Resolving Non-determinism in Choreographies ⋆

Laura Bocchi[1], Hernán Melgratti[2], and Emilio Tuosto[3]

[1] Department of Computing, Imperial College London, UK
[2] Departamento de Computación, FCEyN, Universidad de Buenos Aires - Conicet, Argentina
[3] Department of Computer Science, University of Leicester

Abstract. Resolving non-deterministic choices of choreographies is a crucial task. We introduce a novel notion of realisability for choreographies –called *whole-spectrum implementation*– that rules out deterministic implementations of roles that, no matter which context they are placed in, will never follow one of the branches of a non-deterministic choice. We show that, under some conditions, it is decidable whether an implementation is whole-spectrum. As a case study, we analyse the POP protocol under the lens of whole-spectrum implementation.

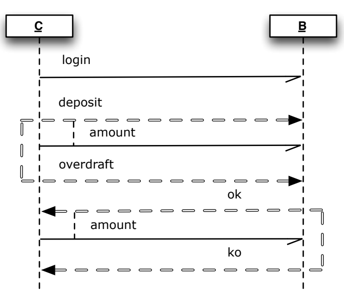
1 Introduction

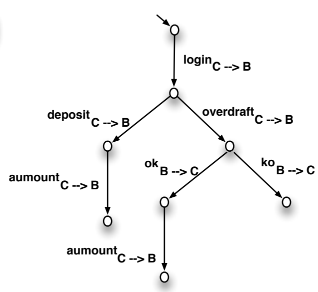
The context A *choreography* describes the expected interactions of a system in terms of the message exchanged between its components (aka *roles*):

> *"Using the Web Services Choreography specification, a contract containing a global definition of the common ordering conditions and constraints under which messages are exchanged, is produced [...]. Each party can then use the global definition to build and test solutions that conform to it. The global specification is in turn realised by combination of the resulting local systems [...]"*

The first part of the excerpt above taken from Kavantzas et al. [2004] envisages a choreography as a global contract regulating the exchange of messages; the last part identifies a distinctive element of choreographies: the global definition can be used to check the conformance of local components so to (correctly) realise the global contract. Choreographies allows for the combination of independently developed distributed components (e.g., services) while hiding implementation details. Moreover, the communication pattern specified in the choreography suffices to check each component.

For illustration, take a simple choreography, hereafter called *ATM*, involving the cash machine of a bank B and a customer C depicted as either of the following diagrams:





In the diagram on the left, the doubly stroked lines represent choices and the dashed lines connect interactions with the branches where they occur. On the right, *ATM* is expressed in terms of the conversation protocols of Fu et al. [2005].

⋆ This work has been partially sponsored by the project EU 7FP under grant agreement no. 295261 (MEALS), ANPCyT Project BID-PICT-2008-00319, Ocean Observatories Initiative and EPSRC EP/K034413/1.

After successful authentication, B offers a deposit and an overdraft service to C. When opting for a deposit, C indicates the amount of money to be deposited. If C asks to overdraft then B can either grant or deny it; in the former case C will communicate the amount of money required.

On realisations A set of processes is a *realisation* of a choreography when the behaviour emerging from their concurrent execution matches the behaviour specified by the choreography. A choreography is *realisable* when it has a realisation.

A realisation of *ATM* can be given using two CCS-like processes Milner [1989] (augmented with internal $\_ \oplus \_$ and external $\_ + \_$ choice operators) for roles B and C:

$$T_{\mathtt{B}} = \mathsf{login}.(\mathsf{deposit}.\mathsf{amount} \ + \ \mathsf{overdraft}.(\overline{\mathsf{ok}}.\mathsf{amount} \ \oplus \ \overline{\mathsf{ko}}))$$
$$T_{\mathtt{C}} = \overline{\mathsf{login}}.(\overline{\mathsf{deposit}}.\overline{\mathsf{amount}} \ \oplus \ \overline{\mathsf{overdraft}}.(\mathsf{ok}.\overline{\mathsf{amount}} \ + \ \mathsf{ko}))$$

In words, $T_{\mathtt{B}}$ specifies that, after C logs in, B waits to interact either on deposit or on overdraft; in the latter case, B non-deterministically decides whether to grant or deny the overdraft; $T_{\mathtt{C}}$ is the dual of $T_{\mathtt{B}}$. Note that *ATM* uses non-determinism to avoid specifying the criteria for B to grant or deny an overdraft. The use of non-determinism is also reflected in realisations, in fact $T_{\mathtt{B}}$ uses the internal choice operator $\_ \oplus \_$ to model the reaction when C requests an overdraft.

Choreographies can be interpreted either as *constraints* or as *obligations* of distributed interactions Lohmann and Wolf [2011]. The former interpretation (aka *partial* Lohmann and Wolf [2011] or weak Su et al. [2007]) admits a realisation if it exhibits a subset of the behaviour. For instance, take

$$T'_{\mathtt{B}} = \mathsf{login}.(\mathsf{deposit}.\mathsf{amount} \ + \ \mathsf{overdraft}.\overline{\mathsf{ko}})$$

then $T'_{\mathtt{B}}$ and $T_{\mathtt{C}}$ form a partial realisation of *ATM* where requests of overdraft are consistently denied. On the contrary, when interpreting choreographies as obligations, a realisation is admissible if it is able to exhibit *all* interaction sequences (hence such realisations are also referred to as *complete* realisations Lohmann and Wolf [2011]). For instance, $T_{\mathtt{B}}$ and $T_{\mathtt{C}}$ form a complete realisation of *ATM*.

The problem Choreographies typically yield non-deterministic specifications; here we explore the problem of resolving their non-determinism. In fact, despite being a valuable abstraction mechanism, non-determinism has to be implemented using deterministic constructs such as conditional branch statements.

