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. The approach known as Design by Contract (DbC) [23] promotes reliable software development through elaboration of type signatures for sequential programs with logical formulae. This paper presents an assertion method which generalises the notion of DbC to multiparty distributed interactions, enabling the specification and verification of distributed multiparty protocols, based on the π-calculus with full recursion. Centring on the notion of *global assertions* and their *projections* onto endpoint assertions, our method allows fully general specifications for typed sessions with session channel passing, constraining the *content* of the exchanged messages, the *choice* of subconversations to follow, and *invariants* on recursions. The paper presents key theoretical foundations of this framework, including a validation algorithm for consistency of global assertions and a sound and relatively complete compositional proof system for verifying a large class of processes against assertions.

1 Introduction

This paper introduces an assertion method for specifying and verifying distributed multiparty interaction protocols, drawing from the idea often known as Design-by-Contract (DbC) for sequential computation. DbC [23] specifies a contract between a user and a program as a set of pre-conditions, post-conditions, and invariants over the program's type signature. Instead of saying "the method fooBar should be invoked with a string and an integer, and then it will return (if ever) another string", DbC allows more precise specifications, such as "if we invoke the method fooBar with a string representing a date d between 2007 and 2008 and an integer n less than 1000 then it will (if ever) return a string representing the date n days after d".

A type signature describes a basic shape of how a program can interact with other programs, stipulating its key *interface* to other components, which may be developed by other programmers. By associating signatures with logical predicates, DbC enables a highly effective framework for specifying, validating and managing systems' behaviour, usable throughout all phases of software development [19, 21, 27]. As a modelling and programming practice, DbC encourages engineers to make contracts among software modules precise [13, 23], and build a system on the basis of these contracts.

The traditional DbC-based approaches are however limited to type signature of sequential procedures. A typical distributed application implements interaction scenarios that are much more complex than, say, request-reply. To build a theory that extends the core idea of DbC to distributed applications, we consider a generalised notion of type signature for distributed interactions centring on the abstraction units, called *sessions*, studied in [3, 18]. A session consists of a structured series of message exchanges among multiple participants. More than one sessions can interleave or run in parallel even in a single application. For example, a session for an electronic commerce can

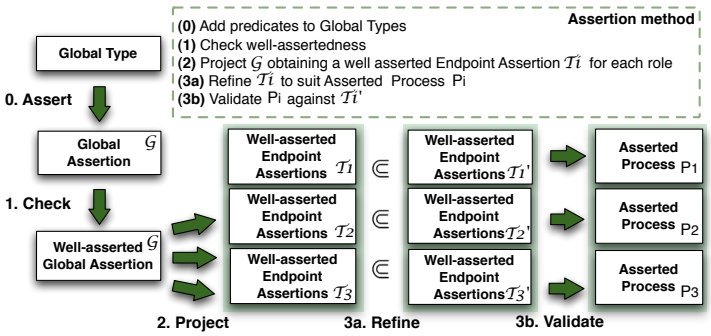

Fig. 1. The assertion method

run interleaved with a session for a financial transaction to settle its payment. Each session follows a stipulated protocol, given as a type called *session type* [3, 18], which prescribes interaction scenarios among its participants.

On this basis, we introduce a theory of assertions for distributed interactions centring on *global assertions*. A global assertion specifies a contract for multiparty sessions by elaborating session types with logical predicates. A session type only specifies the skeleton of interaction scenarios: it does not, for example, refer to constraints on message values except their types. Just as in the traditional DbC, the use of logical predicates allows us to specify more fine-grained constraints on protocols, regarding *content* of messages, how *choice* of sub-conversations is made based on preceding interactions, and what *invariants* may be obeyed in recursive interactions. The key ideas are presented in Figure 1, which we illustrate below.

- **(0,1)** A specification for a multiparty session is given as a *global assertion* $\mathcal{G}$, namely a protocol structure annotated with logical predicates. A minimal semantic criterion, *well-assertedness of* $\mathcal{G}$ (§ 3.1), characterises consistent specifications with respect to the temporal flow of events, to avoid unsatisfiable specifications.
- **(2)** $\mathcal{G}$ is projected onto endpoints, yielding one *endpoint assertion* ($\mathcal{T}_i$) for each participant, specifying the behavioural responsibility of the endpoint (§ 4). The consistency of endpoint assertions are automatically guaranteed once the original global assertion is checked to be well-asserted.
- **(3)** Asserted processes, modelled by the π-calculus annotated with predicates (§ 5.1), are verified against endpoint assertions through a sound and relatively complete compositional proof system (§5.2, §6). Completeness, proved through generation of principal formulae, yields a relative decision procedure for satisfiability.

Our contributions include an algorithmic validation of consistency of global assertions (Propositions 3.2 and 4.3); semantic foundations of global assertions through projection and labelled transitions (Propositions 6.4 and 6.3); a compositional proof system for validating processes against assertions (Theorem 6.5), leading to *assertion-error freedom* (Theorem 6.6) which ensures that the process will meet its obligations assuming that the remaining parties do so. Theorems 6.7 and 7.2 ensure completeness. For readability, we first present a simplified version of our theory and then, in § 7, we extend it to delegation and shared name passing. § 8 concludes with the related work. The omitted definitions are listed in the attached appendix, while the detailed proofs are found in the full version [31].

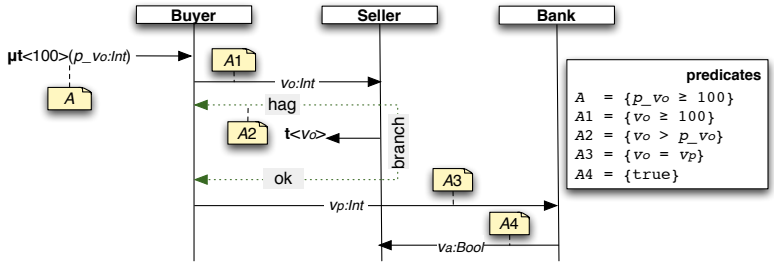

Fig. 2. Global assertion for the protocol.

2 DbC for Distributed Multiparty Interactions

The theory presented in this paper centres on what we call *global assertions* for specifying contracts for multiparty sessions. A global assertion uses a logical formula to prescribe, for each interaction specified in the underlying session type, what the sending party must guarantee, and dually what the receiving party can rely on. Concretely:

1. Each message exchange in a session is associated with a predicate which constrains the values carried in the message (e.g., "the quantity on the invoice from seller to buyer equals the quantity on the order");
2. Each branch in a session is associated with a predicate which constrains the selection of that branch (e.g., "seller chooses the 'sell' option for a product if the ordered quantity does not exceed the stock");
3. Each recursion in a session is associated with an invariant representing an obligation to be maintained by all parties at each repetition of the recursion (e.g., "while negotiating, seller and buyer maintain the price per unit about a fixed threshold").

As an illustration, Figure 2 describes a simple multiparty session among the participants `Buyer`, `Seller`, and `Bank` exchanging messages whose content is represented by the *interaction variables* v_o, v_p (of type Int) and v_a (of type Bool). `Buyer` asynchronously sends an offer v_o, then `Seller` selects either to recursively start negotiating (hag) or to accept the offer (ok). In the latter case, `Buyer` instructs `Bank` to make a payment v_p. Finally, `Bank` sends `Seller` an acknowledgement v_a. The recursion has one parameter p_v_o, initially set to 100, that upon recursive invocation takes the value that v_o has in the instance of recursion which invoked the current one. This allows us to compare the current content of v_o with the one of the previous recursion instance (see $A2$ below).

The predicates attached to the interactions are in a way similar to guards in sequence diagrams (the present framework however models and constrains distributed communications). The example uses five predicates. The recursion invariant A states that p_v_o is always greater or equal than 100. `Buyer` guarantees $A1$ (i.e., the value of v_o is above 100) which, dually, `Seller` relies upon. By $A2$, `Buyer` has to increase the price during negotiations until an agreement is reached. The value of the (last) offer and the payment must be equal by $A3$, while $A4$ does not constrain v_a.

The specification framework naturally extends to more complex situations which involve, among others, delegation (session channel passing), as we shall discuss later.

3 Global Assertions

We use the syntax of logical formulae, often called *predicates*, as follows.

$$A,B \;::=\; e_1 = e_2 \mid e_1 > e_2 \mid \phi(e_1,\ldots,e_n) \mid A \wedge B \mid \neg A \mid \exists v(A)$$

where e_i range over expressions[1] and ϕ over a pre-defined set of atomic predicates with fixed arities and types (such as those on natural numbers [22, §2.8]). The set $var(A)$ denotes the set of *free variables* of A, similarly for $var(e)$. We assume the validity of closed formulae of the *underlying logic* to be decidable.

Global assertions (ranged over by $\mathcal{G}, \mathcal{G}', \ldots$) elaborate global session types in [18] with logical formulae. The syntax is given below:

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

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

Interaction $\mathtt{p} \to \mathtt{p}' : k\,(\tilde{v}\colon \tilde{S})\{A\}.\mathcal{G}$ describes a communication between a sender $\mathtt{p}$ and a receiver $\mathtt{p}'$ via the k^{th} session channel (k is a natural number), followed by $\mathcal{G}$. The variables in the vector $\tilde{v}$ are called *interaction variables* and bind their occurrences in A and in $\mathcal{G}$; interaction variables are sorted by *sorts* S (Bool, Int, ...) that denote types for first-order message values (session delegation and shared name passing is discussed in § 7). The predicate A constrains the content of $\tilde{v}$: the sender $\mathtt{p}$ *guarantees* A and the receiver $\mathtt{p}'$ *relies* on A (following the rely-guarantee paradigm [20]).

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

Recursive assertion $\mu\mathbf{t}\langle \tilde{e}\rangle(\tilde{v}\colon \tilde{S})\{A\}.\mathcal{G}$ (cf. [11], $\mathbf{t}$ is an *assertion variable*) specifies how a recursive session, which may be repeated arbitrarily many times, should be carried out through interactions among participants. The formal parameters $\tilde{v}$ are a vector of pairwise distinct variables (sorted by a vector $\tilde{S}$ of sorts of the same length; each v_i in $\tilde{v}$ has sort S_i of $\tilde{S}$); $\tilde{v}$ bind their free occurrences in A. The *initialisation vector* $\tilde{e}$ denote the initial values for the recursion, each e_i instantiating v_i in $\tilde{v}$. The *recursion invariant* A specifies the condition that needs be obeyed at each recursion instantiation; *recursion instantiation*, of the form $\mathbf{t}\langle \tilde{e}\rangle$, is to be guarded by prefixes, i.e. the underlying recursive types should be contractive. A recursive assertion can be unfolded to an infinite tree, as in the *equi-recursive* view on recursive types [29].

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

We write $\mathtt{p} \in \mathcal{G}$ when $\mathtt{p}$ occurs in $\mathcal{G}$. For the sake of simplicity we avoid linearity-check [3] by assuming that each channel in $\mathcal{G}$ is used (maybe repeatedly) only between two parties: one party for input/branching and by the other for output/selection.

Example 3.1 (Global Assertions). $\mathcal{G}_{neg}$ models the protocol described in § 2. $\mathcal{G}_{neg}$ has recursion parameter p_v_o denoting the offer of `Buyer` in the previous recursion instance; p_v_o is 100 in the first instance. $\{k_1,k_2,k_3,k_4\}$ are channels.

$$\begin{array}{l} \mathcal{G}_{neg} = \mu\mathbf{t}\langle 100\rangle(p\_v_o : \mathsf{Int})\{A\}.\ \mathtt{Buyer} \to \mathtt{Seller}\colon k_1\,(v_o : \mathsf{Int})\{A1\}. \\ \qquad\qquad\qquad\qquad\qquad\qquad \mathtt{Seller} \to \mathtt{Buyer}\colon k_2\{\{A2\}\mathbf{hag}\colon \mathbf{t}\langle v_o\rangle, \{\mathsf{true}\}\mathbf{ok}\colon \mathcal{G}_{ok}\} \\ \mathcal{G}_{ok} \;= \mathtt{Buyer} \to \mathtt{Bank}\colon k_3\,(v_p : \mathsf{Int})\{A3\}.\ \mathtt{Bank} \to \mathtt{Seller}\colon k_4\,(v_a : \mathsf{Bool})\{A4\}.\ \mathsf{end} \end{array}$$

[1] Expressions include sorted variables but not channels.

3.1 Well Asserted Global Assertions

When setting up global assertions as a contract among multiple participants, we should prevent inconsistent specifications, such as those in which it is logically impossible for a participant to meet the specified obligations. Below we give two constraints on predicates of global assertions that guarantee consistency.

Let $I(\mathcal{G})$ be the set of variables occurring in $\mathcal{G}$; a participant $\mathtt{p}$ *knows* $v \in I(\mathcal{G})$ if v occurs in an interaction of $\mathcal{G}$ involving $\mathtt{p}$ (this relation can be computed effectively, see Appendix B). $I(\mathcal{G}) \upharpoonright \mathtt{p}$ denotes the set of variables of $\mathcal{G}$ that $\mathtt{p} \in \mathcal{G}$ knows.

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

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

Consider the following examples:

$$\mathtt{p_A} \to \mathtt{p_B}\colon k_1\,(v : \mathsf{Int})\{\mathsf{true}\}.\ \mathtt{p_B} \to \mathtt{p_C}\colon k_2\,(v' : \mathsf{Int})\{\mathsf{true}\}.\ \mathtt{p_C} \to \mathtt{p_A}\colon k_3\,(z : \mathsf{Int})\{z > v\}.\ \mathsf{end}$$
$$\mathtt{p_A} \to \mathtt{p_B}\colon k_1\,(v : \mathsf{Int})\{v < 10\}\ \mathtt{p_B} \to \mathtt{p_A}\colon k_2\,(z : \mathsf{Int})\{v > z \wedge z > 6\}.\ \mathsf{end}.$$

The first global assertion violates history-sensitivity since $\mathtt{p_C}$ has to send z such that $z > v$ but $\mathtt{p_C}$ does not know v. The second global assertion violates temporal-satisfiability because if $\mathtt{p_A}$ sends $v = 6$, which satisfies $v < 10$, then $\mathtt{p_B}$ will not be able to find a value that satisfies $6 > z \wedge z > 6$.

Assertions satisfying history-sensitivity and temporal-satisfiability are called *well-asserted assertions*. For the formal definitions, including inductive rules to check well-assertedness, see Appendix B.

Proposition 3.2 (Well-assertedness). *Checking well-assertedness of a given global assertion is decidable if the underlying logic is decidable.*

4 Endpoint Assertions and Projection

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

$$\begin{array}{rcl} \mathcal{T} & ::= & k!(\tilde{v}\colon \tilde{S})\{A\};\mathcal{T} \mid \mu\mathbf{t}\langle\tilde{e}\rangle(\tilde{v} : \tilde{S})\{A\}.\mathcal{T} \mid k\&\{\{A_i\}l_i\colon \mathcal{T}_i\}_{i\in I} \mid \mathsf{end} \\ & \mid & k?(\tilde{v}\colon \tilde{S})\{A\};\mathcal{T} \mid \mathbf{t}\langle\tilde{e}\rangle \qquad\qquad\quad \mid k\oplus\{\{A_j\}l_j\colon \mathcal{T}_j\}_{j\in I} \end{array}$$

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

In $k \oplus \{\{A_j\}l_j\colon \mathcal{T}_j\}_{j\in I}$ the selector guarantees A_j when choosing l_j on k; dually $k\&\{\{A_j\}l_j\colon \mathcal{T}_j\}_{i\in I}$ states that A_j can be assumed when branching at k on a label l_j. Assertion $\mu\mathbf{t}\langle\tilde{e}\rangle(\tilde{v} : \tilde{S})\{A\}.\mathcal{T}$ constrains parameters $\tilde{v}$ of type $\tilde{S}$ which are initially take values $\tilde{e}$; the invariant of the recursion is A.

The projection of predicate A on participant $\mathtt{p}$, written $A \upharpoonright \mathtt{p}$, is defined as $\exists V_{ext}(A)$ where $V_{ext} = var(A) \backslash I(\mathcal{G}) \upharpoonright \mathtt{p}$. Also, $\tilde{e} \upharpoonright \mathtt{p}$ are the expressions in $\tilde{e}$ including only such that $var(e'_i) \subseteq I(\mathcal{G}) \upharpoonright \mathtt{p}$. The *projection* function in Definition 4.1 maps global assertions, predicates and participants to endpoint assertions.

Definition 4.1 (Projection). Given $\mathcal{G}$ and A, the *projection of $\mathcal{G}$ for a participant* $\mathtt{p}$ *wrt* A is denoted by $(\mathcal{G})\downarrow_{\mathtt{p}}^{A}$ and, assuming $\mathtt{p}_1 \neq \mathtt{p}_2$, recursively defined as follows.

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

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

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

$$(4)\ (\mu\mathbf{t}\langle\tilde{e}\rangle(\tilde{v}\colon\tilde{S})\{A\}.\mathcal{G})\downarrow_{\mathtt{p}}^{A_P} = \mu\mathbf{t}\langle\tilde{e}\upharpoonright\mathtt{p}\rangle(\tilde{v}\upharpoonright\mathtt{p}\colon S)\{A\upharpoonright\mathtt{p}\}.(\mathcal{G})\downarrow_{\mathtt{p}}^{A_P}$$

$$(5)\ (\mathbf{t}\langle\tilde{e}\rangle)\downarrow_{\mathtt{p}}^{A_P} = \mathbf{t}\langle\tilde{e}\upharpoonright\mathtt{p}\rangle \qquad (6)\ (\mathsf{end})\downarrow_{\mathtt{p}}^{A_P} = \mathsf{end}$$

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

In (1), value passing interactions are projected. For a send, the projection of a predicate A consists of A itself. For a receive, it is not sufficient to verify the non-violation of the current predicate only. Consider the following well-asserted global assertion:

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

The predicate $pay \geqslant cost$ is not meaningful to `Bank` since `Bank` does not know *cost* hence cannot verify it: rather the projection on `Bank` should be $k_2?(pay:\mathsf{Int})\{\exists cost(cost > 10\wedge pay\geqslant cost)\}$ which incorporates the interaction between `Buyer` and `Seller`. Thus in (1) all the past predicates are projected on $\mathtt{p}_2$, existentially quantifying the variables unknown to $\mathtt{p}_2$, so that $\mathtt{p}_2$ can detect violations of predicates coming from interactions it has not participated in. Note (1) yields the strongest precondition for $\mathtt{p}_2$ (the contract is satisfied iff $\mathtt{p}_2$ receives a legal message), avoiding the burden of defensive programming (e.g., the programmer of `Bank` can concentrate on the case $pay \leqslant 10$).

In (2), the "otw" case says the projection should be the same for all branches. In (3), each participant is in at most a single global assertion to ensure each local assertion is single threaded. In (4), the projection to $\mathtt{p}$ is the recursive assertion itself with its predicate projected on $\mathtt{p}$ by existential quantification (see Appendix C for details). Similarly in (5) the projection of a recursive call is itself with its predicate projected.

Example 4.2 (Projection). The projection of $\mathcal{G}_{neg}$ (Example 3.1) on `Seller` is

$$\begin{array}{rcl} \mathcal{T}_{sel} & = & \mu\mathbf{t}\langle 100\rangle(p\_v_o:\mathsf{Int})\{p\_v_o\geqslant 100\};k_1?(v_o:\mathsf{Int})\{B\};\mathcal{T}_2 \\ \mathcal{T}_2 & = & k_2\oplus\{\{v_o > p\_v_o\}\mathbf{hag}\colon \mathbf{t}\langle v_o\rangle, \{\mathsf{true}\}\mathbf{ok}\colon \mathcal{T}_{ok}\} \\ \mathcal{T}_{ok} & = & \mathcal{G}_{ok}\upharpoonright\mathtt{Seller} = k_4?(v_a:\mathsf{Bool})\{B'\} \end{array}$$

where $B = p\_v_o \geqslant 100 \wedge v_o \geqslant 100$ and $B' = \exists p\_v_o.B \wedge v_o = v_p$.

Well-assertedness can be defined on endpoint assertions as for global assertions, characterising the same two principles discussed in §3.1.

Proposition 4.3 (Projections). *Let $\mathcal{G}$ be a well-asserted global assertion. Then for each $\mathtt{p}\in\mathcal{G}$, if $\mathcal{G}\upharpoonright\mathtt{p}$ is defined then $\mathcal{G}\upharpoonright\mathtt{p}$ is also well-asserted.*

5 Compositional Validation of Processes

5.1 The π-Calculus with Assertions

We use the π-calculus with multiparty sessions [18, §2], augmented with predicates for checking (both outgoing and incoming) communications.

$P ::= \overline{a}_{[2..n]}(\tilde{s}).P$	request	$\mid s \lhd \{A\}l;P$	select	$P_{rt} ::= P \mid (\nu \tilde{s})P_{rt}$	
$\mid a_{[p]}(\tilde{s}).P$	accept	$\mid s \rhd \{\{A_i\}l_i\colon P_i\}_{i\in I}$	branch	$\mid s:\tilde{h}$	
$\mid (\nu a)P$	hide	$\mid P \mid Q$	parallel	$\mid \mathsf{errH} \mid \mathsf{errT}$	
$\mid s!\langle\tilde{e}\rangle(\tilde{v})\{A\};P$	send	$\mid \mu X\langle\tilde{e}\tilde{t}\rangle(\tilde{v}\tilde{s}).P$	rec def	$e ::= n \mid e \wedge e' \ldots$	
$\mid s?(\tilde{v})\{A\};P$	receive	$\mid X\langle\tilde{e}\tilde{s}\rangle$	rec call	$n ::= a \mid \mathsf{true} \mid \mathsf{false}$	
$\mid \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q$	conditional	$\mid \mathbf{0}$	idle	$h ::= l \mid \tilde{n}$	

Fig. 3. Syntax of asserted processes

The grammar of *asserted processes* or simply *processes* ($P, Q, \ldots$) is given in Figure 3. On the left, we define *programs*. $\overline{a}_{[2..n]}(\tilde{s}).P$ multicasts a session initiation request to each $a_{[p]}(\tilde{s}).P$ (with $2 \leqslant p \leqslant n$) by multiparty synchronisation through a *shared name* a.[2] Send, receive, and selection, all through a *session channel* s, are associated with a predicate. Branch associates a predicate to each label. Others are standard.

Runtime processes P_{rt}, defined on the right-hand side of the grammar, extend programs with runtime constructs. Process $s:h_1..h_n$ represents messages in transit going through a session 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 sessions/values. The empty queue is written $s:\varnothing$. Processes errH and errT indicate two kinds of run-time assertion violation: errH (for "error here") indicates a violation of a predicate by the process itself; and errT ("error there") indicates a violation by the environment.

The reduction rules with predicate checking are given in Figure 4, which generate $\rightarrow$ by closing the induced relation under $\mid$ and ν and taking terms modulo the standard structural equality[3] [18]. The satisfaction of the predicate is checked at each communication action: *send*, *receive*, *selection* and *branching*, where we write $A \downarrow \mathsf{true}$ (resp. $\tilde{e} \downarrow \tilde{n}$) for a closed formula A (resp. expression $\tilde{e}$) when it evaluates to true (resp. $\tilde{n}$). When initiating a session, [R-LINK] establishes a session through multiparty synchronisation, generating queues (all session channels are hidden at the initiation). The remaining rules are standard, modelling communications in a session via queues [3, 18].

Example 5.1 (Seller's Process). Continuing from Examples 3.1 and 4.2 we present a process that implements $\mathcal{G}_{neg}$, focusing on P_2 of `Seller`. Below, B, B' are as in Example 4.2. We set `Buyer`, `Seller`, `Bank` to be participants `1`, `2`, `3`. We denote $s_1, .., s_4$ with $\tilde{s}$.

$$
\begin{array}{lll}
P_{neg} & = & \overline{a}_{[2,3]}(\tilde{s}).P_1 \mid a_{[2]}(\tilde{s}).P_2 \mid a_{[3]}(\tilde{s}).P_3 \\
P_2 & = & \mu X\langle 100, \tilde{s}\rangle(p\_v_o, \tilde{s}).s_1?(v_o)\{B\};Q_2 \\
Q_2 & = & \mathsf{if}\ e\ \mathsf{then}\ (s_2 \lhd \mathbf{hag}; X\langle v_o, \tilde{s}\rangle)\ \mathsf{else}\ (s_2 \lhd \mathbf{ok}; P_{ok}) \quad \text{where} \quad P_{ok} = s_4?(v_a)\{B'\};\mathbf{0}
\end{array}
$$

Q_2 defines, with e, a policy to select a branch (e.g., $e = \{v_o > 200 \wedge v_o > p\_v_o\}$). While the predicates are known to all parties, e is a local policy to `Seller`.

[2] Session initialisation does not have predicates because we study contracts for individual sessions.

[3] We include $\mu X\langle\tilde{e}\rangle(\tilde{v}\tilde{s}_1 \ldots \tilde{s}_n).P \equiv P[\mu X(\tilde{v}\tilde{s}_1 \ldots \tilde{s}_n).P/X]\,[\tilde{e}/\tilde{v}]$ where $X\langle\tilde{e}'\tilde{s}'\rangle[\mu X(\tilde{v}\tilde{s}_1 \ldots \tilde{s}_n).P/X]$ is defined as $\mu X\langle\tilde{e}'\tilde{s}'\rangle(\tilde{v}\tilde{s}_1 \ldots \tilde{s}_n).P$.

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

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

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

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

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

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

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

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

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

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

Fig. 4. Reduction: non-error cases (top) - error cases (bottom)

5.2 Validation Rules

For validation, we use the judgement of the form $\mathcal{C};\Gamma \vdash P \rhd \Delta$, which reads: *"under $\mathcal{C}$ and Γ, process P is validated against Δ"*. Here, $\mathcal{C}$ is an *assertion environment*, which incrementally records the conjunction of predicates. Γ is a *global assertion assignment* that is a finite function mapping shared names to well-asserted global assertions, writing $\Gamma \vdash a : \mathcal{G}$ when Γ assigns $\mathcal{G}$ to a; and process variables $(X, Y, ...)$ to the specification of their parameters, writing $\Gamma \vdash X : (\tilde{v} : \tilde{S})\mathcal{T}_1 @ \mathtt{p}_1 ... \mathcal{T}_n @ \mathtt{p}_n$ when Γ maps X to the vector of endpoint assertions $\mathcal{T}_1 @ \mathtt{p}_1 ... \mathcal{T}_n @ \mathtt{p}_n$ using the variables $\tilde{v}$ sorted by $\tilde{S}$. And Δ is an *endpoint assertion assignment* which maps the channels for each session, say $\tilde{s}$, to a well-asserted endpoint assertion located at a participant, say $\mathcal{T} @ \mathtt{p}$.

The validation rules are given in Figure 5. In each rule, we assume all occurring (global/endpoint) assertions are well-asserted. The rules validate the process against assertions, simultaneously annotating processes with the interaction predicates from endpoint assertions. We illustrate the key rules.

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

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 [17]. As in the underlying typing [18], rule [CONC] takes a disjoint union of two channel environments, and rule [IDLE] takes Δ which only contains end as endpoint assertions. Rule [HIDE] is standard, assuming a is not specified in $\mathcal{C}$.

Rule [CONSEQ] uses the *refinement* relation $\ni$ on endpoint assertions. If $\mathcal{T} \ni \mathcal{T}'$, $\mathcal{T}$ specifies a *more refined behaviour* than $\mathcal{T}'$, in that $\mathcal{T}$ strengthens the predicates for send/selection, so it emits/selects less; and weakens those for receive/branching, so it can receive/accept more. The formal definition follows this intuition, and is given in Appendix D. Below we illustrate this relation through an example.

