rules are essentially identical with the typing rules for [27], excepting:

1. The rules for delegations in [27] are missing; and
2. In [Var] and [Def], process variables are assigned located types (with participant numbers) rather than unlocated ones as in [27].

The use of located types in the presented system is not essential: we use located endpoint assertions for facilitating algorithmic validation. Delegations can be easily incorporated (cf. §8).

D.3 Proof of Lemma 5.4 (Substitution Lemma)

Below we reproduce the statement of Lemma 5.4 (Substitution Lemma) and present its proofs.

Lemma D.2 (Substitution). *Let* $\mathcal{C}; \Gamma, \tilde{u} : \tilde{S} \vdash P \triangleright \Delta$ *with* Δ *well-asserted. If* $\tilde{u}$ *is free in* P *and* $\tilde{\mathtt{n}}$ *have sorts* $\tilde{S}$ *then* $\mathcal{C}[\tilde{\mathtt{n}}/\tilde{u}]; \Gamma \vdash P[\tilde{\mathtt{n}}/\tilde{u}] \triangleright \Delta[\tilde{\mathtt{n}}/\tilde{u}]$ *and* $\Delta[\tilde{\mathtt{n}}/\tilde{u}]$ *is well-asserted.*

Proof. The proof is by rule induction on validation rules. We proceed by case analysis.

– If $P = s_k!\langle\tilde{e}\rangle(\tilde{v})\{A\}; P'$ then by [Send]

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

$(\mathcal{C} \supset A[\tilde{e}/\tilde{v}])$ implies $(\mathcal{C} \supset A[\tilde{e}/\tilde{v}])[\tilde{\mathtt{n}}/\tilde{u}]$ which is equivalent to:

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

By inductive hypothesis

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

By applying [Send] with premises D.1 and D.2 we obtain

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

The substituted end-point assertion is well-asserted since $\mathcal{C}[\tilde{\mathtt{n}}/\tilde{u}] \supset \exists\tilde{v}(A)[\tilde{\mathtt{n}}/\tilde{u}]$. This follows from $\mathcal{C} \supset \exists\tilde{v}(A)$ which holds by well-assertedness of the (unsubstituted) end-point assertion.

– If $P = s_k?(\tilde{v})\{A\}; P'$ then by [Recv]

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

By inductive hypothesis

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

By applying D.3 as a premise for [Recv] we obtain

$$\begin{array}{l} \mathcal{C}[\tilde{\mathtt{n}}/\tilde{u}]; \Gamma \vdash s_k?(\tilde{v})\{A'[\tilde{\mathtt{n}}/\tilde{u}]\}; P'[\tilde{\mathtt{n}}/\tilde{u}] \rhd \\ \quad \Delta[\tilde{\mathtt{n}}/\tilde{u}], \tilde{s} : \ k?(\tilde{v} : \tilde{S})\{A[\tilde{\mathtt{n}}/\tilde{u}]\}; \mathcal{T}[\tilde{\mathtt{n}}/\tilde{u}] \,@\, \mathtt{p} \end{array}$$

where the substituted end-point assertion is well-asserted since $\mathcal{C}[\tilde{\mathtt{n}}/\tilde{u}] \supset \exists \tilde{v}(A)[\tilde{\mathtt{n}}/\tilde{u}]$.

– If $P = s_k \lhd \{A_j\} l_j; P_j$ then by [Sel]

$$\frac{\mathcal{C} \supset A_j \quad \mathcal{C}; \Gamma, \tilde{u} : \tilde{S} \vdash P_j \rhd \Delta, \tilde{s} : \mathcal{T}_j \,@\, \mathtt{p} \quad j \in I}{\mathcal{C}; \Gamma, \tilde{u} : \tilde{S} \vdash s_k \lhd \{A_j\} l_j; P_j \rhd \Delta, \tilde{s} : k \oplus \{\{A_i\} l_i : \mathcal{T}_i\}_{i \in I} \,@\, \mathtt{p}}$$

By inductive hypothesis

$$\mathcal{C}[\tilde{\mathtt{n}}/\tilde{u}]; \Gamma \vdash P_j[\tilde{\mathtt{n}}/\tilde{u}] \rhd \Delta[\tilde{\mathtt{n}}/\tilde{u}], \tilde{s} : \mathcal{T}[\tilde{\mathtt{n}}/\tilde{u}] \,@\, \mathtt{p}. \tag{D.4}$$

$(\mathcal{C} \supset A_j)$ implies $(\mathcal{C} \supset A_j)[\tilde{\mathtt{n}}/\tilde{u}]$, i.e.

$$\mathcal{C}[\tilde{\mathtt{n}}/\tilde{u}] \supset A_j[\tilde{\mathtt{n}}/\tilde{u}]. \tag{D.5}$$

By applying D.4 and D.5 as a premise for [Sel] we obtain

$$\begin{array}{l} \mathcal{C}[\tilde{\mathtt{n}}/\tilde{u}]; \Gamma \vdash s_k \lhd \{A_j[\tilde{\mathtt{n}}/\tilde{u}]\} l_j; P_j[\tilde{\mathtt{n}}/\tilde{u}] \rhd \\ \quad \Delta[\tilde{\mathtt{n}}/\tilde{u}], \tilde{s} : k \oplus \{\{A_i[\tilde{\mathtt{n}}/\tilde{u}]\} l_i : \mathcal{T}_i[\tilde{\mathtt{n}}/\tilde{u}]\}_{i \in I} \,@\, \mathtt{p} \end{array}$$

where the substituted end-point assertion is well-asserted since $\mathcal{C}[\tilde{\mathtt{n}}/\tilde{u}] \supset (A_1 \vee \ldots A_n)[\tilde{\mathtt{n}}/\tilde{u}]$. This follows from $\mathcal{C} \supset A_1 \vee \ldots \vee A_n$ which holds by well-assertedness of the (unsubstituted) end-point assertion.

– The case for [Branch] is similar to the case of [Recv].

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

$$\frac{\mathcal{C} \wedge e; \Gamma, \tilde{u} : \tilde{S} \vdash Q \rhd \Delta \quad \mathcal{C} \wedge \neg e; \Gamma, \tilde{u} : \tilde{S} \vdash R \rhd \Delta}{\mathcal{C}; \Gamma, \tilde{u} : \tilde{S} \vdash \mathsf{if}\ e\ \mathsf{then}\ Q\ \mathsf{else}\ R \rhd \Delta}$$

By inductive hypothesis

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

By applying D.6 as a premise for [If] we obtain

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

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

– If $P = \overline{a}_{[2..\mathtt{n}]}\,(\tilde{s}).P'$ by [Mcast]

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

By inductive hypothesis

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

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

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

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

– The case for [Macc] is similar to the case for [Mcast].

– If $P = X\langle \tilde{e}\tilde{s}_1..\tilde{s}_n \rangle$ by [Var]

$$\frac{\mathcal{C} \supset A[\tilde{e}/\tilde{v}]}{\begin{array}{c}\mathcal{C}; \Gamma, \tilde{u} : \tilde{S}, X : (\tilde{v} : \tilde{S})\mathcal{T}_1 \mathbin{@} \mathtt{p}_1..\mathcal{T}_n \mathbin{@} \mathtt{p}_n A \vdash X\langle \tilde{e}\tilde{s}_1..\tilde{s}_n \rangle \\ \rhd \Delta, \tilde{s}_1 : \mathcal{T}_1[\tilde{e}/\tilde{v}] \mathbin{@} \mathtt{p}_1, .., \tilde{s}_n : \mathcal{T}_n[\tilde{e}/\tilde{v}] \mathbin{@} \mathtt{p}_n\end{array}}$$

Since $\mathcal{C} \supset A[\tilde{e}/\tilde{v}]$ then also $\mathcal{C}[\tilde{\mathtt{n}}/\tilde{u}] \supset A[\tilde{e}/\tilde{v}][\tilde{\mathtt{n}}/\tilde{u}]$. By applying it as a premise of [Var] we obtain

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

Where $\Delta[\tilde{\mathtt{n}}/\tilde{u}]$ is well-asserted (since it is end only).

The remaining cases for [Conc] and [Def] are straightforward. □

E Appendix for § 7

E.1 On Unfoldings of Recursive Assertions

This section illustrates the notion *unfolding* in recursive endpoint assertions. A key idea follows [19]:

Remark E.1. In the labelled transitions for assertions (Figure 11) and in the refinement relation (Definition 7.4), both given in § 7.1, page 40), recursive endpoint assertions are taken up to their unfoldings (this point was not explicitly noted in an earlier version).

Definition E.2 (unfolding). Write $\mathcal{T}'[(\tilde{v})\mathcal{T}/\mathbf{t}]$ for the result of replacing each occurrence of the form $\mathbf{t}\langle\tilde{e}_i\rangle$ in $\mathcal{T}'$ with $\mathcal{T}[\tilde{e}_i/\tilde{v}]$, with an appropriate renaming of bound interaction variables. Then we define, with $\mathcal{T} \stackrel{\text{def}}{=} \mu\mathbf{t}\langle\tilde{e}\rangle(\Gamma)\{A\}.\mathcal{T}'$,

$$\mathsf{unfold}(\mathcal{T}) \stackrel{\text{def}}{=} \mathcal{G}'[(\tilde{v})\mathcal{T}/\mathbf{t}]$$

which we call the *unfolding of* $\mathcal{T}$.

Throughout the proofs from now on, especially when we consider the labelled transition of endpoint assertions, we consider recursive assertions up to their unfoldings, which, among others, enable visible transitions. For example, given

$$\mathcal{T} \stackrel{\text{def}}{=} \mu\mathbf{t}\langle 2\rangle(v : S)\{v \gtrsim 1\}.k \oplus \{l_1 : \mathbf{t}\langle v+1\rangle, l_2 : \mathsf{end}\}$$

assuming $L = \{p, q\}$ and considering all interaction predicates for $l_{1,2,3}$ are true, we have

$$\mathsf{unfold}(\mathcal{T}) = k \oplus \{l_1 : \mathcal{T}', l_2 : \mathsf{end}\}$$

where we set:

$$\begin{aligned}\mathcal{T}' = \mu\mathbf{t}\langle 3\rangle(v : S)\{v \gtrsim 1\}.\\ k \oplus \{l_1 : \mathbf{t}\langle v+1\rangle, l_2 : \mathsf{end}\}\end{aligned}$$

which allows the assertion to have a visible transition.

$$\overline{a}_{[2..\mathtt{n}]}\,(\tilde{s}).P \xrightarrow{\overline{a}[2..n](\tilde{s})} P \qquad [\textsc{LinkOut}]$$

$$a_{[\mathtt{i}]}\,(\tilde{s}).P_i \xrightarrow{a[i](\tilde{s})} P_i \qquad [\textsc{LinkIn}]$$

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

$$s_k!\langle\tilde{\mathtt{n}}\rangle(\tilde{v})\{A\};P \xrightarrow{s_k!\tilde{\mathtt{n}}} \mathsf{errH} \quad (A[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{false}) \qquad [\textsc{SendErr}]$$

$$s_k?(\tilde{v})\{A\};P \xrightarrow{s_k?\tilde{\mathtt{n}}} P[\tilde{\mathtt{n}}/\tilde{v}] \quad (A[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{true}) \qquad [\textsc{Recv}]$$

$$s_k?(\tilde{v})\{A\};P \xrightarrow{s_k?\tilde{\mathtt{n}}} \mathsf{errT} \quad (A[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{false}) \qquad [\textsc{RecvErr}]$$

$$s_k \lhd \{A\}l;P \xrightarrow{s_k \lhd l} P \quad (A \downarrow \mathsf{true}) \qquad [\textsc{Label}]$$

$$s_k \lhd \{A\}l;P \xrightarrow{s_k \lhd l} \mathsf{errH} \quad (A \downarrow \mathsf{false}) \qquad [\textsc{LabelErr}]$$

$$s_k \rhd \{\{A_i\}l_i\colon P_i\}_{i\in I} \xrightarrow{s_k \rhd l_j} P_j \quad (A_j \downarrow \mathsf{true})_{j\in I} \qquad [\textsc{Branch}]$$

$$s_k \rhd \{\{A_i\}l_i\colon P_i\}_{i\in I} \xrightarrow{s_k \rhd l_j} \mathsf{errT} \quad (A_j \downarrow \mathsf{false})_{j\in I} \qquad [\textsc{BranchErr}]$$

$$P|Q \xrightarrow{\alpha} P'|Q \quad (\text{when } P \xrightarrow{\alpha} P') \qquad [\textsc{Par}]$$

$$(\boldsymbol{\nu} a)P \xrightarrow{\alpha} (\boldsymbol{\nu} a)P' \quad (\text{when } P \xrightarrow{\alpha} P' \text{ and } a \notin \mathsf{fn}\alpha) \qquad [\textsc{NRes}]$$

$$(\boldsymbol{\nu}\tilde{s})P \xrightarrow{\alpha} (\boldsymbol{\nu}\tilde{s})P' \quad (\text{when } P \xrightarrow{\alpha} P') \qquad [\textsc{CRes}]$$

$$(\boldsymbol{\nu} a)P \xrightarrow{(\boldsymbol{\nu} a\tilde{b})s!\tilde{\mathtt{n}}} P' \quad (\text{when } P \xrightarrow{(\boldsymbol{\nu}\tilde{b})s!\tilde{\mathtt{n}}} P' \text{ and } a \in \{\tilde{\mathtt{n}}\}) \qquad [\textsc{BOut}]$$

$$P \xrightarrow{\tau} Q \quad (\text{when } P \to Q) \qquad [\textsc{Tau}]$$

$$\mathsf{def}\ D\ \mathsf{in}\ C[X\langle\tilde{e}\tilde{s}\rangle] \xrightarrow{\alpha} \mathsf{def}\ D\ \mathsf{in}\ Q$$
$$(\text{when } \langle X\langle\tilde{v}\tilde{s}\rangle = P\rangle \in D \text{ and } C[P[\tilde{e}/\tilde{v}]] \xrightarrow{\alpha} Q) \qquad [\textsc{Def}]$$

$$P \xrightarrow{\alpha} Q \quad (\text{when } P' \xrightarrow{\alpha} Q',\ P \equiv P' \text{ and } Q \equiv Q') \qquad [\textsc{Str}]$$

Fig. 15. Labelled Transition for Processes

E.2 Labelled Transition Relation for Processes

E.3 Proof of Proposition 7.5 (Refinement)

This appendix proves Proposition 7.5 (relationship between the refinement relation and the satisfaction), after a lemma.

We first reproduce the definition of refinement (Definition 7.4) for convenience.

Definition E.3. An endpoint assertion is *closed* (resp. *open*) if it does not (resp. it may) contain free variables.

Definition E.4 (Refinement). A binary relation $\mathcal{R}$ over closed well-asserted end-point assertions, taken up to the unfolding of recursive assertions is a *refinement relation* if $\mathcal{T}_1 \mathcal{R} \mathcal{T}_2$ implies one of the following 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 $A_2\sigma \downarrow \mathsf{true}$.
- $\mathcal{T}_1 = k \oplus \{\{A_i\} l_i : \mathcal{T}_i\}_{i \in I}$ and $\mathcal{T}_2 = k \oplus \{\{A_j\} l_j : \mathcal{T}_j\}_{j \in J}$ where for each $i \in I$, we have $l_i = l_j$, $A_i \supset A_j$ and $\mathcal{T}_i \mathcal{R} \mathcal{T}_j$ for some $j \in J$.
- $\mathcal{T}_1 = k \& \{\{A_i\} l_i : \mathcal{T}_i\}_{i \in I}$ and $\mathcal{T}_2 = k \& \{\{A_j\} l_j : \mathcal{T}_j\}_{j \in J}$ where for each $j \in J$, we have $l_i = l_j$, $A_j \supset A_i$ and $\mathcal{T}_i \mathcal{R} \mathcal{T}_j$ for some $j \in J$.