Using again *ATM*, we illustrate that traditional notions of complete realisation are not fully satisfactory. The non-deterministic choice in $T_{\mathtt{B}}$ abstracts away from the actual conditions used in implementations to resolve the choice. This permits, e.g., different banks to adopt different policies depending, for instance, on the type of the clients' accounts. Consider the (deterministic) implementations B_1 and B_2 of $T_{\mathtt{B}}$ below (for brevity, each name refers to the interaction of *ATM* with the same initial):

$$B_i ::= l(c); (d(); a(x); Q + o(); P_i(c)) \quad \text{for } i = 1, 2 \quad (Q \text{ is immaterial})$$
$$P_1(c) ::= \texttt{if } \mathit{check}(c) : \ \overline{ok}.a(x) \texttt{ else } \overline{ko} \qquad \text{and} \qquad P_2(c) ::= \overline{ko}$$

The expression $check(c)$ in P_1 deterministically discriminates if the overdraft should be granted. Clearly both B_1 and B_2 can be used as implementations of $T_{\mathtt{B}}$ in *partial realisations* of the choreography.[4] (as e.g. in Dezani-Ciancaglini and de' Liguoro [2009]).

Conversely, neither B_1 nor B_2 can be used in a *complete realisation*. This is straightforward for B_2 (unable to interact over ok after receiving an overdraft request), but not so evident for B_1. Depending on the credentials c sent by the customer to login, $check(c)$ will evaluate either to `true` or to `false`. Therefore, B_2 will be unable to exhibit both branches. This will be the case for any possible deterministic implementation of *ATM*: only one branch will be matched. Consequently, there is not a complete, deterministic realisation for *ATM*.

We prefer B_1 to B_2 arguing that they are not equally appealing when interpreting choreographies as obligations. In fact, B_2 consistently precludes one of the alternatives while B_1 guarantees only one or the other alternative (provided that $check$ is not the constant map) depending on the deterministic implementation of the role $T_{\mathtt{C}}$.

Contributions and synopsis We introduce *whole-spectrum implementation* (WSI), a new interpretation of choreographies as interaction obligations. A WSI of a role R guarantees that, whenever the choreography allows R to make an internal choice, there is a context (i.e., an implementation of the remaining roles) for which (the implementation of) R chooses such alternative. We illustrate the use of WSI to analyse the POP2 protocol (i.e., choreography § 2.2, implementation § 3.1, and verification § 5.1).

We develop our results in a behavioural typing framework since types directly relate specifications to implementations, but our results can be established in different contexts (c.f. Appendix F). Our technical contributions are a formalisation of WSI and a sound type system that guarantees that typable processes form WSIs. For instance, our type system validates B_1 against $T_{\mathtt{B}}$ while it discards B_2. Typing is decidable if so is the logic expressing internal conditions. We relate a denotational semantics of global types (featuring optional behaviours) to the operational semantics of local types (c.f. Thm. 3). Finally, the strong connection between local types and processes ensures that well-typed processes enjoy whole-spectrum implementability (c.f. Thm. 4).

2 Global and Local Types

Our types elaborate from Lange and Tuosto [2012] and use a more tractable form of iteration (discussed below). We fix a countably infinite set $\mathbb{C}$ of *(session channel) names* ranged over by $u, y, s, \ldots$ and a countably infinite set $\mathbb{P}$ of *(participants) roles* ranged over by $\mathtt{p}, \mathtt{q}, \mathtt{r}, \ldots$ (with $\mathbb{C} \cap \mathbb{P} = \emptyset$). Basic data types, called *sorts*, (e.g., booleans `Bool`, integers `Int`, strings `Str`, record types, etc.) are assumed; `U` ranges over sorts.

Tuples are written in bold font and, abusing notation, we use them to represent their underlying set (e.g., if $\mathbf{y} = (y_1, y_2, y_3)$, we write $y_2 \in \mathbf{y}$ for $y_2 \in \{y_1, y_2, y_3\}$). Let $\#X$ denote the cardinality of a set X. Write $\{\_/\_\}$ for substitutions and in $\{\mathbf{y}/\mathbf{s}\}$ assume that $\mathbf{s}$ and $\mathbf{y}$ have the same length, that the components of $\mathbf{y}$ are pairwise disjoint, and that the i-th element of $\mathbf{y}$ is replaced by the i-th element of $\mathbf{s}$.

[4] For instance, both B_1 and B_2 type-check against $T_{\mathtt{B}}$ considered as a session type due to the fact that subtyping for session types Gay and Hole [2005] is contra-variant with respect to internal choices (and covariant with respect to external choices).

2.1 Types

A *global type term* (GTT, for short) $\mathtt{G}$ is derived by the following grammar:

$$\mathtt{G} ::= \mathtt{p} \to \mathtt{q} : y \langle \mathtt{U} \rangle \;\big|\; \mathtt{G} + \mathtt{G} \;\big|\; \mathtt{G} \,|\, \mathtt{G} \;\big|\; \mathtt{G}; \mathtt{G} \;\big|\; \mathtt{G}^{*^f} \;\big|\; \mathtt{end}$$

In words, a GTT can either be a single interaction, the non-deterministic ($\_ + \_$), parallel ($\_ \mid \_$), or sequential ($\_;\_$) composition of two GTTs, the iteration of a GTT ($\_^{*^{\cdot}}$), or the empty term. Hereafter, we tacitly assume $\mathtt{p} \neq \mathtt{q}$ in any interaction $\mathtt{p} \to \mathtt{q} : y \langle \mathtt{U} \rangle$. As in Castagna et al. [2012], we adopt a form of iteration to statically check for WSI (see § 4); in $\mathtt{G}^{*^f}$, f injectively maps roles in $\mathtt{G}$ to pairs of channels and sorts; i.e., $f(\mathtt{p}) = y\langle \mathtt{U} \rangle$ is used to notify $\mathtt{p} \in \mathtt{G}$ when the iteration ends. We use $\mathrm{cod}(f)$ to denote the set of channels appearing as first component in the image of f.