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

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

$$\frac{\mathcal{C};\Gamma \vdash P \rhd \Delta, \tilde{s}:(\Gamma(a)\upharpoonright \mathtt{p}) @ \mathtt{p} \quad \mathtt{p} \gneq 1}{\mathcal{C};\Gamma \vdash a[\mathtt{p}](\tilde{s}).P \rhd \Delta}\,[\text{MACC}] \qquad \frac{\mathcal{C};\Gamma \vdash P \rhd \Delta, \tilde{s}:(\Gamma(a)\upharpoonright 1) @ 1}{\mathcal{C};\Gamma \vdash \overline{a}[2..\mathtt{n}](\tilde{s}).P \rhd \Delta}\,[\text{MCAST}]$$

$$\frac{\mathcal{C} \wedge e;\Gamma \vdash P \rhd \Delta \quad \mathcal{C} \wedge \neg e;\Gamma \vdash Q \rhd \Delta}{\mathcal{C};\Gamma \vdash \text{if } e \text{ then } P \text{ else } Q \rhd \Delta}\,[\text{IF}] \quad \frac{\mathcal{C};\Gamma \vdash P \rhd \Delta \quad \mathcal{C};\Gamma \vdash Q \rhd \Delta'}{\mathcal{C};\Gamma \vdash P \mid Q \rhd \Delta,\Delta'}\,[\text{CONC}] \quad \frac{\Delta \text{ end only}}{\mathcal{C};\Gamma \vdash \mathbf{0} \rhd \Delta}\,[\text{IDLE}]$$

$$\frac{\mathcal{C};\Gamma, a:\mathcal{G} \vdash P \rhd \Delta \quad a \notin \text{fn}(\mathcal{C},\Gamma,\Delta)}{\mathcal{C};\Gamma \vdash (\nu a:\mathcal{G})P \rhd \Delta}\,[\text{HIDE}] \quad \frac{\mathcal{C}';\Gamma \vdash P \rhd \Delta' \quad \mathcal{C} \supset \mathcal{C}' \quad \Delta' \Supset \Delta}{\mathcal{C};\Gamma \vdash P \rhd \Delta}\,[\text{CONSEQ}]$$

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

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

Fig. 5. Validation rules for program phrases

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

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

$\mathcal{T}_s$ has a stronger obligation on the sent value v, and a weaker reliance on the received value z; while $\mathcal{T}_s$ has a weaker guarantee at **l1** and offers one additional branch.

The refinement relation is decidable if we restrict the use of recursive assertions so that only those in identical shapes are compared as illustrated in Appendix D, which would suffice in many practical settings.

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

Henceforth we write $\Gamma \vdash P \rhd \Delta$ for $\mathsf{true};\Gamma \vdash P \rhd \Delta$.

Example 5.3 (Validating Seller Process). We validate P_{neg} (part of `Seller` from Example 5.1) under $\mathcal{T}_{sel}$ (from Example 3.1). We focus on one branch of Q_2 in P_{neg}. We associate each $s_1,\ldots,s_4$ of P_{neg} to a channel $k_1,\ldots,k_4$ of $\mathcal{T}_{neg}$, respectively. Recall $B = \{p\_v_o \geqslant 100 \wedge v_o \geqslant 100\}$, $A1 = \{v_o > p\_v_o\}$, and $A2 = \{\exists v_p.p\_v_o \geqslant 100 \wedge v_o \geqslant 100 \wedge v_o = v_p\}$. Below $Q_{ok} = s_4?(v_a)\{B'\};\mathbf{0}$.

$$\cfrac{\cfrac{\cfrac{(B\wedge\neg e)\supset A1 \quad \cfrac{\cfrac{\cfrac{-}{(B\wedge\neg e\wedge B'),\Gamma\vdash \mathbf{0}\rhd t:\mathsf{end} @ 2}\,[\text{IDLE}]}{(B\wedge\neg e),\Gamma\vdash s_4?(v_a)\{B'\};\mathbf{0}\rhd\tilde{s}:k_4?(v_a:\mathsf{Int})\{B'\};\mathsf{end} @ 2}\,[\text{RCV}]}{(B\wedge\neg e),\Gamma\vdash Q_{ok}\rhd\tilde{s}:\mathcal{T}_{ok} @ 2}\,\text{(substituting)}}{B\wedge\neg e,\Gamma\vdash s_2\lhd\mathbf{ok};Q_{ok}\rhd\tilde{s}:k_2\oplus\{\{\mathsf{true}\}\mathbf{ok}:\mathcal{T}_{ok},\{A1\}\mathbf{hag}:\mathbf{t}\langle v_o\rangle\} @ 2}\,[\text{SEL}] \quad \ldots}{B,\Gamma\vdash \text{if } e \text{ then } (s_2\lhd\mathbf{hag};X\langle v_o,\tilde{s}\rangle) \text{ else } (s_2\lhd\mathbf{ok};s_4?(v_a)\{B'\};\mathbf{0})\rhd\tilde{s}:\mathcal{T}_2 @ 2}\,[\text{IF}]}{\mathsf{true},\Gamma\vdash s_1?(v_o)\{B\};Q_2\rhd\tilde{s}:k_1?(v_o:\mathsf{Int})\{B\};\mathcal{T}_2 @ 2}\,[\text{RCV}]$$

The ... on the premise of [IF] indicate the missing validation of the first branch. The interested reader may refer to [31] for a complete validation example with recursion.

6 Error-Freedom and Completeness

6.1 Semantics of Assertions

The semantics of asserted processes is formalised as a labelled transition relation which includes the reduction semantics given in § 5.1 and is standard except that predicates are checked for communications. We use the following labels:

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

We write $P \xrightarrow{\alpha} Q$ when P has a one-step transition α to Q. The transition rules are standard synchronous ones [4] except that (*i*) a process moves to errT after an input/branching action if its predicate is violated, (*ii*) a process has the τ-action to errH when it violates the predicate of an output/selection, and (*iii*) $P \to Q$ induces $P \xrightarrow{\tau} Q$.

The semantics of endpoint assertions is defined as another labelled transition relation, of form $\langle \Gamma, \Delta \rangle \xrightarrow{\alpha} \langle \Gamma', \Delta' \rangle$, which reads: *the specification* $\langle \Gamma, \Delta \rangle$ allows *the action* α*, with* $\langle \Gamma', \Delta' \rangle$ *as the specification for its continuation.* In this transition relation, only legitimate (assertion-satisfying) actions are considered.

We use a simulation between the transitions of processes and those of assertions to define the semantic counterpart of $\Gamma \vdash P \rhd \Delta$. In the simulation (Definition 6.1 below), an input/branching action is required to be simulated only for each "valid" value/label, i.e. a type-correct action which does not violate the associated predicate. Thus we demand conformance to a proper behaviour only if the environment behaves properly. If not, the process is allowed to misbehave. Below $\mathsf{erase}(P)$ is the result of erasing all predicates from P. Similarly $\mathsf{erase}(\Gamma)$ and $\mathsf{erase}(\Delta)$ erase predicates from the underlying session types, giving the *typing* environments. We use the erasure to show that the validation can prevent bad behaviour even without runtime predicate checking. Below P is *closed* if it is without free variables.

Definition 6.1 (Conditional Simulation). Let $\mathcal{R}$ be a binary relation whose element relates a closed process P without errH or errT and a pair of assignments $\langle \Gamma, \Delta \rangle$ such that $\mathsf{erase}(\Gamma) \vdash \mathsf{erase}(P) \rhd \mathsf{erase}(\Delta)$ in the typing rules in [18, §4]. Then $\mathcal{R}$ is a *conditional simulation* if, for each $(P, \langle \Gamma, \Delta \rangle) \in \mathcal{R}$:

1. for each input/branching/session input $P \xrightarrow{\alpha} P'$, $\langle \Gamma, \Delta \rangle$ has a respective move at $\mathsf{sbj}(\alpha)$ (the subject of α) and, if $\langle \Gamma, \Delta \rangle \xrightarrow{\alpha} \langle \Gamma', \Delta' \rangle$ then $(P', \langle \Gamma', \Delta' \rangle) \in \mathcal{R}$.
2. for each output/selection/τ/session output move $P \xrightarrow{\alpha} P'$, $\langle \Gamma, \Delta \rangle \xrightarrow{\alpha} \langle \Gamma', \Delta' \rangle$ such that $(P', \langle \Gamma', \Delta' \rangle) \in \mathcal{R}$. If $\mathcal{R}$ is a conditional simulation we write $P \precsim \langle \Gamma, \Delta \rangle$ for $(P, \langle \Gamma, \Delta \rangle) \in \mathcal{R}$.

The conditional simulation requires P to be well-typed against $\mathsf{erase}(\Gamma)$ and $\mathsf{erase}(\Delta)$. Without this condition, the inaction $\mathbf{0}$ would conditionally simulate any Δ. This stringent condition can be dropped, but it does not lose generality since our interest is to build an assertion semantics on the basis of the underlying type discipline.

Definition 6.2 (Satisfaction). Let P be a closed program and Δ an end-point assertion assignment. If $P \precsim \langle \Gamma, \Delta \rangle$ 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.

[4] The synchronous transition suits our present purpose since it can describe how a process places/retrieves messages at/from queues, at which points message content is checked.

The judgement $\Gamma \models P \triangleright \Delta$ in Definition 6.2 states that (1) P will send valid messages or selection labels; and (2) P will continue to behave well (i.e., without going into error) w.r.t. the continuation specification after each valid action in (1) as well as after receiving each valid message/label (i.e. which satisfies an associated predicate). The satisfaction is about partial correctness since if P (is well-typed and) has no visible actions, the satisfaction trivially holds.

6.2 Soundness, Error Freedom and Completeness

To prove soundness of the validation rules, we first extend the validation rules to processes with queues, based on the corresponding typing rules in [3, 18]. This is left to Appendix H. Below $\langle \Gamma, \Delta \rangle$ *allows* a sequence of actions $\alpha_1..\alpha_n$ $(n \geqslant 0)$ if for some $\langle \Gamma', \Delta' \rangle$ we have $\langle \Gamma, \Delta \rangle \xrightarrow{\alpha_1..\alpha_n} \langle \Gamma', \Delta' \rangle$ (and similarly for processes).

Proposition 6.3 (Subject Reduction). *Let $\Gamma \vdash P \triangleright \Delta$ be a closed program and suppose we have $\langle \Gamma, \Delta \rangle \xrightarrow{\alpha_1..\alpha_n} \langle \Gamma', \Delta' \rangle$. Then $P \xrightarrow{\alpha_1..\alpha_n} P'$ implies $\Gamma' \vdash P' \triangleright \Delta'$.*

The proof demands the analysis of the effects of τ-actions on endpoint assertions, observing the reduction at free session channels changes the shape of linear typing [3, 18], hence of endpoint assertions.

The following Proposition says that if a process satisfies a stronger (more refined) specification, it also satisfies a weaker one.

Proposition 6.4 (Refinement). *If $\Gamma \models P \triangleright \Delta$ and $\Delta \ni \Delta'$ then $\Gamma \models P \triangleright \Delta'$.*

The soundness result follows. Its proof uses Propositions 6.3 and 6.4, see [31].

Theorem 6.5 (Soundness of Validation Rules). *Let P be a program. Then $\mathcal{C}; \Gamma \vdash P \triangleright \Delta$ implies $\mathcal{C}; \Gamma \models P \triangleright \Delta$.*

A direct consequence of Theorem 6.5 is the error freedom of validated processes.

Theorem 6.6 (Predicate Error Freedom). *Suppose P is a closed program, $\Gamma \vdash P \triangleright \Delta$ and $P \xrightarrow{\alpha_1..\alpha_n} P'$ such that $\langle \Gamma, \Delta \rangle$ allows $\alpha_1..\alpha_n$. Then P' contains neither* errH *nor* errT.

The proof system is complete relative to the decidability of the underlying logic for processes without hidden shared names. We avoid name restriction since it allows us to construct a process which is semantically equivalent to the inaction if and only if interactions starting from a hidden channel terminates. Since we can simulate arbitrary Turing machines by processes, this immediately violates completeness. In this case, non-termination produces a *dead code*, i.e. part of a process which does not give any visible action, which causes a failure in completeness. [5]

For each program without hiding, we can compositionally construct its "principal assertion assignment" from which we can always generate, up to $\ni$, any sound assertion assignment for the process. Since the construction of principal specifications is compositional, it immediately gives an effective procedure to check $\models$ as far as $\ni$ is decidable (which is relative to the underlying logic). We conclude:

Theorem 6.7 (Completeness of Validation Rules for Programs without Hiding). *For each closed program P without hiding, if $\Gamma \models P \triangleright \Delta$ then $\Gamma \vdash P \triangleright \Delta$. Further $\Gamma \models P \triangleright \Delta$ is decidable relative to the decidability of $\ni$.*

[5] Not all dead codes cause failure in completeness. For example a dead branch in a branching/conditional do not cause this issue since the validation rules can handle it, see Appendix I.

7 Extension with Shared Name Passing and Session Delegation

In this section we extend the theory with delegation (session channel passing) and shared name passing. The incorporation of shared name passing also leads to a generalised completeness result where we can treat name hiding.

Extension in Assertions and Processes. The shared name passing is easily incorporated by extending the grammar of sorts $S, S', \ldots$ with a global assertion $\langle \mathcal{G} \rangle$. Then the syntax of global/endpoint assertions are automatically extended so that they can specify shared name passing, similarly for processes. For example, in global assertions, we can write $\mathtt{p} \to \mathtt{p}' : k\,(v : \langle \mathcal{G} \rangle)\{A\}.\mathcal{G}'$, in which v, standing for a shared name, is constrained by $\mathcal{G}$ (the predicate A is generally trivial but we keep it for uniformity). No change is needed for well-assertedness, projection nor the reduction rules.

To model delegation, we extend the original grammars of global and endpoint assertions (respectively in § 3 and in § 4) and of processes (in § 5.1).

$$\mathcal{G} ::= \ldots \mid \mathtt{p} \to \mathtt{p}' : k\,(\tilde{v} : \mathcal{T}@\mathtt{p})\{A\}.\mathcal{G} \quad \mathcal{T} ::= \ldots \mid k!(\tilde{v} : \mathcal{T}_1@\mathtt{p})\{A\};\mathcal{T}_2 \mid k?(\tilde{v} : \mathcal{T}_1@\mathtt{p})\{A\};\mathcal{T}_2$$
$$P ::= \ldots \mid s!\langle\!\langle \tilde{t} \rangle\!\rangle(\tilde{v} : \mathcal{T}@\mathtt{p})\{A\};P \mid s?(\!(\tilde{v} : \mathcal{T}@\mathtt{p})\!)\{A\};P$$

Above $\mathcal{T}@\mathtt{p}$ constraints the receiver's behaviour for the delegated session, represented by the session channels $\tilde{v}$. For processes, the reduction rules are extended for delegation in the standard way, where the communicated channels are checked against the annotating assertion through refinement: when delegating a session, its publicly stipulated assertion should refine the annotating assertion, dually when receiving [6] (for example, $s!\langle\!\langle \tilde{t} \rangle\!\rangle(\tilde{v} : \mathcal{T}@\mathtt{p})\{A\};P$ with an appropriate queue will reduce successfully if the publicly stipulated local assertion at $\tilde{t}$ refines $\mathcal{T}$, otherwise it reaches errH, dually for input: see Appendix A). The clauses for well-assertedness and projection are as before. An example of processes and assertions with delegation follows.

Example 7.1 (Seller's Process with Delegation). Consider a variation of Example 5.1 where `Seller` delegates the communication with `Bank` to a process $P_{cashier}$.

$$\begin{array}{ll}
P_{neg} & = \overline{a}_{[2,3]}\,(\tilde{s}).P_1 \mid a_{[2]}\,(\tilde{s}).P_2 \mid a_{[3]}\,(\tilde{s}).P_3 \mid P_{cashier} \\
P_2 & = \mu X\langle 100, \tilde{s}\rangle(p_v_o, \tilde{s}).s_1?(v_o : \mathsf{Int})\{B\};Q_2 \\
Q_2 & = \mathsf{if}\ e\ \mathsf{then}\ (s_2 \lhd \mathbf{hag}; X\langle v_o, \tilde{s}\rangle)\ \mathsf{else}\ (k_2 \lhd \mathbf{ok}; t!\langle\!\langle \tilde{s} \rangle\!\rangle(\tilde{v} : \mathcal{T}_{ok} @ 2)\{\mathsf{true}\};\mathbf{0}) \\
P_{cashier} & = t?(\!(\tilde{v} : \mathcal{T}_{ok} @ 2)\!)\{\mathsf{true}\};v_4?(v_a : \mathsf{Bool})\{B''\};\mathbf{0} \\
\hline
\mathcal{T}_{deleg} & = h_1?(\tilde{v} : \mathcal{T}'_{ok} @ 2)\{\mathsf{true}\}.\mathcal{T}'_{ok} \\
\mathcal{T}_{ok} & = k_4?(v_a : \mathsf{Bool})\{B'\}.\mathsf{end} \\
\mathcal{T}'_{ok} & = k_4?(v_a : \mathsf{Bool})\{B''\}.\mathsf{end} \quad \text{where } B'' = B' \wedge (v_p < 100 \supset v_a)
\end{array}$$

Above we assume h_1 corresponds to the channel t used for delegation. Note $B'' \supset B'$ for any B' hence $\mathcal{T}_{ok} \ni \mathcal{T}'_{ok}$. By the refinement of delegation discussed soon (covariant in input), the delegation session in $P_{cashier}$ can be validated w.r.t. $\mathcal{T}_{deleg}$.

Extension in Validation Rules, Soundness and Completeness. For validation, shared name passing can be treated using the same send/receive rules as before, while delegation requires the following two additional rules:

$$\frac{\mathcal{C}; \Gamma \vdash P \rhd \Delta,\ \tilde{s}{:}\mathcal{T}\,@\,\mathtt{p}}{\Gamma \vdash s_k!\langle\!\langle \tilde{t} \rangle\!\rangle(\tilde{v} : \mathcal{T}'@\mathtt{q})\{A\};P \rhd \Delta,\ \tilde{s}{:}k!(\tilde{v} : \mathcal{T}'@\mathtt{q})\{A\};\mathcal{T}\,@\,\mathtt{p},\ \tilde{t}{:}\mathcal{T}'@\,\mathtt{q}}\,[\textsc{Sdel}]$$

$$\frac{\mathcal{C} \wedge A; \Gamma \vdash P \rhd \Delta, \tilde{s}{:}\mathcal{T}\,@\,\mathtt{p},\ \tilde{v}{:}\mathcal{T}'@\,\mathtt{q}}{\mathcal{C}; \Gamma \vdash s_k?(\!(\tilde{v} : \mathcal{T}'@\mathtt{q})\!)\{A\};P \rhd \Delta, \tilde{s}{:}k?(\tilde{v} : \mathcal{T}'@\mathtt{q})\{A\};\mathcal{T}\,@\,\mathtt{p}}\,[\textsc{Rdel}]$$

[6] To maintain decidability of reduction, we assume that $\ni$ is decidable, see Appendix D.

For transitions, since shared name passing can induce name extrusion, we extend the transition labels with $(\nu\tilde{a}:\tilde{G})s!\tilde{n}$. The transition is standard, demanding that, for free names, its publicly declared global assertion coincides with the annotating global type. For delegation we add labels $s!\langle\!\langle\tilde{t}\rangle\!\rangle$ and $s?(\!(\tilde{t})\!)$. The delegation transitions accompany predicate checking as in reductions (see Appendix A). The refinement for delegation is affected by the refinement of annotating assertions contravariantly in output and covariantly in input, in parallel with their typing [25].

Soundness of validation rules (Theorem 6.5) and error freedom (Theorem 6.6) hold for the extension (the proofs in [31] are given for this extended system). Further, through the use of shared name passing, completeness (Theorem 6.7) holds for processes with hiding, which in particular can model the generation of unbounded resources. A *visible program* is a closed program, say P, for which each of the hidden shared names occurring in P is immediately exported outside after the hiding, i.e. each hiding, say (νa), in P is always of the form $(\nu a)k!\langle a\rangle;Q$. Visible programs encompass all well-typed behaviours up to pruning of hidden behaviours (which get exposed by visibility), including dynamic creation of unbounded resources. We conclude:

Theorem 7.2 (Completeness with Delegation and Hiding). *For each closed visible program P in the extended syntax, if $\Gamma \models P \rhd \Delta$ then $\Gamma \vdash P \rhd \Delta$. Further $\Gamma \models P \rhd \Delta$ is decidable relative to the decidability of $\Rightarrow$.*

8 Conclusion and Related Work

Hennessy-Milner logic for the π-calculus. Hennessy-Milner Logic (HML) is an expressive modal logic with an exact semantic characterisation [16]. The presented theory addresses some of the key challenges in practical logical specifications for the π-calculus, unexplored in the context of HML. First, by starting from global assertions, we gain in significant concision of descriptions while enjoying generality within its scope (properties of individual protocols). Previous work [2, 11] show how specifications in HML tend to be lengthy from the practical viewpoint. In fact, the direct use of HML is tantamount to *reversing* the methodology depicted in Figure 1 of § 1: we start from endpoint specifications and later try to check their mutual consistency, which may not easily yield understandable global specifications. Further, since $\Rightarrow$ is decidable for practically important classes assertions [31], the present theory also offers algorithmic validation methods for key engineering concerns [32] including consistency of specifications (cf. §3.1) and correctness of process behaviours with full recursion against non-trivial specifications (cf. Theorem 6.7), whose analogue may not be known for the general HML formulae on the π-calculus. The use of the underlying type structures plays a crucial role in obtaining these methods. From the viewpoint of logical specifications for name passing, the present theory takes an *extensional* approach: we are concerned with what behaviours will unfold starting from given channels, than their (in)equality [11]. While our approach does reflect recommended practices in application-level distributed programming (where the direct use of network addresses is discouraged), it is an interesting topic to study how we can treat names as data as studied in [11].

Corresponding assertions and refinement/dependent types. The work [6] combines session-types with *correspondence assertions*. The type system can check that an asser-

tion **end** L, where L is a list of values (not a logical formula), is matched by the corresponding **begin** effect. The refinement types for channels (e.g. [5]) specify value dependency with logical constraints. For example, one might write $?(x\colon \mathsf{int}, !\{y : \mathsf{int} \mid y > x\})$ using the notations from [14, 33]. This only specifies a dependency at a *single point* (channel), unable to describe *a constraint for a series of interactions among multiple channels*. Our theory, based on multiparty sessions, can verify processes against a contract globally agreed by multiple distributed peers.

Contract-based approaches to communications and functions. Theories of contracts for web services based on advanced behavioural types are proposed in, e.g. [7, 9, 10]. Some authors focus on *compliance* of client and services, often defining compliance in terms of deadlock-freedom. In [1] a type system guaranteeing a progress property of clients is defined. Process calculi and concurrent constraint programming are combined in [8, 12] to model constraints that specify a Service Level Agreement on QoS parameters. Verification of financial protocols are also studied as contracts for functional languages (e.g. [28, 34]). Our theory treats contracts for distributed interactions with the link-mobility, using the π-calculus as an underlying formalism.

The global consistency checking is also used in the advanced security formalisms. In [15] a rely-guarantee technique is applied to a trust-management logic. The main technical difference is that users have to directly annotate each participant with assertions because of the the absence of global assertions. In [4] cryptography is used to ensure integrity of sessions but logical contracts are not considered.

Our approach differs from these preceding works in its use of global assertions for multiparty sessions, and its underpinning by a compositional proof system. This permits us to express and enforce fine-grained contracts of choreographic scenarios. The proposed assertion method builds on and enriches the underlying type discipline: types in [18] cannot deal with different conditions under which different sub-sessions are chosen at a branching point. Global/endpoint assertions can express constraints over message values (including channels), branches and invariants, impossible in [18]. The enriched expressiveness of specifications introduces novel technical challenges. Our consistency conditions for global assertions ensure that the end-point assertions are automatically consistent when projected, on whose basis a sound and complete proof system is built for a large class of name passing process behaviours.

As a different DbC-based approach to concurrency, an extension of DbC has been proposed in [26], using contracts for SCOOP [24] in order to reason about liveness properties of concurrent object-oriented systems. The main difference of our approach from [26] is that our framework captures and specifies for distributed message passing systems while [26] treats shared resources. The notion of pre-/post-conditions and invariants for global assertions centring on communications and the use of projections cannot be found in [26].

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. Berger, K. Honda, and N. Yoshida. Completeness and logical full abstraction for modal logics for the typed π-calculus. In *ICALP'08*, volume 5126 of *LNCS*, pages 99–111, 2008.
3. L. Bettini et al. Global Progress in Dynamically Interfered Multiparty Sessions. In *CONCUR'08*, volume 5201 of *LNCS*, pages 418–433. Springer, 2008.

4. K. Bhargavan, R. Corin, P.-M. Deniélou, C. Fournet, and J. Leifer. Cryptographic protocol synthesis and verification for multiparty sessions. In *CSF*, pages 124–140, 2009.
5. K. Bhargavan, C. Fournet, and A. D. Gordon. Modular verification of security protocol code by typing. In *POPL*, pages 445–456, 2010.
6. E. Bonelli, A. Compagnoni, and E. Gunter. Correspondence assertions for process synchronization in concurrent communications. *JFC*, 15(2):219–247, 2005.
7. M. Bravetti and G. Zavattaro. A foundational theory of contracts for multi-party service composition. *Fundamenta Informaticae*, XX:1–28, 2008.
8. 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, 2007.
9. L. Caires and H. T. Vieira. Conversation types. In *ESOP*, volume 5502 of *LNCS*, pages 285–300, 2009.
10. G. Castagna and L. Padovani. Contracts for mobile processes. In *CONCUR*, volume 5710 of *LNCS*, pages 211–228. Springer, 2009.
11. M. Dam. Proof systems for pi-calculus logics. In *Logic for Concurrency and Synchronisation*, Trends in Logic, Studia Logica Library, pages 145–212. Kluwer, 2003.
12. R. De Nicola et al. A Basic Calculus for Modelling Service Level Agreements. In *Coordination*, volume 3454 of *LNCS*, pages 33 – 48. Springer, 2005.
13. D. S. Frankel. *Model Driven Architecture: Applying MDA to Enterprise Computing*. Wiley, 2003.
14. T. Freeman and F. Pfenning. Refinement types for ml. *SIGPLAN Not.*, 26(6):268–277, 1991.
15. J. D. Guttman et al. Trust management in strand spaces: A rely-guarantee method. In *ESOP*, volume 2986 of *LNCS*, pages 325–339. Springer, 2004.
16. M. Hennessy and R. Milner. Algebraic laws for non-determinism and concurrency. *JACM*, 32(1), 1985.
17. T. Hoare. An axiomatic basis of computer programming. *CACM*, 12, 1969.
18. K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. In *POPL '08*, pages 273–284. ACM, 2008.
19. *The Java Modeling Language (JML) Home Page*.
20. C. B. Jones. Specification and design of (parallel) programs. In *IFIP Congress*, pages 321–332, 1983.
21. K. R. M. Leino. Verifying object-oriented software: Lessons and challenges. In *TACAS*, volume 4424 of *LNCS*, page 2, 2007.
22. E. Mendelson. *Introduction to Mathematical Logic*. Wadsworth Inc., 1987.
23. B. Meyer. Applying "Design by Contract". *Computer*, 25(10):40–51, 1992.
24. B. Meyer. *Object-Oriented Software Construction (Chapter 31)*. Prentice Hall, 1997.
25. D. Mostrous and N. Yoshida. Two Sessions Typing Systems for Higher-Order Mobile Processes. In *TLCA'07*, volume 4583 of *LNCS*, pages 321–335. Springer, 2007.
26. P. Nienaltowski, B. Meyer, and J. S. Ostroff. Contracts for concurrency. *Form. Asp. Comput.*, 21(4):305–318, 2009.
27. OMG. Object constraint language version 2.0, may 2006.
28. S. Peyton Jones et al. Composing contracts: an adventure in financial engineering. In *ICFP*, pages 281–292. ACM, 2000.
29. B. C. Pierce. *Types and Programming Languages*. MIT Press, 2002.
30. 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.
31. Full version of this paper. `http://www.cs.le.ac.uk/people/lb148/fullpaper.html`.
32. SAVARA JBoss Project webpage. `http://www.jboss.org/savara`.
33. H. Xi and F. Pfenning. Dependent types in practical programming. In *POPL*, 1999.
34. D. Xu and S. Peyton Jones. Static contract checking for Haskell. In *POPL*, 2009.

Table of Contents

A Notation

Delegation and Channel Passing Throughout this appendix, we present definitions and discussions for processes and assertions which include shared channel passing and delegations, as discussed in Section 7.

Decidability of Refinement For the extension with delegation, an assertion includes an annotation which is again another assertion, i.e. assertions can be nested. In particular, the refinement over endpoint assertions is used for runtime checking.

The set of extended endpoint assertions are stratified as follows. First, we have the set of endpoint assertions which do not use delegations (the original set of endpoint assertions). Second, we have the set of endpoint assertions which may use endpoint assertions for specifying delegations. And so on.

For the initial set, we assume recursive assertions are always of the same shape, and parameters (variables) bind variables in the predicates for the actions in the body precisely in the same way. Under this condition, the refinement is decidable if the underlying logic is decidable, since we have only to compare the predicates for the same actions individually. The rest is by way of stratification: under the same condition for recursive assertions, the refinement for the second set of assertions again becomes decidable. In this way we obtain a decidable subset of extended assertions.

Further details are found in [31].