If $\mathcal{T}_1 \mathcal{R} \mathcal{T}_2$ for some refinement relation $\mathcal{R}$, we say $\mathcal{T}_1$ is a *refinement* of $\mathcal{T}_2$, denoted $\mathcal{T}_1 \Subset \mathcal{T}_2$. The relation $\Subset$ extends to open endpoint assertions in the standard way.

Note we are taking recursive assertions up to the unfolding (cf. Appendix E.1) in the definition above.
The following notion already appeared in §7.2 (after Theorem 7.11).

Definition E.5. $\langle \Gamma, \Delta \rangle$ allows α *when* $\langle \Gamma, \Delta \rangle \xrightarrow{\alpha} \langle \Gamma', \Delta' \rangle$ *for some* $\langle \Gamma', \Delta' \rangle$.

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 $\mathsf{sbj}(\alpha)$.

Lemma E.6. *Assume* $\Delta \Subset \Delta'$. *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* $\mathsf{sbj}(\alpha)$.

Remark. The end-point assertions in Δ and in Δ' are well-asserted by definition of refinement (Definition 7.4). Notice anyway that it is sufficient that only the end-point assertions in Δ' are well-asserted for this lemma to hold.

Proof. The proof is straightforward from the definition refinement (Definition 7.4). Notice that, up to the unfoldings of recursive assertions, an end-point 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 7.4,

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 their respective continuations. □

Lemma E.7. *Assume $\Delta \Supset \Delta'$ below.*

1. *If $\langle \Gamma, \Delta \rangle \xrightarrow{\alpha} \langle \Gamma_1, \Delta_1 \rangle$ such that α being a value output, selection or the τ-action, then $\langle \Gamma, \Delta' \rangle \xrightarrow{\alpha} \langle \Gamma_1, \Delta'_1 \rangle$ such that $\Delta_1 \Supset \Delta'_1$ again.*
2. *If $\langle \Gamma, \Delta \rangle \xrightarrow{\alpha} \langle \Gamma_1, \Delta_1 \rangle$ such that α being an input or branching, and if (Γ, Δ') allows α, then $\langle \Gamma, \Delta' \rangle \xrightarrow{\alpha} \langle \Gamma_1, \Delta'_1 \rangle$ such that $\Delta_1 \Supset \Delta'_1$ again.*

Proof. The proof is 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 $\tilde{s} : \mathcal{T} \, @ \, \mathtt{p}$ and assume the transition is from this 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 outout, we do not consider the cases of new name import and export, since they only add to Γ the same new elements. Below in each case we use $\Supset$ over endpoint assertions as the refinement relation justifying the original refinement (note $\Supset$ is the largest refinement relation).

(1) If $\Delta = \Delta_{side}, \tilde{s} : k!(\tilde{v} : \tilde{S})\{A_1\}; \mathcal{T}' \, @ \, \mathtt{p}$ then

$$\langle \Gamma, \Delta \rangle \xrightarrow{s_k ! \tilde{\mathtt{n}}} \langle \Gamma, \Delta_{side}, \tilde{s} : \mathcal{T}' \, @ \, \mathtt{p} \rangle \tag{E.1}$$

by [TR-Send]. By hypothesis we have $\Delta \Supset \Delta'$, so by Definition 7.4 we can set

$$\Delta' = \Delta'_{side}, \tilde{s} : k!(\tilde{v} : \tilde{s})\{A_2\}; \mathcal{T}'' \, @ \, \mathtt{p} \tag{E.2}$$

with $\Delta_{side} \Supset \Delta'_{side}$, $\mathcal{T}' \Supset \mathcal{T}''$ and $A_1 \supset A_2$. Since $A_1 \supset A_2$ then

$$A_1[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{true} \supset A_2[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{true}. \tag{E.3}$$

It follows that also the following transition is possible:

$$\langle \Gamma, \Delta' \rangle \xrightarrow{s_k ! \tilde{\mathtt{n}}} \langle \Gamma, \Delta'_{side}, \tilde{s} : \mathcal{T}'' \, @ \, \mathtt{p} \rangle. \tag{E.4}$$

The lemma hold by induction for this case since $\Delta_{side}, \tilde{s} : \mathcal{T}' \, @ \, \mathtt{p} \Supset \Delta'_{side}, \tilde{s} : \mathcal{T}'' \, @ \, \mathtt{p}$.

(2) If $\Delta = \Delta_{side}, \tilde{s}:\ k?(\tilde{v} : \tilde{S})\{A_1\}; \mathcal{T}' \,@\, \mathtt{p}$ then

$$\langle \Gamma, \Delta \rangle \overset{s_k?\tilde{\mathtt{n}}}{\rightarrow} \langle \Gamma, \Delta_{side}, \tilde{s} : \mathcal{T}' \,@\, \mathtt{p} \rangle \tag{E.5}$$

by [Tr-Rec]. Assume further we have

$$\langle \Gamma, \Delta' \rangle \text{ allows } s_k?\tilde{\mathtt{n}}. \tag{E.6}$$

As before, by hypothesis and by Definition 7.4 we can set:

$$\Delta' = \Delta'_{side}, \tilde{s} : k?(\tilde{v} : \tilde{s})\{\}A_2; \mathcal{T}'' \,@\, \mathtt{p} \tag{E.7}$$

such that $\Delta_{side} \ni \Delta'_{side}$, $\mathcal{T}' \ni \mathcal{T}''$ and $A_2 \supset A_1$. By (E.6), however, we also have $A_2[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{true}$. It follows that the following transition is possible:

$$\langle \Gamma, \Delta' \rangle \overset{s_k?\tilde{\mathtt{n}}}{\rightarrow} \langle \Gamma, \Delta'_{side}, \tilde{s} : \mathcal{T}'' \,@\, \mathtt{p} \rangle \tag{E.8}$$

The statement hold since $\Delta_{side}, \tilde{s} : \mathcal{T}' \,@\, \mathtt{p} \rangle \ni \Delta'_{side}, \tilde{s} : \mathcal{T}'' \,@\, \mathtt{p}$.

(3) If $\Delta = \Delta_{side}, \tilde{s} : k \oplus \{\{A_{1i}\}l_i : \mathcal{T}_{1i}\}_{i \in I} \,@\, \mathtt{p}$ then

$$\langle \Gamma, \Delta \rangle \overset{s_k < l_j}{\rightarrow} \langle \Gamma, \Delta_{side}, \tilde{s} : \mathcal{T}_{1j} \,@\, \mathtt{p} \rangle \tag{E.9}$$

by [Tr-Sel]. By hypothesis and by Definition 7.4, we can set

$$\Delta' = \Delta'_{side}, \tilde{s} : k \oplus \{\{A_{1i}\}l_i : \mathcal{T}_{1i}\}_{i \in J} \,@\, \mathtt{p} \tag{E.10}$$

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:

$$\langle \Gamma, \Delta' \rangle \overset{s_k < l_j}{\rightarrow} \langle \Gamma, \Delta'_{side}, \tilde{s} : \mathcal{T}_{2j} \,@\, \mathtt{p} \rangle. \tag{E.11}$$

The lemma hold since $\Delta_{side}, \tilde{s} : \mathcal{T}_{1i} \,@\, \mathtt{p} \ni \Delta'_{side}, \tilde{s} : \mathcal{T}_{2j} \,@\, \mathtt{p}$.

(4) If $\Delta = \Delta_{side}, \tilde{s} : k\&\{\{A_{1i}\}l_i : \mathcal{T}_{1i}\}_{i \in I} \,@\, \mathtt{p}$ then

$$\langle \Gamma, \Delta \rangle \overset{s_k > l_j}{\rightarrow} \langle \Gamma, \Delta_{side}, \tilde{s} : \mathcal{T}_{1j} \,@\, \mathtt{p} \rangle. \tag{E.12}$$

Assume further we have

$$\langle \Gamma, \Delta' \rangle \text{ allows } s_k > l_j. \tag{E.13}$$

By hypothesis and by Definition 7.4, we can set

$$\Delta = \Delta'_{side}, \tilde{s} : k\&\{\{A_{1i}\}l_i : \mathcal{T}_{1i}\}_{i \in J} \,@\, \mathtt{p} \tag{E.14}$$

with $\Delta_{side} \ni \Delta'_{side}$. By (E.13) we know $j \in J$ and $A_{1j} \downarrow \mathsf{true}$. It follows that also the following transition is possible:

$$\langle \Gamma, \Delta' \rangle \overset{s_k > l_j}{\rightarrow} \langle \Gamma, \Delta'_{side}, \tilde{s} : \mathcal{T}_{2j} \,@\, \mathtt{p} \rangle. \tag{E.15}$$

The lemma hold since $\Delta_{side}, \tilde{s} : \mathcal{T}_{1i} \,@\, \mathtt{p} \ni \Delta'_{side}, \tilde{s} : \mathcal{T}_{2j} \,@\, \mathtt{p}$.

(5) The case of [TR-Tau] is immediate since there is no change in assertion environments. □

We now prove Proposition 7.5. We reproduce the statement below.

Proposition E.8 (Refinement). *If* $\Gamma \models P \rhd \Delta$ *and* $\Delta \ni \Delta'$ *then* $\Gamma \models P \rhd \Delta'$.

Proof. The proof is by induction on the transitions of P. We proceed by case analysis.

1 If $P \xrightarrow{\alpha} P'$ by output/selection/τ move, since $\Gamma \models P \rhd \Delta$ then $\langle \Gamma, \Delta \rangle \xrightarrow{\alpha} \langle \Gamma, \Delta_1 \rangle$. By Lemma E.7, $\langle \Gamma, \Delta' \rangle \xrightarrow{\alpha} \langle \Gamma, \Delta'_1 \rangle$ where $\Delta_1 \ni \Delta'_1$.

2 If $P \xrightarrow{\alpha} P'$ by input/branching, since $\Gamma \models P \rhd \Delta$ then $\langle \Gamma, \Delta \rangle$ has the capability of a move at the subject $\mathsf{sbj}(\alpha)$. By Lemma E.6 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 $\Gamma \models P \rhd \Delta'$ 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 end-point assertion in input/branching moves. By Lemma E.7, $\Delta_1 \ni \Delta'_1$. The predicate holds by induction.

□

E.4 Validation Rules of Runtime Processes (definition)

Message Assertions For treating the τ-action (which is identical with reduction), we introduce the validation rules for runtime processes (for the formal definition of runtime processes, see Appendix D.1, page 73). For this purpose we adapt the framework of typing for runtime processes in [5], using *message assertions* (corresponding to *message types* in [5]), which abstract messages in queues. These message assertions are only needed for asserting for processes whose free session channels can have queues, thus inducing reductions: they are not needed for asserting for transition derivatives of programs, cf. Appendix D.1.

We first extend endpoint assertions as follows.

$$\begin{array}{rcl} \mathcal{M} & ::= & k!\langle \tilde{\mathfrak{n}} \rangle \mid k \oplus l \mid \mathcal{M}; \mathcal{M}' \\ \mathcal{T} & ::= & \ldots \mid \mathcal{M} \mid \mathcal{M}; \mathcal{T} \end{array}$$

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

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

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

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

$$\Delta_1,\Delta_2 \to \Delta_1',\Delta_2 \quad (\Delta_1\to\Delta_1')$$

Fig. 16. Reduction Rules for Assertion Assignments

We call $\mathcal{M}$ a *message assertion*, which is simply a sequence of a sending action with a conrete value and a selection action with a concrete label. Using this extended set of endpoint assertions, we further extend several notions. First we use a context $\mathcal{H}[\ \cdot\ ]$ given by the grammar:

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

Next we extend $\circ$ as:

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

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

In the second rule we add a prefix of a message assertion to an endpoint assertion from the head of a queue. In the rule we used the commutative and associative operator $*$ as follows. Below $\varnothing$ is the empty sequence.

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

Reduction of Message Assertions We can now define the rules for asserted reduction for assertion assignments which plays a key role in the proof of Subject Reduction, given in Figure Figure 16. The rules come from [5], elaborated with assertion checking.

1. The first rule non-deterministically instantiates an assertion for sending under the predicate A to the corresponding message assertion with carried values satisfying A.
2. The second rule non-deterministically instantiates an assertion for selection 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.
3. The third rules depict how a sending message assertion interacts with its dual, the assertion for receiving.
4. The fourth rules depict how a selection message assertion interacts with the assertion for branching.
5. The fifth and the sixth rules close the reduction under contexts.

Some comments on the use of non-deterministic instantiation of values and labels in the first and second rules follow.

Remark E.9 (non-deterministic instantiation). The motivation for having the non-deterministic instantiation rules for the assertions for 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. This idea comes from [5]: athe involved non-determinism is particularly natural in the present context since each assertions (say an assertion for sending) describes many, possibly infinite, instances of distinct process behaviours.

Validation for Queues and Session Hiding We list the validation rules for queues and channel hiding in Figure 17, where [NRes] in Figure 17 generalises that of [NRes] in Figure 10 (since if $a \notin \mathsf{fn}\Delta$ then the hiding can be erased by the equality above) hence replacing the original version. The remaining rules from Figure 10 are used as they are except we are now using the extended sets of processes and assertion assignments (accordingly [Conc] now uses the extended $\circ$ defined above). Since message assertions do not involve interaction predicates, these rules are a direct analogue of the typing rules for runtime processes in [5, 27] except message assertions now mention values.

Convention E.10. Henceforth we write $\mathcal{C}; \Gamma \vdash P \rhd \Delta$ for a runtime process P when it is derived by combining the rules of Figure 10 excepting [NRes] and those of Figure 17.

Note that, following [5, 27], the validation of the composability of multiple processes is relegated to the session hiding rule [CRes] rather than to

$$\frac{-}{\Gamma \vdash s_k : \varnothing \ \rhd \tilde{s} : \{\varnothing \,@\, \mathtt{p}\}_{\mathtt{p}}} \quad [\textsc{Qnil}]$$

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

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

$$\frac{\Gamma \vdash P \rhd \Delta, \tilde{s} : \{\mathcal{T}_{\mathtt{p}} \,@\, \mathtt{p}\}_{\mathtt{p} \in I} \quad \{\mathcal{T}_{\mathtt{p}} \,@\, \mathtt{p}\}_{\mathtt{p} \in I} \text{ coherent}}{\Gamma \vdash \ (\boldsymbol{\nu}\, \tilde{s}) P \rhd \Delta} \quad [\textsc{CRes}]$$

Fig. 17. Validation Rules for Runtime Processes

the parallel composition rule [Conc]. By the shape of these rules we immediately observe:

Proposition E.11. *Suppose $\Gamma \vdash P \rhd \Delta$. Then P contains no error.*

E.5 Proof of Proposition 7.8 (Subject Transition for Visible Actions)

In this subsection we list the proofs for Proposition 7.8. The proof hinges on two lemmas: Substitution Lemma (Lemma 5.4, whose statement and proof are given in Appendix D.3, page 77); and Evaluation Lemma, whose statement and proofs are given below. Then we shall prove Proposition 7.8.

Convention E.12 (shape of processes).

1. In this subsection, unless otherwise stated, $P, Q, \ldots$ range over runtime (i.e. general) processes (cf. Appendix D.1) and validation and other judgements are considered for runtime processes (cf. Convention E.10).
2. Further whenever the definitions, statements etc. mention the transition or reduction of processes, we implicitly assume these processes are closed (remember reduction and transition are only defined over closed processes).