For a GTT $\mathtt{G}$, $\mathtt{ch}(\mathtt{G}) \subseteq \mathbb{C}$ are the names, $\mathcal{P}(\mathtt{G})$ are the participants, and $\mathtt{fst}(\mathtt{G})$ are the initially enabled input and output actions of each each participant in $\mathtt{G}$; e.g., in

$$\mathtt{G_f} = \mathtt{p} \to \mathtt{q} : y \langle \mathtt{U} \rangle;\ \mathtt{q} \to \mathtt{s} : z \langle \mathtt{U} \rangle \tag{2.1}$$

$\mathtt{ch}(\mathtt{G_f}) = \{y, z\}$, $\mathcal{P}(\mathtt{G_f}) = \{\mathtt{p}, \mathtt{q}, \mathtt{s}\}$, and $\mathtt{fst}(\mathtt{G_f}) = \{(\mathtt{p}, \overline{y}), (\mathtt{q}, y), (\mathtt{s}, z)\}$. Formal definitions of such maps are standard and relegated in Appendix A.

A *global type* is defined by an equation $\mathcal{G}(\mathbf{y}) \stackrel{\triangle}{=} \mathtt{G}$ where $\mathbf{y} \subseteq \mathbb{C}$ are pairwise distinct names and $\mathtt{ch}(\mathtt{G}) \subseteq \mathbf{y}$. The syntax of global types explicitly mentions names as they are needed when typing processes to check if they form a WSI (c.f. § 5). We write $\mathcal{G}(\mathbf{y})$ when the defining equation of a global type is understood or its corresponding GTT is immaterial; we write $\mathcal{G}$ or $\mathtt{G}$ instead of $\mathcal{G}(\mathbf{y})$ when parameters are understood.

GTTs are taken up to *structural congruence*, defined as the smallest congruence $\equiv$ such that $\_;\_$, $\_ \mid \_$, and $\_ + \_$ form a monoid with identity $\mathtt{end}$ and $\_ \mid \_$ and $\_ + \_$ are commutative. Two global types $\mathcal{G}_1(\mathbf{y}_1) \stackrel{\triangle}{=} \mathtt{G}_1$ and $\mathcal{G}_2(\mathbf{y}_2) \stackrel{\triangle}{=} \mathtt{G}_2$ are structurally equivalent when $\mathtt{G}_1 \equiv \mathtt{G}_2\{\mathbf{y}_2/\mathbf{y}_1\}$, in which case we write $\mathcal{G}_1 \equiv \mathcal{G}_2$.

We define the set of *ready participants* of $\mathtt{G}$ as follows.

$$\mathtt{rdy}(\mathtt{p} \to \mathtt{q} : y \langle \mathtt{U} \rangle) = \{\mathtt{p}\} \quad \mathtt{rdy}(\mathtt{G} + \mathtt{G}') = \mathtt{rdy}(\mathtt{G} \mid \mathtt{G}') = \mathtt{rdy}(\mathtt{G}) \cup \mathtt{rdy}(\mathtt{G}') \quad \mathtt{rdy}(\mathtt{end}) = \emptyset$$
$$\mathtt{rdy}(\mathtt{G};\mathtt{G}') = \mathtt{rdy}(\mathtt{G}),\ \text{if}\ \mathtt{rdy}(\mathtt{G}) \neq \emptyset \quad \mathtt{rdy}(\mathtt{G};\mathtt{G}') = \mathtt{rdy}(\mathtt{G}'),\ \text{if}\ \mathtt{rdy}(\mathtt{G}) = \emptyset \quad \mathtt{G}^{*^f} = \mathtt{rdy}(\mathtt{G})$$

(note that for the GTT (2.1) $\mathtt{rdy}(\mathtt{G_f}) = \{\mathtt{p}\}$). We extend $\mathcal{P}(\_)$ and $\mathtt{rdy}(\_)$ to global types $\mathcal{G}(\mathbf{y}) \stackrel{\triangle}{=} \mathtt{G}$ by defining $\mathcal{P}(\mathcal{G}) = \mathcal{P}(\mathtt{G})$ and $\mathtt{rdy}(\mathcal{G}) = \mathtt{rdy}(\mathtt{G})$.

As customary in session types, we restrict the attention to *well-formed* global types in order to rule out specifications that cannot be implemented distributively. A global type is *well-formed* when it enjoys the following properties: *linearity*, *single threadness*, *single selector* Honda et al. [2008], *knowledge of choice* Castagna et al. [2012], Honda et al. [2008], and *single iteration controller*. All but the last condition are standard. The last condition is specific to our form of iteration; informally, it requires that in each interation there is a unique participant that decides when to exit the loop (see Appendix B for its definition).

A *local type term* (LTT for short) $\mathtt{T}$ is derived by the following grammar:

$$\mathtt{T} ::= \bigoplus_{i \in I} y_i!\mathtt{U}_i;\mathtt{T}_i \;\Big|\; \sum_{i \in I} y_i?\mathtt{U}_i;\mathtt{T}_i \;\Big|\; \mathtt{T}_1;\mathtt{T}_2 \;\Big|\; \mathtt{T}^* \;\Big|\; \mathtt{end}$$

An LTT is either an internal ($\bigoplus$) or external ($\sum$) guarded choice, the sequential composition of LTTs $\_;\_$, an iteration $\_^*$, or the empty term $\mathtt{end}$. The set $\mathtt{ch}(\mathtt{T})$ of channels of $\mathtt{T}$ is standard (see Appendix A).