Recursion Initialisation The proof of completeness is done for a slightly more general syntax for recursion. We can express the syntax presented in the paper with this more general one, as discussed below.

In the syntax of $\mathcal{G}$ in §3, we substitute recursion definition $\mu \mathbf{t}\langle \tilde{e}\rangle(\tilde{v}\colon U)\{A'\}.\mathcal{G}$ with $\mu \mathbf{t}\langle \tilde{u}:A\rangle(\tilde{v}\colon U)\{A'\}.\mathcal{G}$, where $\tilde{u}$ has the same length as $\tilde{v}$, and $\langle \tilde{u}\ :\ A\rangle$ defines the set of valid initialisations for $\tilde{v}$, that is each v_i is initialised with a value for which u_i satisfies A (e.g., $A = \wedge_i(u_i > 0)$). This syntax allows to specify sets of possible initial values for each recursion parameter, other than specific expressions (notice that $\mu \mathbf{t}\langle \tilde{e}\rangle(\tilde{v}\colon U)\{A'\}.\mathcal{G} = \mu \mathbf{t}\langle \tilde{u}:\tilde{u}=\tilde{e}\rangle(\tilde{v}\colon U)\{A'\}.\mathcal{G}$). Similarly we substitute recursion call $\mathbf{t}\langle \tilde{e}\rangle$ with $\mathbf{t}\langle \tilde{u}\ :\ A\rangle$.

In the syntax for $\mathcal{T}$, we substitute recursion definition $\mu \mathbf{t}\langle \tilde{e}\rangle(\tilde{v}:U)\{A'\}.\mathcal{T}$ with $\mu \mathbf{t}\langle \tilde{u}:A\rangle(\tilde{v}:U)\{A'\}.\mathcal{T}$ and recursion call $\mathbf{t}\langle \tilde{e}\rangle$ with $\mathbf{t}\langle \tilde{u}:A\rangle$. Projection becomes:

Definition A.1 (Extended Projection). Given $\mathcal{G}$ and A, the *projection of $\mathcal{G}$ for a participant* $\mathtt{p}$ *wrt* A is denoted by $(\mathcal{G})\downarrow^{A}_{\mathtt{p}}$ and, assuming $\mathtt{p}_1 \neq \mathtt{p}_2$, recursively defined as follows

(1) $$(\mathtt{p}_1 \to \mathtt{p}_2\colon k(\tilde{v}\colon U)\{A\}.\mathcal{G}')\downarrow^{A_{Proj}}_{\mathtt{p}} = \begin{cases} k!(\tilde{v}\colon U)\{A\}.(\mathcal{G}')\downarrow^{A\wedge A_{Proj}}_{\mathtt{p}} & \text{if } \mathtt{p}=\mathtt{p}_1 \\ k?(\tilde{v}\colon U)\{(A\wedge A_{Proj})\upharpoonright \mathtt{p}\}.(\mathcal{G}')\downarrow^{A\wedge A_{Proj}}_{\mathtt{p}} & \text{if } \mathtt{p}=\mathtt{p}_2 \\ (\mathcal{G}')\downarrow^{A\wedge A_{Proj}}_{\mathtt{p}} & \text{otw} \end{cases}$$

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

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

(4) $$(\mu \mathbf{t}\langle \tilde{u}:A\rangle(\tilde{v}\colon U)\{B\}.\mathcal{G})\downarrow^{A_{Proj}}_{\mathtt{p}} = \mu \mathbf{t}\langle \tilde{u}:A\upharpoonright\mathtt{p}\rangle(\tilde{v}\colon U)\{B\upharpoonright\mathtt{p}\}.(\mathcal{G})\downarrow^{A_{Proj}}_{\mathtt{p}}$$

(5) $$(\mathbf{t}\langle \tilde{u}:A\rangle)\downarrow^{A_{Proj}}_{\mathtt{p}} = \mathbf{t}\langle \tilde{u}:(A\wedge A_{Proj})\upharpoonright\mathtt{p}\rangle$$

(6) $$(\mathsf{end})\downarrow^{A_{Proj}}_{\mathtt{p}} = \mathsf{end}$$

If no condition applies, $(\mathcal{G})\downarrow_{\mathtt{p}}^{A}$ is undefined; $\mathcal{G}\upharpoonright\mathtt{p} = (\mathcal{G})\downarrow_{\mathtt{p}}^{\mathsf{true}}$ is the *projection of* $\mathcal{G}$ *on* $\mathtt{p}$. Recall we write $A\upharpoonright\mathtt{p}$ for the projection of predicate A on participant $\mathtt{p}$ defined as $\exists V_{ext}(A)$ where $V_{ext} = var(A)\backslash I(\mathcal{G})\upharpoonright\mathtt{p}$.

Differently from Definition 4.1, the definition above does not change the number of recursion parameters in the projection of recursive assertions if the participant does not know some of the parameters, i.e., cases (4) and (5). This is just a conceptual simplification that does not alter the semantics of endpoint assertions. In fact, a participant that does not know some parameters will simply use them as dummy parameters when recurring.

The reason to consider a more general syntax for recursion is that in the proof of completeness it is necessary to merge endpoint assertions. The merging is needed because, in order to generate the most refined assertion for a given process, we need to "merge" the endpoint assertions that have been generated for the different branches of, for example, a conditional process. In order to merge two recursive process it is necessary to merge the initialisation expressions. The more general syntax allows us to use the logical "and" to merge the initialisations.

Location Annotations for Recursion in Global Assertions We often use an annotation for recursion definition in global assertions (e.g., for checking well-assertedness). Specifically, we annotate recursion definition as,

$$\mu\mathbf{t}\langle\tilde{u}:A\rangle(v_1@\mathtt{L}_1:U_1,...,v_n@\mathtt{L}_n:U_n)\{A'\}.\mathcal{G}$$

where $\mathtt{L}_i$, called *locations*, are sets of the form $\{\mathtt{p},\mathtt{p}'\}$ yielding the participants that either send or receive the value denoted by each parameter variable v_i in $\mathcal{G}$. More specifically, the annotation $v@\{\mathtt{p},\mathtt{p}'\}:U$ for recursion parameter v means that in all recursive calls inside $\mathcal{G}$, the variables that are used to derive the actual value to be assigned to v are all known by both $\mathtt{p}$ and $\mathtt{p}'$ (thus $\mathtt{p}$ and $\mathtt{p}'$ know v inside the recursion body). A location is empty (i.e., $\varnothing$) if a parameter is always assigned an expression with no interaction variables (e.g., $\mathbf{t}\langle u:u=10\rangle$). In that case the value is determined by the global assertion. Each recursion parameter has exactly one location (this does not cause a loss of generality). The annotation can be done automatically (see [31]).

B Well-Assertedness of Global Assertions

B.1 Algorithm for Annotating Recursion parameters

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

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

1. if $\mathcal{G} = \mathtt{p}\to\mathtt{p}' : k\,(\tilde{v})\{A\}.\mathcal{G}'$, then return $\mathsf{L}(\mathcal{G}';\Gamma,\tilde{v}@\{\mathtt{p},\mathtt{p}'\};\mathsf{MK}\backslash\{(\mathtt{p},v),(\mathtt{p}',v)\ :\ v\in\tilde{v}\};\mathsf{C})$
2. if $\mathcal{G} = \mathtt{p}\to\mathtt{p}' : k\,\{\{A_j\}l_j :\ \mathcal{G}_j\}_{j\in J}$ then return $\Cup_{j\in J}\mathsf{L}(\mathcal{G}_j;\Gamma;\mathsf{MK};\mathsf{C})$
3. if $\mathcal{G} = \mathcal{G}_1,\mathcal{G}_2$ then return $\Cup_{i\in\{1,2\}}\mathsf{L}(\mathcal{G}_i;\Gamma;\mathsf{MK};\mathsf{C})$

4. if $\mathcal{G} = \mathbf{t}\langle u_1,\ldots,u_n : A_1,...,A_n\rangle$[7], assuming $\mathsf{C}(\mathbf{t}) = \tilde{v}, V_1,\ldots,V_n$, then return $(\Gamma, \mathsf{MK}, \mathsf{C}[\mathbf{t} \mapsto \tilde{v}, V_1 \cup var(A_1)\backslash u_1,\ldots,V_n \cup var(A_n)\backslash u_n])$
5. if $\mathcal{G} = \mathsf{end}$ then return $(\Gamma, \mathsf{MK}, \mathsf{C})$
6. if $\mathcal{G} = \mu\mathbf{t}\langle u_1,\ldots,u_n : A_1,...,A_n\rangle(\tilde{v})\{A\}.\mathcal{G}'$ then return $\mathsf{L}(\mathcal{G}';\Gamma;\mathsf{MK};\mathsf{C}[\mathbf{t} \mapsto \tilde{v}, var(A_1)\backslash u_1,\ldots,var(A_n)\backslash u_n])$.

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

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

.

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

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

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

B.2 Checking history-sensitivity

We use environments Γ defined by the grammar:

$$\Gamma ::= \varnothing \mid \Gamma, v@\mathtt{L} \mid \Gamma, \mathbf{t} : v_1 @ \mathtt{L_1},\ldots,v_n @ \mathtt{L_n}$$

to assign to an interaction variable v its location $\mathtt{L}$ and to an assertion variables $\mathbf{t}$ a sequence of pairs $v_i @ \mathtt{L_i}$ to handle recursive types. We omit the type annotation from the recursive variables. 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 $u \in var(e)$.

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

$$\frac{\Gamma \vdash \mathcal{G} \quad \Gamma \vdash \mathcal{G}'}{\Gamma \vdash \mathcal{G}, \mathcal{G}'} \qquad \frac{}{\Gamma \vdash \mathsf{end}} \qquad \frac{dom(\Gamma) \cup \tilde{v} \supseteq var(A)\backslash\tilde{u}}{\Gamma, \mathbf{t} : v_1 @ \mathtt{L_1} \ldots v_n @ \mathtt{L_n} \vdash \mathbf{t}\langle\tilde{u} : A\rangle}$$

$$\frac{\Gamma, \mathbf{t} : v_1 @ \mathtt{L_1} \ldots v_n @ \mathtt{L_n} \vdash \mathcal{G} \quad dom(\Gamma) \supseteq var(A) \quad dom(\Gamma, \mathbf{t} : v_1 @ \mathtt{L_1} \ldots v_n @ \mathtt{L_n}) \supseteq var(A')\backslash\tilde{u}}{\Gamma \vdash \mu\mathbf{t}\langle\tilde{u} : A\rangle(v_1 @ \mathtt{L_1},\ldots,v_n @ \mathtt{L_n})\{A'\}.\mathcal{G}}$$

Fig. 6. Checker for history sensitivity on global assertions

The rules in Figure 6 discipline the usage of assertion variables and restricts the set of interaction variables that can be used in each assertion so to enforce the history sensitivity principle. The first two rules require that the sender/selector p must know all the interaction variables of the predicate. The other rules are straightforward. Note that the rules are purely syntactic, hence the verification of history sensitivity of $\mathcal{G}$ is a linear time problem.

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

B.3 Checking temporal satisfiability

To enforce temporal satisfiability, validity of formulae has to be checked. Below we pre-annotate each occurrence of type variables $\mathbf{t}$ with $\mathbf{t}_{A(\tilde{v})}$ where A is the invariant of the recursion binding $\mathbf{t}$ and $\tilde{v}$ are the corresponding formal parameters[8].

Definition B.1 (Well-asserted Global Assertions). We recursively define a boolean function $GSat(\mathcal{G},A)$ as follows:

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

$\mathcal{G}$ is *well asserted* if it satisfies history sensitivity (i.e., Fig. 6) and $GSat(\mathcal{G},\mathsf{true}) = \mathsf{true}$.

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

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

is well-asserted even if only its second branch is satisfiable. Note we do not specify any relationship among the predicates in a branch (such as a XOR relationship to enforce determinism in potential paths) in order to allow vague specifications (as far as consistent). For the same reason, well-assertedness does not prohibit unreachable branches. Note $GSat(\mathcal{G},\mathsf{false}) = \mathsf{true}$ (i.e., global assertions are trivially satisfiable in a bad environment). The remaining clauses of Definition B.1 are intuitive.

Well-assertedness is decidable under the assumptions that the logic is decidable. The fixed shape of the implications (e.g. no alternating quantifiers) suggests that validation can be done efficiently [30]. Since the algorithm is compositional it can be integrated with the checker in Figure 6.

C Well-Assertedness of Endpoint Assertions

As done for global assertions, we give a decision procedure for the satisfiability of endpoint assertion.

Definition C.1 (Well Asserted Endpoint Assertions). We define a boolean function $LSat(\mathcal{T},A)$ recursively as follows:

[8] This annotation is always possible if $\mathcal{G}$ does not have free assertion variables.

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

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

D Refinement

We assume the standard subtyping for session types, in which a subtype describes a more constrained behaviour, e.g. more branches and less selections. More precisely, a super session-type has

- more selection labels and less branch labels than a sub session-type;
- super types for their non-channel values to be sent by outputs, and subtypes for their non-channel values to be inputted (covariant value typing); and
- subtypes of channel values to be sent by outputs and super types for their channel values to be inputted (contravariant channel typing).

Convention D.1. We adopt the standard definition of unfolding unfolding of recursive assertion. The one-time unfolding of $\mu\mathbf{t}\langle\tilde{u}:A\rangle(\tilde{v}:U).\mathcal{T}$ is $\mathcal{T}[\mu\mathbf{t}(\tilde{v}:U).\mathcal{T}/\mathbf{t}]$ where the invariant is omitted since it does not affect the unfolding and

$$\mathbf{t}\langle\tilde{u}:A'\rangle[\mu\mathbf{t}(\tilde{v}:U).\mathcal{T}/\mathbf{t}] \stackrel{\text{def}}{=} \mu\mathbf{t}\langle\tilde{u}:A'\rangle(\tilde{v}:U).\mathcal{T}$$

Definition D.2 (Refinement). A binary relation $\mathcal{R}$ over closed well-asserted endpoint assertions is a *refinement relation* if $\mathcal{T}_1\mathcal{R}\mathcal{T}_2$ implies one of the following conditions holds, up to the unfoldings of recursive assertions.

- $\mathcal{T}_1 = k!(\tilde{v}:\mathcal{T}@\mathtt{p})\{A_1\};\mathcal{T}_1'$, $\mathcal{T}_2 = k!(\tilde{v}:\mathcal{T}'@\mathtt{p})\{A_2\};\mathcal{T}_2'$ s.t. $A_1 \supset A_2$, $\mathcal{T}_1'\sigma\mathcal{R}\mathcal{T}_2'\sigma$ and $\mathcal{T}\mathcal{R}\mathcal{T}'$ for each $\sigma = [\tilde{t}/\tilde{v}]$ with $A_1\sigma\downarrow\mathsf{true}$.
- $\mathcal{T}_1 = k!(\tilde{v}:U)\{A_1\};\mathcal{T}_1'$ and $\mathcal{T}_2 = k!(\tilde{v}:U)\{A_2\};\mathcal{T}_2'$ (where U is not a located endpoint assertion) 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 $A_1\sigma\downarrow\mathsf{true}$.
- $\mathcal{T}_1 = k?(\tilde{v}:\mathcal{T}@\mathtt{p})\{A_1\};\mathcal{T}_1'$ and $\mathcal{T}_2 = k?(\tilde{v}:\mathcal{T}'@\mathtt{p})\{A_2\};\mathcal{T}_2'$ s.t. $A_2 \supset A_1$, $\mathcal{T}_1'\sigma\mathcal{R}\mathcal{T}_2'\sigma$ and $\mathcal{T}'\mathcal{R}\mathcal{T}$ or each $\sigma = [\tilde{t}/\tilde{v}]$ with $A_2\sigma\downarrow\mathsf{true}$.
- $\mathcal{T}_1 = k?(\tilde{v}:U)\{A_1\};\mathcal{T}_1'$ and $\mathcal{T}_2 = k?(\tilde{v}:U)\{A_2\};\mathcal{T}_2'$ (where U is not a located endpoint assertion) s.t. $A_2 \supset A_1$ and $\mathcal{T}_1'\sigma\mathcal{R}\mathcal{T}_2'\sigma$ for each $\sigma = [\tilde{\mathtt{n}}/\tilde{v}]$ with $A_2\sigma\downarrow\mathsf{true}$.
- $\mathcal{T}_1 = k\oplus\{\{A_{1i}\}l_{1i}:\mathcal{T}_{1i}\}_{i\in I}$ and $\mathcal{T}_2 = k\oplus\{\{A_{2j}\}l_{2j}:\mathcal{T}_{2j}\}_{j\in J}$ where $I \subset J$, $A_{1i} \supset A_{2i}$ and $\mathcal{T}_{1i}\mathcal{R}\mathcal{T}_{2i}$ $(i \in I)$.
- $\mathcal{T}_1 = k\&\{\{A_{1i}\}l_{1i}:\mathcal{T}_{1i}\}_{i\in I}$ and $\mathcal{T}_2 = k\&\{\{A_{2j}\}l_{2j}:\mathcal{T}_{2j}\}_{j\in J}$ where $J \subset I$, $A_{2j} \supset A_{1j}$ and $\mathcal{T}_{1j}\mathcal{R}\mathcal{T}_{2j}$ $(j \in J)$.

where predicates are evaluated in fixed environments Γ and Δ.

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

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

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

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

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

$$s_k!\langle\!\langle\tilde{t}\rangle\!\rangle(\tilde{v})\{A\};P \xrightarrow{s!\langle\!\langle\tilde{t}\rangle\!\rangle} P[\tilde{t}/\tilde{v}] \quad (A[\tilde{t}/\tilde{v}] \downarrow \mathsf{true}, \tilde{t} \notin \mathrm{fn}(A) \cup \mathrm{fn}(P)) \qquad \text{[Deleg]}$$

$$s_k?(\!(\tilde{v})\!)\{A\};P \xrightarrow{s?(\!(\tilde{t})\!)} P[\tilde{t}/\tilde{v}] \quad (A[\tilde{t}/\tilde{v}] \downarrow \mathsf{true} \wedge \tilde{t} \notin \mathrm{fn}(A) \cup \mathrm{fn}(P)) \qquad \text{[SRec]}$$

$$s_k \lhd \{A\}l : P \xrightarrow{s_k \lhd l} P \quad (A \downarrow \mathsf{true}) \qquad \text{[Sel]}$$

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

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

$$(\nu a : \langle G \rangle)P \xrightarrow{\alpha} (\nu a : \langle G \rangle)P' \quad (\text{when } P \xrightarrow{\alpha} P' \text{ and } a \notin \mathrm{fn}(\alpha)) \qquad \text{[NRes]}$$

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

$$(\nu a : \langle G \rangle)P \xrightarrow{(\nu a\tilde{b}:\langle G\rangle\langle\tilde{G}_1\rangle)s!\tilde{n}} P' \quad (\text{when } P \xrightarrow{(\nu\tilde{b}:\langle\tilde{G}_1\rangle)s!\tilde{n}} P' \text{ and } a \in \{\tilde{n}\}) \qquad \text{[BOut]}$$

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

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

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

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

$$s_k!\langle\!\langle\tilde{t}\rangle\!\rangle(\tilde{v})\{A\};P \xrightarrow{\tau} \mathsf{errH} \quad (A[\tilde{t}/\tilde{v}] \downarrow \mathsf{false}) \qquad \text{[DelegErr]}$$

$$s_k?(\!(\tilde{v})\!)\{A\};P \xrightarrow{s_k?(\!(\tilde{t})\!)} \mathsf{errT} \quad (A[\tilde{t}/\tilde{v}] \downarrow \mathsf{false} \wedge \tilde{t} \notin \mathrm{fn}(A) \cup \mathrm{fn}(P)) \qquad \text{[SRecErr]}$$

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

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

Fig. 7. Labelled Transition for Processes

E Labelled Transition System for Asserted Processes

See Figure 7. We assume that in rule [Deleg] the sender cannot send a channel that is free in the continuation (i.e., $\tilde{t} \notin \mathrm{fn}(P)$). Note [LinkOut] and [LinkIn] do not capture multicasting. This does not lead to the lack of soundness [31].

F Labelled Transition System for Assertions

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

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

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

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

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

$$\frac{A[\tilde{t}/\tilde{v}] \downarrow \mathsf{true} \quad \tilde{t} \cap (\mathrm{fn}(\mathcal{T}) \cup \mathrm{fn}(A)) = \varnothing}{\langle \Gamma, (\Delta, \tilde{s} : k!(\tilde{v} : \mathcal{T}' @ \mathtt{q})\{A\}; \mathcal{T} @ \mathtt{p}, \tilde{t} : \mathcal{T}' @ \mathtt{q}) \rangle \xrightarrow{s_k ! \langle\!\langle \tilde{t} \rangle\!\rangle} \langle \Gamma, (\Delta, \tilde{s} : \mathcal{T}[\tilde{t}/\tilde{v}] @ \mathtt{p}) \rangle} \quad [\text{TR-DELEG}]$$

$$\frac{A[\tilde{t}/\tilde{v}] \downarrow \mathsf{true} \quad \tilde{t} \cap (\mathrm{fn}(\mathcal{T}) \cup \mathrm{fn}(A)) = \varnothing}{\langle \Gamma, (\Delta, \tilde{s} : k?(\tilde{v} : \mathcal{T}' @ \mathtt{q})\{A\}; \mathcal{T} @ \mathtt{p}) \rangle \xrightarrow{s_k ? (\!(\tilde{t})\!)} \langle \Gamma, (\Delta, \tilde{s} : \mathcal{T}[\tilde{t}/\tilde{v}] @ \mathtt{p}, \tilde{t} : \mathcal{T}' @ \mathtt{q}) \rangle} \quad [\text{TR-SREC}]$$

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

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

Fig. 8. Labelled transition for endpoint assertions

G Type Discipline Underlying Definition 6.1

The present theory builds on the underlying type discipline from [18]. The practical significance of this approach is that this enables us to fix the shape of type signature when specifying fine-grained behavioural properties. Theoretically this allows us to build the theory of assertions, including its semantic basis, concentrating on high-level logical specifications on the basis of the abstraction already provided by type signatures.

Write $\mathsf{erase}(P)$ for the unasserted (typable) process underlying P, i.e. the result of taking off all predicates and associated variables from P. Similarly we write $\mathsf{erase}(\Gamma)$ and $\mathsf{erase}(\Delta)$. In Definition 6.1 (conditional simulation), a simulation relates P to $\langle \Gamma, \Delta \rangle$, under the condition $\mathsf{erase}(\Gamma) \vdash \mathsf{erase}(P) \rhd \mathsf{erase}(\Delta)$ in the type discipline from [18]. Thus a simulation is inherently typed, allowing the dispensation of such issues as type errors in our formulation.

H Validation Rules of Runtime Processes

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

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

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

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

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

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

Fig. 9. Reduction Rules for Assertion Assignments

Message Assertions For treating the τ-action (which is identical with reduction), we introduce the validation rules for runtime processes. For this purpose we adapt the framework of typing for runtime processes in [3], using *message assertions* (corresponding to *message types* in [3]), which abstract messages in queues.

We first extend endpoint assertions as follows.

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

We call $\mathcal{M}$ a *message assertion*, which is simply a sequence of a sending action with a concrete 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:

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

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

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.

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

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 9. The rules come from [3], 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. The forth rule is for a delegation.
4. The fifth rules depict how a selection message assertion interacts with the assertion for branching.
5. The sixth and seventh 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 H.1 (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 [3]: the 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.

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

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

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

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

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

Fig. 10. Validation Rules for Runtime Processes

Validation for Queues and Session Hiding We list the validation rules for queues and channel hiding in Figure 10, where [NRES] in Figure 10 generalises that of [NRES] in Figure 5 (since if $a \notin \mathrm{fn}\Delta$ then the hiding can be erased by the equality above) hence replacing the original version. The remaining rules from Figure 5 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 [3, 18] except message assertions now mention values.

Convention H.2. 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 5 excepting [NRES] and those of Figure 10.

Note that, following [3, 18], the validation of the composability of multiple processes is relegated to the session hiding rule [CRES] rather than to the parallel composition rule [CONC]. By the shape of these rules we immediately observe:

Proposition H.3. *Suppose* $\Gamma \vdash P \rhd \Delta$. *Then P contains no error.*

I Completeness: Examples of Dead Code

The proof of completeness is given in [31]. Here we list basic examples which illustrate why we consider restricted classes of processes in our completeness results.

Example I.1 (Hiding). Consider the following process which tries to initiate a session at a and a is hidden: $P = (\nu\, a)\overline{a}[2,\ldots,n](s).P_{BAD}$ Assume P has the correct interaction structure with respect to Γ,Δ (i.e., $\mathsf{erase}(\Gamma) \vdash \mathsf{erase}(P) \rhd \mathsf{erase}(\Delta)$) but its sub-processes P_{BAD} sends a violating value. We observe that P cannot perform any transition (thus trivially satisfies $\Gamma \models P \rhd \Delta$), but since P_{BAD} sends a violating message, $\Gamma \not\vdash P \rhd \Delta$.

Along the same veign, we can construct a process which, when a certain Turing machine terminates, emits at a hidden channel, whose reception by another process leads to an output at a visible channel.

Other cases of dead code (i.e., unreachable paths) do not violate completeness. Consider the process $\mathsf{if\ true\ then}\ P_{GOOD}\ \mathsf{else}\ P_{BAD}$ where $\mathsf{erase}(P_{GOOD})$ and $\mathsf{erase}(P_{BAD})$ are well-typed but P_{BAD} contains predicate violations. Since the assertion environment for the branch P_{BAD} would includes $\neg\mathsf{true}$, then the validation would be successful for that branch despite the violations. This case is illustrated by example I.2.

Example I.2 (Unreachable Paths).

$$\begin{array}{ll}\mathcal{T} & = s!(v:\mathsf{Int})\{v > 10\};\mathsf{end}\\ P & = \overline{a}[2](s).\mathsf{if\ true\ then}\ s!\langle 20\rangle(v)\{v > 10\};\mathbf{0}\ \mathsf{else}\ s!\langle 0\rangle(v)\{v > 10\};\mathbf{0}\end{array}$$

The process P can be validated against $\mathcal{T}$ since the second branch (i.e., the one that violates the predicate $v > 10$) would be validated since the assertion environment would include $\neg\mathsf{true}$ thus $\neg\mathsf{true} \supset 0 > 10$.

Unreachable paths in endpoint assertions, e.g., where a branch of a select/branching has a false predicate [9], are similarly handled.

[9] An endpoint assertion with predicate false in one of its branches can be well-assertedness as long as there is at least one path forward.

J Proof of Proposition 4.3 (Projection Preserves Well-Assertedness)

Proposition 4.3 immediately follows from Lemma J.1. By Lemma J.1, *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 quantiers which make them weaker).

Lemma J.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. By induction on the projection rules, proceeding by case analysis on $\mathcal{G}$.