Lemma E.13 (Evaluation). *If* $\mathcal{C}; \Gamma \vdash P(e) \triangleright \Delta(\tilde{e})$ *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. The proof is by rule induction on the validation rules (Figures 10 and 17). We proceed by case analysis. Recall that by decidability of underlying logic (Convention 3.3), we 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 $A[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{true}$.

- If $P(\tilde{e}) = s_k!\langle\tilde{e}\rangle(\tilde{v})\{A\}; P'$ then

$$P(\tilde{\mathtt{n}}) = s_k!\langle\tilde{n}\rangle(\tilde{v})\{A\}; P',$$

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

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

$$\mathcal{C} \supset A[\tilde{\mathtt{n}}/\tilde{v}]. \tag{E.16}$$

  By inductive hypothesis

$$\mathcal{C}; \Gamma \vdash P'[\tilde{\mathtt{n}}/\tilde{e}] \triangleright \Delta'[\tilde{\mathtt{n}}/\tilde{e}], \tilde{s}\colon \mathcal{T}[\tilde{\mathtt{n}}/\tilde{e}] \,@\, \mathtt{p}. \tag{E.17}$$

  By applying (E.16) and (E.17) to the validation rule [Send] the lemma holds for this case.
- 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].
- $P(e) = \mathsf{if}\ e\ \mathsf{then}\ Q\ \mathsf{else}\ R$ the property holds by induction since $\mathcal{C} \wedge e \downarrow \mathsf{true}$ is equivalent to $\mathcal{C} \wedge n \downarrow \mathsf{true}$.
- If $P(\tilde{e}) = s_k!\langle\tilde{e'}\rangle(\tilde{v})\{A\}; P'$, $P(\tilde{e}) = X\langle\tilde{e'}\tilde{s}_1..\tilde{s}_n\rangle$, $P(e) = \mathsf{if}\ e'\ \mathsf{then}\ Q\ \mathsf{else}\ R$, multicast session request, session acceptance, value reception, label selection, label branching, parallel composition, inaction, hiding, recursion, message queue or error the property holds straightforwardly by induction.

This exhausts all cases. □

We now prove Proposition 7.8 in §7, the subject transition for visible transitions. We reproduce the statement in the following.

Proposition E.14 (Subject Transition for 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 10 and 17, showing a stronger result which adds to the statement:

> If $P \xrightarrow{\alpha} P'$ and $\Gamma \vdash P \rhd \Delta$ 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 15 and the transition rules for end-point assertions in Figure 11. Assume we have:

1. $\Gamma \vdash P \rhd \Delta$ (which stands for $\mathsf{true}; \Gamma \vdash P \rhd \Delta$)
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. By Convention E.12 (2), we assume all processes concerned are closed. Further below notice $\mathcal{C}$ in the conclusion of each rule should be true by our assumption.

Rule [Send]: In this case, we derive $\mathcal{C}; \Gamma \vdash P \rhd \Delta$ with:

$$\mathcal{C} = \mathsf{true} \tag{E.18}$$

$$P = s_k!\langle \tilde{e} \rangle(\tilde{v})\{A\}; Q \tag{E.19}$$

$$\Delta = \Delta_0, \tilde{s} : k!(\tilde{v} : \tilde{S})\{A\}; \mathcal{T} \, @ \, \mathtt{p}. \tag{E.20}$$

By the first premise of [Send] and (E.18) we have:

$$\mathsf{true} \supset A[\tilde{e}/\tilde{v}] \tag{E.21}$$

Since P is closed, we can set $\tilde{e} \downarrow \mathtt{n}$. By (E.21) we infer:

$$A[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{true}. \tag{E.22}$$

It follows that P can move only by [Send] (i.e., not [SendErr]), hence, setting $\alpha = s_k!\tilde{\mathtt{n}}$:

$$P \quad \xrightarrow{\alpha} \quad Q[\tilde{\mathtt{n}}/\tilde{v}] \quad \stackrel{\text{def}}{=} \quad P' \tag{E.23}$$

Now Δ can move by [TR-Send]:

$$\langle \Gamma, \Delta \rangle \xrightarrow{\alpha} \langle \Gamma, \ (\Delta_0, \tilde{s} : \mathcal{T}[\tilde{\mathtt{n}}/\tilde{v}] \, @ \, \mathtt{p}) \rangle \tag{E.24}$$

By the second premise of [Send] in Figure 10, we have

$$\mathsf{true}; \Gamma \vdash Q[\tilde{e}/\tilde{v}] \rhd \Delta_0, \tilde{s} : \mathcal{T}[\tilde{e}/\tilde{v}] \, @ \, \mathtt{p} \tag{E.25}$$

By Lemma E.13 (Evaluation Lemma), (E.25) immediately gives:

$$\mathsf{true}; \Gamma \vdash Q[\tilde{\mathtt{n}}/\tilde{v}] \rhd \Delta_0, \tilde{s} : \mathcal{T}[\tilde{\mathtt{n}}/\tilde{v}] \, @ \, \mathtt{p} \tag{E.26}$$

as required.

Rule [Rcv]: In this case the conclusion is $\mathcal{C}; \Gamma \vdash P \rhd \Delta$ with, as well as $\mathcal{C} = \mathsf{true}$ as before:

$$P = s_k?(\tilde{v})\{A\}; Q \tag{E.27}$$

$$\Delta = \Delta_0, \tilde{s}\colon\ k?(\tilde{v} : \tilde{S})\{A\}; \mathcal{T} \mathbin{@} \mathtt{p} \tag{E.28}$$

By the shape of P we can set $\alpha = s_k?\tilde{\mathtt{n}}$ for which we have, by [TR-Rec]:

$$A[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{true} \tag{E.29}$$

$$\langle \Gamma, \Delta \rangle \xrightarrow{\alpha} \langle \Gamma \cdot \tilde{a} : \tilde{\mathcal{G}},\ \ \Delta_0, \tilde{s}\colon\ \mathcal{T}[\tilde{\mathtt{n}}/\tilde{v}] \mathbin{@} \mathtt{p} \rangle \tag{E.30}$$

Thus P can move only by [Recv] (not by [RecvErr]), obtaining:

$$P \xrightarrow{\alpha} Q[\tilde{\mathtt{n}}/\tilde{v}] \tag{E.31}$$

Now the premise of [Rcv] in Figure 10 says:

$$\mathsf{true} \wedge A\ ;\ \Gamma \vdash Q \rhd \Delta_0, \tilde{s}\colon\ \mathcal{T} \mathbin{@} \mathtt{p} \tag{E.32}$$

By Lemma 5.4 (Substitution Lemma) we obtain

$$\mathsf{true} \wedge A[\tilde{\mathtt{n}}/\tilde{v}]; \Gamma, \tilde{v} : \tilde{S} \vdash Q[\tilde{\mathtt{n}}/\tilde{v}] \rhd \Delta_0, \tilde{s}\colon\ \mathcal{T}[\tilde{\mathtt{n}}/\tilde{v}] \mathbin{@} \mathtt{p} \tag{E.33}$$

By (E.29) and by [Conseq] we obtain

$$\mathsf{true}; \Gamma, \tilde{v} : \tilde{S} \vdash Q[\tilde{\mathtt{n}}/\tilde{v}] \rhd \Delta_0, \tilde{s}\colon\ \mathcal{T}[\tilde{\mathtt{n}}/\tilde{v}] \mathbin{@} \mathtt{p} \tag{E.34}$$

as required.

Rule [Sel]: We can set $\mathcal{C}; \Gamma \vdash P \rhd \Delta$ such that, as well as $\mathcal{C} = \mathsf{true}$:

$$P = s_k \lhd \{A_j\} l_j; P_j \tag{E.35}$$

$$\Delta = \Delta_0, \tilde{s}\colon\ k \oplus \{\{A_i\} l_i : \mathcal{T}_i\}_{i \in I} \mathbin{@} \mathtt{p} \tag{E.36}$$

By the premise of the rule we have:

$$\mathsf{true} \supset A_j \tag{E.37}$$

hence $A_j \downarrow \mathsf{true}$, therefore P can move only by [Label] (i.e., not [LabelErr]). Thus we set $\alpha = s_k! < l_j$ and we have

$$P \xrightarrow{\alpha} P_j \tag{E.38}$$

The following assertion transition is also possible by [TR-Select]:

$$\langle \Gamma, \Delta \rangle \xrightarrow{\alpha} \langle \Gamma, \Delta_0, \tilde{s}\colon \mathcal{T}_j \mathbin{@} \mathtt{p} \rangle. \tag{E.39}$$

By the second premise of [Label] in Figure 10 we get

$$\mathsf{true}; \Gamma \vdash P_j \rhd \Delta_0, \tilde{s}\colon \mathcal{T}_j \mathbin{@} \mathtt{p} \tag{E.40}$$

as required.

Rule [Branch]: In this case we have $\mathsf{true}; \Gamma \vdash P \rhd \Delta$ such that

$$P = s_k \rhd \{\{A_i\} l_i \colon P_i\}_{i \in I} \tag{E.41}$$
$$\Delta = \Delta_0, \tilde{s} \colon k \& \{\{A_i\} l_i : \mathcal{T}_i\}_{i \in I} \, @ \, \mathtt{p} \tag{E.42}$$

By the shape of P we can set $\alpha = s_k < l_j$ for which we have, by [TR-Choice]:

$$A_j \downarrow \mathsf{true} \tag{E.43}$$
$$\langle \Gamma, \Delta \rangle \xrightarrow{\alpha} \langle \Gamma \cdot \tilde{a} : \tilde{\mathcal{G}}, \quad \Delta_0, \tilde{s} \colon \ \mathcal{T}_j \, @ \, \mathtt{p} \rangle \tag{E.44}$$

Thus P can move only by [Branch] (not by [BranchErr]), obtaining:

$$P \xrightarrow{\alpha} P_j \tag{E.45}$$

Now the premise of [Branch] in Figure 10 says:

$$\mathsf{true} \wedge A_j \ ; \ \Gamma \vdash P_j \rhd \Delta_0, \tilde{s} \colon \ \mathcal{T}_j \, @ \, \mathtt{p} \tag{E.46}$$

By (E.43) and [Conseq] we obtain:

$$\mathsf{true}; \Gamma, \tilde{v} \colon \tilde{S} \vdash Q[\tilde{\mathtt{n}}/\tilde{v}] \rhd \Delta_0, \tilde{s} \colon \ \mathcal{T}[\tilde{\mathtt{n}}/\tilde{v}] \, @ \, \mathtt{p} \tag{E.47}$$

as required.

Rule [Mcast]: In this case we have $\mathsf{true}; \Gamma \vdash P \rhd \Delta$ such that, combining with the premises of the rule:

$$P = \overline{a}_{[2..\mathtt{n}]}\,(\tilde{s}).Q \tag{E.48}$$
$$\Gamma \vdash a \colon \mathcal{G} \tag{E.49}$$
$$\mathsf{true}; \Gamma \vdash Q \rhd \Delta, \tilde{s} \colon (\mathcal{G} \upharpoonright \mathtt{1}) \, @ \, \mathtt{1} \tag{E.50}$$

By the shape of P we can set $\alpha = \overline{a}[2..n](\tilde{s})$ and

$$\overline{a}_{[2..\mathtt{n}]}\,(\tilde{s}).Q \xrightarrow{\alpha} Q \tag{E.51}$$

By (E.49) the following transition is possible using [TR-LinkOut]:

$$\langle \Gamma, \Delta \rangle \xrightarrow{\alpha} \langle \Gamma, \Delta, \tilde{s} \colon (\mathcal{G} \upharpoonright \mathtt{1}) \, @ \, \mathtt{1} \rangle \tag{E.52}$$

as required.

Rule [Macc]: Similar to the case [Mcast] above.

Rule [Par] Immediate, since the visible transition for $P|Q$ 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.

Rules [NRes], [CRes] and [BOut]: In each case, direct from the induction hypothesis.

Rule [Conseq]: Suppose the conclusion is $\mathsf{true}; \Gamma \vdash P \rhd \Delta$ which is derived from

$$\mathsf{true}; \Gamma \vdash P \rhd \Delta_0 \tag{E.53}$$

$$\Delta_0 \ni \Delta \tag{E.54}$$

Now first suppose the concerned visible action α is neither a receive action nor a branching. Now suppose

$$P \xrightarrow{\alpha} P' \tag{E.55}$$

$$\langle \Gamma, \Delta \rangle \xrightarrow{\alpha} \langle \Gamma', \Delta' \rangle \tag{E.56}$$

By induction hypothesis and by (E.53), (E.56) gives us:

$$\langle \Gamma, \Delta_0 \rangle \xrightarrow{\alpha} \langle \Gamma', \Delta'_0 \rangle \tag{E.57}$$

for some Δ'_0 for which we have, by induction hypothesis

$$\mathsf{true}; \Gamma' \vdash P' \rhd \Delta'_0 \tag{E.58}$$

Since the assertion transition is deterministic and by Lemma E.7 we know:

$$\Delta'_0 \ni \Delta'. \tag{E.59}$$

By (E.59) and (E.58) we can use [Conseq] to reach

$$\mathsf{true}; \Gamma' \vdash P' \rhd \Delta' \tag{E.60}$$

as required. This exhausts all cases. □

E.6 Proof of Lemma 7.9 (Subject Reduction)

τ-action (1): Refinement and Message Assertions We extend $\Rightarrow$ to message assertions as follows, again taking recursive assertions up to their unfolding. The following is Definition 7.4 except for the last four clauses which are about message assertions.

Definition E.15 (Refinement for Message Assertions). A binary relation $\mathcal{R}$ over closed end-point assertions is a *refinement relation* if $\mathcal{T}_1 \mathcal{R} \mathcal{T}_2$ implies one of the four conditions in Definition 7.4 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 $\mathcal{T}_1' \mathcal{R} \mathcal{T}_2'[\tilde{\mathtt{n}}/\tilde{v}]$ again
- $\mathcal{T}_1 = k \oplus l_j; \mathcal{T}'$ 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, as before, we say $\mathcal{T}_1$ is a *refinement* of $\mathcal{T}_2$, denoted $\mathcal{T}_1 \Rightarrow \mathcal{T}_2$.

Assertion assignments used for refinements now include non-singleton assignments (i.e. $\tilde{s}$ may be assigned more than two endpoint assertions for different participants): in spite of this, the non-trivial refinement of endpoint assertions is only applied to singleton assignments [12], as made explicit in the following.

Definition E.16 (refinement on extended assertion assignments). We define $\Delta \Rightarrow \Delta'$ where Δ and Δ' may possibly contain non-singleton assertions as follows:

$$\begin{array}{rcll} \Delta & \Rightarrow & \Delta & \\ \tilde{s} : \mathcal{T}@p & \Rightarrow & \tilde{s} : \mathcal{T}'@p & (\mathcal{T} \Rightarrow \mathcal{T}') \\ \Delta_1, \Delta_2 & \Rightarrow & \Delta_1', \Delta_2' & (\Delta_i \Rightarrow \Delta_i', i = 1, 2) \end{array}$$

We say Δ *refines* Δ' if $\Delta \Rightarrow \Delta'$.