A *local type* is defined by an equation $\mathcal{T}(\mathbf{y}) \stackrel{\triangle}{=} \mathtt{T}$ where $\mathbf{y}$ are pairwise distinct names and $\mathtt{ch}(\mathtt{T}) \subseteq \mathbf{y}$. Hereafter, we write $\mathcal{T}(\mathbf{y})$ when the defining equation of a local type is understood or its corresponding LTT is immaterial; we may write $\mathcal{T}$ or $\mathtt{T}$ instead of $\mathcal{T}(\mathbf{y})$ when parameters are understood. We overload $\equiv$ to denote the structural congruence over local types defined as the least congruence such that internal and external choice are associative, commutative and have $\mathtt{end}$ as identity, while $\_;\_$ is associative. In the following, we consider types up-to structural congruence.

The projection operation extracts the local types from a global type. For a well-formed GTT $\mathtt{G}$ and $\mathtt{r} \in \mathbb{P}$, $\mathtt{G}\!\upharpoonright\!\mathtt{r}$ is the *projection* of $\mathtt{G}$ on $\mathtt{r}$ and it is defined homomorphically on $\_ \oplus \_$, $\_ + \_$, and $\_;\_$ and as follows on the remaining constructs:

$$\mathtt{G}\!\upharpoonright\!\mathtt{r} = \begin{cases} y!\mathtt{U} \quad (\text{ resp. } y?\mathtt{U}) & \text{if } \mathtt{G} = \mathtt{r} \to \mathtt{p} : y\,\langle \mathtt{U}\rangle \quad (\text{ resp. if } \mathtt{G} = \mathtt{p} \to \mathtt{r} : y\,\langle \mathtt{U}\rangle) \\ \mathtt{G}_i\!\upharpoonright\!\mathtt{r} & \text{if } \mathtt{G} = \mathtt{G}_1 \mid \mathtt{G}_2 \text{ and } \mathtt{r} \not\in \mathcal{P}(\mathtt{G}_j) \text{ with } j \neq i \in \{1,2\} \\ (\mathtt{G}_1\!\upharpoonright\!\mathtt{r})^*; b_1!\mathtt{U}_1; \ldots; b_n!\mathtt{U}_\mathtt{n} & \text{if } \mathtt{G} = \mathtt{G}_1^{*^f},\ \mathrm{cod}(f) = \{b_1\langle \mathtt{U}_1\rangle, \ldots, b_n\langle \mathtt{U}_\mathtt{n}\rangle\},\ \text{and } \mathtt{r} \in \mathtt{rdy}(\mathtt{G}_1) \\ (\mathtt{G}_1\!\upharpoonright\!\mathtt{r})^*; b?\mathtt{U} & \text{if } \mathtt{G} = \mathtt{G}_1^{*^f},\ f(\mathtt{r}) = b\langle \mathtt{U}\rangle,\ \text{and } \mathtt{r} \not\in \mathtt{rdy}(\mathtt{G}_1) \\ \mathtt{end} & \text{if } \mathtt{G} = \mathtt{p} \to \mathtt{q} : y\,\langle \mathtt{U}\rangle \text{ and } \mathtt{r} \neq \mathtt{p}, \mathtt{q} \text{ or if } \mathtt{G} = \mathtt{end} \\ \mathtt{end} & \text{if } \mathtt{G} = \mathtt{G}_1^{*^f} \text{ and } \mathtt{r} \not\in \mathcal{P}(\mathtt{G}_1) \text{ or } f(\mathtt{r}) \text{ is undefined} \end{cases}$$

Our projection is total on well-formed global types. All but the clauses for the projections of iteration in the definition of $\_\!\upharpoonright\!\_$ are straightforward (c.f. Honda et al. [2008]). Each iteration has a unique participant $\mathtt{r} \in \mathtt{rdy}(\mathtt{G}_1)$ (by well-formedness) dictating when to stop the iteration, and a number of 'passive' participants. Projection sends messages from $\mathtt{r}$ to each passive participant to signal the termination of the iteration. The *projection* $\mathcal{G}(\mathbf{y})\upharpoonright \mathtt{r}$ of a global type $\mathcal{G}(\mathbf{y}) \stackrel{\triangle}{=} \mathtt{G}$ with respect to $\mathtt{r}$ is a local type $\mathcal{T}(\mathbf{y}) \stackrel{\triangle}{=} \mathtt{T}$ where $\mathtt{T} = \mathtt{G}\!\upharpoonright\!\mathtt{r}$.

Example 1. Let $\mathtt{G} = \mathtt{G}_\mathtt{f}^{*^f}$, with $\mathtt{G}_\mathtt{f}$ defined in (2.1), $f(\mathtt{q}) = b_1\langle \mathtt{U}_1\rangle$ and $f(\mathtt{s}) = b_2\langle \mathtt{U}_2\rangle$. Then, the projections of $\mathtt{G}$ are

$$\mathtt{G}\!\upharpoonright\!\mathtt{p} = (y!\mathtt{U})^*; b_1!\mathtt{U}_1; b_2!\mathtt{U}_2 \qquad \mathtt{G}\!\upharpoonright\!\mathtt{q} = (y?\mathtt{U}; z!\mathtt{U})^*; b_1?\mathtt{U}_1 \qquad \mathtt{G}\!\upharpoonright\!\mathtt{s} = (z?\mathtt{U})^*; b_2?\mathtt{U}_2$$