Values sending/receiving. If $\mathcal{G} = \mathtt{p}_1 \to \mathtt{p}_2 \colon k\,(\tilde{v} : U)\{A\}.\mathcal{G}'$ then we have two cases.
Case $\mathtt{p} = \mathtt{p}_1$**.** By Definition A.1, $\mathcal{G} \upharpoonright \mathtt{p} = k!(\tilde{v} : U)\{A\};\mathcal{T}'$. 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

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

By (J.1) and C.1, $LSat(\mathcal{G} \upharpoonright \mathtt{p},A_{\mathcal{T}}) = LSat(\mathcal{T},A_{\mathcal{T}} \wedge A)$. 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 A.1) $\mathcal{G} \upharpoonright \mathtt{p} = k?(\tilde{v} : U)\{\exists V_{ext}(A_{\mathcal{T}} \wedge A)\};\mathcal{T}'$. 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

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

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

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

By (J.3) and C.1, $LSat(\mathcal{G} \upharpoonright \mathtt{p},A_{\mathcal{T}}) = LSat(\mathcal{T},A_{\mathcal{T}} \wedge A)$. 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 A.1, $\mathcal{G} \upharpoonright \mathtt{p} = k \oplus \{\{A_j\}l_j : \mathcal{T}_j\}_{j\in J}$. 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

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

By (J.4) and C.1, $LSat(\mathcal{G} \upharpoonright \mathtt{p},A_{\mathcal{T}}) = \wedge_{j\in J} LSat(\mathcal{T}_j,A_{\mathcal{T}} \wedge A_j)$. 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 A.1, $\mathcal{G} \upharpoonright \mathtt{p} = k\&\{\{\exists V_{ext}(A_{\mathcal{T}} \wedge A_j)\}l_j : \mathcal{T}_j\}_{j\in J}$. 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

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

By weakening the consequence of J.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

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

By (J.6) and C.1, $LSat(\mathcal{G} \upharpoonright \mathtt{p}, A_{\mathcal{T}}) = \wedge_{j \in J} LSat(\mathcal{T}_j, A_{\mathcal{T}} \wedge A_j)$. 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 $\mathcal{G} = \mu \mathbf{t}\langle \tilde{u} : A \rangle (\tilde{v} : U)\{B\}.\mathcal{G}'$ then $\mathcal{G} \upharpoonright \mathtt{p} = \mu \mathbf{t}\langle \tilde{u} : A \upharpoonright \mathtt{p} \rangle (\tilde{v} : U)\{B \upharpoonright \mathtt{p})\}.\mathcal{T}'$. By well formedness of $\mathcal{G}$

$$A_{\mathcal{G}} \wedge A[\tilde{v}/\tilde{u}] \supset B[\tilde{v}/\tilde{u}]. \tag{J.7}$$

By weakening (J.7) where $\tilde{w}$ is the vector of variables in $var(A) \cup var(B)$ but not in $\mathcal{G} \upharpoonright \mathtt{p}$ (i.e., they are not known by $\mathtt{p}$):

$$\exists \tilde{w}(A_{\mathcal{G}} \wedge A[\tilde{v}/\tilde{u}] \supset B[\tilde{v}/\tilde{u}]). \tag{J.8}$$

Equation J.8 is stronger than

$$A_{\mathcal{G}} \wedge \exists \tilde{w} A[\tilde{v}/\tilde{u}] \supset \exists \tilde{w} B[\tilde{v}/\tilde{u}]. \tag{J.9}$$

where $\tilde{w}$ does not appear free in the assertion environment $A_{\mathcal{G}}$ (interaction variables have fresh names). Equation J.9 is equivalent to

$$A_{\mathcal{G}} \wedge \exists A[\tilde{v}/\tilde{u}] \upharpoonright \mathtt{p} \supset \exists B[\tilde{v}/\tilde{u}] \upharpoonright \mathtt{p}. \tag{J.10}$$

Finally, since by hypothesis $A_{\mathcal{T}} \supset A_{\mathcal{G}}$, by strengthening the hypothesis of equation J.10:

$$A_{\mathcal{T}} \wedge \exists A[\tilde{v}/\tilde{u}] \upharpoonright \mathtt{p} \supset \exists B[\tilde{v}/\tilde{u}] \upharpoonright \mathtt{p}. \tag{J.11}$$

Type Variable. If $\mathcal{G} = t_{A(\tilde{v})}\langle \tilde{u} : A' \rangle$, then the projection is $t_{B(\tilde{v})}\langle \tilde{u} : B' \rangle$ where $B = A \upharpoonright \mathtt{p}$ and $B' = A' \upharpoonright \mathtt{p}$. The proof proceeds like in the case of recursion definition.

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.

K Proof of Proposition 6.4 (Refinement)

We prove Proposition 6.4 (relationship between the refinement relation and the satisfaction), after a lemma. The full definition of refinement is presented in § D.2.

Definition K.1. An endpoint assertion is *closed* (resp. *open*) if it does not (resp. it may) contain free variables.

Definition K.2. $\langle \Gamma, \Delta \rangle$ allows α *when* $\langle \Gamma, \Delta \rangle \xrightarrow{\alpha} \langle \Gamma', \Delta' \rangle$ *for some* $\langle \Gamma', \Delta' \rangle$.

Definition K.3. *Below we illustrate how to perform an unfolding of recursive assertion. It follows the standard definition. Assume given a recursive assertion as follows (omitting an invariant since it does not affect it as far as it is well-asserted):*

$$\mu \mathbf{t}\langle u : A \rangle (v : S).\mathcal{T}$$

Then its one-time unfolding is:

$$\mathcal{T}[\mu \mathbf{t}(v : S).\mathcal{T}/\mathbf{t}]$$

where we define this substitution starting from:

$$\mathbf{t}\langle u : A'\rangle[\mu \mathbf{t}(v : S).\mathcal{T}/\mathbf{t}] \stackrel{def}{=} \mu \mathbf{t}\langle u : A'\rangle(v : S).\mathcal{T}$$

The rest being homomorphic.

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 K.1. *Assume $\Delta \ni \Delta'$. If $\langle \Gamma, \Delta\rangle$ is capable of moving at the subject $\mathsf{sbj}(\alpha)$ then $\langle \Gamma, \Delta'\rangle$ is also capable to move at the subject $\mathsf{sbj}(\alpha)$.*

Remark. The endpoint assertions in Δ and in Δ' are well-asserted by definition of refinement (Definition D.2). Notice anyway that it is sufficient that only the endpoint assertions in Δ' are well-asserted for this lemma to hold.

Proof. The proof is straightforward from the definition refinement (Definition D.2). Notice that, up to the unfoldings of recursive assertions, an endpoint assertion may differ from its refinement only in (a) the predicates in case of input/output/selection/branching and (b) the sets of possible labels/branches in case of selection/branching. By Definition D.2, 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 K.2. *Assume $\Delta \ni \Delta'$ 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 $\Delta_1 \ni \Delta_1'$ 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 $\Delta_1 \ni \Delta_1'$ 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 $\ni$ over endpoint assertions as the refinement relation justifying the original refinement (note $\ni$ is the largest refinement relation).

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

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

by [TR-SEND]. By hypothesis we have $\Delta \ni \Delta'$, so by Definition D.2 we can set

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

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

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

It follows that also the following transition is possible:

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

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

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

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

by [TR-REC]. Assume further we have

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

As before, by hypothesis and by Definition D.2 we can set:

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

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

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

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

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

$$\langle \Gamma, \Delta \rangle \xrightarrow{s_k \lhd l_j} \langle \Gamma, \Delta_{side}, \tilde{s} : \mathcal{T}_{1j} \mathbin{@} \mathtt{p} \rangle \tag{K.9}$$

by [TR-SEL]. By hypothesis and by Definition D.2, we can set

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

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:

$$\langle \Gamma, \Delta' \rangle \xrightarrow{s_k \lhd l_j} \langle \Gamma, \Delta'_{side}, \tilde{s} : \mathcal{T}_{2j} \mathbin{@} \mathtt{p} \rangle. \tag{K.11}$$

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

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

$$\langle \Gamma, \Delta \rangle \xrightarrow{s_k > l_j} \langle \Gamma, \Delta_{side}, \tilde{s} : \mathcal{T}_{1j} \text{ @ p} \rangle. \tag{K.12}$$

Assume further we have

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

By hypothesis and by Definition D.2, we can set

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

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

$$\langle \Gamma, \Delta' \rangle \xrightarrow{s_k > l_j} \langle \Gamma, \Delta'_{side}, \tilde{s} : \mathcal{T}_{2j} \text{ @ p} \rangle. \tag{K.15}$$

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

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

We now prove Proposition 6.4. We reproduce the statement below.

Proposition K.1 (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 K.2, $\langle \Gamma, \Delta' \rangle \xrightarrow{\alpha} \langle \Gamma, \Delta'_1 \rangle$ 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 K.1 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 endpoint assertion in input/branching moves. By Lemma K.2, $\Delta_1 \ni \Delta'_1$. The predicate holds by induction.

L Proof of Proposition 6.3 (Subject Reduction)

L.1 Substitution Lemma

The substitution lemma uses the following lemma.

Lemma L.1. *If* $\mathcal{T}$ *is well-asserted under* $\Gamma, \tilde{u} : U$ *and* $\tilde{\mathtt{n}} : U$ *then* $\mathcal{T}[\tilde{\mathtt{n}}/\tilde{u}]$ *is well-asserted.*

Proof. The proof trivially follows from the fact that *LSat* universally quantifies the free variables of $\mathcal{T}$.

Lemma L.2 (Substitution).

Let $\mathcal{C};\Gamma, u : U \vdash P \triangleright \Delta$ with Δ well-asserted. If u is free in P and $\mathtt{n}$ has type U then $\mathcal{C}[\mathtt{n}/u];\Gamma \vdash P[\mathtt{n}/u] \triangleright \Delta[\mathtt{n}/u]$ and $\Delta[\mathtt{n}/u]$ is well-asserted.

Proof. The proof is by rule induction on validation rules. We proceed by case analysis o the rules in Figure 5. Assume $u : U'$.

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

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

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

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

Moreover, by inductive hypothesis, we have

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

By applying [SND] with premises L.1 and L.2 we obtain

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

The substituted endpoint assertion is well-asserted by Lemma L.1.

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

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

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

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

By applying L.3 as a premise for [RCV] we obtain

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

The substituted endpoint assertion is well-asserted by Lemma L.1.

– If $P = s_k!\langle\!\langle\tilde{t}\rangle\!\rangle(\tilde{v} : \mathcal{T}' \mathbin{@} \mathtt{p})\{A\};P'$ then by the validation rule for delegation [SDEL]

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

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

$$\mathcal{C}[\mathtt{n}/u];\Gamma \vdash P'[\tilde{t}/\tilde{v}][\mathtt{n}/u] \triangleright (\Delta, \tilde{s}\colon \mathcal{T} \mathbin{@} \mathtt{p})[\mathtt{n}/u]. \tag{L.4}$$

By applying [SDEL] with premise L.4 we obtain

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

The substituted endpoint assertion is well-asserted by Lemma L.1.

– The case for [RDEL] is similar.
– If $P = s_k \triangleleft \{A_j\} l_j : P_j$ then by [SEL]

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

By inductive hypothesis

$$\mathcal{C}[\mathtt{n}/u]; \Gamma \vdash P_j[\mathtt{n}/u] \rhd (\Delta, \tilde{s} \colon \mathcal{T} \mathbin{@} \mathtt{p})[\mathtt{n}/u]. \tag{L.5}$$

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

$$\Gamma \vdash \mathcal{C}[\mathtt{n}/u] \supset A_j[\mathtt{n}/u]. \tag{L.6}$$

By applying L.5 and L.6 as a premise for [SEL] we obtain

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

where the substituted endpoint assertion is well-asserted by Lemma L.1
– The case for [BRA] is similar to the case of [SEL].

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

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

By inductive hypothesis

$$\begin{aligned} &(\mathcal{C} \wedge e)[\mathtt{n}/u]; \Gamma \vdash Q[\mathtt{n}/u] \rhd \Delta[\mathtt{n}/u] \text{ and} \\ &(\mathcal{C} \wedge \neg e)[\mathtt{n}/u]; \Gamma \vdash R[\mathtt{n}/u] \rhd \Delta[\mathtt{n}/u] \end{aligned} \tag{L.7}$$

By applying L.7 as a premise for [IF] we obtain

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

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

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

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

By inductive hypothesis

$$\mathcal{C}[\mathtt{n}/u]; \Gamma \vdash P'[\mathtt{n}/u] \rhd \Delta[\mathtt{n}/u], \tilde{s} \colon (\mathcal{G} \upharpoonright \mathtt{1}) \mathbin{@} \mathtt{1}[\mathtt{n}/u]. \tag{L.8}$$

Also $\Gamma \vdash a \colon \mathcal{G}[\mathtt{n}/u]$ (trivially since $\mathcal{G}$ does not have free variables). By applying L.8 as a premise for [MCAST] we obtain

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

where the substituted endpoint 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]

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

Without loss of generality we assume $u \notin \tilde{v}$. Since $\mathcal{T}_1[\tilde{e}/\tilde{v}]...\mathcal{T}_n[\tilde{e}/\tilde{v}]$ are well-typed under $\Gamma, u : U', \tilde{v} : U$ and $\mathtt{n} : U'$ then $\mathcal{T}_1[\tilde{e}/\tilde{v}][\mathtt{n}/u]...\mathcal{T}_n[\tilde{e}/\tilde{v}][\mathtt{n}/u]$ are also well-typed. Also, $\mathcal{T}_1[\tilde{e}/\tilde{v}][\mathtt{n}/u]...\mathcal{T}_n[\tilde{e}/\tilde{v}][\mathtt{n}/u]$ are well-asserted by Lemma L.1.
By applying $\mathcal{T}_1[\tilde{e}/\tilde{v}][\mathtt{n}/u]...\mathcal{T}_n[\tilde{e}/\tilde{v}][\mathtt{n}/u]$ as a premise of [VAR] we obtain

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

The remaining cases for [CONC], [IDLE], [HIDE], [CONSEQ] and [REC] are straightforward.

L.2 Evaluation Lemma

Lemma L.3 (Evaluation). *If* $\mathcal{C};\Gamma \vdash P(\tilde{e}) \rhd \Delta(\tilde{e})$ *and* $\tilde{e} \downarrow \tilde{\mathtt{n}}$ *then we have* $\mathcal{C};\Gamma \vdash P[\tilde{\mathtt{n}}/\tilde{e}] \rhd \Delta[\tilde{\mathtt{n}}/\tilde{e}]$.

Proof. The proof is by rule induction on the validation rules (Figures 5 and 10). We proceed by case analysis. By decidability of underlying logic, we can write $A[\tilde{e}/\tilde{v}] \downarrow \mathsf{true}$ when a closed formula $A[\tilde{e}/\tilde{v}]$ evaluates to true. Note that if we further have $\tilde{e} \downarrow \tilde{\mathtt{n}}$ then we have $A[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{true}$.

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

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

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

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

$$\mathcal{C} \supset A[\tilde{\mathtt{n}}/\tilde{v}]. \tag{L.9}$$

By inductive hypothesis

$$\mathcal{C};\Gamma \vdash P'[\tilde{\mathtt{n}}/\tilde{e}] \rhd \Delta'[\tilde{\mathtt{n}}/\tilde{e}], \tilde{s}\colon\ \mathcal{T}[\tilde{\mathtt{n}}/\tilde{e}] @ \mathtt{p}. \tag{L.10}$$

By applying (L.9) and (L.10) 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 $P(\tilde{e}) = s_k!\langle e'\rangle(v)\{A\};P'$, $P(\tilde{e}) = X\langle \tilde{e}'\tilde{s}_1..\tilde{s}_n\rangle$, $P(e) = \mathsf{if}\ e'\ \mathsf{then}\ Q\ \mathsf{else}\ R$, 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.

L.3 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 D.2 except for the last four clauses which are about message assertions.

Definition L.4 (Refinement for Message Assertions). A binary relation $\mathcal{R}$ over closed endpoint assertions is a *refinement relation* if $\mathcal{T}_1 \mathcal{R} \mathcal{T}_2$ implies one of the four conditions in Definition D.2 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 [10], as made explicit in the following.

Definition L.5 (refinement on extended assertion assignments). We define $\Delta \Rightarrow \Delta'$ where Δ and Δ' may possibly contain non-singleton assertions as follows:

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

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 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 11. We observe:

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

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 8 (page 23). 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 9.
3. Rule [TR-TAU-SELECT] is similar. The rule corresponds to the second reduction rule in Figure 9.
4. Rule [TR-TAU-VAL] is for interaction, corresponding to the third reduction rule in Figure 9.
5. Rule [TR-TAU-BRA] corresponds to the third reduction rule in Figure 9.

Accordingly we also use the $\tau_{\mathtt{free}}$-actions for processes, by dividing the rule [TAU] in Figure 7, presented in Figure 12. 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 7 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. [11]

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

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

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

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

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

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

Fig. 11. Labelled Transition for Message Assertions

Proposition L.6 (extended transitions).

[11] 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).

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

Proof. (1) is by definition. (2) is also direct from the definitions. For example $\langle \Gamma, \Delta \rangle \overset{\tau}{\rightarrow} \langle \Gamma', \Delta' \rangle$ 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 K. 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.

$$\frac{P \rightarrow P' \ \text{ at a shared name or a hidden session channel}}{P \overset{\tau}{\rightarrow} P'} \qquad [\textsc{Tau}]$$

$$\frac{P \rightarrow P' \ \text{ at a free session channel}}{P \overset{\tau_{\mathtt{free}}}{\Rightarrow} P'} \qquad [\textsc{TauFSC}]$$

Fig. 12. Refined τ-Transitions (replacing [TAU] in Fig. 7)

Definition L.7 (extended conditional simulation and satisfaction). We extend the notion of the conditional simulation in Definition 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 6.2 (§6.1, page 10).

Proposition L.8 (extended assertion transition and refinement).

1. *The same statement as given in Lemma K.2 holds for assertion assignments with message assertions, adding the clause:*
   If $\langle \Gamma, \Delta \rangle \overset{\tau_{\mathtt{free}}}{\rightarrow} \langle \Gamma, \Delta_1 \rangle$ *then* $\langle \Gamma, \Delta' \rangle \overset{\tau_{\mathtt{free}}}{\rightarrow} \langle \Gamma, \Delta'_1 \rangle$ *or* $\langle \Gamma, \Delta' \rangle \overset{\tau_{\mathtt{free}}\tau_{\mathtt{free}}}{\rightarrow} \langle \Gamma, \Delta'_1 \rangle$ *such that* $\Delta_1 \ni \Delta'_1$ *again.*
2. *The same statement as given in Proposition K.1 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 L.6 (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 K.2 except the pairs introduced in Definition L.4. 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 K.2:

$$\Delta = \Delta_{side}, \tilde{s} : k!\langle \tilde{\mathtt{n}} \rangle; \mathcal{T}_1 \mathbin{@} \mathtt{p} \tag{L.11}$$

$$\Delta' = \Delta'_{side}, \tilde{s} : k!(\tilde{v} : \tilde{S})\{\}A; \mathcal{T}_1' \mathbin{@} \mathtt{p} \tag{L.12}$$

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

$$\langle \Gamma, \Delta \rangle \xrightarrow{s_k!\tilde{\mathtt{n}}} \langle \Gamma, \Delta_{side}, \tilde{s} : \mathcal{T}' \mathbin{@} \mathtt{p} \rangle \tag{L.13}$$

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

$$\langle \Gamma, \Delta' \rangle \xrightarrow{s_k!\tilde{\mathtt{n}}} \langle \Gamma, \Delta'_{side}, \tilde{s} : \mathcal{T}_1' \mathbin{@} \mathtt{p} \rangle. \tag{L.14}$$

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

$$\langle \Gamma, \Delta \rangle \xrightarrow{\tau_{\mathtt{free}}} \langle \Gamma, \Delta_1 \rangle \tag{L.15}$$

First assume in (L.15) that this action is induced by the reduction from

$$\tilde{s} : \mathcal{H}[k!(\tilde{v} : \tilde{S})\{\}A; \mathcal{T}] \tag{L.16}$$

in Δ to its instantiation

$$\tilde{s} : \mathcal{H}[k!\langle \tilde{\mathtt{n}} \rangle; \mathcal{T}[\tilde{\mathtt{n}}/\tilde{v}] \mathbin{@} \mathtt{p}] \tag{L.17}$$

in Δ_1 such that

$$A[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{true}. \tag{L.18}$$

Then Δ' will have the corresponding reduction from

$$\tilde{s} : \mathcal{H}[k!(\tilde{v} : \tilde{S})\{\}A'; \mathcal{T}'] \tag{L.19}$$

in Δ', because, by the definition of refinement, we have $A \supset A'$, hence by (L.18) we obtain $A'[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{true}$ too, so that we obtain the corresponding instantiation:

$$\tilde{s} : \mathcal{H}[k!\langle \tilde{\mathtt{n}} \rangle; \mathcal{T}'[\tilde{\mathtt{n}}/\tilde{v}] \mathbin{@} \mathtt{p}] \tag{L.20}$$

for which we have, by definition,

$$k!\langle \tilde{\mathtt{n}} \rangle; \mathcal{T}[\tilde{\mathtt{n}}/\tilde{v}] \ \ni \ k!\langle \tilde{\mathtt{n}} \rangle; \mathcal{T}'[\tilde{\mathtt{n}}/\tilde{v}] \tag{L.21}$$

as required. On the other hand if the transition in (L.15) is induced by the following redex in Δ

$$\tilde{s} : \mathcal{H}[k!\langle \tilde{\mathtt{n}} \rangle; \mathcal{T}_a \mathbin{@} \mathtt{p}, k?(\tilde{v})\{A_b\}; \mathcal{T}_b \mathbin{@} \mathtt{q}] \tag{L.22}$$

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

$$\tilde{s} : \mathcal{H}[\mathcal{T}_a \mathbin{@} \mathtt{p}, \mathcal{T}_b \mathbin{@} [\tilde{\mathtt{n}}/\tilde{v}] \mathbin{@} \mathtt{q}] \tag{L.23}$$

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

$$\tilde{s}\colon \mathcal{H}[k!\langle\tilde{\mathtt{n}}\rangle; \mathcal{T}_a'@\mathtt{p}, k?(\tilde{v})\{A_b'\}; \mathcal{T}_b'@\mathtt{q}] \tag{L.24}$$

such that

$$\mathcal{T}_a \supset \mathcal{T}_a' \tag{L.25}$$
$$A_b' \supset A_b \tag{L.26}$$
$$\mathcal{T}_b[\tilde{\mathtt{m}}/\tilde{v}] \ni \mathcal{T}_b'[\tilde{\mathtt{m}}/\tilde{v}] \quad (\text{if } A_b'[\tilde{\mathtt{m}}/\tilde{v}] \downarrow \mathsf{true}) \tag{L.27}$$

Thus (L.24) 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:

$$\tilde{s}\colon \mathcal{H}[k!(\tilde{v}:\tilde{S})\{\}A_a'; \mathcal{T}_a'@\mathtt{p}, k?(\tilde{v})\{A_b'\}; \mathcal{T}_b'@\mathtt{q}] \tag{L.28}$$

such that

$$A_a \supset A_a'[\tilde{\mathtt{n}}/\tilde{v}] \tag{L.29}$$
$$\mathcal{T}_a[\tilde{\mathtt{m}}/\tilde{v}] \ni \mathcal{T}_b'[\tilde{\mathtt{m}}/\tilde{v}] \quad (\text{if } A_a[\tilde{\mathtt{m}}/\tilde{v}] \downarrow \mathsf{true}) \tag{L.30}$$
$$A_b' \supset A_b \tag{L.31}$$
$$\mathcal{T}_b[\tilde{\mathtt{m}}/\tilde{v}]\mathcal{T}_b'[\tilde{\mathtt{m}}/\tilde{v}] \quad (\text{if } A_b'[\tilde{\mathtt{m}}/\tilde{v}] \downarrow \mathsf{true}) \tag{L.32}$$

By (L.29) we know (L.33) has the reduction into:

$$\tilde{s}\colon \mathcal{H}[k!\langle\tilde{\mathtt{n}}\rangle; \mathcal{T}_a'[\tilde{\mathtt{m}}/\tilde{v}]@\mathtt{p}, k?(\tilde{v})\{A_b'\}; \mathcal{T}_b'@\mathtt{q}] \tag{L.33}$$

We further use (L.31) to obtain the reduction from (L.33) into:

$$\tilde{s}\colon \mathcal{H}[\mathcal{T}_a'[\tilde{\mathtt{m}}/\tilde{v}]@\mathtt{p}, \mathcal{T}_b'@[\tilde{\mathtt{n}}/\tilde{v}]@\mathtt{q}] \tag{L.34}$$

as required.

Corollary L.9 (Assertion Reduction and Coherence). *If Δ is coherent and $\Delta \to \Delta'$ or equivalently $\langle\Gamma,\Delta\rangle \xrightarrow{\tau_{\mathtt{free}}} \langle\Gamma,\Delta'\rangle$, 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:

$$\tilde{s}\colon \mathcal{H}[k!(\tilde{v}:\tilde{S})\{\}A; \mathcal{T}@\mathtt{p},] \tag{L.35}$$

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

$$k!(\tilde{v}:\tilde{S})\{\}A; \mathcal{T} \ni \mathcal{G}\upharpoonright\mathtt{p} \tag{L.36}$$

similarly for other endpoint assertions under $\tilde{s}$. Now consider we have a reduction from (L.35) by the first rule in Figure 9 into:

$$\tilde{s}\colon \mathcal{H}[k!\langle\tilde{\mathtt{n}}\rangle; \mathcal{T}[\tilde{\mathtt{n}}/\tilde{v}]@\mathtt{p},] \tag{L.37}$$

where we have

$$A[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{true} \tag{L.38}$$

By (L.36) and (L.38) and because $\ni$ is transitive we obtain:

$$k!\langle\tilde{\mathtt{n}}\rangle;\mathcal{T}[\tilde{\mathtt{n}}/\tilde{v}] \ni \mathcal{G}\upharpoonright \mathtt{p} \tag{L.39}$$

as required. For the other case, the reduction involves a pair. Assume Δ has a redex

$$\tilde{s} : \mathcal{H}[k!\langle\tilde{\mathtt{n}}\rangle\mathcal{T}_a, k?(\tilde{v})\{A_b\};\mathcal{T}_b] \tag{L.40}$$

As before, by coherence we can set:

$$k!\langle\tilde{\mathtt{n}}\rangle;\mathcal{T}_a \ni \mathcal{G}\upharpoonright \mathtt{p} \tag{L.41}$$

$$k?(\tilde{v})\{A_b\};\mathcal{T}_b \ni \mathcal{G}\upharpoonright \mathtt{q} \tag{L.42}$$

Note we can safely assume $\mathcal{G}$ has the shape (up to permutation of utterly unordered actions):

$$\mathcal{G} = \mathtt{p} \to \mathtt{q} : (\tilde{v} : \tilde{S})\{A'\}.\mathcal{G}' \tag{L.43}$$

hence we can assume:

$$\mathcal{G}\upharpoonright \mathtt{p} = k!(\tilde{v} : \tilde{S})\{\}A';(\mathcal{G}'\upharpoonright \mathtt{p}) \tag{L.44}$$

$$\mathcal{G}\upharpoonright \mathtt{q} = k?(\tilde{v})\{A'\};(\mathcal{G}'\upharpoonright \mathtt{q}) \tag{L.45}$$

such that, by Definition L.4, $A' \supset A_b$ and $A'[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{true}$ and hence

$$\mathcal{T}_a \ni \mathcal{G}'\upharpoonright \mathtt{p}[\tilde{\mathtt{n}}/\tilde{v}] \tag{L.46}$$

$$\mathcal{G}'\upharpoonright \mathtt{q}[\tilde{\mathtt{n}}/\tilde{v}] \supset \mathcal{T}_b[\tilde{\mathtt{n}}/\tilde{v}] \tag{L.47}$$

Now consider the reduction from (L.40) into:

$$\tilde{s} : \mathcal{H}[\mathcal{T}_a, \mathcal{T}_b[\tilde{\mathtt{n}}/\tilde{v}]] \tag{L.48}$$

By (L.46) and (L.47) we obtain

$$\mathcal{T}_a \ni \mathcal{G}'[\tilde{\mathtt{n}}/\tilde{v}]\upharpoonright \mathtt{p} \tag{L.49}$$

$$\mathcal{T}_b[\tilde{\mathtt{n}}/\tilde{v}] \ni \mathcal{G}'[\tilde{\mathtt{n}}/\tilde{v}]\upharpoonright \mathtt{q} \tag{L.50}$$

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:

$$\mathcal{G}'[\tilde{\mathtt{n}}/\tilde{v}]\upharpoonright \mathtt{r} \;=\; \mathcal{G}\upharpoonright \mathtt{r} \tag{L.51}$$

hence as required.

We shall also use the following result later.

Lemma L.10. *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 L.11. *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* $P_1 \overset{a[2..n](\tilde{s})}{\longrightarrow} P_1'$ *and* $P_i \overset{a[i](\tilde{s})}{\longrightarrow} P_i'$ *for* $2 \leqslant i \leqslant n$*, with* $P' \equiv C[(\nu\tilde{s})(P_1'|..|P_n')]$.
3. $P \equiv C[Q|s:\tilde{h}]$ *such that* $Q \overset{s!\mathtt{n}}{\longrightarrow} Q'$ *and* $P' \equiv C[Q'|s:\tilde{h}\cdot\mathtt{n}]$.
4. $P \equiv C[Q|s:\tilde{h}\cdot\mathtt{n}]$ *such that* $Q \overset{k?\mathtt{n}}{\longrightarrow} Q'$ *and* $P' \equiv C[Q'|s:\tilde{h}]$.
5. $P \equiv C[Q|s:\tilde{h}]$ *such that* $Q \overset{s\triangleleft l}{\longrightarrow} Q'$ *and* $Q' \equiv C[Q'|s:\tilde{h}\cdot l]$.
6. $P \equiv C[Q|s:\tilde{h}\cdot l]$ *such that* $Q \overset{l\triangleright l}{\longrightarrow} Q'$ *and* $P' \equiv C[Q'|s:\tilde{h}]$.

Proof. Immediate from the corresponding reduction rules.