We also define transitions involving message assertions. In particular, a non-singleton assignment — where we have two or more endpoint assertions with compensating channels for a single session — may have

[12] This restriction is not essential but is natural from a semantic viewpoint and enables a cleaner technical development.

a transition which represents a reduction at a free session channel. For conceptual clarity and technical convenience, we add the following new action for this transition.

$$\alpha \quad ::= \quad ..... \quad | \quad \tau_{\mathtt{free}}$$

$\tau\langle\tilde{s}\rangle$ says that a reduction of assertions as we have defined has taken place at a free session channel (we do not need to mention a specific channel). Such a transition can be non-deterministic (i.e. can have more than one derivatives for a single transition starting from the same source).

The transition rules which involve message assertions, both visible ones and invisible ones, are given in Figure 18. We observe:

1. The first two rules for visible transitions, [TR-M-Send] and [TR-M-Sel], are straightforward. These transitions are defined only over singleton assertions, just as in Figure 11 (page 39). Thus they are never induced at say $\tilde{s}$ when have more than 1 endpoint assertions assigned to $\tilde{s}$.
2. Rule [TR-Tau-Send] uses the $\tau_{\mathtt{free}}$-action label. "$\mathcal{H}$ non-trivial" says that this rule is applied only when we have more than one assertions under $\tilde{s}$. The rule corresponds to the first reduction rule in Figure 16.
3. Rule [TR-Tau-Select] is similar. The rule corresponds to the second reduction rule in Figure 16.
4. Rule [TR-Tau-Val] is for interaction, corresponding to the third reduction rule in Figure 16.
5. Rule [TR-Tau-Bra] corresponds to the third reduction rule in Figure 16.

Accordingly we also use the $\tau_{\mathtt{free}}$-actions for processes, by dividing the rule [Tau] in Figure 15, presented in Figure 19. The rules simply divide the reduction into two cases, depending on whether it is at a free session channel or othersise.

These two silent transitions are consistent with those in Figure 15 since program phrases and their derivatives never have reduction at free session channels: thus τ is used for the universal cases, while $\tau_{\mathtt{free}}$ is only used when session channels are not yet hidden. [13]

Proposition E.17 (extended transitions).

[13] For both assertions and processes, we can merge this $\tau_{\mathtt{free}}$-transition and the τ-action and can still establish all the main technical results, with no essential change in arguments. The purpose of using this action is for conceptual clarity, so that the τ-transition continues to denote the (assertion-wise) deterministic transition while incorporating the silent action at free session channels (which no derivative of program phrases has but is needed to analyse its behaviour).

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

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

$$\frac{A[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{true}}{\tilde{s}:\mathcal{H}[k!(\tilde{v} : \tilde{S})\{A\}; \mathcal{T}] \overset{\tau_{\mathtt{free}}}{\to} \tilde{s}:\mathcal{H}[k!\langle \tilde{\mathtt{n}}\rangle; \mathcal{T}[\tilde{\mathtt{n}}/\tilde{v}]]} \quad \text{[TR-Tau-Send]}$$

$$\frac{A_j \downarrow \mathsf{true} \quad j \in I}{\tilde{s}:\mathcal{H}[k \oplus \{\{A_i\} l_i : \mathcal{T}_i\}_{i\in I}] \overset{\tau_{\mathtt{free}}}{\to} \tilde{s}:\mathcal{H}[k \oplus l_j; \mathcal{T}_j]} \quad \text{[TR-Tau-Sel]}$$

$$\frac{A[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{true}}{\tilde{s}:\mathcal{H}[k!\langle \tilde{\mathtt{n}}\rangle; \mathcal{T} @ \mathtt{p},\ k?(\tilde{v})\{A\}; \mathcal{T}' @ \mathtt{q}] \overset{\tau_{\mathtt{free}}}{\to} \tilde{s}:\mathcal{H}[\mathcal{T} @ \mathtt{p}, \mathcal{T}'[\tilde{\mathtt{n}}/\tilde{v}] @ \mathtt{q}]} \quad \text{[TR-Tau-Val]}$$

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

Fig. 18. Labelled Transition for Message Assertions

1. (coincidence with reduction, 1) $\langle \Gamma, \Delta\rangle \overset{\tau_{\mathtt{free}}}{\to} \langle \Gamma, \Delta'\rangle$ *iff* $\Delta \to \Delta'$.
2. (coincidence with reduction, 2) $P \to Q$ *iff* $P \overset{\tau}{\to} Q$ *or* $P \overset{\tau_{\mathtt{free}}}{\to} Q$.
3. (determinism of non-$\tau_{\mathtt{free}}$-actions) *Suppose* $\langle \Gamma, \Delta\rangle \overset{\alpha}{\to} \langle \Gamma', \Delta'\rangle$ *such that* $\alpha \neq \tau_{\mathtt{free}}$. *Then* $\langle \Gamma, \Delta\rangle \overset{\alpha}{\to} \langle \Gamma'', \Delta''\rangle$ *implies* $\Gamma' = \Gamma''$ *and* $\Delta' = \Delta''$.

Proof. (1) is by definition. (2) is also direct from the definitions. For example $\langle \Gamma, \Delta\rangle \overset{\tau}{\to} \langle \Gamma', \Delta'\rangle$ implies $\Gamma' = \Gamma$ and $\Delta' = \Delta$, similarly for others. □

By (1) above, we can safely identify a $\tau_{\mathtt{free}}$-action from $\langle \Gamma, \Delta\rangle$ and a reduction from Δ.

Using these extended transition relations, we generalise our results in Appendix E.3. Recall below that we still use only singleton assertion assignments for visible transitions, as discussed in (1) above. The $\tau_{\mathtt{free}}$ transition takes place only when there is a non-singleton assignment.

$$\frac{P \to P' \ \text{ at a shared name or a hidden session channel}}{P \xrightarrow{\tau} P'} \qquad [\textsc{Tau}]$$

$$\frac{P \to P' \ \text{ at a free session channel}}{P \xrightarrow{\tau_{\mathtt{free}}} P'} \qquad [\textsc{TauFSC}]$$

Fig. 19. Refined τ-Transitions (replacing [Tau] in Fig. 15)

Definition E.18 (extended conditional simulation and satisfaction). We extend the notion of the conditional simulation in Definition 7.2 as follows:

1. $\mathcal{R}$ in the definition now relates P which can be a (closed) runtime process without errH or errT; and $\langle \Gamma, \Delta \rangle$) where Γ is an environment and Δ may include non-singleton assignments.
2. In (2), we include the case of $\tau_{\mathtt{free}}$.

Using this extended conditional simulation, the satisfaction relation $\Gamma \models P \rhd \Delta$ with P a runtime process and Δ containing possibly non-singleton assignments, is defined by precisely the same clauses as in Definition 7.3 (§7.1, page 40).

Proposition E.19 (extended assertion transition and refinement).

1. *The same statement as given in Lemma E.7 holds for assertion assignments with message assertions, adding the clause:*
   If $\langle \Gamma, \Delta \rangle \overset{\tau_{\mathtt{free}}}{\to} \langle \Gamma, \Delta_1 \rangle$ *then* $\langle \Gamma, \Delta' \rangle \overset{\tau_{\mathtt{free}}}{\to} \langle \Gamma, \Delta'_1 \rangle$ *or* $\langle \Gamma, \Delta' \rangle \overset{\tau_{\mathtt{free}}\tau_{\mathtt{free}}}{\to} \langle \Gamma, \Delta'_1 \rangle$ *such that* $\Delta_1 \ni \Delta'_1$ *again.*
2. *The same statement as given in Proposition E.8 holds for assertion assignments extended with message assertions.*

Remark. In (1) above, we only have to *find* one appropriate Δ'_1 which corresponds to Δ_1, due to the non-determinism, cf. Proposition E.17 (3). Further notice Δ' may need two $\tau_{\mathtt{free}}$-actions for catching up with the reduction of Δ, since Δ can have already instantiated a send/select assertion which may still be abstract in Δ' (see the proofs below).

Proof. For (1), the proof is identical to the proof of Lemma E.7 except the pairs introduced in Definition E.15. 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 E.7:

$$\Delta = \Delta_{side}, \tilde{s} : k!\langle \tilde{\mathtt{n}} \rangle; \mathcal{T}_1 \,@\, \mathtt{p} \tag{E.61}$$

$$\Delta' = \Delta'_{side}, \tilde{s} : k!(\tilde{v} : \tilde{S})\{A\}; \mathcal{T}'_1 \,@\, \mathtt{p} \tag{E.62}$$

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

$$\langle \Gamma, \Delta \rangle \stackrel{s_k!\tilde{\mathtt{n}}}{\rightarrow} \langle \Gamma, \Delta_{side}, \tilde{s} : \mathcal{T}' \,@\, \mathtt{p} \rangle \tag{E.63}$$

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

$$\langle \Gamma, \Delta' \rangle \stackrel{s_k!\tilde{\mathtt{n}}}{\rightarrow} \langle \Gamma, \Delta'_{side}, \tilde{s} : \mathcal{T}'_1 \,@\, \mathtt{p} \rangle. \tag{E.64}$$

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

$$\langle \Gamma, \Delta \rangle \stackrel{\tau_{\mathtt{free}}}{\rightarrow} \langle \Gamma, \Delta_1 \rangle \tag{E.65}$$

First assume in (E.65) that this action is induced by the reduction from

$$\tilde{s} : \mathcal{H}[k!(\tilde{v} : \tilde{S})\{A\}; \mathcal{T}] \tag{E.66}$$

in Δ to its instantiation

$$\tilde{s} : \mathcal{H}[k!\langle \tilde{\mathtt{n}} \rangle; \mathcal{T}[\tilde{\mathtt{n}}/\tilde{v}] \,@\, \mathtt{p}] \tag{E.67}$$

in Δ_1 such that

$$A[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{true}. \tag{E.68}$$

Then Δ' will have the corresponding reduction from

$$\tilde{s} : \mathcal{H}[k!(\tilde{v} : \tilde{S})\{A'\}; \mathcal{T}'] \tag{E.69}$$

in Δ', because, by the definition of refinement, we have $A \supset A'$, hence by (E.68) we obtain $A'[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{true}$ too, so that we obtain the corresponding instantiation:

$$\tilde{s} : \mathcal{H}[k!\langle \tilde{\mathtt{n}} \rangle; \mathcal{T}'[\tilde{\mathtt{n}}/\tilde{v}] \,@\, \mathtt{p}] \tag{E.70}$$

for which we have, by definition,

$$k!\langle \tilde{\mathtt{n}} \rangle; \mathcal{T}[\tilde{\mathtt{n}}/\tilde{v}] \;\Supset\; k!\langle \tilde{\mathtt{n}} \rangle; \mathcal{T}'[\tilde{\mathtt{n}}/\tilde{v}] \tag{E.71}$$

as required. On the other hand if the transition in (E.65) is induced by the following redex in Δ

$$\tilde{s} : \mathcal{H}[k!\langle \tilde{\mathtt{n}} \rangle; \mathcal{T}_a @ \mathtt{p},\ k?(\tilde{v})\{A_b\}; \mathcal{T}_b @ \mathtt{q}] \tag{E.72}$$

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

$$\tilde{s} : \mathcal{H}[\mathcal{T}_a @ \mathtt{p},\ \mathcal{T}_b @ [\tilde{\mathtt{n}}/\tilde{v}] @ \mathtt{q}] \tag{E.73}$$

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

$$\tilde{s} : \mathcal{H}[k!\langle \tilde{\mathtt{n}} \rangle; \mathcal{T}'_a @ \mathtt{p},\ k?(\tilde{v})\{A'_b\}; \mathcal{T}'_b @ \mathtt{q}] \tag{E.74}$$

such that

$$\mathcal{T}_a \supset \mathcal{T}'_a \tag{E.75}$$

$$A'_b \supset A_b \tag{E.76}$$

$$\mathcal{T}_b[\tilde{\mathtt{m}}/\tilde{v}] \ni \mathcal{T}'_b[\tilde{\mathtt{m}}/\tilde{v}] \quad (\text{if } A'_b[\tilde{\mathtt{m}}/\tilde{v}] \downarrow \mathsf{true}) \tag{E.77}$$

Thus (E.74) can have the corresponding reduction, hence $\langle \Gamma, \Delta' \rangle$ can have the corresponding $\tau_{\mathtt{free}}$-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:

$$\tilde{s} : \mathcal{H}[k!(\tilde{v} : \tilde{S})\{A'_a\}; \mathcal{T}'_a @ \mathtt{p}, k?(\tilde{v})\{A'_b\}; \mathcal{T}'_b @ \mathtt{q}] \tag{E.78}$$

such that

$$A_a \supset A'_a[\tilde{\mathtt{n}}/\tilde{v}] \tag{E.79}$$

$$\mathcal{T}_a[\tilde{\mathtt{m}}/\tilde{v}] \ni \mathcal{T}'_b[\tilde{\mathtt{m}}/\tilde{v}] \quad (\text{if } A_a[\tilde{\mathtt{m}}/\tilde{v}] \downarrow \mathsf{true}) \tag{E.80}$$

$$A'_b \supset A_b \tag{E.81}$$

$$\mathcal{T}_b[\tilde{\mathtt{m}}/\tilde{v}]\mathcal{T}'_b[\tilde{\mathtt{m}}/\tilde{v}] \quad (\text{if } A'_b[\tilde{\mathtt{m}}/\tilde{v}] \downarrow \mathsf{true}) \tag{E.82}$$

By (E.79) we know (E.83) has the reduction into:

$$\tilde{s} : \mathcal{H}[k!\langle \tilde{\mathtt{n}} \rangle; \mathcal{T}'_a[\tilde{\mathtt{m}}/\tilde{v}] @ \mathtt{p},\ k?(\tilde{v})\{A'_b\}; \mathcal{T}'_b @ \mathtt{q}] \tag{E.83}$$

We further use (E.81) to obtain the reduction from (E.83) into:

$$\tilde{s} : \mathcal{H}[\mathcal{T}'_a[\tilde{\mathtt{m}}/\tilde{v}] @ \mathtt{p},\ \mathcal{T}'_b @ [\tilde{\mathtt{n}}/\tilde{v}] @ \mathtt{q}] \tag{E.84}$$

as required. □

Corollary E.20 (Assertion Reduction and Coherence). *If Δ is coherent and $\Delta \to \Delta'$ or equivalently $\langle \Gamma, \Delta \rangle \overset{\tau_{\mathsf{free}}}{\to} \langle \Gamma, \Delta' \rangle$, 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:

$$\tilde{s} : \mathcal{H}[k!(\tilde{v} : \tilde{S})\{A\}; \mathcal{T}@\mathtt{p},] \tag{E.85}$$

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

$$k!(\tilde{v} : \tilde{S})\{A\}; \mathcal{T} \Subset \mathcal{G} \upharpoonright \mathtt{p} \tag{E.86}$$

similarly for other endpoint assertions under $\tilde{s}$. Now consider we have a reduction from (E.85) by the first rule in Figure 16 into:

$$\tilde{s} : \mathcal{H}[k!\langle \tilde{\mathtt{n}} \rangle; \mathcal{T}[\tilde{\mathtt{n}}/\tilde{v}]@\mathtt{p},] \tag{E.87}$$

where we have

$$A[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{true} \tag{E.88}$$

By (E.86) and (E.88) and because $\Subset$ is transitive we obtain:

$$k!\langle \tilde{\mathtt{n}} \rangle; \mathcal{T}[\tilde{\mathtt{n}}/\tilde{v}] \Subset \mathcal{G} \upharpoonright \mathtt{p} \tag{E.89}$$

as required. For the other case, the reduction involves a pair. Assume Δ has a redex

$$\tilde{s} : \mathcal{H}[k!\langle \tilde{\mathtt{n}} \rangle \mathcal{T}_a, k?(\tilde{v})\{A_b\}; \mathcal{T}_b] \tag{E.90}$$