2.2 Running example

We illustrate our approach on a real yet tractable protocol, the Post Office Protocol - Version 2 (POP2) Butler et al. [1985] between a client and a mail server. We describe POP2 with the following choreography where $\mathtt{G}_{\mathtt{EXIT}} = \mathtt{S} \to \mathtt{C} : \mathtt{BYE}\,\langle\rangle$:

$$\begin{aligned} \mathtt{G}_{\mathtt{POP}} &= \mathtt{C} \to \mathtt{S} : \mathtt{QUIT}\,\langle\rangle; \mathtt{G}_{\mathtt{EXIT}} \ +\ \mathtt{C} \to \mathtt{S} : \mathtt{HELO}\,\langle \mathtt{Str}\rangle; \mathtt{G}_{\mathtt{MBOX}} \\ \mathtt{G}_{\mathtt{MBOX}} &= \mathtt{S} \to \mathtt{C} : \mathtt{R}\,\langle \mathtt{Int}\rangle; \mathtt{G}_{\mathtt{NMBR}} \ +\ \mathtt{S} \to \mathtt{C} : \mathtt{E}\,\langle\rangle; \mathtt{G}_{\mathtt{EXIT}} \\ \mathtt{G}_{\mathtt{NMBR}} &= (\mathtt{C} \to \mathtt{S} : \mathtt{FOLD}\,\langle \mathtt{Str}\rangle; \mathtt{S} \to \mathtt{C} : \mathtt{R}\,\langle \mathtt{Int}\rangle \\ &\qquad + \mathtt{C} \to \mathtt{S} : \mathtt{READ}\,\langle \mathtt{Int}\rangle; \mathtt{S} \to \mathtt{C} : \mathtt{R}\,\langle \mathtt{Int}\rangle; \mathtt{G}_{\mathtt{SIZE}})^{*\mathtt{S}\mapsto\mathtt{QUIT}\langle\rangle}; \mathtt{G}_{\mathtt{EXIT}} \\ \mathtt{G}_{\mathtt{SIZE}} &= (\mathtt{C} \to \mathtt{S} : \mathtt{RETR}\,\langle\rangle; \mathtt{S} \to \mathtt{C} : \mathtt{MSG}\,\langle \mathtt{Data}\rangle.\mathtt{G}_{\mathtt{XFER}} \\ &\quad + \mathtt{C} \to \mathtt{S} : \mathtt{READ}\,\langle \mathtt{Int}\rangle; \mathtt{S} \to \mathtt{C} : \mathtt{R}\,\langle \mathtt{Int}\rangle)^{*\mathtt{S}\mapsto\mathtt{FOLD}\langle \mathtt{Str}\rangle}; \mathtt{S} \to \mathtt{C} : \mathtt{R}\,\langle \mathtt{Int}\rangle \\ \mathtt{G}_{\mathtt{XFER}} &= \mathtt{C} \to \mathtt{S} : \mathtt{ACKS}\,\langle\rangle; \mathtt{S} \to \mathtt{C} : \mathtt{R}\,\langle \mathtt{Int}\rangle + \mathtt{C} \to \mathtt{S} : \mathtt{ACKD}\,\langle\rangle; \mathtt{S} \to \mathtt{C} : \mathtt{R}\,\langle \mathtt{Int}\rangle \\ &\qquad + \mathtt{C} \to \mathtt{S} : \mathtt{NACK}\,\langle\rangle; \mathtt{S} \to \mathtt{C} : \mathtt{R}\,\langle \mathtt{Int}\rangle \end{aligned}$$

The protocol $\mathtt{G}_{\mathtt{POP}}$ starts with $\mathtt{C}$ sending $\mathtt{S}$ either an empty message along channel $\mathtt{QUIT}$ to quit the session, or a string on channel $\mathtt{HELO}$ representing $\mathtt{C}$'s password. In the first case, the protocol ends as per $\mathtt{G}_{\mathtt{EXIT}}$ while in the latter case the $\mathtt{G}_{\mathtt{MBOX}}$ is executed.

In $\mathtt{G}_{\mathtt{MBOX}}$, the server S either sends the number of messages in the default mailbox or it signals an error and ends the session as per $\mathtt{G}_{\mathtt{EXIT}}$. In the former case, $\mathtt{G}_{\mathtt{NMBR}}$ establishes that C repeatedly asks either (a) to enter a folder (sending the folder's name on FOLD) and then receiving back the number of messages in that folder, or (b) to request a message by sending its index along READ and then receiving back the length of the message. In case (a), the loop is immediately repeated after S's reply, in case (b) the protocol continues as $\mathtt{G}_{\mathtt{SIZE}}$ where another loop starts with C either (a) retrieving the message or (b) asking for another message (by interacting again on READ). For (a), C signals on RETR that it is ready to receive data that are sent by S on MSG (sort Data abstracts away the format of messages specified in Crocker [1982]); after these interactions the choreography continues as $\mathtt{G}_{\mathtt{XFER}}$ where the transmission is acknowledged by C with the interactions in $\mathtt{G}_{\mathtt{XFER}}$: ACKS keeps the message in the mailbox, ACKD deletes the message, NACK notifies that the message has not been received and must be kept in the mailbox; after any acknowledgement, S sends C the length of the next message. After some iterations in $\mathtt{G}_{\mathtt{SIZE}}$, C specifies a different folder and repeats $\mathtt{G}_{\mathtt{NMBR}}$.