By Lemma L.11 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 L.12. *Assume below all transitions are typed under the implicit typing.*

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

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 L.11 (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

$$\Gamma \vdash P|s:\tilde{h} \rhd \Delta \tag{L.52}$$

By Lemma L.10, we safely assume the last rule applied is [Conc]. Thus we can assume, for some Δ_0 and Δ_1:

$$\Gamma \vdash P \rhd \Delta_0 \tag{L.53}$$

$$\Gamma \vdash s:\tilde{h} \rhd \Delta_1 \tag{L.54}$$

$$\Delta_0 \circ \Delta_1 = \Delta \tag{L.55}$$

Now consider the transition

$$P \xrightarrow{s!\tilde{\mathtt{n}}} P' \tag{L.56}$$

By (L.53) we observe Δ_0 has the shape, with $s = s_k$:

$$\Delta_0 = \tilde{s}\colon \mathcal{H}\big[k!(e)\{\tilde{v}\}A; \mathcal{T}@\mathtt{p}\big], \Delta_{00} \tag{L.57}$$

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

$$\Delta_0' = \tilde{s}\colon \mathcal{H}\big[\mathcal{T}\big[\tilde{\mathtt{n}}/\tilde{v}\big]\big], \Delta_{00} \tag{L.58}$$

Now the assertion Δ_1 for the queue has the shape, omitting the vacuous "end":

$$\Delta_1 = \tilde{s}\colon \mathcal{H}\big[\mathcal{M}@\mathtt{p}\big] \tag{L.59}$$

hence the addition of the values to this queue, $s\!:\!\tilde{h}\cdot\tilde{\mathtt{n}}$, must have the endpoint assertion:

$$\Delta_1' = \tilde{s}\colon \mathcal{H}\big[k!\langle\tilde{\mathtt{n}}\rangle; \mathcal{M}@\mathtt{p}\big] \tag{L.60}$$

Setting $\Delta' = \Delta_0' \circ \Delta_1'$, we know:

$$\Gamma \vdash P' | s\!:\!\tilde{h}\cdot\tilde{\mathtt{n}} \rhd \Delta' \tag{L.61}$$

By (L.58) and (L.60) and the type composition $\circ$ and the type reduction $\rightarrow$, we obtain

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

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

$$\Gamma \vdash P | s\!:\!\tilde{h}\cdot\tilde{\mathtt{n}} \rhd \Delta \tag{L.62}$$

Again by Lemma L.10, we safely assume the last rule applied is [Conc]. Thus we can assume, for some Δ_0 and Δ_1:

$$\Gamma \vdash P \rhd \Delta_0 \tag{L.63}$$

$$\Gamma \vdash s\!:\!\tilde{h}\cdot\tilde{\mathtt{n}} \rhd \Delta_1 \tag{L.64}$$

$$\Delta_0 \circ \Delta_1 = \Delta \tag{L.65}$$

Now consider the transition

$$P \xrightarrow{s?\tilde{\mathtt{n}}} P' \tag{L.66}$$

As before, we can infer, from (L.63) and (L.66) the shape of Δ_0 as follows, with $s = s_k$:

$$\Delta_0 = \tilde{s}\colon \mathcal{H}\big[k?(\tilde{v})\{A\}; \mathcal{T}@\mathtt{p}\big], \Delta_{00} \tag{L.67}$$

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

$$\Delta_0' = \tilde{s}\colon \mathcal{H}\big[\mathcal{T}\big[\tilde{\mathtt{n}}/\tilde{v}\big]\big], \Delta_{00} \tag{L.68}$$

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

$$\Delta_1 = \tilde{s}\colon \mathcal{H}[k!\langle \tilde{\mathrm{n}} \rangle; \mathcal{M}@\mathrm{p}] \tag{L.69}$$

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

$$\Delta_1' = \tilde{s}\colon \mathcal{H}[\mathcal{M}@\mathrm{p}] \tag{L.70}$$

Note this is symmetric to the case (1) above. As before, setting $\Delta' = \Delta_0' \circ \Delta_1'$, we know:

$$\Gamma \vdash P'|s:\tilde{h} \rhd \Delta' \tag{L.71}$$

By (L.68) and (L.70) and the type composition $\circ$ and the type reduction $\rightarrow$, we obtain

$$\begin{aligned} \Delta_0' \circ \Delta_1' \ &= \ \tilde{s}\colon \mathcal{H}[\mathcal{T}[\tilde{\mathrm{n}}/\tilde{v}]],\ \Delta_{00},\ \Delta_1 \\ &\leftarrow \Delta_0, \Delta_1 \end{aligned}$$

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 L.1, the subject reduction.

Lemma L.1 (Subject Reduction). *Suppose $\Gamma \vdash P \rhd \Delta$ and $P \rightarrow P'$. Then $\Gamma \vdash P' \rhd \Delta'$ such that either $\Delta' = \Delta$ or $\Delta \rightarrow \Delta'$*

We prove a stronger statement. Below we use the refined τ-transition rules in Figure 12.

Lemma L.13. *Suppose $\Gamma \vdash P \rhd \Delta$.*

1. *If $P \xrightarrow{\tau} P'$ by Figure 12 then $\Gamma \vdash P' \rhd \Delta$ again.*
2. *If $P \xrightarrow{\tau_{\mathtt{free}}} P'$ by Figure 12 then $\Gamma \vdash P' \rhd \Delta'$ such that $\Delta \rightarrow \Delta'$.*

Remark. Via Proposition L.6, Lemma L.13 above entails Lemma L.1.

Proof. Assume

$$\mathsf{true};\Gamma_0 \vdash P \rhd \Delta_0 \quad \text{and} \quad P \xrightarrow{\tau} P'. \tag{L.72}$$

Each of the six cases in Lemmas L.11 are possible, which we inspect one by one. Below let $C[\ \cdot\ ]$ is an appropriate reduction context.

1. Conditional. By Lemmas L.11 (1) assume $P = C[R]$ where

$$R = \mathsf{if}\ e\ \mathsf{then}\ Q_1\ \mathsf{else}\ Q_2 \tag{L.73}$$

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:

$$\mathsf{true};\Gamma \vdash R \rhd \Delta \tag{L.74}$$

By Lemma L.10 we can assume R is inferred by [IF] of Figure 5. Hence we have

$$\mathsf{true} \wedge e;\Gamma \vdash Q_1 \rhd \Delta \tag{L.75}$$

By [CONSEQ] we get

$$\mathsf{true};\Gamma \vdash Q_1 \rhd \Delta \tag{L.76}$$

as required. Dually for the case of $e \downarrow \mathsf{false}$.

2. Link. By Lemmas L.11 (2) we set $P = C[R]$ where

$$R = P_1|...|P_n \tag{L.77}$$

with their initialization actions compensating each other, as given in Lemmas L.11 (2), i.e.

$$P_1 \xrightarrow{\bar{a}[2..n](\tilde{s})} P_1' \tag{L.78}$$

$$P_i \xrightarrow{a[i](\tilde{s})} P_i' \quad (2 \leqslant i \leqslant n) \tag{L.79}$$

As before, we can safely set:

$$\mathsf{true};\Gamma \vdash R \rhd \Delta \tag{L.80}$$

By Lemma L.10 we can assume R is inferred by [IF] of Figure 5 by consecutive applications of [CONC], hence we safely assume:

$$\mathsf{true};\Gamma \vdash P_i \rhd \Delta_i \tag{L.81}$$

such that $\Delta_1 \circ .. \circ \Delta_n = \Delta$. By Lemma L.12 (1) and (2), we have, with $\Gamma(a) = \mathcal{G}$,

$$\mathsf{true};\Gamma \vdash P_i' \rhd \Delta_i, \tilde{s} : (\mathcal{G} \upharpoonright i)@i \tag{L.82}$$

Hence

$$\mathsf{true};\Gamma \vdash P_1|..|P_n \rhd \Delta, \tilde{s} : \{(\mathcal{G} \upharpoonright i)@i\}_{1 \leqslant i \leqslant n} \tag{L.83}$$

Since $\{(\mathcal{G} \upharpoonright i)@i\}_{1 \leqslant i \leqslant n}$ is obviously coherent, we have

$$\mathsf{true};\Gamma \vdash (\nu \tilde{s})(P_1|..|P_n) \rhd \Delta \tag{L.84}$$

as required.

3. Send. By Lemmas L.11 (3) we set

$$P = [Q|s:\tilde{h}] \tag{L.85}$$

with

$$Q \xrightarrow{s!\tilde{\mathfrak{n}}} Q' \tag{L.86}$$

As above we can safely set

$$\mathsf{true};\Gamma \vdash Q|s:\tilde{h} \rhd \Delta \tag{L.87}$$

By Lemmas L.12 (3), (L.86) and (L.87), we infer:

$$\mathsf{true};\Gamma \vdash Q'|s:\tilde{h} \cdot \tilde{\mathfrak{n}} \rhd \Delta' \tag{L.88}$$

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 12, we know that this $\tilde{s}$ in R are hidden in P. Assume therefore, without loss of generality:

$$P \equiv C'[(\nu \tilde{s})(Q|s:\tilde{h}|R)] \tag{L.89}$$

$$\Gamma_1 \vdash Q|s:\tilde{h}|R \rhd \Delta_1 \tag{L.90}$$

$$\Delta' \text{ coherent and} \Delta_1 = \Delta \circ \Delta_{01} \tag{L.91}$$

By Lemma L.9 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 L.11 (4) we set

$$P = C[Q|s:\tilde{h}\cdot \mathfrak{n}] \tag{L.92}$$

with

$$Q \overset{s?\mathfrak{n}}{\to} Q' \tag{L.93}$$

As above we can safely set

$$\mathsf{true};\Gamma \vdash Q|s:\tilde{h}\cdot \mathfrak{n} \rhd \Delta \tag{L.94}$$

As before, by Lemmas L.12 (4), (L.93) and (L.94), we get:

$$\mathsf{true};\Gamma \vdash Q'|s:\tilde{h} \rhd \Delta' \tag{L.95}$$

such that $\Delta \to \Delta'$ where the only change is at $\tilde{s}$ which contains s. Again by Figure 12, we can set, without loss of generality:

$$P \equiv C'[(\nu \tilde{s})(Q|s:\tilde{h}|R)] \tag{L.96}$$

$$\Gamma_1 \vdash Q|s:\tilde{h}|R \rhd \Delta_1 \tag{L.97}$$

$$\Delta' \text{ coherent and} \Delta_1 = \Delta \circ \Delta_{01} \tag{L.98}$$

As before, by Lemma L.9 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 L.11 (5) and Lemmas L.12 (5) instead of Lemmas L.11 (3) and Lemmas L.12 (3), respectively.

6. Branch. The argument exactly follows Case ***4.Receive*** above except using Lemmas L.11 (6) and Lemmas L.12 (6) instead of Lemmas L.11 (4) and Lemmas L.12 (4), respectively.

Finally the $\tau_{\mathtt{free}}$-reduction,

$$\mathsf{true};\Gamma_0 \vdash P \rhd \Delta_0 \quad \text{and} \quad P \overset{\tau_{\mathtt{free}}}{\to} P'. \tag{L.99}$$

rather than (L.72), precisely follow the same reasoning as given in the cases of ***3***..6 above, excepting we do not have to hide $\tilde{s}$.

L.4 Subject Transition

In this subsection we list the proofs for Proposition 6.3. The proof hinges on two lemmas: Substitution Lemma (Lemma L.2, whose statement and proof are given in Appendix L.1, page 32); and Evaluation Lemma (Lemma L.3, whose statement and proof are given in Appendix L.2, page 34)).

Convention L.14 (shape of processes).

1. In this subsection, unless otherwise stated, $P, Q, \ldots$ range over runtime (i.e. general) processes (cf. Appendix H) and validation and other judgements are considered for runtime processes (cf. Convention H.2).
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).

L.5 Subject Transition for Visible Transitions

We now prove Proposition 6.3 in §6, the subject transition for visible transitions. We reproduce the statement in the following.

Proposition L.15 (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 5 and 10, showing a stronger result which adds to the statement:

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

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

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

We proceed by the case analysis depending on the last rule used for deriving this judgement. By Convention L.14 (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 $\mathcal{C}; \Gamma \vdash P \triangleright \Delta$ with:

$$\mathcal{C} = \mathsf{true} \tag{L.100}$$
$$P = s_k!\langle e \rangle(v)\{A\}; Q \tag{L.101}$$
$$\Delta = \Delta_0, \tilde{s}\colon k!(\tilde{v}\colon \tilde{S})\{\tilde{e}\}A; \mathcal{T} \mathbin{@} \mathtt{p}. \tag{L.102}$$

By the first premise of [SEND] and (L.100) we have:

$$\mathsf{true} \supset A[\tilde{e}/\tilde{v}] \tag{L.103}$$

Since P is closed, we can set $\tilde{e} \downarrow \mathtt{n}$. By (L.103) we infer:

$$A[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{true}. \tag{L.104}$$

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

$$P \xrightarrow{\alpha} Q[\tilde{\mathtt{n}}/\tilde{v}] \stackrel{\text{def}}{=} P' \tag{L.105}$$

Now Δ can move by [TR-SEND]:

$$\langle \Gamma, \Delta \rangle \xrightarrow{\alpha} \langle \Gamma,\ (\Delta_0, \tilde{s}\colon \mathcal{T}[\tilde{\mathtt{n}}/\tilde{v}] \mathbin{@} \mathtt{p}) \rangle \tag{L.106}$$

By the second premise of [SEND] in Figure 5, we have

$$\mathsf{true}; \Gamma \vdash Q[\tilde{e}/\tilde{v}] \triangleright \Delta_0, \tilde{s}\colon \mathcal{T}[\tilde{e}/\tilde{v}] \mathbin{@} \mathtt{p} \tag{L.107}$$

By Lemma L.3 (Evaluation Lemma), (L.107) immediately gives:

$$\mathsf{true}; \Gamma \vdash Q[\tilde{\mathtt{n}}/\tilde{v}] \triangleright \Delta_0, \tilde{s}\colon \mathcal{T}[\tilde{\mathtt{n}}/\tilde{v}] \mathbin{@} \mathtt{p} \tag{L.108}$$

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:

$$P = s_k?(v)\{A\};Q \tag{L.109}$$
$$\Delta = \Delta_0, \tilde{s}\colon\ k?(\tilde{v} : \tilde{S})\{A\};\mathcal{T}\,@\,\mathtt{p} \tag{L.110}$$

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

$$A[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{true} \tag{L.111}$$
$$\langle \Gamma, \Delta\rangle \xrightarrow{\alpha} \langle \Gamma \cdot \tilde{a} : \tilde{\mathcal{G}},\ \Delta_0, \tilde{s}\colon\ \mathcal{T}[\tilde{\mathtt{n}}/\tilde{v}]\,@\,\mathtt{p}\rangle \tag{L.112}$$

Thus P can move only by [RECV] (not by [RECVERR]), obtaining:

$$P \xrightarrow{\alpha} Q[\tilde{\mathtt{n}}/\tilde{v}] \tag{L.113}$$

Now the premise of [RCV] in Figure 5 says:

$$\mathsf{true} \wedge A\ ;\ \Gamma \vdash Q \rhd \Delta_0, \tilde{s}\colon\ \mathcal{T}\,@\,\mathtt{p} \tag{L.114}$$

By Lemma L.2 (Substitution Lemma) we obtain

$$\mathsf{true} \wedge A[\tilde{\mathtt{n}}/\tilde{v}];\Gamma, \tilde{v}\colon\tilde{S} \vdash Q[\tilde{\mathtt{n}}/\tilde{v}] \rhd \Delta_0, \tilde{s}\colon\ \mathcal{T}[\tilde{\mathtt{n}}/\tilde{v}]\,@\,\mathtt{p} \tag{L.115}$$

By (L.111) and by [CONSEQ] we obtain

$$\mathsf{true};\Gamma, \tilde{v}\colon\tilde{S} \vdash Q[\tilde{\mathtt{n}}/\tilde{v}] \rhd \Delta_0, \tilde{s}\colon\ \mathcal{T}[\tilde{\mathtt{n}}/\tilde{v}]\,@\,\mathtt{p} \tag{L.116}$$

as required.

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

$$P = s_k \lhd \{A_j\}l_j : P_j \tag{L.117}$$
$$\Delta = \Delta_0, \tilde{s}\colon\ k \oplus \{\{A_i\}l_i : \mathcal{T}_i\}_{i\in I}\,@\,\mathtt{p} \tag{L.118}$$

By the premise of the rule we have:

$$\mathsf{true} \supset A_j \tag{L.119}$$

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

$$P \xrightarrow{\alpha} P_j \tag{L.120}$$

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

$$\langle \Gamma, \Delta\rangle \xrightarrow{\alpha} \langle \Gamma, \Delta_0, \tilde{s}\colon\ \mathcal{T}_j\,@\,\mathtt{p}\rangle. \tag{L.121}$$

By the second premise of [LABEL] in Figure 5 we get

$$\mathsf{true};\Gamma \vdash P_j \rhd \Delta_0, \tilde{s}\colon\ \mathcal{T}_j\,@\,\mathtt{p} \tag{L.122}$$

as required.

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

$$P = s_k \rhd \{\{A_i\} l_i \colon P_i\}_{i \in I} \tag{L.123}$$

$$\Delta = \Delta_0, \tilde{s} \colon k \& \{\{A_i\} l_i : \mathcal{T}_i\}_{i \in I} \mathbin{@} \mathtt{p} \tag{L.124}$$

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

$$A_j \downarrow \mathsf{true} \tag{L.125}$$

$$\langle \Gamma, \Delta \rangle \xrightarrow{\alpha} \langle \Gamma \cdot \tilde{a} : \tilde{\mathcal{G}},\ \Delta_0, \tilde{s} \colon\ \mathcal{T}_j \mathbin{@} \mathtt{p} \rangle \tag{L.126}$$

Thus P can move only by [BRANCH] (not by [BRANCHERR]), obtaining:

$$P \xrightarrow{\alpha} P_j \tag{L.127}$$

Now the premise of [BRANCH] in Figure 5 says:

$$\mathsf{true} \wedge A_j \ ;\ \Gamma \vdash P_j \rhd \Delta_0, \tilde{s} \colon\ \mathcal{T}_j \mathbin{@} \mathtt{p} \tag{L.128}$$

By (L.125) and [CONSEQ] we obtain:

$$\mathsf{true};\Gamma, \tilde{v} \colon \tilde{S} \vdash Q[\tilde{\mathtt{n}}/\tilde{v}] \rhd \Delta_0, \tilde{s} \colon\ \mathcal{T}[\tilde{\mathtt{n}}/\tilde{v}] \mathbin{@} \mathtt{p} \tag{L.129}$$

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:

$$P = \overline{a}_{[2..\mathtt{n}]}(\tilde{s}).Q \tag{L.130}$$

$$\Gamma \vdash a \colon \mathcal{G} \tag{L.131}$$

$$\mathsf{true};\Gamma \vdash Q \rhd \Delta, \tilde{s} \colon (\mathcal{G} \upharpoonright \mathtt{1}) \mathbin{@} \mathtt{1} \tag{L.132}$$

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

$$\overline{a}_{[2..\mathtt{n}]}(\tilde{s}).Q \xrightarrow{\alpha} Q \tag{L.133}$$

By (L.131) the following transition is possible using [TR-LINKOUT]:

$$\langle \Gamma, \Delta \rangle \xrightarrow{\alpha} \langle \Gamma, \Delta, \tilde{s} \colon (\mathcal{G} \upharpoonright \mathtt{1}) \mathbin{@} \mathtt{1} \rangle \tag{L.134}$$

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

$$\mathsf{true};\Gamma \vdash P \rhd \Delta_0 \tag{L.135}$$

$$\Delta_0 \ni \Delta \tag{L.136}$$

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

$$P \xrightarrow{\alpha} P' \tag{L.137}$$

$$\langle \Gamma, \Delta \rangle \xrightarrow{\alpha} \langle \Gamma', \Delta' \rangle \tag{L.138}$$

By induction hypothesis and by (L.135), (L.138) gives us:

$$\langle \Gamma, \Delta_0 \rangle \xrightarrow{\alpha} \langle \Gamma', \Delta_0' \rangle \tag{L.139}$$

for some Δ_0' for which we have, by induction hypothesis

$$\mathsf{true};\Gamma' \vdash P' \rhd \Delta_0' \tag{L.140}$$

Since the assertion transition is deterministic and by Lemma K.1 we know:

$$\Delta_0' \ni \Delta'. \tag{L.141}$$

By (L.141) and (L.140) we can use [CONSEQ] to reach

$$\mathsf{true};\Gamma' \vdash P' \rhd \Delta' \tag{L.142}$$

as required. This exhausts all cases.

M Semantics of Open Judgement

In this section, we define the semantics of open judgement for satisfiability,

$$\mathcal{C};\Gamma \models P \rhd \Delta \tag{M.1}$$

We define the notion incrementally, starting from the judgement on closed processes based on conditional simulation given in Definition 6.2, page 10 and adding variables of different kinds. First we consider value variables. We first make clear the kinds of variables we treat, from Sections 3 and 5.1:

Definition M.1 (variables). A *value variable* is a variable for a name or an atomic value, for which we used $v, w, ..$ as well as $x, y, ...$ A *process variable* X is a variable standing for a (possibly parameterised) process. An *assertion variable* is a variable standing for an assertion, written $\mathbf{t}$.

When we think about open judgements, we need to think about these different kinds of variables. Below *asssertion erasure* of a process P (resp. Γ, resp. Δ) is the result of removing all predicate annotations from P (resp. Γ, resp. Δ), resulting in a typed process (resp. a shared typing assignment, resp. a linear typing assignment).

Definition M.2 (open judgement, 1). Let P be an open asserted process and none of Γ, Δ, $\mathcal{T}_i$ and P contains value variables, process variables nor assertion variables. Then assuming the typability of the assertion erasure of P under the assertion erasure of Γ, Δ, and the induced assignment to X, we write:

$$\Gamma, X : (\tilde{v} : \tilde{S})\mathcal{T}_1 @ \mathtt{p}_1 ... \mathcal{T}_n @ \mathtt{p}_n \models P \rhd \Delta$$

when, for each parametrised process $Q(\tilde{x}, \tilde{s}_1, ..., \tilde{s}_n)$ (with each $\tilde{s}_i$ as induced by the use of channels in $\mathcal{T}_i$), whose free variables are in $\tilde{x}$ and otherwise well-typed under the above typing, such that for each σ mapping $\tilde{x}$ to values,

$$\Gamma \models Q\sigma \rhd \uplus_i \tilde{s}_i : \mathcal{T}_1\sigma @ \mathtt{p}_1$$

we have

$$\Gamma \models P[Q(\tilde{x}, \tilde{s}_1, ..., \tilde{s}_n)/X] \rhd \Delta$$

in the sense of Definition 6.2, page 10. Similarly when a judgement contains more than one free process variables.

Based on this, we extend to the

Definition M.3 (open judgement, 2). Let P be an open asserted process and none of Γ, Δ and P contains assertion variables. Then we write, assuming the typability of the assertion erasure of P under the assertion erasure of Γ and Δ:

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

when, for each closing σ which maps all free value variables in Γ (hence in $\mathcal{C}$ and P) to values respecting types, we have either $\mathcal{C}\sigma$ is false or, if it is true:

$$\Gamma\sigma \models P\sigma \rhd \Delta\sigma$$

in the sense of Definition M.2 above.

Finally we introduce assertion variables, which we interpret based on substitution by closed well-asserted assertions so that the resulting assertions are again well-asserted.[12]

Definition M.4 (open judgement, 3). Let P be an open asserted process and none of Γ, Δ and P contains assertion variables. Then we write, assuming typability as above,

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

when, for each σ which maps each free assertion variable with an instantiation, say $\mathbf{t}\langle\vec{e}\rangle$, to a parametrised assertion $A(\tilde{x})$ such that types for $\tilde{x}$ in A match $\tilde{e}$, we have

$$\mathcal{C}; \Gamma\sigma \models P\sigma \rhd \Delta\sigma$$

in the sense of Definition M.3 above.

[12] We can directly interpret assertion variables as transition relations, though a resulting difference, if any, does not concern us here, since well-assertedness implies inhabitation, while our interests are only those transitions representable by well-asserted assertions, which can capture, for example, all well-typed transitions without any constraints.

Observe we could have made all definitions into one: the division into three definitions is to make clear what substitutions are involved.

Definition M.5 (Closure of judgement). Let $\mathcal{C};\Gamma \vdash P \rhd \Delta$. Then its *closure by substitution* σ, 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 $Q(\tilde{v}\tilde{s})$ (so that each call $X\langle\tilde{e}\tilde{s}\rangle$ is instantiated into $Q(\tilde{e}\tilde{s})$), such that we have $\mathcal{C}, \Gamma, \tilde{v} : \tilde{S} \models Q(\tilde{v}\tilde{s}) \rhd \tilde{s}_1 : \tilde{T}_1 @ \mathtt{p}_1 ... \tilde{s}_n : \tilde{T}_n @ \mathtt{p}_n$.

Similarly we define a closure of an open satisfaction judgement.

Remark. Note that, in the last clause above, we use the satisfaction $\models$ when considering substitutions for process variables, cf. [2]. This corresponds to Definition M.2.

N Proof of Theorem 6.5 (Soundness)

We prove Theorem 6.5 (soundness of validation rules). The soundness proof relies on basic properties of the proof system for validation in Figures 5 and 10.

Lemma N.1. *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 N.2 (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 5) except at the end of the derivation.*

Proof. The premise of [DEF] for Q (the main process) reads:

$$\mathcal{C};\Gamma, X : (\tilde{v} : \tilde{S})\mathcal{T}_1 @ \mathtt{p1} \ldots \mathcal{T}_n @ \mathtt{p}_n \vdash Q \rhd \Delta \tag{N.1}$$

Note the only condition it demands is the assumption contains

$$X : (\tilde{v} : \tilde{S})\mathcal{T}_1 @ \mathtt{p1} \ldots \mathcal{T}_n @ \mathtt{p}_n \tag{N.2}$$

and the only effect of the application of [DEF] is we lose this assumption. Since no other rules use this assumption, by Lemma N.1, we can always permute an application of any rule with [DEF] to obtain the same conclusion.

Definition N.3. 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 N.4. *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 6.5, whose statement is reproduced below.