As before, by coherence we can set:

$$k!\langle \tilde{\mathtt{n}} \rangle; \mathcal{T}_a \Subset \mathcal{G} \upharpoonright \mathtt{p} \tag{E.91}$$

$$k?(\tilde{v})\{A_b\}; \mathcal{T}_b \Subset \mathcal{G} \upharpoonright \mathtt{q} \tag{E.92}$$

Note we can safely assume $\mathcal{G}$ has the shape (up to permutation of utterly unordered actions):

$$\mathcal{G} = \mathtt{p} \to \mathtt{q} : (\tilde{v} : \tilde{S})\{A'\}.\mathcal{G}' \tag{E.93}$$

hence we can assume:

$$\mathcal{G} \upharpoonright \mathtt{p} = k!(\tilde{v} : \tilde{S})\{A'\}; (\mathcal{G}' \upharpoonright \mathtt{p}) \tag{E.94}$$

$$\mathcal{G} \upharpoonright \mathtt{q} = k?(\tilde{v})\{A'\}; (\mathcal{G}' \upharpoonright \mathtt{q}) \tag{E.95}$$

such that, by Definition E.15, $A' \supset A_b$ and $A'[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{true}$ and hence

$$\mathcal{T}_a \ \ni \ \mathcal{G}' \upharpoonright \mathtt{p}[\tilde{\mathtt{n}}/\tilde{v}] \tag{E.96}$$

$$\mathcal{G}' \upharpoonright \mathtt{q}[\tilde{\mathtt{n}}/\tilde{v}] \ \supset \ \mathcal{T}_b[\tilde{\mathtt{n}}/\tilde{v}] \tag{E.97}$$

Now consider the reduction from (E.90) into:

$$\tilde{s} : \mathcal{H}[\mathcal{T}_a, \mathcal{T}_b[\tilde{\mathtt{n}}/\tilde{v}]] \tag{E.98}$$

By (E.96) and (E.97) we obtain

$$\mathcal{T}_a \ \ni \ \mathcal{G}'[\tilde{\mathtt{n}}/\tilde{v}] \upharpoonright \mathtt{p} \tag{E.99}$$

$$\mathcal{T}_b[\tilde{\mathtt{n}}/\tilde{v}] \ \ni \ \mathcal{G}'[\tilde{\mathtt{n}}/\tilde{v}] \upharpoonright \mathtt{q} \tag{E.100}$$

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:

$$\mathcal{G}'[\tilde{\mathtt{n}}/\tilde{v}] \upharpoonright \mathtt{r} \ = \ \mathcal{G} \upharpoonright \mathtt{r} \tag{E.101}$$

hence as required. □

We shall also use the following result later.

Lemma E.21. *Suppose $\mathcal{C}; \Gamma \vdash P|Q \rhd \Delta$ is derived. Then there is always a derivation with the same conclusion and with the same or lesser length than the original derivation such that the last rule applied is [Conc]. Similarly for the remaining syntactic shapes.*

Proof. By the shape of the validation rules, the last rules applied to derive this judgement can only be the application of [Conc] followed by zero or more [Conseq]. However since $\ni$ is only applied point-wise, and this does not affect the composability by $\circ$ (which is based on linear compatibility), we can first apply the same refinement point-wise then finally apply [Conc], to obtain exactly the same final conclusion. The same reasoning holds for other rules. □

τ-action (2): Key Lemmas The current definition of τ-action as well as $\tau_{\mathtt{free}}$-action is not based on compatible visible actions but is defined from reduction. The following lemma shows that, in spite of this, the $\tau/\tau_{\mathtt{free}}$-action is indeed derivable from complementary visible actions except for initiation and conditionals). Let $C[\ ]$ denote a reduction context.

Lemma E.22. *If $P \to P'$ then one of the following cases hold:*

1. $P \equiv C[\mathsf{if}\ e\ \mathsf{then}\ Q_1\ \mathsf{else}\ Q_2]$ *such that* $P' \equiv C[Q_1]$ *(if* $e \downarrow \mathsf{true}$*) or* $P' \equiv C[Q_2]$ *(if* $e \downarrow \mathsf{false}$*).*
2. $P \equiv C[P_1|..|P_n]$ *such that* $P_1 \overset{a[2..n](\tilde{s})}{\rightarrow} P_1'$ *and* $P_i \overset{a[i](\tilde{s})}{\rightarrow} P_i'$ *for* $2 \leqslant i \leqslant n$*, with* $P' \equiv C[(\boldsymbol{\nu}\tilde{s})(P_1'|..|P_n')]$.
3. $P \equiv C[Q|s:\tilde{h}]$ *such that* $Q \overset{s!\tilde{\mathfrak{n}}}{\rightarrow} Q'$ *and* $P' \equiv C[Q'|s:\tilde{h}\cdot\tilde{\mathfrak{n}}]$.
4. $P \equiv C[Q|s:\tilde{h}\cdot\tilde{\mathfrak{n}}]$ *such that* $Q \overset{k?\tilde{\mathfrak{n}}}{\rightarrow} Q'$ *and* $P' \equiv C[Q'|s:\tilde{h}]$.
5. $P \equiv C[Q|s:\tilde{h}]$ *such that* $Q \overset{s\triangleleft l}{\rightarrow} Q'$ *and* $Q' \equiv C[Q'|s:\tilde{h}\cdot l]$.
6. $P \equiv C[Q|s:\tilde{h}\cdot l]$ *such that* $Q \overset{l\triangleright l}{\rightarrow} Q'$ *and* $P' \equiv C[Q'|s:\tilde{h}]$.

Proof. Immediate from the corresponding reduction rules. □

By Lemma E.22 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.

Lemma E.23. *Assume below all transitions are typed under the implicit typing.*

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

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

Remark. In the third clause above, we do not include the case of bound outputs since we do not need them in Lemma E.22 (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)). For (3), suppose we have

$$\Gamma \vdash P|s:\tilde{h} \rhd \Delta \tag{E.102}$$

By Lemma E.21, we safely assume the last rule applied is [Conc]. Thus we can assume, for some Δ_0 and Δ_1:

$$\Gamma \vdash P \ \rhd \ \Delta_0 \tag{E.103}$$

$$\Gamma \vdash s:\tilde{h} \ \rhd \ \Delta_1 \tag{E.104}$$

$$\Delta_0 \circ \Delta_1 = \Delta \tag{E.105}$$

Now consider the transition

$$P \xrightarrow{s!\tilde{\mathtt{n}}} P' \tag{E.106}$$

By (E.103) we observe Δ_0 has the shape, with $s = s_k$:

$$\Delta_0 = \tilde{s}:\mathcal{H}[k!(e)\{A\};\mathcal{T}@\mathtt{p}], \Delta_{00} \tag{E.107}$$

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

$$\Delta_0' = \tilde{s}:\mathcal{H}[\mathcal{T}[\tilde{\mathtt{n}}/\tilde{v}]], \Delta_{00} \tag{E.108}$$

Now the assertion Δ_1 for the queue has the shape, omitting the vacuous "end":

$$\Delta_1 = \tilde{s}:\mathcal{H}[\mathcal{M}@\mathtt{p}] \tag{E.109}$$

hence the addition of the values to this queue, $s:\tilde{h}\cdot\tilde{\mathtt{n}}$, must have the endpoint assertion:

$$\Delta_1' = \tilde{s}:\mathcal{H}[k!\langle\tilde{\mathtt{n}}\rangle;\mathcal{M}@\mathtt{p}] \tag{E.110}$$

Setting $\Delta' = \Delta_0' \circ \Delta_1'$, we know:

$$\Gamma \ \vdash \ P'|s:\tilde{h}\cdot\tilde{\mathtt{n}} \ \rhd \ \Delta' \tag{E.111}$$

By (E.108) and (E.110) and the type composition $\circ$ and the type reduction $\rightarrow$, we obtain

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

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

For (3), suppose we have

$$\Gamma \vdash P | s : \tilde{h} \cdot \tilde{\mathtt{n}} \rhd \Delta \tag{E.112}$$

Again by Lemma E.21, we safely assume the last rule applied is [Conc]. Thus we can assume, for some Δ_0 and Δ_1:

$$\Gamma \vdash P \ \rhd \ \Delta_0 \tag{E.113}$$

$$\Gamma \vdash s : \tilde{h} \cdot \tilde{\mathtt{n}} \ \rhd \ \Delta_1 \tag{E.114}$$

$$\Delta_0 \circ \Delta_1 = \Delta \tag{E.115}$$

Now consider the transition

$$P \stackrel{s?\tilde{\mathtt{n}}}{\rightarrow} P' \tag{E.116}$$

As before, we can infer, from (E.113) and (E.116) the shape of Δ_0 as follows, with $s = s_k$:

$$\Delta_0 = \tilde{s} : \mathcal{H}[k?(\tilde{v})\{A\}; \mathcal{T}@\mathtt{p}], \Delta_{00} \tag{E.117}$$

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

$$\Delta_0' = \tilde{s} : \mathcal{H}[\mathcal{T}[\tilde{\mathtt{n}}/\tilde{v}]], \Delta_{00} \tag{E.118}$$

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

$$\Delta_1 = \tilde{s} : \mathcal{H}[k!\langle\tilde{\mathtt{n}}\rangle; \mathcal{M}@\mathtt{p}] \tag{E.119}$$

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

$$\Delta_1' = \tilde{s} : \mathcal{H}[\mathcal{M}@\mathtt{p}] \tag{E.120}$$

Note this is symmetric to the case (1) above. As before, setting $\Delta' = \Delta_0' \circ \Delta_1'$, we know:

$$\Gamma \ \vdash \ P' | s : \tilde{h} \ \rhd \ \Delta' \tag{E.121}$$

By (E.118) and (E.120) and the type composition $\circ$ and the type reduction $\rightarrow$, we obtain

$$\begin{aligned} \Delta_0' \circ \Delta_1' \ = \ & \tilde{s} : \mathcal{H}[\mathcal{T}[\tilde{\mathtt{n}}/\tilde{v}]], \ \Delta_{00}, \ \Delta_1 \\ & \leftarrow \Delta_0, \Delta_1 \end{aligned}$$

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

τ-action (2): Subject Reduction We now establish Lemma E.24, the subject reduction.

Lemma E.24 (Subject Reduction). *Suppose $\Gamma \vdash P \rhd \Delta$ and $P \to P'$. Then $\Gamma \vdash P' \rhd \Delta'$ such that either $\Delta' = \Delta$ or $\Delta \to \Delta'$*

We prove a stronger statement. Below we use the refined τ-transition rules in Figure 19.

Lemma E.25. *Suppose $\Gamma \vdash P \rhd \Delta$.*

1. *If $P \xrightarrow{\tau} P'$ by Figure 19 then $\Gamma \vdash P' \rhd \Delta$ again.*
2. *If $P \xrightarrow{\tau_{\mathtt{free}}} P'$ by Figure 19 then $\Gamma \vdash P' \rhd \Delta'$ such that $\Delta \to \Delta'$.*

Remark. Via Proposition E.17, Lemma E.25 above entails Lemma E.24.

Proof. Assume

$$\mathsf{true}; \Gamma_0 \vdash P \rhd \Delta_0 \quad \text{and} \quad P \xrightarrow{\tau} P'. \tag{E.122}$$

Each of the six cases in Lemmas E.22 are possible, which we inspect one by one. Below let $C[\ \cdot\ ]$ is an appropriate reduction context.

1. Conditional. By Lemmas E.22 (1) assume $P = C[R]$ where

$$R = \mathsf{if}\ e\ \mathsf{then}\ Q_1\ \mathsf{else}\ Q_2 \tag{E.123}$$

such that if $e \downarrow \mathsf{true}$ then $P' = Q_1$. Since $C[\ \cdot\ ]$ is a reduction context we know R is closed. Therefore we can safely set:

$$\mathsf{true}; \Gamma \vdash R \rhd \Delta \tag{E.124}$$

By Lemma E.21 we can assume R is inferred by [If] of Figure 10. Hence we have

$$\mathsf{true} \wedge e; \Gamma \vdash Q_1 \rhd \Delta \tag{E.125}$$

By [Conseq] we get

$$\mathsf{true}; \Gamma \vdash Q_1 \rhd \Delta \tag{E.126}$$

as required. Dually for the case of $e \downarrow \mathsf{false}$.

2. Link. By Lemmas E.22 (2) we set $P = C[R]$ where

$$R = P_1|...|P_n \tag{E.127}$$

with their initialization actions compensating each other, as given in Lemmas E.22 (2), i.e.

$$P_1 \overset{\overline{a}[2..n](\tilde{s})}{\rightarrow} P_1' \tag{E.128}$$

$$P_i \overset{a[i](\tilde{s})}{\rightarrow} P_i' \quad (2 \leqslant i \leqslant n) \tag{E.129}$$

As before, we can safely set:

$$\mathsf{true}; \Gamma \vdash R \rhd \Delta \tag{E.130}$$

By Lemma E.21 we can assume R is inferred by [If] of Figure 10 by consecutive applications of [Conc], hence we safely assume:

$$\mathsf{true}; \Gamma \vdash P_i \rhd \Delta_i \tag{E.131}$$

such that $\Delta_1 \circ .. \circ \Delta_n = \Delta$. By Lemma E.23 (1) and (2), we have, with $\Gamma(a) = \mathcal{G}$,

$$\mathsf{true}; \Gamma \vdash P_i' \rhd \Delta_i, \tilde{s} : (\mathcal{G} \upharpoonright i)@i \tag{E.132}$$

Hence

$$\mathsf{true}; \Gamma \vdash P_1|..|P_n \rhd \Delta, \tilde{s} : \{(\mathcal{G} \upharpoonright i)@i\}_{1 \leqslant i \leqslant n} \tag{E.133}$$

Since $\{(\mathcal{G} \upharpoonright i)@i\}_{1 \leqslant i \leqslant n}$ is obviously coherent, we have

$$\mathsf{true}; \Gamma \vdash (\boldsymbol{\nu} \tilde{s})(P_1|..|P_n) \rhd \Delta \tag{E.134}$$

as required.

3. Send. By Lemmas E.22 (3) we set

$$P = [Q|s:\tilde{h}] \tag{E.135}$$

with

$$Q \overset{s!\tilde{\mathfrak{n}}}{\rightarrow} Q' \tag{E.136}$$

As above we can safely set

$$\mathsf{true}; \Gamma \vdash Q|s:\tilde{h} \rhd \Delta \tag{E.137}$$

By Lemmas E.23 (3), (E.136) and (E.137), we infer:

$$\mathsf{true}; \Gamma \vdash Q'|s:\tilde{h} \cdot \tilde{\mathfrak{n}} \rhd \Delta' \tag{E.138}$$

such that $\Delta \to \Delta'$ where the only change is at $\tilde{s}$ which contains s. Since P reduces to P' by τ-transition rather than $\tau_{\mathtt{free}}$-transition, by Figure 19, we know that this $\tilde{s}$ in R are hidden in P. Assume therefore, without loss of generality:

$$P \equiv C'[(\boldsymbol{\nu}\tilde{s})(Q|s:\tilde{h}|R)] \tag{E.139}$$

$$\Gamma_1 \vdash Q|s:\tilde{h}|R \rhd \Delta_1 \tag{E.140}$$

$$\Delta' \text{ coherent and} \Delta_1 = \Delta \circ \Delta_{01} \tag{E.141}$$

By Lemma E.20 and noting $\Delta_1 \to \Delta' \circ \Delta_{01}$ we know $\Delta_1 = \Delta' \circ \Delta_{01}$ is also coherent, hence as required.