The projection $\mathtt{T}_{\mathtt{S}} = \mathtt{G}_{\mathtt{POP}} \upharpoonright \mathtt{S}$ of $\mathtt{G}_{\mathtt{POP}}$ onto the server is below; $\mathtt{G}_{\mathtt{POP}} \upharpoonright \mathtt{C}$ is dual.

$$
\begin{aligned}
\mathtt{T}_{\mathtt{S}} &= \mathtt{QUIT?}; \mathtt{T}_{\mathtt{EXIT}} \\
&\quad + \mathtt{HELO?Str}; \mathtt{T}_{\mathtt{MBOX}} \\
\mathtt{T}_{\mathtt{MBOX}} &= \mathtt{R!Int}; \mathtt{T}_{\mathtt{NMBR}} \oplus \mathtt{E!}; \mathtt{T}_{\mathtt{EXIT}} \\
\mathtt{T}_{\mathtt{EXIT}} &= \mathtt{BYE!}
\end{aligned}
\qquad
\begin{aligned}
\mathtt{T}_{\mathtt{NMBR}} &= (\mathtt{FOLD?Str}; \mathtt{R!Int} + \mathtt{READ?Int}; \mathtt{R!Int}; \mathtt{T}_{\mathtt{SIZE}})^*; \\
&\quad\ \mathtt{QUIT?}; \mathtt{T}_{\mathtt{EXIT}} \\
\mathtt{T}_{\mathtt{SIZE}} &= (\mathtt{RETR?}; \mathtt{MSG!Data}; \mathtt{T}_{\mathtt{XFER}} + \mathtt{READ?Int}; \mathtt{R!Int})^*; \\
&\quad\ \mathtt{FOLD?Str}; \mathtt{R!int} \\
\mathtt{T}_{\mathtt{XFER}} &= \mathtt{ACKS?}; \mathtt{R!Int} + \mathtt{ACKD?}; \mathtt{R!Int} + \mathtt{NACK?}; \mathtt{R!Int}
\end{aligned}
$$

The messages in $\mathtt{T}_{\mathtt{S}}$ are as in $\mathtt{G}_{\mathtt{POP}}$ and S iterates until C's signals on QUIT or on FOLD.

In Ex. 2 we present, for illustrative purpose, a multiparty variant of $\mathtt{G}_{\mathtt{POP}}$ where the authentication is outsourced.

Example 2. A multiparty variant of POP2 is given by $\mathtt{G}'_{\mathtt{POP}}$ below where S uses a third-party authentication service A:

$$
\begin{aligned}
\mathtt{G}'_{\mathtt{POP}} &= \mathtt{C} \to \mathtt{S} : \mathtt{QUIT}\,\langle\rangle; \mathtt{G}_{\mathtt{EXIT}} \ + \ \mathtt{C} \to \mathtt{S} : \mathtt{HELO}\,\langle \mathtt{Str} \rangle; \mathtt{G}'_{\mathtt{MBOX}} \\
\mathtt{G}'_{\mathtt{MBOX}} &= \mathtt{S} \to \mathtt{A} : \mathtt{REQ}\,\langle \mathtt{Str} \rangle; \mathtt{A} \to \mathtt{S} : \mathtt{RES}\,\langle \mathtt{Bool} \rangle; \\
&\quad\ \mathtt{S} \to \mathtt{C} : \mathtt{R}\,\langle \mathtt{Int} \rangle; \mathtt{G}_{\mathtt{NMBR}} \ + \ \mathtt{S} \to \mathtt{C} : \mathtt{E}\,\langle\rangle; \mathtt{G}_{\mathtt{EXIT}}
\end{aligned}
$$

where, on RES, A sends the result of the authentication of C ($\mathtt{G}_{\mathtt{NMBR}}$ and $\mathtt{G}_{\mathtt{EXIT}}$ remain unchanged). The projection of $\mathtt{G}'_{\mathtt{POP}}$ on S is

$$
\begin{gathered}
\mathtt{T}'_{\mathtt{S}} = \mathtt{QUIT?}; \mathtt{T}_{\mathtt{EXIT}} \ + \ \mathtt{HELO?Str}; \mathtt{T}_{\mathtt{AUTH}} \\
\mathtt{T}_{\mathtt{AUTH}} = \mathtt{REC!Str}; \mathtt{RES?Bool}; \mathtt{T}'_{\mathtt{MBOX}} \qquad \mathtt{T}'_{\mathtt{MBOX}} = \mathtt{R!Int}; \mathtt{T}_{\mathtt{NMBR}} \ + \ \mathtt{E!}; \mathtt{T}_{\mathtt{EXIT}}
\end{gathered}
\qquad \blacklozenge
$$

2.3 Behaviour of types

The semantics of local types is given in terms of *specifications*, namely pairs of partial functions Γ and Δ such that Γ maps session names to global types and names to sorts, and Δ maps tuples of session names to local types. We use $\Gamma \bullet \Delta$ to denote a specification and adopt the usual syntactic notations for environments:

$$
\Gamma ::= \emptyset \ \big|\ \Gamma, u : \mathcal{G} \ \big|\ \Gamma, x : \mathtt{U} \qquad\qquad \Delta ::= \emptyset \ \big|\ \Delta, \mathbf{s} : \mathcal{T}
$$

as usual, when writing $\Delta, \mathbf{s} : \mathcal{T}$, $\mathbf{s} \notin \mathrm{dom}(\Delta)$ is implicitly assumed (likewise for $\Gamma, \_ : \_$) and $\Delta_1, \Delta_2 \equiv \Delta_2, \Delta_1$.