Theorem N.5 (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:

$$(P_\sigma, \langle \Gamma_\sigma, \Delta_\sigma \rangle) \tag{N.3}$$

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 2), by rule induction on the validation rules. Assume

$$\mathcal{C};\Gamma \vdash P \rhd \Delta \tag{N.4}$$

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 5, we can set

$$P = s_k!\langle e \rangle(v)\{A\};P' \tag{N.5}$$

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

$$\Delta = \Delta', \tilde{s} : k!(\tilde{v} : \tilde{S})\{\tilde{\mathrm{n}}\}A; \mathcal{T} \mathbin{@} \mathrm{p}. \tag{N.6}$$

By the definition of closure

$$P_\sigma = s_k!\langle e_\sigma \rangle(v)\{A_\sigma\};P'_\sigma \tag{N.7}$$

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

$$s_k!\langle e_\sigma \rangle(v)\{A_\sigma\};P'_\sigma \xrightarrow{s_k!\tilde{\mathrm{n}}} P'_\sigma. \tag{N.8}$$

It follows that

$$\Delta_\sigma = \Delta'_\sigma, \tilde{s} : k!(\tilde{v} : \tilde{S})\{\tilde{\mathrm{n}}\}A_\sigma; \mathcal{T}_\sigma \mathbin{@} \mathrm{p}. \tag{N.9}$$

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

$$\langle \Gamma, \Delta'_\sigma, \tilde{s} : k!(\tilde{v} : \tilde{S})\{\tilde{\mathrm{n}}\}A_\sigma; \mathcal{T}_\sigma \mathbin{@} \mathrm{p} \rangle \xrightarrow{s_k!\tilde{\mathrm{n}}} \langle \Gamma, \Delta'_\sigma, \tilde{s} : \mathcal{T}_\sigma \mathbin{@} \mathrm{p} \rangle, \tag{N.10}$$

It follows by induction,

$$(P'_\sigma, \langle \Gamma, \Delta'_\sigma, \tilde{s} : \mathcal{T}_\sigma \mathbin{@} \mathrm{p} \rangle) \in R. \tag{N.11}$$

Case [RCV]. By [RCV] in Figure 5, and setting $P = s_k?(v)\{A\};P'$, we have

$$\mathcal{C};\Gamma \vdash P \rhd \Delta \tag{N.12}$$

with $\Delta = \Delta', \tilde{s} : k?(\tilde{v} : \tilde{S})\{A\};\mathcal{T} \,@\, \mathrm{p}$. Let $P_\sigma = s_k?(v)\{A_\sigma\};P'_\sigma$. It follows that

$$\Delta_\sigma = \Delta'_\sigma, \tilde{s} : k?(\tilde{v} : \tilde{S})\{A_\sigma\};\mathcal{T}_\sigma \,@\, \mathrm{p}. \tag{N.13}$$

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

$$s_k?(v)\{A_\sigma\};P'_\sigma \xrightarrow{s_k!\tilde{n}} P'_\sigma \tag{N.14}$$

and by [TR-REC] in Figure 8,

$$\langle \Gamma, \Delta'_\sigma, \tilde{s} : k?(\tilde{v} : \tilde{S})\{A_\sigma\};\mathcal{T}_\sigma \,@\, \mathrm{p}\rangle \xrightarrow{s_k?\tilde{n}} \langle \Gamma, \Delta'_\sigma, \tilde{s} : \mathcal{T}_\sigma \,@\, \mathrm{p}\rangle. \tag{N.15}$$

It follows by induction

$$(P'_\sigma, \langle \Gamma_1, \Delta_\sigma, \tilde{s} : \mathcal{T}_\sigma \,@\, \mathrm{p}\rangle) \in R. \tag{N.16}$$

Case [SEL]. Let $P = s_k \lhd \{A_j\}l_j : P_j$. By [SEL] in Figure 5,

$$\mathcal{C};\Gamma \vdash s_k \lhd \{A_j\}l_j : P_j \rhd \Delta \tag{N.17}$$

with $\Delta = \Delta', \tilde{s} : k \oplus \{\{A_i\}l_i : \mathcal{T}_i\}_{i\in I} \,@\, \mathrm{p}$ and $j \in I$. It follows $\Delta_\sigma = \Delta'_\sigma, \tilde{s} : k \oplus \{\{A_{i\sigma}\}l_i : \mathcal{T}_{i\sigma}\}_{i\in I} \,@\, \mathrm{p}$. 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 7 with label $s_k < l_j$ and, since $A_j \downarrow \mathsf{true}$ by well formedness of P and $A_j \implies A_{j\sigma}$ then

$$s_k \lhd \{A_{j\sigma}\}l_j : P_{j\sigma} \xrightarrow{s_k<l_j} P_{j\sigma}. \tag{N.18}$$

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

$$\langle \Gamma, \Delta'_\sigma, \tilde{s} : k \oplus \{\{A_{i\sigma}\}l_i : \mathcal{T}_{i\sigma}\}_{i\in I} \,@\, \mathrm{p}\rangle \xrightarrow{s_k<l_j} \langle \Gamma, \Delta'_\sigma, \tilde{s} : \mathcal{T}_{j\sigma} \,@\, \mathrm{p}\rangle. \tag{N.19}$$

By induction,

$$(P_{j\sigma}, \langle \Gamma, \Delta'_\sigma, \tilde{s} : \mathcal{T}_{j\sigma} \,@\, \mathrm{p}\rangle) \in R. \tag{N.20}$$

Case [BRANCH]. We can set $P = s_k \rhd \{\{A_i\}l_i : P_i\}_{i\in I}$ then $P_\sigma = s_k \rhd \{\{A_{i\sigma}\}l_i : P_{i\sigma}\}_{i\in I}$.

By [BRANCH] in Figure 5,

$$\mathcal{C};\Gamma \vdash \; s_k \rhd \{\{A_i\}l_i : P_i\}_{i\in I} \rhd \Delta \tag{N.21}$$

with $\Delta = \Delta', \tilde{s} : k\&\{\{A_i\}l_i : \mathcal{T}_i\}_{i\in I} \,@\, \mathrm{p}$. It follows $\Delta_\sigma = \Delta'_\sigma, \tilde{s} : k\&\{\{A_{i\sigma}\}l_i : \mathcal{T}_{i\sigma}\}_{i\in I} \,@\, \mathrm{p}$.

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 7:

$$s_k \lhd \{A_{j\sigma}\}l_j : P_{j\sigma} \xrightarrow{s_k>l_j} P_{j\sigma}, \tag{N.22}$$

and by [TR-CHOICE] in Figure 8:

$$\langle \Gamma, \Delta'_\sigma, \tilde{s} : k\&\{\{A_{i\sigma}\}l_i : \mathcal{T}_{i\sigma}\}_{i\in I} \,@\, \mathrm{p}\rangle \xrightarrow{k>l_j} \langle \Gamma, \Delta'_\sigma, \tilde{s} : \mathcal{T}_{j\sigma} \,@\, \mathrm{p}\rangle. \tag{N.23}$$

By induction,

$$(P_{j\sigma}, \langle \Gamma, \Delta'_\sigma, \tilde{s} : \mathcal{T}_{j\sigma} \,@\, \mathrm{p}\rangle) \in R. \tag{N.24}$$

Case [MCAST]. In this case $P = \overline{a}_{[2..n]}(\tilde{s}).P'$ and we can set:

$$\Gamma \vdash \overline{a}_{[2..n]}(\tilde{s}).P \triangleright \Delta. \tag{N.25}$$

Let $P_\sigma = \overline{a}_{[2..n]}(\tilde{s}).P'_\sigma$. Process P_σ can only move because of rule [LINKOUT] in Figure 7:

$$\overline{a}_{[2..n]}(\tilde{s}).P'_\sigma \xrightarrow{\overline{a}[2..n](\tilde{s})} P'_\sigma. \tag{N.26}$$

By [TR-LINKOUT] in Figure 8,

$$\langle \Gamma, \Delta_\sigma \rangle \xrightarrow{\overline{a}[2..n](\tilde{s})} \langle \Gamma_1, \Delta_\sigma, \tilde{s} : (\mathcal{G} \upharpoonright 1)_\sigma @ \mathtt{p}_1 \rangle. \tag{N.27}$$

By induction hypothesis we have

$$(P'_\sigma, \langle \Gamma_1, \Delta_\sigma, \tilde{s} : (\mathcal{G} \upharpoonright 1)_\sigma @ \mathtt{p}_1 \rangle) \in R. \tag{N.28}$$

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 L.13.
Case [IF]. With $P = \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q$ then by Definition M.5:

$$P_\sigma = \mathsf{if}\ e_\sigma\ \mathsf{then}\ Q_\sigma\ \mathsf{else}\ R_\sigma, \tag{N.29}$$

By [IF] in Figure 5, such that

$$\mathcal{C}; \Gamma \vdash P \triangleright \Delta \tag{N.30}$$

$$\mathcal{C} \wedge e; \Gamma \vdash Q \triangleright \Delta \tag{N.31}$$

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 7) with label is τ. Let us consider the case in which the transition happens by rule [IFT] (the case with [IFF] is symmetric):

$$\mathsf{if}\ e_\sigma\ \mathsf{then}\ Q_\sigma\ \mathsf{else}\ R_\sigma \xrightarrow{\tau} Q_\sigma \quad \text{with } \tilde{e}_\sigma \downarrow \mathsf{true}. \tag{N.32}$$

By induction, since $e_\sigma \downarrow \mathsf{true}$ and moreover e_σ does not have free variables.

$$(Q_\sigma, \langle \Gamma, \Delta_\sigma \rangle) \in R. \tag{N.33}$$

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

$$P = X\langle \tilde{e}, \tilde{s_1}, ..., \tilde{s_n} \rangle \tag{N.34}$$

with

$$\Gamma(X) = (\tilde{v} : \tilde{S})\tilde{\mathcal{T}}. \tag{N.35}$$

Then by well formedness of P, $\mathcal{C} \implies A[\tilde{e}/\tilde{v}] \downarrow \mathsf{true}$. P_σ is a process such that

$$\mathcal{C}_\sigma, \Gamma_\sigma \vdash P_\sigma[\tilde{e}/\tilde{v}] \triangleright \Delta'_\sigma, \tilde{s}_1 : \mathcal{T}_{1\sigma} @ \mathtt{p}_1, .., \tilde{s}_n : \mathcal{T}_{n\sigma} @ \mathtt{p}_n. \tag{N.36}$$

Notice that

$$\Delta'_\sigma, \tilde{s}_1 : \mathcal{T}_{1\sigma} @ \mathtt{p}_1, .., \tilde{s}_n : \mathcal{T}_{n\sigma} @ \mathtt{p}_n = \Delta_\sigma \tag{N.37}$$

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 N.2, 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 N.4, we know the premise and conclusion of an application of [DEF] is well-initiated in the sense of Definition N.3. Note that the reductions of such processes are completely characterised by $\xrightarrow{\tau}$ (cf. Proposition L.6), and the correspoinding τ-transition for assertions is deterministic. Assume that we have

$$\begin{array}{l}\mathcal{C};\Gamma,X:(\tilde{v}:\tilde{S})\mathcal{T}_1@\mathtt{p}_1\ldots\mathcal{T}_n@\mathtt{p}_n,\tilde{v}:\tilde{S}\models P\\ \rhd\tilde{s_1}:\mathcal{T}_1@\mathtt{p}_1\ldots\tilde{s_n}:\mathcal{T}_n@\mathtt{p}_n\end{array}\tag{N.38}$$

Further we also assume

$$\mathcal{C};\Gamma,X:(\tilde{v}:\tilde{S})\mathcal{T}_1@\mathtt{p1}\ldots\mathcal{T}_n@\mathtt{p}_n\models Q\rhd\Delta\tag{N.39}$$

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 [18, §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. [13] We also set P^ω to be the recursively defined agent itself:

$$\mathsf{def}\ X(\tilde{x})\ =\ P\ \mathsf{in}\ P.\tag{N.40}$$

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).

Using Lemma N.1, we first note that, for any $\langle\Gamma,\Delta\rangle$ and $\mathcal{C}$, we have $\mathcal{C};\Gamma\models P^0\rhd\Delta$. Thus we apply this to (N.38) and replace X in P by $(\tilde{v}\tilde{s}_1..\tilde{s}_n)P^0$:

$$\mathcal{C};\Gamma\models P^1\rhd\tilde{s_1}:\mathcal{T}_1@\mathtt{p}_1\ldots\tilde{s_n}:\mathcal{T}_n@\mathtt{p}_n\tag{N.41}$$

This can again be used for (N.38) (noting the environment Γ can always be taken as widely as possible in [VAR]):

$$\mathcal{C};\Gamma\models P^2\rhd\tilde{s_1}:\mathcal{T}_1@\mathtt{p}_1\ldots\tilde{s_n}:\mathcal{T}_n@\mathtt{p}_n\tag{N.42}$$

[13] For example, writing $a(s).S$ for $a[2](s).S$ and choosing a and s to be fresh, let $P'\stackrel{\text{def}}{=}(\nu a)(a(s).P)$ then $P'\approx\mathbf{0}$.

In this way we know

$$\mathcal{C};\Gamma \models P^n \rhd \tilde{s_1} : \mathcal{T}_1 \mathbin{@} \mathtt{p}_1 \ldots \tilde{s_n} : \mathcal{T}_n \mathbin{@} \mathtt{p}_n \tag{N.43}$$

for an arbitrary n. By applying this to (N.39), we obtain:

$$\mathcal{C};\Gamma \models Q^n \rhd \Delta \tag{N.44}$$

for an arbitrary n. Now assume, for simplicity, that there are no free variables in Q (hence in Q^n) and therefore $\mathcal{C} = \mathsf{true}$ (the reasoning is precisely the same by applying a closing substitution). We can then construct a relation taking each node in the transitions from Q^ω and relating it to the derivative of $\langle \Gamma, \Delta \rangle$, by observing that assertions' transitions are always deterministic for the given process and its transition derivatives. Let the resulting relation be $\mathcal{R}$. Since any finite trace of Q^ω is in some Q^n, the conditions of Definition 2 hold at each step, hence $\mathcal{R}$ is a conditional simulation, hence done.

Case [CONSEQ]. By Proposition 6.4 (Proposition K.1 in Appendix K, page 29).

Cases [QNIL], [QVAL] and [QSEL] of Figure 10. Again these processes (queues) do not have transitions.[14]

Case [CRES]. By Lemma L.9. This exhausts all cases.

As an immediate corollary we obtain:

Theorem N.6 (Soundeness for Programs).
Let P be a program. Then $\Gamma \vdash P \rhd \Delta$ implies $\Gamma \models P \rhd \Delta$.

O Proof of Theorems 6.7 and 7.2 (Completeness)

O.1 General Structure of Completeness Proof

For completeness, we introduce the *generation rules* for (visible) programs and program phrases by which we can derive a "principal" formula. The principal formula of P (if one can be generated) is the most refined assertion assignment against which P can be validated.

The general intuition is that, for every P, Γ and Δ such that $\Gamma \models P \rhd \Delta$, we can generate Δ' such that $\Gamma \vdash P \rhd \Delta'$ and $\Delta' \ni \Delta$ (thus $\Gamma \vdash P \rhd \Delta$ by rule [CONSEQ]). To be more precise, we generate an assignment Δ' which is parametric with respect to a number of *predicate variables*, and we show that there exist a substitution of the predicate variables in Δ' that produces an assignment refining Δ (see O.2).

The completeness proof is done in the following steps:

1. We define a generalised sequent and generation rules. We show that the derivability in this system is subsumed by the derivability in our validation rules.
2. We show it is sound w.r.t. the satisfiablity.
3. We show it is complete w.r.t. the satisfiablity; we show that the principal formula is a representative formula for the process (i.e. the principal formula under appropriate instantiation refines (w.r.t. $\ni$) all satisfiable formulae).

[14] The behaviours of queues are taken into account as part of $\tau_{\mathtt{free}}$-actions.

O.2 Predicate Variables and Extended Predicates

The generation rules for principal formulae use sequents with predicate variables with fixed arities. There is no need for manipulation of these predicate variables during the generation. We need predicate variables since we cannot rely on a specific predicate when we stipulate a constraint on an input value: if we concretize it, we may lose principality (i.e., the principal formula is not the strongest), since the stronger an input constraint is, the stronger a related output constraint is.

Consider, for instance, the following assertion

$$\mathcal{G} = \mathtt{p} \to \mathtt{p}' \colon k_1\,(v \colon \mathsf{Int})\{v > 0\}.\mathtt{p}' \to \mathtt{p} \colon k_2\,(u \colon \mathsf{Int})\{u > 1\}.\mathsf{end}$$

where the projection on $\mathtt{p}'$ is

$$\mathcal{G} \upharpoonright \mathtt{p}' = k_1?(v : \mathsf{Int})\{v > 0\}; k_2!(u : \mathsf{Int})\{u > 1\}; \mathsf{end}$$

and the following (unasserted) process implementing $\mathtt{p}'$

$$P = s_1?(v).s_2!\langle v + 1\rangle(u); \mathbf{0}.$$

Assume the generated principal formula includes the following endpoint assertion for P:

$$\mathcal{T} = k_1?(v : \mathsf{Int})\{A\}; k_2!(u : \mathsf{Int})\{B\}; \mathsf{end}.$$

The intuition is: for $\mathcal{T}$ to be the strongest formula validating P, the predicate A for the input should as weak as possible, whereas the predicate B for the output should be as strong as possible. Assume we set A to be true and B to be $v + 1 = u$. Notice $\mathcal{T}$ would not be the principal formula since it would not be a refinement of $\mathcal{G} \upharpoonright \mathtt{p}'$. In the derivation rules, we leave the constraint A open, by using a predicate variable.

Extended predicates are defined as predicates where *predicate variables* can occur; predicate variables are ranged over by $\phi(\tilde{v})$ and are meant to be replaced by normal predicates A such that $\mathrm{fn}(A) \subseteq \tilde{v}$.

O.3 Generalised Sequent

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

1. $\mathcal{C}; \Gamma_0 \vdash^{\star} \mathsf{erase}(P) \blacktriangleright \Delta$. This is used for generation of principal formulae and reads: *"Under $\mathcal{C}$ as constraints on values and Γ_0 as public contracts for shared names, P has the principal formula Δ"*.
   Note predicates in these assertions use predicate variables, defined in §O.5, page 60.

2. $\mathcal{C}; \Gamma \vdash^{ext} P \rhd \Delta$. This is the same provability using the validation rules in Figure 5 *except* using the syntax of predicates incorporating predicate variables and endpoint assertion variables.
3. $\mathcal{C}; \Gamma \models^{ext} P \rhd \Delta$. Again this is the same satisfiability as we defined in §6.1 *except* using the syntax of predicates incorporating predicate variables (the semantics of predicate variables is taken in the standard way, taking satisfiability as in §6.1 under all closing substitutions).

In brief (1) is the sequent for generation described in O.5 while (2) and (3) are the sequents for validation/satisfiability obtained extending the logic with predicate variables.

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

O.4 Two Merge Operations

This subsection is a technical discussion introducing and studying two merge operations used in the generation rules. These operations "merge" endpoint assertions. After introducing the operations, we establish their basic properties. This subsection is technical, needed only for the proofs of completeness, hence may as well be skipped until the proof of the theorem.

Convention O.1 (shape of recursive assertions). In this subsection and henceforth we assume two recursive assertions to be merged are always in the same shape. Since the shape of recursive assertions to be generated rely on the shape of recursions in the original process, this assumption means semantically neutral assumption (up to a simple transformation) of recursions in processes. Since generated formulae are equivalent for different shapes of recursions, this does not lose generality.

Merging Assertions (1)

Definition O.2 (Merge of Endpoint Assertions). The function $\sqcup\!\!\sqcup$ takes two end-point assertions and *merges* them; it is recursively defined as follows:

- $k!(\tilde{v}:U)\{A\};\mathcal{T}_1' \sqcup\!\!\sqcup k!(\tilde{v}:U)\{B\};\mathcal{T}_2' \stackrel{def}{=} k!(\tilde{v})\{A\vee B\};\mathcal{T}_1'\sqcup\!\!\sqcup\mathcal{T}_2'$
  where U has the form $\tilde{S}$ or $\langle\mathcal{G}\rangle$
- $k!(\tilde{v}:\mathcal{T}@\mathtt{p})\{A\};\mathcal{T}_1' \sqcup\!\!\sqcup k!(\tilde{v}:\mathcal{T}@\mathtt{p})\{B\};\mathcal{T}_2' \stackrel{def}{=} k!(\tilde{v}:\mathcal{T}@\mathtt{p})\{A\vee B\};\mathcal{T}_1'\sqcup\!\!\sqcup\mathcal{T}_2'$
- $k?(\tilde{v}:U)\{A\};\mathcal{T}_1' \sqcup\!\!\sqcup k?(\tilde{v}:U)\{B\};\mathcal{T}_2' \stackrel{def}{=} k?(\tilde{v}:U)\{A\wedge B\};\mathcal{T}_1'\sqcup\!\!\sqcup\mathcal{T}_2'$
- $k?(\tilde{v}:\mathcal{T}@\mathtt{p})\{A\};\mathcal{T}_1' \sqcup\!\!\sqcup k?(\tilde{v}:\mathcal{T}@\mathtt{p})\{B\};\mathcal{T}_2' \stackrel{def}{=} k?(\tilde{v}:\mathcal{T}@\mathtt{p})\{A\wedge B\};\mathcal{T}_1'\sqcup\!\!\sqcup\mathcal{T}_2'$
- $k\oplus\{\{A_i\}\ l_i:\mathcal{T}_i\}_{i\in I} \sqcup\!\!\sqcup k\oplus\{\{B_i\}\ l_i:\mathcal{T}_i'\}_{i\in I} \stackrel{def}{=} k\oplus\{\{A_i\vee B_i\}\ l_i:\mathcal{T}_i\sqcup\!\!\sqcup\mathcal{T}_i'\}_{i\in I}$
- $k\&\{\{l_i\}\mathcal{T}_i : i\in I\}_{A_i} \sqcup\!\!\sqcup k\&\{\{l_i\}\mathcal{T}_i' : i\in I\}_{B_i} \stackrel{def}{=} k\&\{\{l_i\}\mathcal{T}_i\sqcup\!\!\sqcup\mathcal{T}_i' : i\in I\}_{A_i\wedge B_i}$
- $\mu\mathbf{t}\langle\tilde{u}_1:A\rangle(\tilde{v}_1:U_1)\{\mathsf{true}\}.\mathcal{T}_1' \sqcup\!\!\sqcup \mu\mathbf{t}\langle\tilde{u}_2:B\rangle(\tilde{v}_2:U_2)\{\mathsf{true}\}.\mathcal{T}_2'$
  $\stackrel{def}{=} \mu\mathbf{t}\langle\tilde{u}_1\tilde{u}_2:A\wedge B\rangle(\tilde{v}_1\tilde{v}_2:U_1U_2)\{\mathsf{true}\}.\mathcal{T}_1'\sqcup\!\!\sqcup\mathcal{T}_2'$
  where we assume $\tilde{v_1}\cap\tilde{v_2}=\tilde{u_1}\cap\tilde{u_2}=\varnothing$.
- $\mathbf{t}\langle\tilde{u}_1:A\rangle \sqcup\!\!\sqcup \mathbf{t}\langle\tilde{u}_2:B\rangle \stackrel{def}{=} \mathbf{t}\langle\tilde{u}_1\tilde{u}_2:A\wedge B\rangle$ where we assume $\tilde{u_1}\cap\tilde{u_2}=\varnothing$.
- $\mathsf{end}\sqcup\!\!\sqcup\mathsf{end}=\mathsf{end}$.

The merge function $\mathcal{T}_1\sqcup\!\!\sqcup\mathcal{T}_2$ returns the strongest endpoint assertion $\mathcal{T}$ (if any) such that $\mathcal{T}_1\ni\mathcal{T}$ and $\mathcal{T}_2\ni\mathcal{T}$. Observe that in the cases for delegation (2nd and 4th clauses in Definition O.2), the type $\mathcal{T}@\mathtt{p}$ of $\tilde{v}$ has to be the same in the arguments of $\sqcup\!\!\sqcup$; in fact, such assertions are supposed to validate a same process P and, different types in such clauses would not be consistent with the validations of P.

Definition O.3. The merge operation is extended to assignments Δ in the obvious way, i.e. $\Delta_1\sqcup\!\!\sqcup\Delta_2$ is the pointwise merge of Δ_1 and Δ_2.

$\Delta_1 \uplus \Delta_2$ returns an assignment Δ, assigning all the sessions occurring in Δ_1 or Δ_2. For every $\tilde{s} : \mathcal{T}_1 \,@\, \mathtt{p}$ in Δ_1 and $\tilde{s} : \mathcal{T}_2 \,@\, \mathtt{p}$, Δ maps $\tilde{s}$ to $\mathcal{T}_1 \uplus \mathcal{T}_2 \,@\, \mathtt{p}$. We extend $\uplus$ to multi-ary compositions in the obvious way, writing e.g. $\uplus_i \mathcal{T}_i$. We note:

Lemma O.4. *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be closed local assertions.*

1. $\mathcal{T}_1 \;\sqsupseteq\; \mathcal{T}_1 \uplus \mathcal{T}_2$.
2. *If $\mathcal{T}_1 \sqsupseteq \mathcal{T}'$ and $\mathcal{T}_2 \sqsupseteq \mathcal{T}'$ then $\mathcal{T}_1 \uplus \mathcal{T}_2 \;\sqsupseteq\; \mathcal{T}'$.*

Proof. We prove by induction that, under arbitrary closing substitution, the statements hold for open local assertions. The only non-trivial case is recursive assertions.

We first show (1). By our assumption, $\mathcal{T}_1$ and $\mathcal{T}_2$ are of the same shape in their recursions. We use their finite unfoldings starting from End. We write the i-th unfolding, $\mathcal{T}_1^{(i)}$ and $\mathcal{T}_2^{(i)}$, and use induction on i. The base case is immediate.

$$\mathsf{end} \sqsupseteq \mathsf{end} \uplus \mathsf{end} \tag{O.1}$$

Suppose we have

$$\mathcal{T}_1^{(n)} \;\sqsupseteq\; \mathcal{T}_1^{(n)} \uplus \mathcal{T}_2^{(n)} \tag{O.2}$$

Since $\mathcal{T}_1^{(n+1)}$ is obtained by substituting the assertion variable concerned in $\mathcal{T}_1$ with $\mathcal{T}_1^{(n)}$, and since this substitution is co-variant for the substituting formula, writing the substitution on $\mathcal{T}_2$ by $\mathcal{T}_1^{(n)}$ as $\mathcal{T}_2'$, we obtain:

$$\mathcal{T}_1^{(n+1)} \quad \sqsupseteq \quad \mathcal{T}_1^{(n+1)} \uplus \mathcal{T}_2' \quad \sqsupseteq \quad \mathcal{T}_1^{(n+1)} \uplus \mathcal{T}_2^{(n+1)} \tag{O.3}$$

as required. We now observe that the transitions of $\mathcal{T}_1$ and $\mathcal{T}_2$ are determined by (all of) their finite partial traces since their transitions are deterministic. Hence the above result shows we can construct a refinement relation between $\mathcal{T}_1$ and $\mathcal{T}_1 \uplus \mathcal{T}_2$, by relating each finite node based on the finite unfoldings, as required.

For (2), assume

$$\mathcal{T}_{1,2} \sqsupseteq \mathcal{T}' \tag{O.4}$$

Since each $\mathcal{T}_i$ is characterised by its finite partial traces, this means a refinement is constructed relating the transition nodees of their finite partial to those in $\mathcal{T}'$. At each finite node, $\mathcal{T}_1 \uplus \mathcal{T}_2$ has actions which are the exact combination of $\mathcal{T}_1$ and $\mathcal{T}_2$, they can be related to the corresponding node of $\mathcal{T}'$ so that its actions locally refine those of $\mathcal{T}'$, hence as required. □

Lemma O.5. *$\uplus$ is commutative and associative.*

Proof. Mechanically by induction on open local assertions under arbitrary closing substitutions. □

Lemma O.6. *Let $\{\mathcal{T}_i\}_{i \in I}$ a set of endpoint assertions for which the merge is defined. If for each $i \in I$ $\mathcal{T}_i \sqsupseteq \mathcal{T}$, then $\uplus\{\mathcal{T}_i\}_{i \in I} \sqsupseteq \mathcal{T}$.*

Proof. By Lemma O.4 and Lemma O.5. □

Merging Assertions (2) We also need a refined merging function for endpoint assertions when we need to generate principal assertions for the conditional construct (if-then-else).

Definition O.7 (Parametric Merge of Endpoint Assertions). *The parametric merge of two end-point assertions $\mathcal{T}_1$ and $\mathcal{T}_2$ wrt condition e* (written $\mathcal{T}_1 \sqcup^e \mathcal{T}_2$) is defined recursively as follows:

- $k!(\tilde{v}:U)\{A\};\mathcal{T}_1' \sqcup^e k!(\tilde{v}:U)\{B\};\mathcal{T}_2' \stackrel{def}{=} k!(\tilde{v}:U)\{(e\wedge A)\vee(\neg e\wedge B)\};\mathcal{T}_1'\sqcup^e\mathcal{T}_2'$ where U has the form $\tilde{S}$ or $\langle\mathcal{G}\rangle$
- $k!(\tilde{v}:\mathcal{T}@\mathtt{p})\{A\};\mathcal{T}_1' \sqcup^e k!(\tilde{v}:\mathcal{T}@\mathtt{p})\{B\};\mathcal{T}_2'$ $\stackrel{def}{=} k!(\tilde{v}:\mathcal{T}@\mathtt{p})\{(e\wedge A)\vee(\neg e\wedge B)\};\mathcal{T}_1'\sqcup^e\mathcal{T}_2'$
- $k?(\tilde{v}:U)\{A\};\mathcal{T}_1' \sqcup^e k?(\tilde{v}:U)\{B\};\mathcal{T}_2' \stackrel{def}{=} k?(\tilde{v}:U)\{(e\supset A)\wedge(\neg e\supset B)\};\mathcal{T}_1'\sqcup^e\mathcal{T}_2'$ where U has the form $\tilde{S}$ or $\langle\mathcal{G}\rangle$
- $k?(\tilde{v}:\mathcal{T}@\mathtt{p})\{A\};\mathcal{T}_1' \sqcup^e k?(\tilde{v}:\mathcal{T}@\mathtt{p})\{B\};\mathcal{T}_2' \stackrel{def}{=}$ $k?(\tilde{v}:\mathcal{T}@\mathtt{p})\{(e\supset A)\wedge(\neg e\supset B)\};\mathcal{T}_1'\sqcup^e\mathcal{T}_2'$
- $k\oplus\{\{A_i\}\,l_i:\mathcal{T}_i\}_{i\in I} \sqcup^e k\oplus\{\{B_i\}\,l_i:\mathcal{T}_i'\}_{i\in I} \stackrel{def}{=} k\oplus\{\{(e\wedge A_i)\vee(\neg e\wedge B_i)\}\,l_i:\mathcal{T}_i\sqcup^e\mathcal{T}_i'\}_{i\in I}$
- $k\&\{\{A_i\}l_i : i\in I\}_{\mathcal{T}_i} \sqcup^e k\&\{\{B_i\}l_i:\mathcal{T}_i'\}_{i\in I} \stackrel{def}{=} k\&\{\{(e\supset A_i)\wedge(\neg e\supset B_i)\}l_i:(\mathcal{T}_i\sqcup^e\mathcal{T}_i')\}_{i\in I}$
- $\mu\mathbf{t}\langle\tilde{u}_1:A\rangle(\tilde{v}_1:U_1)\{\mathsf{true}\}.\mathcal{T}_1' \sqcup^e \mu\mathbf{t}\langle\tilde{u}_2:B\rangle(\tilde{v}_2:U_2)\{\mathsf{true}\}.\mathcal{T}_2' \stackrel{def}{=}$ $\mu\mathbf{t}\langle\tilde{u}_1\tilde{u}_2:(e\wedge A)\wedge(\neg e\wedge B)\rangle(\tilde{v}_1\tilde{v}_2:U_1U_2)\{\mathsf{true}\}.\mathcal{T}_1'\sqcup^e\mathcal{T}_2'$ where $\tilde{v_1}\cap\tilde{v_2}=\tilde{u_1}\cap\tilde{u_2}=\varnothing$.
- $\mathbf{t}\langle\tilde{u}_1:A\rangle \sqcup^e \mathbf{t}\langle\tilde{u}_2:B\rangle \stackrel{def}{=} \mathbf{t}\langle\tilde{u}_1,\tilde{u}_2:(e\wedge A)\vee(\neg e\wedge B)\rangle$ where $\tilde{u_1}\cap\tilde{u_2}=\varnothing$.
- $\mathsf{end}\sqcup^e\mathsf{end}=\mathsf{end}$.