4. Receive. By Lemmas E.22 (4) we set

$$P = C[Q|s:\tilde{h} \cdot \mathtt{n}] \tag{E.142}$$

with

$$Q \overset{s?\mathtt{n}}{\to} Q' \tag{E.143}$$

As above we can safely set

$$\mathsf{true}; \Gamma \vdash Q|s:\tilde{h} \cdot \mathtt{n} \rhd \Delta \tag{E.144}$$

As before, by Lemmas E.23 (4), (E.143) and (E.144), we get:

$$\mathsf{true}; \Gamma \vdash Q'|s:\tilde{h} \rhd \Delta' \tag{E.145}$$

such that $\Delta \to \Delta'$ where the only change is at $\tilde{s}$ which contains s. Again by Figure 19, we can set, without loss of generality:

$$P \equiv C'[(\boldsymbol{\nu}\tilde{s})(Q|s:\tilde{h}|R)] \tag{E.146}$$

$$\Gamma_1 \vdash Q|s:\tilde{h}|R \rhd \Delta_1 \tag{E.147}$$

$$\Delta' \text{ coherent and} \Delta_1 = \Delta \circ \Delta_{01} \tag{E.148}$$

As before, by Lemma E.20 and noting $\Delta_1 \to \Delta' \circ \Delta_{01}$ we know $\Delta_1 = \Delta' \circ \Delta_{01}$ is also coherent, hence done.

5. Select. The argument exactly follows Case ***3.Send*** above using Lemmas E.22 (5) and Lemmas E.23 (5) instead of Lemmas E.22 (3) and Lemmas E.23 (3), respectively.

6. Branch. The argument exactly follows Case ***4.Receive*** above except using Lemmas E.22 (6) and Lemmas E.23 (6) instead of Lemmas E.22 (4) and Lemmas E.23 (4), respectively.

Finally the $\tau_{\mathtt{free}}$-reduction,

$$\mathsf{true}; \Gamma_0 \vdash P \rhd \Delta_0 \quad \text{and} \quad P \stackrel{\tau_{\mathtt{free}}}{\Longrightarrow} P'. \tag{E.149}$$

rather than (E.122), precisely follow the same reasoning as given in the cases of ***3***..6 above, excepting we do not have to hide $\tilde{s}$. □

E.7 Proof of Theorem 7.10 (Soundness)

We prove Theorem 7.10. We first define the standard notion of closing substitutions adapted to the present context, which include different kinds of variables. Below recall a process is *open* if it contains zero or more free variables and/or free process variables.

Definition E.26 (Closure of P). Let P be an open process and $\mathcal{C}; \Gamma \vdash P \rhd \Delta$. The *closure of P by substitution σ*, or simply the *closure of P*, denoted P_σ, is the process obtained by applying to P the substitution σ such that:

- for each free interaction variable declared in Γ, σ instantiate it into a value that satisfy Γ and $\mathcal{C}$ (i.e., it has the correct type and does not violate the assertion environment), and
- for each process variable X s.t. $\Gamma(X) = (\tilde{v} : \tilde{S})\tilde{T}$, the substitution σ instantiates X into a parametrised process $P(\tilde{v}\tilde{s})$ (so that each call $X\langle\tilde{e}\tilde{s}\rangle$ is instantiated into $P(\tilde{e}\tilde{s})$), such that we have $\mathcal{C}, \Gamma, \tilde{v} : \tilde{S} \models P(\tilde{v}\tilde{s}) \rhd \tilde{s}_1 : \tilde{T}_1 @ \mathtt{p}_1 ... \tilde{s}_n : \tilde{T}_n @ \mathtt{p}_n$.

Remark. Note that, in the last clause above, we use the satisfaction $\models$ when considering substitutions for process variables, cf. [4].

Definition E.27 (Closure or Δ). The closure of Δ, which we denote by Δ_σ, is the asserted local type obtained by applying the substitution σ for free interaction/process variables in Δ.

Definition E.28 (Closure or Γ). The *closure of Γ*, denoted Γ_σ, is obtained by removing from Γ all the variables substituted by σ.

The soundness proof relies on basic properties of the proof system for validation in Figures 10 and 17.

Lemma E.29. *Suppose say $\mathcal{C}; \Gamma \vdash S \rhd \Delta$ is derived and, in its derivation, $\mathcal{C}_0; \Gamma_0 \vdash S_0 \rhd \Delta_0$ is used, hence S_0 occurs in S. Suppose $\mathcal{C}_0; \Gamma_0 \vdash S_0' \rhd \Delta_0$ where S_0' and S_0 have the identical typing. Then we can replace the occurrence of S_0 by S_0', with the result written S', such that $\mathcal{C}; \Gamma \vdash S' \rhd \Delta$ is derivable.*

Proof. By noting that, in each step of derivation, the only thing that matters is the assertion environment, the assertion assignment, and the assertion context, in addition to the typing. □

Lemma E.30 (postponement of [Def]). *Suppose $\Gamma \vdash P \rhd \Delta$ is derived. Then there is a derivation tree of the same or less length such that there are no applications of [Def] (cf. Figure 10) except at the end of the derivation.*

Proof. The premise of [Def] for Q (the main process) reads:

$$\mathcal{C}; \Gamma, X : (\tilde{v} : \tilde{S})\mathcal{T}_1 @ \mathtt{p1} \dots \mathcal{T}_n @ \mathtt{p_n} \vdash Q \rhd \Delta \tag{E.150}$$

Note the only condition it demands is the assumption contains

$$X : (\tilde{v} : \tilde{S})\mathcal{T}_1 @ \mathtt{p1} \dots \mathcal{T}_n @ \mathtt{p_n} \tag{E.151}$$

and the only effect of the application of [Def] is we lose this assumption. Since no other rules use this assumption, by Lemma E.29, we can always permute an application of any rule with [Def] to obtain the same conclusion. □

Definition E.31. We say $\mathcal{C}; \Gamma \vdash P \rhd \Delta$ is *well-initiated* if Δ contains only singleton assignments and, moreover, P has no queue at a free session channel.

Lemma E.32. *The second premise of [Def] is well-initiated if and only if its conclusion is well-initiated.*

Proof. Because the assertion assignment (Δ) does not change and the process (Q) does not change except adding the new definition. □

We now prove Theorem 7.10, whose statement is reproduced below.

Theorem E.33 (Soundeness for Open Processes).
Let P be a program phrase. Then $\mathcal{C}; \Gamma \vdash P \rhd \Delta$ implies $\mathcal{C}; \Gamma \models P \rhd \Delta$.

Proof. Let $\mathcal{R}$ be the relation collection all the pairs of the form:

$$(P_\sigma, \langle \Gamma_\sigma, \Delta_\sigma \rangle) \tag{E.152}$$

such that $\mathcal{C}; \Gamma \vdash P \rhd \Delta$ with P being a sub-term of a multi-step $\xrightarrow{\alpha}$-derivative of a program phrase and Δ an assertion assignment possibly containing non-singleton assignments, and σ being its closing substitution. We show $\mathcal{R}$ is a conditional simulation (in the extended sense defined in Definition 7.2), by rule induction on the validation rules. Assume

$$\mathcal{C}; \Gamma \vdash P \rhd \Delta \tag{E.153}$$

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 [SEND]. By [SENDR] in Figure 10, we can set

$$P = s_k!\langle \tilde{e} \rangle(\tilde{v})\{A\}; P' \tag{E.154}$$

where $\mathcal{C}; \Gamma \vdash P \rhd \Delta$ such that

$$\Delta = \Delta', \tilde{s} : k!(\tilde{v} : \tilde{S})\{A\}; \mathcal{T} @ \mathtt{p}. \tag{E.155}$$

By the definition of closure

$$P_\sigma = s_k!\langle \tilde{e_\sigma} \rangle(\tilde{v})\{A_\sigma\}; P'_\sigma \tag{E.156}$$

where $\tilde{e}_\sigma \downarrow \tilde{\mathtt{n}}$. Process P_σ can only move because of rule [SEND] (Figure 15) with label is $s_k!\tilde{\mathtt{n}}$. Since $A[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{true}$ by [SEND] and $A[\tilde{\mathtt{n}}/\tilde{v}] \supset A_\sigma[\tilde{\mathtt{n}}/\tilde{v}]$,

$$s_k!\langle \tilde{e_\sigma} \rangle(\tilde{v})\{A_\sigma\}; P'_\sigma \xrightarrow{s_k!\tilde{\mathtt{n}}} P'_\sigma. \tag{E.157}$$

It follows that

$$\Delta_\sigma = \Delta'_\sigma, \tilde{s} : k!(\tilde{v} : \tilde{S})\{A_\sigma\}; \mathcal{T}_\sigma @ \mathtt{p}. \tag{E.158}$$

By [TR-SEND] in Figure 11, since $A_\sigma[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{true}$,

$$\langle \Gamma, \Delta'_\sigma, \tilde{s} : k!(\tilde{v} : \tilde{S})\{A_\sigma\}; \mathcal{T}_\sigma @ \mathtt{p} \rangle \xrightarrow{s_k!\tilde{\mathtt{n}}} \langle \Gamma, \Delta'_\sigma, \tilde{s} : \mathcal{T}_\sigma @ \mathtt{p} \rangle, \tag{E.159}$$

It follows by induction,

$$(P'_\sigma, \langle \Gamma, \Delta'_\sigma, \tilde{s} : \mathcal{T}_\sigma @ \mathtt{p} \rangle) \in R. \tag{E.160}$$

Case [Rcv]. By [Rcv] in Figure 10, and setting $P = s_k?(\tilde{v})\{A\}; P'$, we have

$$\mathcal{C}; \Gamma \vdash P \rhd \Delta \tag{E.161}$$

with $\Delta = \Delta', \tilde{s} : k?(\tilde{v} : \tilde{S})\{A\}; \mathcal{T} \,@\, \mathtt{p}$. Let $P_\sigma = s_k?(\tilde{v})\{A_\sigma\}; P'_\sigma$. It follows that

$$\Delta_\sigma = \Delta'_\sigma, \tilde{s} : k?(\tilde{v} : \tilde{S})\{A_\sigma\}; \mathcal{T}_\sigma \,@\, \mathtt{p}. \tag{E.162}$$

Process P_σ can only move because of rule [Rcv] in Figure 15 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 15,

$$s_k?(\tilde{v})\{A_\sigma\}; P'_\sigma \xrightarrow{s_k!\tilde{\mathtt{n}}} P'_\sigma \tag{E.163}$$

and by [TR-Rec] in Figure 11,

$$\langle \Gamma, \Delta'_\sigma, \tilde{s} : k?(\tilde{v} : \tilde{S})\{A_\sigma\}; \mathcal{T}_\sigma \,@\, \mathtt{p} \rangle \xrightarrow{s_k?\tilde{\mathtt{n}}} \langle \Gamma, \Delta'_\sigma, \tilde{s} : \mathcal{T}_\sigma \,@\, \mathtt{p} \rangle. \tag{E.164}$$

It follows by induction

$$(P'_\sigma, \langle \Gamma_1, \Delta_\sigma, \tilde{s} : \mathcal{T}_\sigma \,@\, \mathtt{p} \rangle) \in R. \tag{E.165}$$

Case [Sel]. Let $P = s_k \lhd \{A_j\} l_j; P_j$. By [Sel] in Figure 10,

$$\mathcal{C}; \Gamma \vdash s_k \lhd \{A_j\} l_j; P_j \rhd \Delta \tag{E.166}$$

with $\Delta = \Delta', \tilde{s} : k \oplus \{\{A_i\} l_i : \mathcal{T}_i\}_{i \in I} \,@\, \mathtt{p}$ and $j \in I$. It follows $\Delta_\sigma = \Delta'_\sigma, \tilde{s} : k \oplus \{\{A_{i\sigma}\} l_i : \mathcal{T}_{i\sigma}\}_{i \in I} \,@\, \mathtt{p}$. We have $P_\sigma = s_k \lhd \{A_{j\sigma}\} l_j; P_{j\sigma}$ Process P_σ can only move because of rule [Label] in Figure 15 with label $s_k < l_j$ and, since $A_j \downarrow \mathsf{true}$ by well formedness of P and $A_j \supset A_{j\sigma}$ then

$$s_k \lhd \{A_{j\sigma}\} l_j; P_{j\sigma} \xrightarrow{s_k < l_j} P_{j\sigma}. \tag{E.167}$$

By [TR-Sel] in Figure 11, since $A_j \downarrow \mathsf{true}$

$$\langle \Gamma, \Delta'_\sigma, \tilde{s} : k \oplus \{\{A_{i\sigma}\} l_i : \mathcal{T}_{i\sigma}\}_{i \in I} \,@\, \mathtt{p} \rangle \xrightarrow{s_k < l_j} \langle \Gamma, \Delta'_\sigma, \tilde{s} : \mathcal{T}_{j\sigma} \,@\, \mathtt{p} \rangle. \tag{E.168}$$

By induction,

$$(P_{j\sigma}, \langle \Gamma, \Delta'_\sigma, \tilde{s} : \mathcal{T}_{j\sigma} \,@\, \mathtt{p} \rangle) \in R. \tag{E.169}$$

Case [Branch]. We can set $P = s_k \rhd \{\{A_i\}l_i\colon P_i\}_{i\in I}$ then $P_\sigma = s_k \rhd \{\{A_{i\sigma}\}l_i\colon P_{i\sigma}\}_{i\in I}$.

By [Branch] in Figure 10,

$$\mathcal{C}; \Gamma \vdash \ s_k \rhd \{\{A_i\}l_i\colon P_i\}_{i\in I} \rhd \Delta \tag{E.170}$$

with $\Delta = \Delta', \tilde{s}\colon k\&\{\{A_i\}l_i : \mathcal{T}_i\}_{i\in I}\,@\,\mathtt{p}$. It follows $\Delta_\sigma = \Delta'_\sigma, \tilde{s}\colon k\&\{\{A_{i\sigma}\}l_i : \mathcal{T}_{i\sigma}\}_{i\in I}\,@\,\mathtt{p}$.

Process P_σ can only move because of rule [Branch] with label $s_k > l_j$. 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 $s_k > l_j$, that is when $A_{j\sigma} \downarrow \mathsf{true}$. Assuming $A_{j\sigma} \downarrow \mathsf{true}$, by [Branch] in Figure 15:

$$s_k \lhd \{A_{j\sigma}\}l_j; P_{j\sigma} \overset{s_k > l_j}{\rightarrow} P_{j\sigma}, \tag{E.171}$$

and by [TR-Choice] in Figure 11:

$$\langle \Gamma, \Delta'_\sigma, \tilde{s}\colon k\&\{\{A_{i\sigma}\}l_i : \mathcal{T}_{i\sigma}\}_{i\in I}\,@\,\mathtt{p}\rangle \overset{k > l_j}{\rightarrow} \langle \Gamma, \Delta'_\sigma, \tilde{s} : \mathcal{T}_{j\sigma}\,@\,\mathtt{p}\rangle. \tag{E.172}$$

By induction,

$$(P_{j\sigma}, \langle \Gamma, \Delta'_\sigma, \tilde{s}\colon \mathcal{T}_{j\sigma}\,@\,\mathtt{p}\rangle) \in R. \tag{E.173}$$

Case [Mcast]. In this case $P = \overline{a}_{[2..\mathtt{n}]}\,(\tilde{s}).P'$ and we can set:

$$\Gamma \vdash \overline{a}_{[2..\mathtt{n}]}\,(\tilde{s}).P \rhd \Delta. \tag{E.174}$$

Let $P_\sigma = \overline{a}_{[2..\mathtt{n}]}\,(\tilde{s}).P'_\sigma$. Process P_σ can only move because of rule [LinkOut] in Figure 15:

$$\overline{a}_{[2..\mathtt{n}]}\,(\tilde{s}).P'_\sigma \overset{\overline{a}[2..n](\tilde{s})}{\rightarrow} P'_\sigma. \tag{E.175}$$

By [TR-LinkOut] in Figure 11,

$$\langle \Gamma, \Delta_\sigma\rangle \overset{\overline{a}[2..n](\tilde{s})}{\rightarrow} \langle \Gamma_1, \Delta_\sigma, \tilde{s} : (\mathcal{G} \upharpoonright 1)_\sigma\,@\,\mathtt{p}_1\rangle. \tag{E.176}$$

By induction hypothesis we have

$$(P'_\sigma, \langle \Gamma_1, \Delta_\sigma, \tilde{s} : (\mathcal{G} \upharpoonright 1)_\sigma\,@\,\mathtt{p}_1\rangle) \in R. \tag{E.177}$$

Case [Macc]. This case is essentially identical to the case [Mcast] above.