The semantics of specifications is generated by the rules in Fig. 1 using the labels

$$
\alpha ::= \overline{u}^n \mathbf{s} \ \mid \ u_i \mathbf{s} \ \mid \ \overline{s}\mathbf{v} \ \mid \ s\mathbf{v} \ \mid \ \tau \tag{2.3}
$$

$$\frac{\Gamma(u) \equiv \mathcal{G}(\mathbf{y})}{\Gamma \bullet \Delta \xrightarrow{\overline{u}^n \mathbf{y}} \Gamma \bullet \Delta, \mathbf{y} : \mathcal{G}(\mathbf{y}) \upharpoonright 0} [\mathtt{TReq}] \qquad \frac{\Gamma(u) \equiv \mathcal{G}(\mathbf{y})}{\Gamma \bullet \Delta \xrightarrow{u_i \mathbf{y}} \Gamma \bullet \Delta, \mathbf{y} : \mathcal{G}(\mathbf{y}) \upharpoonright \mathtt{i}} [\mathtt{TAcc}]$$

$$\frac{\mathtt{v} : \mathtt{U}_j \quad s_j \in \mathbf{s} \quad j \in I}{\Gamma \bullet \Delta, \mathbf{s} : \bigoplus_{i \in I} s_i ! \mathtt{U}_i ; \mathcal{T}_i \xrightarrow{\overline{s_j} \mathtt{v}} \Gamma \bullet \Delta, \mathbf{s} : \mathcal{T}_j} [\mathtt{TSend}] \qquad \frac{\mathtt{v} : \mathtt{U}_j \quad s_j \in \mathbf{s} \quad j \in I}{\Gamma \bullet \Delta, \mathbf{s} : \sum_{i \in I} s_i ? \mathtt{U}_i ; \mathcal{T}_i \xrightarrow{s_j \mathtt{v}} \Gamma \bullet \Delta, \mathbf{s} : \mathcal{T}_j} [\mathtt{TRec}]$$

$$\frac{\Gamma \bullet \Delta, \mathbf{s} : \mathcal{T} \xrightarrow{\alpha} \Gamma \bullet \Delta, \mathbf{s} : \mathcal{T}'}{\Gamma \bullet \Delta, \mathbf{s} : \mathcal{T} ; \mathcal{T}'' \xrightarrow{\alpha} \Gamma \bullet \Delta, \mathbf{s} : \mathcal{T}' ; \mathcal{T}''} [\mathtt{TSeq}] \qquad \frac{\Gamma \bullet \Delta_1 \xrightarrow{\tau} \Gamma \bullet \Delta_1'}{\Gamma \bullet \Delta_1, \Delta_2 \xrightarrow{\tau} \Gamma \bullet \Delta_1', \Delta_2} [\mathtt{TPar}]$$

$$\Gamma \bullet \Delta, \mathbf{s} : \mathcal{T}^* \xrightarrow{\tau} \Gamma \bullet \Delta, \mathbf{s} : \mathtt{end} [\mathtt{TLoop1}] \qquad \Gamma \bullet \Delta, \mathbf{s} : \mathcal{T}^* \xrightarrow{\tau} \Gamma \bullet \Delta, \mathbf{s} : \mathcal{T} ; \mathcal{T}^* [\mathtt{TLoop2}]$$

Fig. 1. Labelled transitions for specifications

that respectively represent the request on u for the initialisation of a session among $n + 1$ roles, the acceptance of joining a session of u as the i-th role, the sending of a value on s, the reception of a value on s, and the silent step.

Intuitively, the rules of Fig. 1 specify how a single participant behaves in a session $\mathbf{s}$ and are instrumental for type checking processes. Rules [TReq] and [TAcc] allow a specification to initiate a new session by projecting (on 0 and i, resp.) the global type associated[5] to name u in Γ. By [TSend], if types are respected, a specification can send any value on one of the names in a branch of an internal choice. Dually, [TRec] accounts for the reception of a value. Note that values occur only on the label of the transitions and are not instantiated in the local types. Rule [TSeq] is trivial. Rule [TPar] allows part of a specification to make a transition. Finally, an iterative local type can either stop by rule [TLoop1] or arbitrarily repeat itself by rule [TLoop2].

3 Processes and Systems

As we will see (Def. 1 in § 4), global types are implemented by *systems*. Our systems exchange values specified by *expressions* having the following syntax:

$$e \ ::= \ x \mid \mathtt{v} \mid e_1 \ \mathtt{op} \ e_2 \qquad\qquad \ell \ ::= \ [e_1, \ldots, e_n] \ \mid \ e_1..e_2$$

An expression e is either a variable, or a value, or else the composition of expressions (we assume that expressions are implicitly sorted and do not include names). Lists $[e_1, \ldots, e_n]$ and numerical ranges $e_1..e_2$ are used for iteration; in the former case, all the items of a list have the same sort, in the latter case, both expressions are integers and the value of e_1 is smaller than or equal to the value of e_2. The empty list is denoted as ε and the operations $\mathtt{hd}(\ell)$ and $\mathtt{tl}(\ell)$ respectively return the head and tail of ℓ (defined as usual). We write $\mathrm{var}(e)$ and $\mathrm{var}(\ell)$ for the set of variables of e and ℓ.

The syntax of processes and systems below relies on *queues* of basic values M and input-guarded non-deterministic sequential process N, respectively defined as

$$M ::= \emptyset \ \mid \ \mathtt{v} \cdot M \qquad N ::= \sum_{i \in I} y_i(x_i); P_i$$

[5] The use of $\equiv$ in the premises caters for α-conversion of names $\mathbf{y}$. Also, $\mathbb{P}$ is the set of natural numbers (0 is the initiator of sessions) and for readability, in examples we use names to denote participants.

where $i \neq j \in I \implies y_i \neq y_j$; we define $\mathbf{0} \triangleq \sum_{i\in\emptyset} y_i(x_i); P_i$.