The parametric merge takes a boolean expression e as parameter that is the guard of the conditional. The following properties hold.

Lemma O.8. *Let $\approx = \ni \cap \ni^{-1}$. The following properties of $\sqcup^e$ hold.*

- $\mathcal{T}_1\sqcup^e\mathcal{T}_2 \approx \mathcal{T}_1$ *if* $e\downarrow\mathsf{true}$.
- $\mathcal{T}_1\sqcup^e\mathcal{T}_2 \approx \mathcal{T}_2$ *if* $e\downarrow\mathsf{false}$.

Proof. Mechanical from the definition. □

O.5 Generation of Principal Assertions

Judgement for Generation The rules use judgements of the form

$$\mathcal{C};\Gamma_0 \vdash^\star P \blacktriangleright \Gamma;\Delta \tag{O.5}$$

Some remarks are due about each part of the sequent and their meaning.

1. P is a process without predicate annotations;
2. Γ_0 is a global assertion assignment assignment;
3. Γ maps a subset of the shared names in $\mathrm{dom}(\Gamma_0)$ to finite sets of local assertions, writing $\Gamma\vdash a:\langle\mathcal{T}@\mathtt{p}\rangle$ if Δ maps a to a set containing $\langle\mathcal{T}@\mathtt{p}\rangle$; (intuitively, $\Gamma\vdash a:\langle\mathcal{T}@\mathtt{p}\rangle$ states that P has engaged in the session initiating at a as participant $\mathtt{p}$).
4. Δ is an endpoint assertion assignment;

The rules in Figure 15 (cf. page 72) induce an algorithm that takes in input Γ_0, $\mathcal{C}$ and P and generates "most general assertions" Γ and Δ for P under the conditions $\mathcal{C}$ and assignement Γ_0. We remark that the principal general assertion of a program may not exist, in which case the algorithm is supposed to return 'error'.

Without loss of generality, we assume the standard bound variable convention for interaction and assertion variables. The generation rules use the (resp. parametric) merging to merge assertions, in branching and parallel composition (resp. in conditional).

Generation Rules The generation rules are given in Figure 15, page 72. Each rule is naturally obtained, where under the left-hand side environment we derive the right-hand side principal formulae for processes inductively. Because the target is program phrases, we do not need composition of linear channels in the parallel composition, which is crucial for the tractable derivation of principal formulae. We illustrate some of the key rules. The first rule is for session initialisation at a shared name. We only present the accept rule, since the dual rule is similar.

$$\frac{\mathcal{C};\Gamma_0 \vdash^\star P \blacktriangleright \Gamma;\ \Delta, \tilde{s} : \mathcal{T} \mathbin{@} \mathtt{p} \quad \mathcal{T} \ni \Gamma_0(a) \upharpoonright p}{\mathcal{C};\Gamma_0 \vdash^\star a_{[\mathtt{p}]}(\tilde{s}).P \blacktriangleright \Delta} \quad [\textsc{Acc}]$$

Intuitively, in order to generate the principal assertion for a program engaging in a session a as participant $\mathtt{p}$, the assumption first construct the corresponding session types, and checks that such types is a refinement of the projection on $\mathtt{p}$ of the global type assigned to a by Γ_0 (namely what $\mathcal{T} \ni \Gamma_0(a) \upharpoonright p$).

The generation rule for the output prefix is

$$\frac{\mathcal{C};\Gamma_0 \vdash^\star P \blacktriangleright \Gamma;\ \Delta, \tilde{s} : \mathcal{T} \mathbin{@} \mathtt{p}}{\mathcal{C};\Gamma_0 \vdash^\star s_k!\langle \tilde{e} \rangle(\tilde{v});P \blacktriangleright \Gamma;\ \Delta,\ \tilde{s} : k!(\tilde{v})\{\mathcal{C} \wedge \tilde{v} = \tilde{e}\};\mathcal{T} \mathbin{@} \mathtt{p}} \quad [\textsc{Snd}]$$

Namely, when expressions $\tilde{e}$ are sent, the algorithm computes the most general assertion of the continuation and prefixes it with the corresponding send assertion where the computed predicate is the context $\mathcal{C}$ in conjunction with the equation $\tilde{v} = \tilde{e}$ that is the strongest constraints on the interaction variables $\tilde{v}$.

Another remarkable rule for the generation algorithm is the rule for the input action for which we introduce the notation $\exists \tilde{v}(\mathcal{T})$ for the *existential closure of interaction variables on assertions* as follows:

$$\exists \tilde{v}(\mathcal{T}) = \begin{cases} k!(\tilde{u} : \tilde{S})\{\exists \tilde{v}.A\};\mathcal{T} & k!(\tilde{u} : \tilde{S})\{A\};\mathcal{T} \\ k \oplus \{\{\exists \tilde{v}.A_j\}l_j : \mathcal{T}_j\}_{j \in I} & k \oplus \{\{A_j\}l_j : \mathcal{T}_j\}_{j \in I} \\ \mu \mathbf{t}\langle \tilde{e} \rangle(\tilde{u} : \tilde{S})\{\exists \tilde{v}.A\}.\mathcal{T} & \mu \mathbf{t}\langle \tilde{e} \rangle(\tilde{u} : \tilde{S})\{A\}.\mathcal{T} \\ \mathcal{T} & \text{otherwise} \end{cases}$$

Note that only the predicates for output, selection, and recursion are affected, hence the following proposition holds.

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

$$\mathcal{T} \ni \exists \tilde{v}.(\mathcal{T})$$

Proof. Trivially from the definition of refinement and existential closure. □

The generation rule for the input prefix introduces predicate variables as follows:

$$\frac{\mathcal{C} \wedge \Phi(\tilde{v}); \Gamma_0 \vdash^\star P \blacktriangleright \Gamma;\ \Delta, \tilde{s}\!:\!\mathcal{T} \mathbin{@} \mathtt{p}}{\mathcal{C}; \Gamma_0 \vdash^\star s_k?(\tilde{v}); P \blacktriangleright \Gamma;\ \exists\tilde{\tilde{v}}(\Delta),\ \tilde{s}\!:\!k?(\tilde{v})\{\Phi(\tilde{v})\}; \mathcal{T} \mathbin{@} \mathtt{p}} \quad [\textsc{Rcv}]$$

where $\exists\tilde{\tilde{v}}(\Delta) = \{a : \exists\tilde{\tilde{v}}(\mathcal{T} \mathbin{@} \mathtt{p}) \mathbin{@} \mathtt{p} \ :\ \mathcal{T} \mathbin{@} \mathtt{p} \in \Delta(a)\}$. Intuitively, predicate variables "mark" interaction variables of input prefixes with an unknown constraint ϕ (supposed to be freshly introduced at each application of [Rcv]), see §O.2 for further illustration.

The variable rule is simple: we do not introduce any assumption except "unknown" (i.e. assertion variable) and later close it in the recursion rule.

$$\frac{}{\mathcal{C}; \Gamma_0, X : (\tilde{v} : \tilde{S})\mathbf{t}^X\langle\tilde{v}\rangle \mathbin{@} \mathtt{p} \vdash^\star X\langle\tilde{e}\tilde{s}_1..\tilde{s}_n\rangle \blacktriangleright \varnothing;\ \tilde{s}\!:\!\mathbf{t}^X\langle\tilde{u} : \tilde{u} = \tilde{e} \ \wedge\ \mathcal{C}\rangle \mathbin{@} \mathtt{p}} [\textsc{Var}] \quad \text{(O.6)}$$

Above we present the rule for a single session for brevity. Note the rule is indeed strongest since whatever assertion we use to instantiate $\mathbf{t}$, under that assumption, we have the strongest assertion for the session at $\tilde{s}$. This $\mathbf{t}$ is closed by the recursion rule:

$$\frac{\mathcal{C}; \Gamma_0, X : (\tilde{v} : \tilde{S})\mathbf{t}^X\langle\tilde{v}\rangle \mathbin{@} \mathtt{p} \vdash^\star P \blacktriangleright \Delta, \tilde{s}\!:\!\mathcal{T} \mathbin{@} \mathtt{p}}{\mathcal{C}; \Gamma_0 \vdash^\star \mu X\langle\tilde{e}\rangle(\tilde{v}\tilde{s}_1..\tilde{s}_n).P \blacktriangleright \Gamma; \Delta, \tilde{s} : \mu\mathbf{t}_X\langle\tilde{u} : \tilde{u} = \tilde{e}\rangle(\tilde{v})\{\mathsf{true}\}.\mathcal{T} \mathbin{@} \mathtt{p}} [\textsc{Rec}] \quad \text{(O.7)}$$

That this gives a strongest assertion, is proved in Lemma O.14, the last case.

Now we illustrate how this generation rule leads to completeness. Suppose, for a visible program P,

$$\mathsf{true}; \Gamma_0 \vdash^\star P \blacktriangleright \Delta \quad \text{(O.8)}$$

is a sequent derivable by the generation rules. If Γ conforms to Γ_0 (i.e. if $\mathcal{T} \mathbin{@} \mathtt{p} \in \Gamma(a)$ then we have $\mathcal{T} \Subset \Gamma_0(a) \restriction \mathtt{p}$) and

$$\Gamma_0 \models P \rhd \Delta' \quad \text{(O.9)}$$

then we can always find a substitution of predicate variables to concrete ones, say ξ, such that

$$\Delta\xi \Subset \Delta' \quad \text{(O.10)}$$

which easily leads to completeness for visible processes.

Convention O.9. Hereafter, we assume that ξ maps predicate variables $\phi(\tilde{v})$ to assertions A such that $\mathrm{fn}(A) \subseteq \tilde{v}$ and endpoint assertion variables $\mathbf{t}$ to well-asserted endpoint assertions.

O.6 Completeness

In the completeness proof below we use the convention in § O.5 and several observations. The first Lemma is obvious from the definition.

Lemma O.10. $\mathcal{C}, \Gamma \vdash^{ext} P \rhd \Delta \ \supset\ \mathcal{C}\xi, \Gamma\xi \vdash P \rhd \Delta\xi$. *for each* ξ *such that* $\Gamma\xi$ *and* $\Delta\xi$ *are well-asserted.*

Below we note the transitions induced by assertions are deterministic, hence can be completely characterised by its partial traces.

Lemma O.11. *Let $\mathcal{T}$ be a closed recursive assertion and write $\mathcal{T}^{(i)}$ for the $i-th$ unfolding of $\mathcal{T}$ (starting from* End*). Then the union of all partial finite traces of $\mathcal{T}$ coincide with the union of all partial finite traces of all $\mathcal{T}^{(i)}$.*

Proof. First for each i we immediately have the inclusion of the partial finite traces of $\mathcal{T}^{(i)}$ in the partial traces of $\mathcal{T}$. Next any partial finite trace of $\mathcal{T}$ is surely inside $\mathcal{T}^{(i)}$, thus as required. □

We now show the generation rules are provable by the extended validation rules (hence in particular sound). This provablity result later leads to completeness.

Lemma O.12 (Provability by Validation Rules). *If $\mathcal{C};\Gamma_0 \vdash^\star \mathsf{erase}(P) \blacktriangleright \Delta$, then $\mathcal{C},\Gamma_0 \vdash^{\mathsf{ext}} P \rhd \Delta$.*

Proof. We show each generation rules in Figure 15 is an instance of the corresponding validation rule in Figure 14: if the assumption is read as a sequent with $\vdash^{ext}$ rather than $\vdash^\star$, then the same holds for the conclusion, which is enough for the soundness of the each rule in Figure 14.

[SND] By inductive hypothesis

$$\mathcal{C},\Gamma_0 \vdash^{ext} P \rhd \Delta, \tilde{s}; \mathcal{T} \,@\, \mathtt{p} \tag{O.11}$$

and by Substitution Lemma, we have

$$\mathcal{C}[\tilde{e}/\tilde{v}],\Gamma_0 \vdash^{ext} P[\tilde{e}/\tilde{v}] \rhd \Delta[\tilde{e}/\tilde{v}] \tag{O.12}$$

Also,

$$\mathcal{C} \supset (\mathcal{C} \wedge \{\tilde{v} = \tilde{e}\})[\tilde{e}/\tilde{v}] \tag{O.13}$$

trivially holds. Hence we can apply the extended validation rule [SND] and obtain the thesis.

[RCV/RCVNAME] Easy by inductive hypothesis and straightforward application of extended validation rule [RCV]. For the existential elimination, observe that:

1. all occurrences of the abstracted variable are in send/select and recursion instantiation; and
2. all recursion instantiation is used in send/select inside the recursion body.

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

[SNDNAME] By inductive hypothesis $\mathcal{C},\Gamma_0 \vdash^{ext} P \rhd \Delta, \tilde{s}; \mathcal{T} \,@\, \mathtt{p}$; since the assertion generated by [SNDNAME] is $\mathcal{C}$ itself, we can apply [SND] of extended validation and obtain the thesis.

[SEL] The proof follows the case [SND] considering that $\mathcal{C} \supset \mathcal{C}$ is a tautology. Note that I in the validation rule is the singleton $\{j\}$.

[BRA] As in [RCV].

[ACC] Immediate.

[MCAST] By inductive hypothesis from $\mathcal{C};\Gamma_0 \vdash^\star \mathsf{erase}(P) \blacktriangleright \Delta, \tilde{s} : \mathcal{T} \mathbin{@} \mathtt{1}$ it follows

$$\mathcal{C}, \Gamma_0 \vdash^{ext} P \rhd \Delta, \tilde{s} : \Gamma(a) \upharpoonright \mathtt{1} \mathbin{@} \mathtt{1} \tag{O.14}$$

where $\Gamma(a) \upharpoonright \mathtt{1} = \mathcal{T}$. Hence, by [MCAST]of extended validation, we have

$$\mathcal{C}, \Gamma_0 \vdash^{ext} \overline{a}[\mathtt{2..n}](\tilde{s}).P \rhd \Delta \tag{O.15}$$

which, because we have $\Gamma_0, a : \langle \mathcal{T} \mathbin{@} \mathtt{1} \rangle = \Gamma_0$, is equivalent to,

$$\mathcal{C}, \Gamma_0, a : \langle \mathcal{T} \mathbin{@} \mathtt{1} \rangle \vdash^{ext} P \rhd \Delta \tag{O.16}$$

as required.

[IF] By inductive hypothesis we have

$$\mathcal{C} \wedge e \vdash^{ext} P \rhd \Delta_1 \tag{O.17}$$

and

$$\mathcal{C} \wedge \neg e \vdash^{ext} Q \rhd \Delta_2. \tag{O.18}$$

By Lemma O.8 we know

$$\Delta_1 \between \Delta_2 \ \ni \ \Delta_1 \quad \wedge \quad \Delta_1 \between \Delta_2 \ \ni \ \Delta_2 \tag{O.19}$$

Therefore we have both

$$\mathcal{C} \wedge e \vdash^{ext} P \rhd \Delta_1 \between \Delta_2 \tag{O.20}$$

and

$$\mathcal{C} \wedge \neg e \vdash^{ext} Q \rhd \Delta_1 \between \Delta_2 \tag{O.21}$$

by applying rule [CONSEC]. By applying the extended validation rule [IF]we are done.

[CONC] Similar to [IF], applying Lemma O.6.

[INACT, HIDE, VAR] Immediate from the corresponding validation rules.

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

[DEL-OUT/DEL-OUT] Trivial by inductive hypothesis and straightforward application of extended validation rule [SDEL] (resp. [RDEL]).

This exhausts all cases. □

If Γ_0 (resp. Γ) is an assignment of shared names to sets of located endpoint assertions (resp. global assertions) and Δ and Δ' map tuples of sessions channels to located endpoint assertions, we write $\langle \Gamma_0, \Delta \rangle \ni \langle \Gamma, \Delta' \rangle$ if (1) Γ_0 conforms to Γ and (2) $\Delta(\tilde{s}) \ni \Delta'(\tilde{s})$ for all $\tilde{s} \in \mathrm{dom}(\Delta)$.

Definition O.13. We say that ξ is a *concretising substitution* if no predicate variables occur in its codomain.

Lemma O.14 (Completeness via Refinement). *Let $\mathcal{C};\Gamma_0 \models P \rhd \Delta_0$ be an open judgement. If $\mathcal{C};\Gamma_0 \vdash^\star \mathsf{erase}(P) \blacktriangleright \Delta$ then there exists a concretising substitution ξ such that, for any closing substitution σ such that $\mathcal{C}\sigma$ is true,*

– $\Delta\xi\sigma$ *is well asserted, and*
– $\Delta\xi\sigma \Subset \Delta_0$*, (thus* $\mathcal{C}\xi; \Gamma_0 \vdash P \rhd \Delta_0$*).*

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

[ACC/MCAST] We show [ACC]. The proof for [MCAST] is analogue. By hypothesis:

$$\mathcal{C}; \Gamma_0 \models a_{[\mathrm{p}]}(\tilde{s}).P \rhd \Delta_0 \tag{O.22}$$

$$\mathcal{C}; \Gamma_0 \vdash^{\star} a_{[\mathrm{p}]}(\tilde{s}).\mathsf{erase}(P) \blacktriangleright \Delta \tag{O.23}$$

By O.22 and one step of conditional simulation

$$\mathcal{C}; \Gamma_0 \models P \rhd \Delta_0, \tilde{s} : \Gamma_0(a) \upharpoonright \mathrm{p} \mathbin{@} \mathrm{p}. \tag{O.24}$$

By inductive hypothesis

$$\mathcal{C}, \Gamma_0 \vdash^{\star} \mathsf{erase}(P) \blacktriangleright \Delta, \tilde{s} : \mathcal{T} \mathbin{@} \mathrm{p} \text{ where } \Gamma = \Gamma', a : \mathcal{T} \mathbin{@} \mathrm{p}. \tag{O.25}$$

and

$$\langle \Gamma', \Delta, \tilde{s} : \mathcal{T} \mathbin{@} \mathrm{p} \rangle \Subset \langle \Gamma_0, \Delta_0, \tilde{s} : \Gamma_0(a) \upharpoonright \mathrm{p} \mathbin{@} \mathrm{p} \rangle \tag{O.26}$$

which implies $\mathcal{T} \Subset \Gamma_0(a) \upharpoonright \mathrm{p}$ and $\Delta \Subset \Delta_0$ as required.

[SND] Assume we have:

$$\mathcal{C}; \Gamma_0 \models s_k!\langle \tilde{e} \rangle(\tilde{v})\{A\}; P \rhd \Delta_0, \tilde{s} : k!(\tilde{v} : \tilde{S})\{A\}; \mathcal{T}_0 \mathbin{@} \mathrm{p} \tag{O.27}$$

$$\mathcal{C}, \Gamma_0 \vdash^{\star} s_k!\langle \tilde{e} \rangle(\tilde{v}); \mathsf{erase}(P) \blacktriangleright \Delta,\ \tilde{s} : k!(\tilde{v} : \tilde{S})\{\mathcal{C} \wedge \tilde{v} = \tilde{e}\}; \mathcal{T} \mathbin{@} \mathrm{p}. \tag{O.28}$$

Below for brevity we use asserted labelled transition for open processes, which stands for the family its instantiations into closed processes as defined before. We do not mention these substitutions since for each substitution the same reasoning applies. We focus on the local assertion at $\tilde{s}$. Below $\Subset$ is extended to open assertions First, from (O.27), we consider one-step (send) transition for the send action in question, from (in fact under a closing substitution)

$$\mathcal{T}_{\text{before, model}} = k!(\tilde{v} : \tilde{S})\{A\}; \mathcal{T}_0 \mathbin{@} \mathrm{p} \tag{O.29}$$

to

$$\mathcal{T}_{\text{after, model}} = \mathcal{T}_0[\tilde{n}/\tilde{v}] \mathbin{@} \mathrm{p} \tag{O.30}$$

where $\tilde{n}$ are the constants which satisfy A, such that

$$\mathcal{T}_{\text{before, model}} \xrightarrow{k!\langle \tilde{n} \rangle} \mathcal{T}_{\text{after, model}} \tag{O.31}$$

Second, by $\mathcal{C} \wedge \{\tilde{v} = \tilde{e}\} \supset A$, the local assertion in (O.28) at $\tilde{s}$, i.e.

$$\mathcal{T}_{before,gen} = k!(\tilde{v} : \tilde{S})\{\mathcal{C} \wedge \tilde{v} = \tilde{e}\}; \mathcal{T} \mathbin{@} \mathrm{p} \tag{O.32}$$

has the corresponding action, reaching:

$$\mathcal{T}_{\text{after, gen}} = \mathcal{T}[\tilde{n}/\tilde{v}] \mathbin{@} \mathtt{p} \tag{O.33}$$

such that

$$\mathcal{T}_{\text{before, gen}} \overset{k!\langle \tilde{n} \rangle}{\rightarrow} \mathcal{T}_{after,gen}. \tag{O.34}$$

Third, through (each instantiation of) the corresponding transition, the process itself can also reach a term smaller in size, as:

$$s_k!\langle \tilde{e} \rangle(\tilde{v}); P \overset{s_k!\langle \tilde{n} \rangle}{\rightarrow} P[\tilde{n}/\tilde{v}] \tag{O.35}$$

Note this process has the local assertion (O.33) at $\tilde{s}$ (while at other linear/shared channels being invariant), which is direct from the rule (here we use the observation that the generation rules are closed under substitution, which is immediate by their shape). Thus, by induction hypothesis, (each substitution instance of) (O.33) is stronger than (O.30), which, when combined for all closing substitutions, implies (possibly open) (O.33) refines (possibly open) (O.30). Thus by the definition of refinement we know (O.32) refines (O.29), as required.

[SEL/SND-DEL] Similar to [SND].

[RCV] If the following two judgments hold

$$\mathcal{C}; \Gamma_0 \models s_k?(\tilde{v})\{A\}; P \rhd \Delta_0, \tilde{s} : k?(\tilde{v})\{A\}; \mathcal{T}_0 \mathbin{@} \mathtt{p} \tag{O.36}$$

$$\mathcal{C}; \Gamma_0 \vdash^{\star} s_k?(\tilde{v}); \mathsf{erase}(P) \blacktriangleright \exists \tilde{v}(\Delta),\ \tilde{s} : k?(\tilde{v})\{\Phi(\tilde{v})\}; \mathcal{T} \mathbin{@} \mathtt{p} \tag{O.37}$$

Let $\sigma = [\tilde{n}/\tilde{v}]$ be a substitution such that $\mathcal{C} \wedge A\sigma$ holds; by conditional simulation, (O.36) implies that

$$\mathcal{C} \wedge A\sigma; \Gamma_0 \models P\sigma \rhd \Delta_0, \tilde{s} : \mathcal{T}_0\sigma \mathbin{@} \mathtt{p} \tag{O.38}$$

holds. Also, by definition of generation rule, the following judgment must be used in the derivation of (O.37)

$$\mathcal{C} \wedge \Phi(\tilde{v}); \Gamma_0 \vdash^{\star} \mathsf{erase}(P) \blacktriangleright \Delta,\ \tilde{s} : \mathcal{T} \mathbin{@} \mathtt{p} \tag{O.39}$$

where $\Phi(\tilde{v})$ is a fresh predicate variable.

Noticing that applying[15] substitution $[A/\Phi(\tilde{v})]$ to (O.39), by inductive hypothesis we can conclude that there is a substitution ξ such that $(\Delta, \tilde{s} : \mathcal{T} \mathbin{@} \mathtt{p})[A/\Phi(\tilde{v})]\xi\sigma$ is well-asserted, and that $(\Delta, \tilde{s} : \mathcal{T} \mathbin{@} \mathtt{p})[A/\Phi(\tilde{v})]\xi\sigma \ni \Delta_0, \tilde{s} : \mathcal{T}_0\sigma \mathbin{@} \mathtt{p}$. Since,

$$(k?(\tilde{v})\{\Phi(\tilde{v})\}; \mathcal{T} \mathbin{@} \mathtt{p})[A/\Phi(\tilde{v})]\xi\sigma = k?(\tilde{v})\{A\}; \mathcal{T}[A/\Phi(\tilde{v})]\xi\sigma \mathbin{@} \mathtt{p}$$

then trivially

$$k?(\tilde{v})\{A\}; \mathcal{T}[A/\Phi(\tilde{v})]\xi\sigma \mathbin{@} \mathtt{p} \quad \ni \quad k?(\tilde{v})\{A\}; \mathcal{T}_0\sigma \mathbin{@} \mathtt{p}$$

To conclude the proof, we have to show that $(\exists \tilde{v}(\Delta))[A/\Phi(\tilde{v})]\xi\sigma \ni \Delta_0$; this is equivalent to show that the predicates in output, select, and recursion types in $\exists \tilde{v}(\Delta)$ implies the corresponding predicates in Δ under the substitution $[A/\Phi(\tilde{v})]\xi\sigma$. In fact,

[15] The substitution of predicate variables is syntactic and not capture-avoiding (e.g., $\exists v.x > 0 \wedge \Phi(v)[v = x/\Phi(v)]$ yields $\exists v.x > 0 \wedge x = v$.

the assertions in $\exists\tilde{v}(\Delta)$ have the same shape as those in Δ but for the predicates in output, select, and recursion types where variables $\tilde{v}$ get existentially quantified. For each of this predicates, say B, we have to prove that

$$\exists\tilde{v}(B)[A/\Phi(\tilde{v})]\xi\sigma \implies B[A/\Phi(\tilde{v})]\xi\sigma \tag{O.40}$$

Assume that (O.40) does not hold, then we have $B[A/\Phi(\tilde{v})]\xi\sigma$ is false while the hypothesis of (O.40) holds. Two cases are possible: either $\Phi(\tilde{v})$ occurs in B or not. In the former case, by inspection of the generation rules, $\Phi(\tilde{v})$ has to occur in a conjunction and not under any negation; hence, under the substitution $[A/\Phi(\tilde{v})]\xi\sigma$ it would be equivalent to true. For the same reason, the occurrences of $\Phi(\tilde{v})$ in $\exists\tilde{v}(B)$ would be equivalent to true as $\Phi(\tilde{v})$ is replaced by A and $A[\tilde{n}/\tilde{v}]$ holds by hypothesis. This implies that also the antecedent of (O.40) is false, hence the contradiction. If $\Phi(\tilde{v})$ does not occur in B, then hypothesis and conclusion of (O.40) are equivalent, hence we get a contradiction.

[RCV-DELL] Similar to [RCV].

[IF] Suppose that $\mathcal{C};\Gamma_0 \models \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \rhd \Delta$ and $\mathcal{C},\Gamma_0 \vdash^\star \mathsf{if}\ e\ \mathsf{then}\ \mathsf{erase}(P)\ \mathsf{else}\ \mathsf{erase}(Q) \blacktriangleright \Delta_0$. Then we have

$$\mathcal{C} \wedge e;\Gamma_0 \models P \rhd \Delta \quad \text{and} \quad \mathcal{C} \wedge \neg e;\Gamma_0 \models Q \rhd \Delta \tag{O.41}$$

$$\mathcal{C} \wedge e \vdash^\star P \blacktriangleright \Delta_0' \quad \text{and} \quad \mathcal{C} \wedge \neg e \vdash^\star Q \blacktriangleright \Delta_0'' \tag{O.42}$$

where $\Delta_0' \amalg^e \Delta_0'' = \Delta_0$.