Case [Conc]. The cases of independent actions are direct from the induction hypothesis. If the reduction takes place by interaction, then we use Lemma E.25.

Case [IF]. With $P = \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q$ then by Definition E.26:

$$P_\sigma = \mathsf{if}\ e_\sigma\ \mathsf{then}\ Q_\sigma\ \mathsf{else}\ R_\sigma, \tag{E.178}$$

By [IF] in Figure 10, such that

$$\mathcal{C}; \Gamma \vdash P \rhd \Delta \tag{E.179}$$

$$\mathcal{C} \wedge e; \Gamma \vdash Q \rhd \Delta \tag{E.180}$$

We note $P_\sigma = \mathsf{if}\ e_\sigma\ \mathsf{then}\ Q_\sigma\ \mathsf{else}\ R_\sigma$. Process P_σ can move because of either [IFT] or [IFF] (Figure 15) with label is τ. Let us consider the case in which the transition happens by rule [IFT] (the case with [IFF] is symmetric):

$$\mathsf{if}\ e_\sigma\ \mathsf{then}\ Q_\sigma\ \mathsf{else}\ R_\sigma \xrightarrow{\tau} Q_\sigma \quad \text{with } \tilde{e}_\sigma \downarrow \mathsf{true}. \tag{E.181}$$

By induction, since $e_\sigma \downarrow \mathsf{true}$ and moreover e_σ does not have free variables.

$$(Q_\sigma, \langle \Gamma, \Delta_\sigma \rangle) \in R. \tag{E.182}$$

as required.

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

Case [NRES]. Immediate from induction hypothesis.

Case [VAR]. We can set

$$P = X\langle \tilde{e}, \tilde{s_1}, ..., \tilde{s_n} \rangle \tag{E.183}$$

with

$$\Gamma(X) = (\tilde{v} : \tilde{S})\tilde{\mathcal{T}}. \tag{E.184}$$

Then by well formedness of P, $\mathcal{C} \supset A[\tilde{e}/\tilde{v}] \downarrow \mathsf{true}$. P_σ is a process such that

$$\mathcal{C}_\sigma, \Gamma_\sigma \vdash P_\sigma[\tilde{e}/\tilde{v}] \rhd \Delta'_\sigma, \tilde{s}_1 : \mathcal{T}_{1\sigma}\, @\, \mathtt{p}_1, .., \tilde{s}_n : \mathcal{T}_{n\sigma}\, @\, \mathtt{p}_n. \tag{E.185}$$

Notice that

$$\Delta'_\sigma, \tilde{s}_1 : \mathcal{T}_{1\sigma}\, @\, \mathtt{p}_1, .., \tilde{s}_n : \mathcal{T}_{n\sigma}\, @\, \mathtt{p}_n = \Delta_\sigma \tag{E.186}$$

where Δ_σ is the closure of the asserted local type of P. The property holds straightforwardly by the cases for the other process types.

Case [NRes]. Immediate from induction hypothesis.

Case [Def]. This case is proved by the standard syntactic approximation of a recursion. By Lemma E.30, we can assume, in all derivations for processes in $\mathcal{R}$, the application of Rule [Def] 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 E.32, we know the premise and conclusion of an application of [Def] is well-initiated in the sense of Definition E.31. Note that the reductions of such processes are completely characterised by $\xrightarrow{\tau}$ (cf. Proposition E.17), and the corrospoinding τ-transition for assertions is deterministic. Assume that we have

$$\begin{array}{l}\mathcal{C}; \Gamma, X : (\tilde{v} : \tilde{S})\mathcal{T}_1 @ \mathtt{p}_1 \ldots \mathcal{T}_n @ \mathtt{p}_n, \tilde{v} : \tilde{S} \models P \\ \rhd \tilde{s_1} : \mathcal{T}_1 @ \mathtt{p}_1 \ldots \tilde{s_n} : \mathcal{T}_n @ \mathtt{p}_n\end{array} \tag{E.187}$$

Further we also assume

$$\mathcal{C}; \Gamma, X : (\tilde{v} : \tilde{S})\mathcal{T}_1 @ \mathtt{p1} \ldots \mathcal{T}_n @ \mathtt{p}_n \models Q \rhd \Delta \tag{E.188}$$

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 [27, §2], we first approximate the recursion by the following hierarchy:

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

Above P' is chosen as the process which is typed by the same typing as P and which has no visible action. [14] We also set P^ω to be the recursively defined agent itself:

$$\mathsf{def}\ X(\tilde{x}) \ = \ P \ \mathsf{in}\ P. \tag{E.189}$$

In the conclusion of [Def] we abstract the process variable X by the def 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).

[14] For example, writing $a(s).S$ for $a[2](s).S$ and choosing a and s to be fresh, let $P' \stackrel{\text{def}}{=} (\boldsymbol{\nu} a)(a(s).P)$ then $P' \approx \mathbf{0}$.

Using Lemma E.29, we first note that, for any (Γ, Δ) and $\mathcal{C}$, we have $\mathcal{C}; \Gamma \models P^0 \rhd \Delta$. Thus we apply this to (E.187) and replace X in P by $(\tilde{v}\tilde{s}_1..\tilde{s}_n)P^0$:

$$\mathcal{C}; \Gamma \models P^1 \rhd \tilde{s_1} : \mathcal{T}_1 \mathbin{@} \mathtt{p}_1 \ldots \tilde{s_n} : \mathcal{T}_n \mathbin{@} \mathtt{p}_n \tag{E.190}$$

This can again be used for (E.187) (noting the environment Γ can always be taken as widely as possible in [Var]):

$$\mathcal{C}; \Gamma \models P^2 \rhd \tilde{s_1} : \mathcal{T}_1 \mathbin{@} \mathtt{p}_1 \ldots \tilde{s_n} : \mathcal{T}_n \mathbin{@} \mathtt{p}_n \tag{E.191}$$

In this way we know

$$\mathcal{C}; \Gamma \models P^n \rhd \tilde{s_1} : \mathcal{T}_1 \mathbin{@} \mathtt{p}_1 \ldots \tilde{s_n} : \mathcal{T}_n \mathbin{@} \mathtt{p}_n \tag{E.192}$$

for an arbitrary n. By applying this to (E.188), we obtain:

$$\mathcal{C}; \Gamma \models Q^n \rhd \Delta \tag{E.193}$$

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 (Γ, Δ), 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 7.2 hold at each step, hence $\mathcal{R}$ is a conditional simulation, hence done.

Case [Conseq]. By Proposition 7.5 (Proposition E.8 in Appendix E.3, page 85).

Cases [Qnil], [Qval] and [Qsel] of Figure 17. Again these processes (queues) do not have transitions.[15]

Case [CRes]. By Lemma E.20. This exhausts all cases. □

As an immediate corollary we obtain:

Theorem E.34 (Soundeness for Programs).
Let P be a program. Then $\Gamma \vdash P \rhd \Delta$ implies $\Gamma \models P \rhd \Delta$.

[15] The behaviours of queues are taken into account as part of $\tau_{\mathtt{free}}$-actions.

E.8 Proof of Theorem 7.15 (1)-(5) (Monitoring, 2)

This section proves the Erasure Theorem 7.15. It has five clauses, from (1) to (5). Except for (5) (the erasure theorem), all results are straightforward, hence we first list the proof sketch for (1) to (4), then give the full proof of (5).

Proof of Theorem 7.15 (1, 2). These two are based on essentially the same observation. Here we discuss the reasoning for (1) which equally applies to (2). Assume P_1 has an active output at a session channel s which moves to P_2 which has an input at dual s. We assume the latter's predicate is violated, and show in fact in that case the former's predicate should be violated. This is because the sender's predicate is always the same as, or stronger than, the predicate for the corresponding receiver. Observe the consequence rule (anti-refinement) always strengthens an input predicate and weakens an output predicate: hence when elaborating the initial closed program P, such dual pair can only has actually a stronger output predicate and a weaker input predicate (note when we abstract away these actions by shared prefix, we should have exactly the same formula modulo preceding third-party formulae, which are safely made trivial since the preceding successful communications), hence done.

Proof of Theorem 7.15 (3, 4). Both are an immediate corollary of Theorem 7.11, with, in the case of (4), noting that P and $\mathsf{erase}(P)$ have exactly the same reductions except for the error reductions.

Proof of Theorem 7.15 (5). This is the major proof of this section. Below we recall $\mathsf{erase}(P)$ denotes the result of replacing all predicates in P with true.

Lemma E.35. *If P is a well-asserted process, then for any substitution σ, $\mathsf{erase}(P\sigma) = \mathsf{erase}(P)\sigma$.*

Proof. By induction on the structure of P, observing that $\mathsf{erase}(P)$ does not change variables, channels, or names of P. □

The next lemma gives a general condition of the transition derivatives of a program, which is used to form the closure (bisimulation) in the proof of Proposition E.38. Below we recall the asserted transition $\Gamma \vdash P \rhd \Delta \xrightarrow{\alpha} \Gamma' \vdash P' \rhd \Delta'$ is defined as:

$$P \xrightarrow{\alpha} P' \text{ and } \langle \Gamma, \Delta \rangle \xrightarrow{\alpha} \langle \Gamma, \Delta' \rangle.$$

Note that, in the case of the τ-action, a shared input action and a shared output action are always allowed by $\langle \Gamma, \Delta \rangle$ (as far as well-typed), and for an output/selection action, any such action by P such that $\Gamma \vdash P \triangleright \Delta$ is also always allowed by $\langle \Gamma, \Delta \rangle$.

Lemma E.36. *Let P be a program and suppose we have $\Gamma \vdash P \triangleright \Delta$ with $\Delta = \varnothing$. If we have:*

$$\Gamma \vdash P \triangleright \Delta \stackrel{\alpha_{1..}\alpha_n}{\to} \Gamma' \vdash P' \triangleright \Delta', \tag{E.194}$$

i.e. if P' is a multi-step derivative of a program P by asserted transitions, then (1) Δ' contains only singleton assignments for each session (i.e. of the form $\tilde{s} : \mathcal{T}@\mathtt{p}$); and (2) P' does not contain a queue at a free session channel.

Proof. (1) is immediate by the definition of asserted transitions. For (2), assume Q is a validated process in which no queue exists at its free session channel and assume Q has an asserted transition $Q \stackrel{l}{\to} R$. If it is τ-transition, either it is at a hidden session channel in which case there is no new free session channel arises: or it is a τ-transition at a shared channel (initialisation) in which case again we (create new queues under hidden session channels but) do not create free session channels. For visible transitions, if it receives an initialisation ([LinkIn]), then it creates free session channels but they are without queues. Similarly when it requests an initialisation ([LinkOut]), hence again R does not have a queue at a free session channel. □

We now establish the key lemma. Recall below we say $\langle \Gamma, \Delta \rangle$ *allows* α when $\langle \Gamma, \Delta \rangle \stackrel{\alpha}{\to} \langle \Gamma, \Delta' \rangle$ for some $\langle \Gamma', \Delta' \rangle$.

Lemma E.37. *Let P be a closed process such that $\Gamma \vdash P \triangleright \Delta$ and which does not own a queue at a free session channel. Then*

1. *If $P \stackrel{\alpha}{\to} P'$ does not induce an error then $\mathsf{erase}(P) \stackrel{\alpha}{\to} \mathsf{erase}(P')$.*
2. *If $\mathsf{erase}(P) \stackrel{\alpha}{\to} Q$ with α being an output, a selection, the τ-action or an action at a shared channel, then $P \stackrel{\alpha}{\to} P'$ such that $Q \equiv \mathsf{erase}(P')$.*
3. *If $\mathsf{erase}(P) \stackrel{\alpha}{\to} Q$ with α being an input or branching action and $\langle \Gamma, \Delta \rangle$ allows α, then $P \stackrel{\alpha}{\to} P'$ such that $\mathsf{erase}(P') \equiv Q$.*

Remark. Above observe that $\mathsf{erase}(P)$ allows any transition which the underlying process can do, since the interaction predicate associated with each action always trivially holds.

Proof. For (1), all transitions of P are identical with those of $\mathsf{erase}(P)$ except those inducing an assertion error, i.e. those in which the associated predicate is violated. But by $\Gamma \vdash P \rhd \Delta$, Proposition 7.8 and Lemma 7.9 means all transitions allowed by $\langle \Gamma, \Delta \rangle$ never induce errors. For substitutions induced by an input in [Recv], we use Lemma E.35.

(2) is because, in brief, by the definition of transitions of assertions in Figure 11, an output, selection, the τ-action and an action at a shared name is always allowed by $\langle \Gamma, \Delta \rangle$. Thus by Proposition 7.8 and Lemma 7.9, P also has this action, without an assertion error. The argument for (3) is similar, noting any input and branching action of $\mathsf{erase}(P)$ allowed by $\langle \Gamma, \Delta \rangle$ is again an action of P without violating the associated predicate. Below we give a detailed case analysis for (2) and (3), since they are at the core of the bisimilarity result for establishing Proposition E.38.

Let P be a closed process without queues at a free session channel and $\Gamma \vdash P \rhd \Delta$. Assume further

$$\mathsf{erase}(P) \xrightarrow{\alpha} Q$$

is obtained by a derivation tree Υ from the rules in 15. We proceed by induction on the structure of Υ.