The thesis easily follows by the inductive hypothesis (as both Δ_0' and Δ_0'' are well-asserted and refine Δ under some substitutions hence we can invoke Lemma O.8).

[CONC] Assuming $\mathcal{C};\Gamma \models P_1 \mid P_2 \rhd \Delta_1, \Delta_2$ with Δ_1 and Δ_2 disjoint, we have

$$\mathcal{C};\Gamma \models P_1 \rhd \Delta_1 \qquad \text{and} \qquad \mathcal{C};\Gamma \models P_2 \rhd \Delta_2$$

By inductive hypothesis

$$\mathcal{C};\Gamma \blacktriangleright \mathsf{erase}(P_1) \vdash^\star \Delta_1' \qquad \text{and} \qquad \mathcal{C};\Gamma \blacktriangleright \mathsf{erase}(P_2) \vdash^\star \Delta_2'$$

where $\Delta_1' \ni \Delta_1$ and $\Delta_2' \ni \Delta_2$. Therefore $\Delta_1', \Delta_2' \ni \Delta_1, \Delta_2$.

[INACT] Immediate.

[HIDE] By definition of visible process, each hide is followed by a send of the hidden name. The proof proceeds as in the case of [SND]. Assumptions:

$$\mathcal{C};\Gamma \models (\nu a)s_k!(v)\langle a\rangle\{A\}; P \rhd \Delta, \tilde{s} : k!(v : S)\{A\}; \mathcal{T} \mathbin{@} \mathrm{p} \tag{O.43}$$

$$\mathcal{C},\Gamma \vdash^\star (\nu a)s_k!(v)\langle a\rangle\{A\}; \mathsf{erase}(P) \blacktriangleright \Delta_0, \tilde{s} : k!(\tilde{v} : \tilde{S})\{\mathcal{C} \wedge \tilde{v} = \tilde{e}\}; \mathcal{T}_0 \mathbin{@} \mathrm{p}. \tag{O.44}$$

By (O.43), note that for each $\sigma \models \mathcal{C}$ we have $\sigma\dot{v} \mapsto e\sigma \models A\sigma$ (here $\models$ denotes logical interpretation) and

$$\mathcal{C};\Gamma_\sigma \models P_\sigma \rhd \Delta, \tilde{s} : \mathcal{T}[a/v]\sigma \mathbin{@} \mathrm{p}. \tag{O.45}$$

Hence for Substitution Lemma (Lemma L.2) for each substitution σ

$$\mathcal{C};\Gamma \models P \rhd \Delta, \tilde{s} : \mathcal{T}[a/v]\sigma \mathbin{@} \mathrm{p}. \tag{O.46}$$

By assumption (O.44) and construction rule

$$\mathcal{C}, \Gamma \vdash^{\star} \mathsf{erase}(P) \blacktriangleright \Delta_0, \tilde{s} : \mathcal{T}_0 \mathbin{@} \mathrm{p}. \tag{O.47}$$

By inductive hypothesis

$$\Delta_0, \tilde{s} : \mathcal{T}_0 \mathbin{@} \mathrm{p} \quad \ni \quad \Delta, \tilde{s} : \mathcal{T}[a/v] \mathbin{@} \mathrm{p} \tag{O.48}$$

which implies the requested result by

$$\mathcal{C} \supset A[a/v] \quad i.e., \mathcal{C} \wedge \{v = a\} \supset A \quad \text{from (O.43).} \tag{O.49}$$

[VAR] Straightforward.

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

Now write $\ni_n$ for the refinement considered up to the n-length finite traces (since transition is deterministic, considering traces and relating each node suffices). We show for each n, the principal assertion refines a(ny) satisfied assertion. We carry out the argument with a single target session for legibility, which immediately extends to the n cases. Assume:

$$\mathcal{C}; \Gamma \models \mu X \langle \tilde{e}, \tilde{s} \rangle (\tilde{v}, \tilde{s}).P \rhd \tilde{s} : \mathcal{T} \mathbin{@} \mathrm{p} \tag{O.50}$$

and further assume:

$$\mathcal{C}; \Gamma \vdash^{\star} \mu X \langle \tilde{e}, \tilde{s} \rangle (\tilde{v}, \tilde{s}).\mathsf{erase}(P) \rhd \tilde{s} : \mathcal{T}_0 \mathbin{@} \mathrm{p} \tag{O.51}$$

where by construction

$$\mathcal{T}_0 = \mu \mathbf{t} (\tilde{v}) \langle \tilde{e} \rangle . \mathcal{T}_1 \tag{O.52}$$

Now by construction we can also assume $\mathcal{T}$ is also a recursive assertion. By instantiation, we observe:

$$\mathcal{C}; \Gamma_0, X : (\tilde{v} : \tilde{S}) \mathbf{t} \langle \tilde{v} \rangle \sigma \mathbin{@} \mathrm{p} \vdash^{\star} \mathsf{erase}(P) \blacktriangleright \Delta, \tilde{s} : \mathcal{T}_0^{(1)} \mathbin{@} \mathrm{p} \tag{O.53}$$

where $\mathcal{T}^{(1)}$ is the first unfolding of $\mathcal{T}_0$ and σ is $[\mathsf{End}/\mathbf{t}]$. Since P has the same initial finite transitions (up to reaching X) as $\mu X \langle \tilde{e}, \tilde{s} \rangle (\tilde{v}, \tilde{s}).P$, this means up to some finite n, we have

$$\mathcal{T}_0^{(1)} \ \ni_n \ \mathcal{T} \tag{O.54}$$

Because $\ni_i$ is contravariant under substitutions, (O.56) serves as a basis, and by instantiating (O.53) by them, we obtain:

$$\mathcal{T}_0^{(2)} \ \ni_{n+n'} \ \mathcal{T} \qquad (n' \geqslant 1) \tag{O.55}$$

and so on, obtainin for any large m, we have

$$\mathcal{T}_0^{(m)} \ \ni_m \ \mathcal{T} \tag{O.56}$$

By the determinacy of transitions induced by assertions, $\mathcal{T}_0$ is completely characterised by the finite traces of $\cup_i \mathcal{T}_0^{(i)}$. Thus we obtain:

$$\mathcal{T}_0 \ \supset_n \ \mathcal{T}, \tag{O.57}$$

as required.

This exhausts all cases. □

P Full Validation Rules (with Delegation)

Figure 14 offers the full validation rules, adding to the rules in Figure 5 (in Page 9) two rules for delegation.

Q Generation Rules

Figure 15 presents the full generation rules including those for delegation.

- We assume the standard bound name convention. In particular, each binding introduced in the conclusion indicates that the corresponding bound variables/names only occur in its scope.
- For legibility we separate the send/receive rules to those which do not contain name passing and those which only pass names.
- In [RECV], $(\exists v)\Delta$ prefix $\exists v$ to each predicate inside $\{$ and $\}$, i.e. if we have $\{A\}$ then it is replaced by $\{(\exists v)A\}$ then it sets
- In [DEL-OUT] the assignment in the conclusion is defined iff Δ does not assign any local assertion located at $\mathtt{p}'$ for the session channels in $\tilde{t}$.
- Following Convention O.1, we assume the shape of recursions for the same sessions or the sessions for the same shared type of the same role are always identical.

$$\frac{\mathcal{C};\Gamma_0 \vdash^\star P \blacktriangleright \Delta, \tilde{s} : \mathcal{T} \mathbin{@} \mathtt{p} \quad \Gamma_0(a) \upharpoonright p \ni \mathcal{T}}{\mathcal{C};\Gamma_0 \vdash^\star a_{[\mathtt{p}]}(\tilde{s}).P \blacktriangleright \Delta}[\textsc{acc}]$$

$$\frac{\mathcal{C};\Gamma_0 \vdash^\star P \blacktriangleright \Delta, \tilde{s}{:}\mathcal{T} \mathbin{@} \mathtt{1} \quad \Gamma_0(x) \upharpoonright 1 \ni \mathcal{T}}{\mathcal{C};\Gamma_0 \vdash^\star P \blacktriangleright \Delta}[\textsc{mcast}]$$

$$\frac{\mathcal{C};\Gamma_0 \vdash^\star P \blacktriangleright \Delta, \tilde{s}{:}\mathcal{T} \mathbin{@} \mathtt{p}}{\mathcal{C};\Gamma_0 \vdash^\star s_k!\langle\tilde{e}\rangle(\tilde{v});P \blacktriangleright \Delta,\ \tilde{s}{:}k!(\tilde{v})\{\mathcal{C} \wedge \tilde{v} = \tilde{e}\};\mathcal{T} \mathbin{@} \mathtt{p}}[\textsc{Snd}]$$

$$\frac{\mathcal{C} \wedge \Phi(\tilde{v});\Gamma_0 \vdash^\star P \blacktriangleright \Delta, \tilde{s}{:}\mathcal{T} \mathbin{@} \mathtt{p}}{\mathcal{C};\Gamma_0 \vdash^\star s_k?(\tilde{v});P \blacktriangleright \exists\tilde{v}(\Delta),\ \tilde{s}{:}k?(\tilde{v})\{\Phi(\tilde{v})\};\mathcal{T} \mathbin{@} \mathtt{p}}[\textsc{Rcv}]$$

$$\frac{\mathcal{C};\Gamma_0 \vdash^\star P \blacktriangleright \Delta, \tilde{s}{:}\mathcal{T} \mathbin{@} \mathtt{p}}{\mathcal{C};\Gamma_0 \vdash^\star s_k!\langle\!\langle\tilde{t}\rangle\!\rangle(\tilde{v} : \mathcal{T}' \mathbin{@} \mathtt{p}');P \blacktriangleright \Delta, \tilde{s}{:}k!(\tilde{v} : \mathcal{T}' \mathbin{@} \mathtt{p}')\{\mathcal{C}\};\mathcal{T} \mathbin{@} \mathtt{p}, \tilde{t} : \mathcal{T}' \mathbin{@} \mathtt{p}'}[\textsc{Del-Out}]$$

$$\frac{\mathcal{C} \wedge \Phi(\tilde{v});\Gamma_0 \vdash^\star P \blacktriangleright \Delta, \tilde{s}{:}\mathcal{T} \mathbin{@} \mathtt{p}, \tilde{v}{:}\mathcal{T}' \mathbin{@} \mathtt{p}' \qquad \mathcal{T}' \ni \mathcal{T}''}{\mathcal{C};\Gamma_0 \vdash^\star s_k?(\!(\tilde{v} : \mathcal{T}'' \mathbin{@} \mathtt{p}')\!);P \blacktriangleright \Delta, \tilde{s}{:}k?(\tilde{v} : \mathcal{T}'' \mathbin{@} \mathtt{p}')\{\Phi(\tilde{v})\};\mathcal{T} \mathbin{@} \mathtt{p}}[\textsc{Del-In}]$$

$$\frac{\mathcal{C};\Gamma_0 \vdash^\star P \blacktriangleright \Delta, \tilde{s}{:}\mathcal{T} \mathbin{@} \mathtt{p} \quad \Gamma_0(a) = \mathcal{G}}{\mathcal{C};\Gamma_0 \vdash^\star s_k!\langle a\rangle(v);P \blacktriangleright \Delta, \tilde{s}{:}k!(v : \langle\mathcal{G}\rangle)\{\mathcal{C}\};\mathcal{T} \mathbin{@} \mathtt{p}}[\textsc{SndName}]$$

$$\frac{\mathcal{C} \wedge \Phi(v);\Gamma_0, v : \langle\mathcal{G}\rangle \vdash^\star P \blacktriangleright \Delta, \tilde{s}{:}\mathcal{T} \mathbin{@} \mathtt{p}}{\mathcal{C};\Gamma_0 \vdash^\star s_k?(v : \langle\mathcal{G}\rangle);P \blacktriangleright \Delta, \tilde{s}{:}k?(v : \langle\mathcal{G}\rangle)\{\Phi(v)\};\mathcal{T} \mathbin{@} \mathtt{p}}[\textsc{RcvName}]$$

$$\frac{\mathcal{C};\Gamma_0 \vdash^\star P \blacktriangleright \Delta, \tilde{s}{:}\mathcal{T} \mathbin{@} \mathtt{p}}{\mathcal{C};\Gamma_0 \vdash^\star s_k \lhd l;P \blacktriangleright \Delta, \tilde{s}{:}k \oplus \{\mathcal{C}\}l : \mathcal{T} \mathbin{@} \mathtt{p}}[\textsc{sel}]$$

$$\frac{\mathcal{C} \wedge \Phi_i(\tilde{v}_i);\Gamma_0 \vdash^\star P_i \blacktriangleright \Delta_i, \tilde{s}{:}\mathcal{T}_i \mathbin{@} \mathtt{p} \quad \forall i \in I}{\mathcal{C};\Gamma_0 \vdash^\star s_k \rhd \{l_i : P_i\}_{i\in I} \blacktriangleright \sqcup_i \{\Delta_i\}_{i\in I}, \tilde{s}{:}\, k\&\{\{\Phi_i(\tilde{v}_i)\}l_i : \mathcal{T}_i\} \mathbin{@} \mathtt{p}}[\textsc{bra}]$$

$$\frac{\mathcal{C} \wedge e;\Gamma_0 \vdash^\star P_1 \blacktriangleright \Delta_1 \qquad \mathcal{C} \wedge \neg e;\Gamma_0 \vdash^\star P_2 \blacktriangleright \Delta_2}{\mathcal{C};\Gamma_0 \vdash^\star \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \blacktriangleright \Delta_1 \sqcup^e \Delta_2}[\textsc{if}]$$

$$\frac{\mathcal{C};\Gamma_0 \vdash^\star P_i \blacktriangleright \Delta_i \quad (i = 1,2)}{\mathcal{C};\Gamma_0 \vdash^\star P_1 \mid P_2 \blacktriangleright \Delta_1, \Delta_2}[\textsc{conc}] \qquad \frac{\Delta \quad \mathsf{end\ only}}{\mathcal{C};\Gamma_0 \vdash^\star \mathbf{0} \blacktriangleright \Delta}[\textsc{inact}]$$

$$\frac{\mathcal{C};\Gamma_0, a : \langle\mathcal{G}\rangle \vdash^\star P \blacktriangleright \Delta}{\mathcal{C};\Gamma_0 \vdash^\star (\nu a : \langle\mathcal{G}\rangle)P \blacktriangleright \Delta}[\textsc{hide}]$$

$$\frac{-}{\mathcal{C};\Gamma_0, X : (\tilde{v} : \tilde{S})\mathbf{t}_1^X\langle\tilde{v}\rangle \mathbin{@} \mathtt{p}_1..\mathbf{t}_n^X\langle\tilde{v}\rangle \mathbin{@} \mathtt{p}_n \vdash^\star X\langle\tilde{e}\tilde{s}_1..\tilde{s}_n\rangle \blacktriangleright \uplus_{1\leqslant i\leqslant n} \tilde{s}_i : \mathbf{t}_i^X\langle\tilde{u} : \tilde{u} = \tilde{e}\ \wedge\ \mathcal{C}\rangle \mathbin{@} \mathtt{p}_i}[\textsc{Var}]$$

$$\frac{\mathcal{C};\Gamma_0, X : (\tilde{v} : \tilde{S})\mathbf{t}_1^X\langle\tilde{v}\rangle \mathbin{@} \mathtt{p}_1..\mathbf{t}_n^X\langle\tilde{v}\rangle \mathbin{@} \mathtt{p}_n \vdash^\star P \blacktriangleright \Delta, \uplus_{1\leqslant i\leqslant n}\tilde{s}_i : \mathcal{T}_i \mathbin{@} \mathtt{p}_i}{\mathcal{C};\Gamma_0 \vdash^\star \mu X\langle\tilde{e}\rangle(\tilde{v}\tilde{s}_1..\tilde{s}_n).P \blacktriangleright \Delta, \uplus_{1\leqslant i\leqslant n}\tilde{s}_i : \mu\mathbf{t}_X\langle\tilde{u} : \tilde{u} = \tilde{e}\rangle(\tilde{v})\{\mathsf{true}\}.\mathcal{T}_i \mathbin{@} \mathtt{p}_i}[\textsc{Rec}]$$

Fig. 13. Generation rules for programs

$$\frac{\mathcal{C};\Gamma \vdash P \rhd \Delta, \tilde{s}\colon (\Gamma(a) \upharpoonright \mathtt{p}) \mathbin{@} \mathtt{p} \quad \mathtt{p} \not\geqslant 1}{\mathcal{C};\Gamma \vdash a_{[\mathtt{p}]}(\tilde{s}).P \rhd \Delta}[\textsc{Macc}]$$

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

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

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

$$\frac{\mathcal{C};\Gamma \vdash P \rhd \Delta,\ \tilde{s}\colon \mathcal{T} \mathbin{@} \mathtt{p}}{\Gamma \vdash s_k!\langle\!\langle \tilde{t} \rangle\!\rangle(\tilde{v} : \mathcal{T}' \mathbin{@} \mathtt{q})\{A\}; P \rhd \Delta, \tilde{s}\colon k!(\tilde{v} : \mathcal{T}' \mathbin{@} \mathtt{q})\{A\}; \mathcal{T} \mathbin{@} \mathtt{p}, \tilde{t}\colon \mathcal{T}' \mathbin{@} \mathtt{q}}[\textsc{Sdel}]$$

$$\frac{\mathcal{C} \wedge A; \Gamma \vdash P \rhd \Delta, \tilde{s}\colon \mathcal{T} \mathbin{@} \mathtt{p},\ \tilde{v}\colon \mathcal{T}' \mathbin{@} \mathtt{q}}{\mathcal{C};\Gamma \vdash s_k?(\!(\tilde{v} : \mathcal{T}' \mathbin{@} \mathtt{q})\!)\{A\}; P \rhd \Delta, \tilde{s}\colon k?(\tilde{v} : \mathcal{T}' \mathbin{@} \mathtt{q})\{A\}; \mathcal{T} \mathbin{@} \mathtt{p}}[\textsc{Rdel}]$$

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

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

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

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

$$\frac{\mathcal{C}; \Gamma, a : \mathcal{G} \vdash P \rhd \Delta \quad a \notin \mathrm{fn}(\mathcal{C}, \Gamma, \Delta)}{\mathcal{C}; \Gamma \vdash (\nu a : \mathcal{G})P \rhd \Delta}[\textsc{Hide}]$$

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

$$\frac{\mathcal{C}\ ;\ \Gamma, X : (\tilde{v} : U)\mathcal{T}_1 \mathbin{@} \mathtt{p}_1..\mathcal{T}_n \mathbin{@} \mathtt{p}_n \vdash P \rhd \tilde{s_1} : \mathcal{T}_1 \mathbin{@} \mathtt{p}_1..\tilde{s_n} : \mathcal{T}_n \mathbin{@} \mathtt{p}_n}{\mathcal{C}\ ;\ \Gamma \vdash \mu X\langle \tilde{e}\tilde{s}_1..\tilde{s}_n \rangle(\tilde{v}\tilde{s}_1..\tilde{s}_n).P \rhd\ \tilde{s_1} : \mathcal{T}_1[\tilde{e}/\tilde{v}] \mathbin{@} \mathtt{p}_1..\tilde{s_n} : \mathcal{T}_n[\tilde{e}/\tilde{v}] \mathbin{@} \mathtt{p}_n}[\textsc{Rec}]$$

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

Fig. 14. Validation rules for program phrases with delegation

$$\frac{\mathcal{C};\Gamma_0 \vdash^\star P \blacktriangleright \Delta, \tilde{s} : \mathcal{T} \mathbin{@} \mathtt{p} \quad \Gamma_0(a) \upharpoonright p \ni \mathcal{T}}{\mathcal{C};\Gamma_0 \vdash^\star a_{[\mathtt{p}]}(\tilde{s}).P \blacktriangleright \Delta}[\textsc{acc}]$$

$$\frac{\mathcal{C};\Gamma_0 \vdash^\star P \blacktriangleright \Delta, \tilde{s}{:}\mathcal{T} \mathbin{@} \mathtt{1} \quad \Gamma_0(x) \upharpoonright 1 \ni \mathcal{T}}{\mathcal{C};\Gamma_0 \vdash^\star P \blacktriangleright \Delta}[\textsc{mcast}]$$

$$\frac{\mathcal{C};\Gamma_0 \vdash^\star P \blacktriangleright \Delta, \tilde{s}{:}\mathcal{T} \mathbin{@} \mathtt{p}}{\mathcal{C};\Gamma_0 \vdash^\star s_k!\langle\tilde{e}\rangle(\tilde{v});P \blacktriangleright \Delta,\ \tilde{s}{:}k!(\tilde{v})\{\mathcal{C} \wedge \tilde{v} = \tilde{e}\};\mathcal{T} \mathbin{@} \mathtt{p}}[\textsc{Snd}]$$

$$\frac{\mathcal{C} \wedge \Phi(\tilde{v});\Gamma_0 \vdash^\star P \blacktriangleright \Delta, \tilde{s}{:}\mathcal{T} \mathbin{@} \mathtt{p}}{\mathcal{C};\Gamma_0 \vdash^\star s_k?(\tilde{v});P \blacktriangleright \exists\tilde{v}(\Delta),\ \tilde{s}{:}k?(\tilde{v})\{\Phi(\tilde{v})\};\mathcal{T} \mathbin{@} \mathtt{p}}[\textsc{Rcv}]$$

$$\frac{\mathcal{C};\Gamma_0 \vdash^\star P \blacktriangleright \Delta, \tilde{s}{:}\mathcal{T} \mathbin{@} \mathtt{p}}{\mathcal{C};\Gamma_0 \vdash^\star s_k!\langle\!\langle\tilde{t}\rangle\!\rangle(\tilde{v} : \mathcal{T}' \mathbin{@} \mathtt{p}');P \blacktriangleright \Delta, \tilde{s}{:}k!(\tilde{v} : \mathcal{T}' \mathbin{@} \mathtt{p}')\{\mathcal{C}\};\mathcal{T} \mathbin{@} \mathtt{p}, \tilde{t} : \mathcal{T}' \mathbin{@} \mathtt{p}'}[\textsc{Del-Out}]$$

$$\frac{\mathcal{C} \wedge \Phi(\tilde{v});\Gamma_0 \vdash^\star P \blacktriangleright \Delta, \tilde{s}{:}\mathcal{T} \mathbin{@} \mathtt{p}, \tilde{v}{:}\mathcal{T}' \mathbin{@} \mathtt{p}' \qquad \mathcal{T}' \ni \mathcal{T}''}{\mathcal{C};\Gamma_0 \vdash^\star s_k?(\!(\tilde{v} : \mathcal{T}'' \mathbin{@} \mathtt{p}')\!);P \blacktriangleright \Delta, \tilde{s}{:}k?(\tilde{v} : \mathcal{T}'' \mathbin{@} \mathtt{p}')\{\Phi(\tilde{v})\};\mathcal{T} \mathbin{@} \mathtt{p}}[\textsc{Del-In}]$$

$$\frac{\mathcal{C};\Gamma_0 \vdash^\star P \blacktriangleright \Delta, \tilde{s}{:}\mathcal{T} \mathbin{@} \mathtt{p} \quad \Gamma_0(a) = \mathcal{G}}{\mathcal{C};\Gamma_0 \vdash^\star s_k!\langle a\rangle(v);P \blacktriangleright \Delta, \tilde{s}{:}k!(v : \langle\mathcal{G}\rangle)\{\mathcal{C}\};\mathcal{T} \mathbin{@} \mathtt{p}}[\textsc{SndName}]$$

$$\frac{\mathcal{C} \wedge \Phi(v);\Gamma_0, v : \langle\mathcal{G}\rangle \vdash^\star P \blacktriangleright \Delta, \tilde{s}{:}\mathcal{T} \mathbin{@} \mathtt{p}}{\mathcal{C};\Gamma_0 \vdash^\star s_k?(v : \langle\mathcal{G}\rangle);P \blacktriangleright \Delta, \tilde{s}{:}k?(v : \langle\mathcal{G}\rangle)\{\Phi(v)\};\mathcal{T} \mathbin{@} \mathtt{p}}[\textsc{RcvName}]$$

$$\frac{\mathcal{C};\Gamma_0 \vdash^\star P \blacktriangleright \Delta, \tilde{s}{:}\mathcal{T} \mathbin{@} \mathtt{p}}{\mathcal{C};\Gamma_0 \vdash^\star s_k \lhd l;P \blacktriangleright \Delta, \tilde{s}{:}k \oplus \{\mathcal{C}\}l{:}\mathcal{T} \mathbin{@} \mathtt{p}}[\textsc{sel}]$$

$$\frac{\mathcal{C} \wedge \Phi_i(\tilde{v}_i);\Gamma_0 \vdash^\star P_i \blacktriangleright \Delta_i, \tilde{s}{:}\mathcal{T}_i \mathbin{@} \mathtt{p} \quad \forall i \in I}{\mathcal{C};\Gamma_0 \vdash^\star s_k \rhd \{l_i{:}\ P_i\}_{i\in I} \blacktriangleright \sqcup_i \{\Delta_i\}_{i\in I}, \tilde{s}{:}\ k\&\{\{\Phi_i(\tilde{v}_i)\}l_i : \mathcal{T}_i\} \mathbin{@} \mathtt{p}}[\textsc{bra}]$$

$$\frac{\mathcal{C} \wedge e;\Gamma_0 \vdash^\star P_1 \blacktriangleright \Delta_1 \qquad \mathcal{C} \wedge \neg e;\Gamma_0 \vdash^\star P_2 \blacktriangleright \Delta_2}{\mathcal{C};\Gamma_0 \vdash^\star \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \blacktriangleright \Delta_1 \sqcup^e \Delta_2}[\textsc{if}]$$

$$\frac{\mathcal{C};\Gamma_0 \vdash^\star P_i \blacktriangleright \Delta_i \quad (i = 1,2)}{\mathcal{C};\Gamma_0 \vdash^\star P_1 \mid P_2 \blacktriangleright \Delta_1, \Delta_2}[\textsc{conc}] \qquad \frac{\Delta \quad \mathsf{end\ only}}{\mathcal{C};\Gamma_0 \vdash^\star \mathbf{0} \blacktriangleright \Delta}[\textsc{inact}]$$

$$\frac{\mathcal{C};\Gamma_0, a : \langle\mathcal{G}\rangle \vdash^\star P \blacktriangleright \Delta}{\mathcal{C};\Gamma_0 \vdash^\star (\nu a : \langle\mathcal{G}\rangle)P \blacktriangleright \Delta}[\textsc{hide}]$$

$$\frac{-}{\mathcal{C};\Gamma_0, X : (\tilde{v} : \tilde{S})\mathbf{t}_1^X\langle\tilde{v}\rangle \mathbin{@} \mathtt{p}_1..\mathbf{t}_n^X\langle\tilde{v}\rangle \mathbin{@} \mathtt{p}_n \vdash^\star X\langle\tilde{e}\tilde{s}_1..\tilde{s}_n\rangle \blacktriangleright \uplus_{1\leqslant i\leqslant n} \tilde{s}_i : \mathbf{t}_i^X\langle\tilde{u} : \tilde{u} = \tilde{e}\ \wedge\ \mathcal{C}\rangle \mathbin{@} \mathtt{p}_i}[\textsc{Var}]$$

$$\frac{\mathcal{C};\Gamma_0, X : (\tilde{v} : \tilde{S})\mathbf{t}_1^X\langle\tilde{v}\rangle \mathbin{@} \mathtt{p}_1..\mathbf{t}_n^X\langle\tilde{v}\rangle \mathbin{@} \mathtt{p}_n \vdash^\star P \blacktriangleright \Delta, \uplus_{1\leqslant i\leqslant n}\tilde{s}_i : \mathcal{T}_i \mathbin{@} \mathtt{p}_i}{\mathcal{C};\Gamma_0 \vdash^\star \mu X\langle\tilde{e}\rangle(\tilde{v}\tilde{s}_1..\tilde{s}_n).P \blacktriangleright \Delta, \uplus_{1\leqslant i\leqslant n}\tilde{s}_i : \mu\mathbf{t}_X\langle\tilde{u} : \tilde{u} = \tilde{e}\rangle(\tilde{v})\{\mathsf{true}\}.\mathcal{T}_i \mathbin{@} \mathtt{p}_i}[\textsc{Rec}]$$

Fig. 15. Generation rules for programs