- If Υ ends with the application of [LinkOut] (resp. [LinkIn]) then $\mathsf{erase}(P) = \overline{a}_{[2..\mathtt{n}]}(\tilde{s}).Q \xrightarrow{\overline{a}[2..n](\tilde{s})} Q$ (resp. $\mathsf{erase}(P) = a_{[\mathtt{i}]}(\tilde{s}).Q \xrightarrow{a[i](\tilde{s})} Q$) for some Q; then the thesis follows by observing that $P \xrightarrow{\overline{a}[2..n](\tilde{s})} P'$ (resp. $P \xrightarrow{a[i](\tilde{s})} P'$) for a process P' such that $Q = \mathsf{erase}(P)'$ and by Proposition 7.8.
- If Υ is the application of rule [Send] to $\mathsf{erase}(P)$ then $P = s_k!\langle \tilde{\mathtt{n}} \rangle(\tilde{v})\{A\}; Q$. Since by hypothesis $\langle \Gamma, \Delta \rangle \xrightarrow{s_k!\tilde{\mathtt{n}}} \langle \Gamma', \Delta' \rangle$ implies that $A[\tilde{\mathtt{n}}/\tilde{v}]$ holds true (by the premises of rule [TR-Send]), therefore $P \xrightarrow{s_k!\tilde{\mathtt{n}}} Q[\tilde{\mathtt{n}}/\tilde{v}]$. The proof ends by noticing that $\mathsf{erase}(Q[\tilde{\mathtt{n}}/\tilde{v}]) = \mathsf{erase}(Q)[\tilde{\mathtt{n}}/\tilde{v}]$ (by Lemma E.35) is the process reached by $\mathsf{erase}(P)$ and by Proposition 7.8.
- The cases when Υ is the application of rule [Recv], [Label], [Branch] are similar to the case for rule [Send].
- If Υ ends with the application of [Par], then P has shape $P_1 | P_2$ (hence $\mathsf{erase}(P) = \mathsf{erase}(P_1) | \mathsf{erase}(P_2)$). Since $\Gamma \vdash P \rhd \Delta$, by rule [Conc] (Figure 10), there must be Δ_1 and Δ_2 such that $\Delta_1 \asymp \Delta_2$, $\Delta = \Delta_1 \circ \Delta_2$, and $\Gamma \vdash P_i \rhd \Delta_i$ for $i = 1, 2$. Assume that

$$\Gamma \vdash \mathsf{erase}(P_1) \rhd \Delta_1 \xrightarrow{\alpha} \Gamma' \vdash Q \rhd \Delta', \tag{E.195}$$

then by inductive hypothesis we have

$$\Gamma \vdash P_1 \rhd \Delta_1 \xrightarrow{\alpha} \Gamma' \vdash P' \rhd \Delta' \tag{E.196}$$

where $\mathsf{erase}(P') = Q$, hence done. Hence

$$\langle \Gamma, \Delta_1 \circ \Delta_2 \rangle \xrightarrow{\alpha} \langle \Gamma, \Delta' \circ \Delta_2 \rangle \tag{E.197}$$

and, by rule [Par], $P_1|P_2 \xrightarrow{\alpha} P'|P_2$, which concludes the proof for this case.

– If Υ ends with the application of the rule [NRes] then P has either (i) the shape $(\boldsymbol{\nu} a : \mathcal{G})Q$ or (ii) $(\boldsymbol{\nu}\tilde{s})Q$.
We first consider the case (i). Let

$$\Gamma \vdash (\boldsymbol{\nu} a : \mathcal{G})\mathsf{erase}(Q) \rhd \Delta \xrightarrow{\alpha} \Gamma' \vdash (\boldsymbol{\nu} a : \mathcal{G})Q' \tag{E.198}$$

be the last transition in Υ. Since $\Gamma \vdash P \rhd \Delta$, $\Gamma, a : \mathcal{G} \vdash Q \rhd \Delta$ by rule [NRes] in Figure 10 and, by inductive hypothesis,

$$\Gamma, a : \mathcal{G} \vdash Q \rhd \Delta \xrightarrow{\alpha} \Gamma' \vdash Q'' \rhd \Delta' \tag{E.199}$$

with $a \notin \mathsf{fn}\alpha$ and $\mathsf{erase}(Q'') = Q'$. Hence, by rule [NRes], $P \xrightarrow{\alpha} (\boldsymbol{\nu} a : \mathcal{G})Q''$.
For the case (ii), let

$$\Gamma \vdash (\boldsymbol{\nu}\tilde{s})\mathsf{erase}(Q) \rhd \Delta \xrightarrow{\alpha} \Gamma' \vdash (\boldsymbol{\nu}\tilde{s})Q' \tag{E.200}$$

be the last transition in Υ. Since $\Gamma \vdash P \rhd \Delta$,

$$\Gamma \vdash Q \rhd \Delta, \tilde{s} : \{\mathcal{T}_{\mathtt{p}} \mathbin{@} \mathtt{p}\}_{\mathtt{p} \in I} \tag{E.201}$$

for $\{\mathcal{T}_{\mathtt{p}} \mathbin{@} \mathtt{p}\}_{\mathtt{p} \in I}$ coherent (rule [CRes] § 7.2) which, by inductive hypothesis, has a transition α to $\Gamma' \vdash Q'' \rhd \Delta'$ with $\tilde{s} \cap \mathsf{fn}\alpha = \varnothing$ and $\mathsf{erase}(Q'') = Q'$, hence done.

– The case [BOut] is similar to (i) of the previous case.
– The case when the last rule of Υ is [Def] straightforwardly follows by induction and by the fact that $\mathsf{erase}()$ homomorphically extends to contexts, noting the transition of defined agents can be derived by unfolding.
– Similarly for the case [Str].
– The case the transition is derived by [Tau]. Suppose:

$$\Gamma \vdash \mathsf{erase}(P) \rhd \Delta \xrightarrow{\tau} \Gamma' \vdash Q \rhd \Delta'. \tag{E.202}$$

By Rule [Tau] this also means

$$\mathsf{erase}(P) \to Q \tag{E.203}$$

By Lemma E.36, P (nor Q) does not contain queues at free session channels. Hence by Lemma 7.9 as well as noting that the case of the reduction $\Delta \to \Delta'$ in Lemma 7.9 can only be caused when there is a reduction at a free session channel, we know $\Gamma = \Gamma'$ and $\Delta' = \Delta$ in (E.202). Now by (E.203) we know

$$P \to P' \tag{E.204}$$

for some P', by a derivation Υ' corresponding to Υ, where Υ' may possibly differ from Υ in the use of one of the error rules, i.e. the rules in Figure 9. But by $\Gamma \vdash P \rhd \Delta$ and Lemma 7.9, the process P' in (E.204) cannot contain an assertion error, hence Υ' does not use an error reduction rule. Hence by (1) above, we have $Q \equiv \mathsf{erase}(P')$, as required.

This exhausts all cases. □

We now prove:

Lemma E.38 (originally Theorem 7.15-(5)). *Let P be a program and $\Gamma \vdash P \rhd \Delta$. Then we have (1) $\Gamma \models \mathsf{erase}(P) \rhd \Delta$ and (2) $\Gamma \vdash P \sim \mathsf{erase}(P) \rhd \Delta$.*

Proof. For the proof of (1), assume $\mathcal{S}$ witnesses the conditional similarity for P and $\langle \Gamma, \Delta \rangle$. We then create a new relation $\mathcal{S}'$ by the following construction:

$$(P, \langle \Gamma, \Delta \rangle) \in \mathcal{S} \quad \Longrightarrow \quad (\mathsf{erase}(P), \langle \Gamma, \Delta \rangle) \in \mathcal{S}' \tag{E.205}$$

This is a conditional simulation because: by Lemma E.37 (1, 2), each output/selection/τ/free-shared-channel action of $\mathsf{erase}(P)$ is matched by P, with the resulting pair again in the closure; similarly for input/branching action that is allowed by $\langle \Gamma, \Delta \rangle$, using Lemma E.37 (1, 3), hence as required.

To prove (2) we define $\mathcal{E}$ as:

$$\mathcal{E} = \{(P, \mathsf{erase}(P)) \mid \Gamma \vdash P \rhd \Delta\}$$

such that each P is a closed process without queues at each of its free session channels and that Δ is a collection of singleton assignments, i.e. of

the form $\tilde{s} : \mathcal{T}@\mathtt{p}$ at each session. By Lemma E.36, this covers all multiple step derivatives of programs by asserted transitions. Let $(P, \mathsf{erase}(P)) \in \mathcal{E}$ and $\Gamma \vdash P \rhd \Delta$. First assume:

$$\Gamma \vdash P \rhd \Delta \xrightarrow{\alpha} \Gamma' \vdash Q \rhd \Delta' \tag{E.206}$$

Observe that $\langle \Gamma, \Delta \rangle \xrightarrow{\alpha} \langle \Gamma', \Delta' \rangle$ and $\Gamma \vdash P \rhd \Delta$. By Lemma E.37 we know

$$\mathsf{erase}(P) \xrightarrow{\alpha} \mathsf{erase}(Q). \tag{E.207}$$

and the resulting pair is again in the closure, done. For the other direction, i.e. to show any asserted transition of $\mathsf{erase}(P)$ is matched by one of P, assume:

$$\mathsf{erase}(P) \xrightarrow{\alpha} R. \tag{E.208}$$

where $\langle \Gamma, \Delta \rangle \xrightarrow{\alpha} \langle \Gamma, \Delta' \rangle$. By Lemma E.37 (1,2,3) we know

$$\Gamma \vdash P \rhd \Delta \xrightarrow{\alpha} \Gamma' \vdash P' \rhd \Delta' \tag{E.209}$$

with $R \equiv \mathsf{erase}(P')$, hence as required. □

E.9 Proof of Theorem 7.13 (Effective Validation)

We prove Theorem 7.13, whose statement is reproduced below. Recall that we let [NRes] in Figure 10 use the annotation in Convention 7.1 and we dispense with [Conseq]. Write $\Gamma \vdash^{\star} P \rhd \Delta$ for the resulting validation rules.

Theorem E.39. *Given Γ, P and Δ, the provability of $\Gamma \vdash^{\star} P \rhd \Delta$ is decidable.*

Proof. We rely on an initial environment Γ that maps the multicast channels to well-formed and well-asserted global assertions. The validation of P against Δ is done by applying the rules in Figure 10. We define the algorithm in terms of a recursive function from processes, environments, assertion environments and end-point assertions to booleans. This is done by defining an algorithm Val given below. We distinguish a number of cases depending on P and we define $\textsc{Val}(P, \mathcal{C}, \Gamma, \Delta)$ as follows:

– If $P = s_k!\langle \tilde{e} \rangle(\tilde{v})\{A\}; P'$

$$\begin{cases} \text{If } \Delta = \Delta', \tilde{s} : k!(\tilde{v} : \tilde{S})\{A\}; \mathcal{T}' \,@\, \mathtt{p} \text{ and } \mathcal{C} \supset A[\tilde{e}/\tilde{v}] \text{ then} \\ \qquad \mathsf{return}\ \textsc{Val}(P', \mathcal{C}, \Gamma, \Delta', \tilde{s} : \mathcal{T}'[\tilde{e}/\tilde{v}] \,@\, \mathtt{p}) \\ \text{else } \mathsf{return}\ \mathsf{false}. \end{cases}$$

– If $P = s_k?(\tilde{v})\{A\}; P'$
$$\begin{cases}\text{If } \Delta = \Delta', \tilde{s} : k?(\tilde{v} : \tilde{S})\{A\}; \mathcal{T}' \,@\, \mathtt{p} \text{ then} \\ \qquad \mathsf{return}\ \textsc{Val}(P', \mathcal{C} \wedge A, \Delta', \tilde{s} : \mathcal{T}' \,@\, \mathtt{p}) \\ \text{else } \mathsf{return}\ \mathsf{false}.\end{cases}$$
– If $P = s_k \lhd \{A_j\} l_j; P_j$
$$\begin{cases}\text{If } \Delta = \Delta', \tilde{s} : k \oplus \{\{A_i\} l_i : \mathcal{T}_i\}_{i \in I} \,@\, \mathtt{p} \text{ and } \mathcal{C} \supset A_j \text{ then} \\ \qquad \mathsf{return}\ VAL(P_j, \mathcal{C}, \Gamma, \Delta', \tilde{s} : \mathcal{T}_j \,@\, \mathtt{p}) \\ \text{else } \mathsf{return}\ \mathsf{false}.\end{cases}$$
– If $P = s_k \rhd \{\{A_i\} l_i \colon P_i\}_{i \in I}$
$$\begin{cases}\text{If } \Delta = \Delta', \tilde{s} : k \& \{\{A_i\} l_i : \mathcal{T}_i\}_{i \in I} \,@\, \mathtt{p} \text{ then} \\ \qquad \mathsf{return}\ \wedge_{\forall i \in I} \textsc{Val}(P_i, \mathcal{C} \wedge A_i, \Gamma, \Delta', \tilde{s} : \mathcal{T}_i \,@\, \mathtt{p}) \\ \text{else } \mathsf{return}\ \mathsf{false}.\end{cases}$$
– If $P = \mathsf{close}\ \tilde{s}; P$
$$\begin{cases}\text{If } \Delta = \Delta', \tilde{s} : \mathsf{end} \,@\, \mathtt{p} \text{ then } \mathsf{return}\ \textsc{Val}(P', \mathcal{C}, \Gamma \Delta') \\ \text{else } \mathsf{return}\ \mathsf{false}.\end{cases}$$
– If $P = \overline{a}_{[2..n]}\,(\tilde{s}).P'$ then
$\mathsf{return}\ \textsc{Val}(P', \mathcal{C}, \Gamma, (\Delta, \tilde{s} : \Gamma(a) \upharpoonright 1 \,@\, 1))$.
– If $P = a_{[\mathtt{p}]}\,(\tilde{s}).P'$ then
$\mathsf{return}\ \textsc{Val}(P', \mathcal{C}, \Gamma, (\Delta, \tilde{s} : \Gamma(a) \upharpoonright \mathtt{p} \,@\, \mathtt{p}))$.
– If $P = Q_{\Delta} \mid R_{\Delta'}$ (where we assume Q and P annotated with Δ and Δ', respectively, which does not lose generality since by inspecting the free session channels of Q and R we can automatically find the partition of the original assertion assignment) then
$\mathsf{return}\ \textsc{Val}(Q, \mathcal{C}, \Gamma, \Delta) \wedge \textsc{Val}(R, \mathcal{C}, \Gamma, \Delta')$.
– If $P = \mathsf{if}\ e\ \mathsf{then}\ Q\ \mathsf{else}\ R$ then
$\mathsf{return}\ \textsc{Val}(Q, \mathcal{C} \wedge e, \Gamma, \Delta) \wedge \textsc{Val}(R, \mathcal{C} \wedge \neg e, \Gamma, \Delta)$.
– If $P = X\langle \tilde{e} \tilde{s}_1 .. \tilde{s}_n \rangle$
$$\begin{cases}\text{If } \mathcal{C} \supset A[\tilde{e}/\tilde{n}] \text{ then } \mathsf{return}\ \mathsf{true} \\ \text{else } \mathsf{return}\ \mathsf{false}.\end{cases}$$
– If $P = \mathsf{def}\ X(\tilde{v}\tilde{s}_1 \ldots \tilde{s}_n) = P'\ \mathsf{in}\ X\langle \tilde{e}\tilde{s}_1 \ldots \tilde{s}_n \rangle$
$$\begin{cases}\text{If we can find } \mathcal{T}_i,\ 1 \leqslant i \leqslant n, \text{ such that} \\ \Delta = \Delta', \tilde{s}_1 : \mathcal{T}_1[\tilde{e}/\tilde{v}] \,@\, \mathtt{p}_1, .., \tilde{s}_n : \mathcal{T}_n[\tilde{e}/\tilde{v}] \,@\, \mathtt{p}_n \\ \text{and } \mathcal{C} \supset A[\tilde{e}/\tilde{v}] \text{ then} \\ \qquad \mathsf{return}\ \textsc{Val}(P', \mathcal{C}, (\Gamma, X(\tilde{v} : \tilde{S})\tilde{\mathcal{T}}_1 \,@\, \mathtt{p}_1 .. \tilde{\mathcal{T}}_n \,@\, \mathtt{p}_n)) \\ \text{else } \mathsf{return}\ \mathsf{false}.\end{cases}$$

In the last line deciding if such $\mathcal{T}_i$ exists or not is easy by checkinhg all occurrences of $\tilde{e}$ in the resulting formulae and testing them one by one. These rules immediately induce an effective algorithm. $\square$