The syntax of systems S and processes P is

$$P, Q ::= u_i(\mathbf{y}).P \mid \overline{u}^n(\mathbf{y}).P \mid N \mid \overline{s}e \mid \texttt{if } e:\ P \texttt{ else } Q \\ \mid P; P \mid \texttt{for } x \texttt{ in } \ell :\ P \mid \texttt{do } N \texttt{ until } b(x)$$

$$S \ ::= P \mid (\nu\mathbf{s})S \mid S \mid S \mid s : M$$

All constructions but loops are straightforward. In $\texttt{for } x \texttt{ in } \ell :\ P$, the body P is executed for each element in ℓ, while $\texttt{do } N \texttt{ until } b(x)$ repeats N until a message on b is received. Intuitively, the former construct is executed by the (unique) role that decides when to exit the iteration while the latter construct is used by the "passive" roles in the loop (see § 2.1 and § 5). Given a process P, $\texttt{fv}(P)$ denotes the set of all variables appearing outside the scope of input prefixes in P. Also, we extend $\mathrm{var}(\_)$ to systems in the obvious way. In $(\nu\mathbf{s})S$, names $\mathbf{s}$ are bound (the set $\texttt{fc}(S)$ of free session names of S is defined as expected); a system S is *closed* when $\texttt{fc}(S) = \emptyset$ and it is *initial* when S does not contain runtime constructs, namely new session $(\nu\mathbf{s})S'$ and queues $s : M$. Formally, S is initial iff for each s and S', if $S \equiv (\nu\mathbf{s})S'$ then $\mathbf{s} \not\subseteq \texttt{fc}(S')$.

The structural congruence $\equiv$ is the least congruence over systems closed with respect to α-conversion, such that $\_ \mid \_$ and $\_ + \_$ are associative, commutative and have $\mathbf{0}$ as identity, $\_;\_$ is associative and has $\mathbf{0}$ as identity, and the following axioms hold:

$$(\nu\mathbf{s})\mathbf{0} \equiv \mathbf{0} \qquad (\nu\mathbf{s})(\nu\mathbf{s}')S \equiv (\nu\mathbf{s}')(\nu\mathbf{s})S \qquad (\nu\mathbf{s})(S \mid S') \equiv S \mid (\nu\mathbf{s})S',\ \text{when } \mathbf{s} \not\subseteq \texttt{fc}(S)$$

The operational semantics of systems is in Fig. 2 where a store σ records the values assigned to variables, $e \downarrow \sigma$ is the evaluation of e (defined if $\mathrm{var}(e) \subseteq \mathrm{dom}(\sigma)$ and undefined otherwise), and $\sigma[x \mapsto \texttt{v}]$ is the update of σ at x with $\texttt{v}$. Labels are obtained by extending the grammar in (2.3) with the production $\alpha ::= e \vdash \alpha$ where e is a boolean expression used in conditional transitions $\langle S, \sigma\rangle \xrightarrow{e\vdash\alpha} \langle S', \sigma'\rangle$ representing the fact that $\langle S, \sigma\rangle$ has an α-transition to $\langle S', \sigma'\rangle$ provided that $e \downarrow \sigma$ actually holds. We may write α instead of $\texttt{true} \vdash \alpha$ and $e \wedge e' \vdash \alpha$ instead of $e \vdash (e' \vdash \alpha)$.

We comment on the rules in Fig. 2 where $\texttt{fc}(\alpha)$ is defined as $\texttt{fc}(\overline{u}^n\mathbf{s}) = \texttt{fc}(u_i\mathbf{s}) = \{u\}$, $\texttt{fc}(\overline{s}\texttt{v}) = \texttt{fc}(s\texttt{v}) = \{s\}$, and $\texttt{fc}(\tau) = \emptyset$. Rules [SReq] and [SAcc] are for requesting and accepting new sessions; in their continuations, newly created session names $\mathbf{s}$ replace $\mathbf{y}$. Rule [SRec] is for receiving messages in an early style approach (variables are assigned when firing input prefixes); note that the store is updated by recording that x is assigned $\texttt{v}$. Rule [SSend] is for sending values. Rules [SThen] and [SElse] handle 'if' statements as expected; their only peculiarity is that the guard is recorded on the label of the transition: this is instrumental for the correspondence between systems and their types (c.f. § 6). Rules [$\texttt{SFor}_1$], [$\texttt{SFor}_2$], [$\texttt{Sloop}_1$], [$\texttt{Sloop}_2$] unfold the corresponding iterative program in an expected way. Except for session initialisation, the remaining rules are standard. Rule [SInit] allows n roles to synchronise with $\overline{u}^n(\mathbf{y}_0).P_0$; in the continuation of each role i, the bound names $\mathbf{y}_i$ are replaced with a tuple of freshly chosen session names for which the corresponding queues are created. Such queues are used to exchange values as prescribed by rules [$\texttt{SCom}_1$] and [$\texttt{SCom}_2$]. Rule [SInit] requires the synchronisation of all roles. Since processes are single-threaded, this is only possible when each process plays exactly one role in that session. Note that the

$$\frac{\mathbf{s} \notin \mathtt{fc}(P)}{\langle \overline{u}^n(\mathbf{y}).P, \sigma\rangle \xrightarrow{\overline{u}^n \mathbf{s}} \langle P\{\mathbf{y}/\mathbf{s}\}, \sigma\rangle}[\mathtt{SReq}] \qquad \frac{\ell \downarrow \sigma \neq \varepsilon \;\; \langle P, \sigma[x \mapsto \mathtt{hd}(\ell \downarrow \sigma)]\rangle \xrightarrow{e\vdash\alpha} \langle P', \sigma'\rangle}{\langle \mathtt{for}\ x\ \mathtt{in}\ \ell\colon P, \sigma\rangle \xrightarrow{e\vdash\alpha} \langle P'; \mathtt{for}\ x\ \mathtt{in}\ \mathtt{tl}(\ell)\colon P, \sigma'\rangle}[\mathtt{SFor}_2]$$

$$\frac{\mathbf{s} \notin \mathtt{fc}(P)}{\langle u_i(\mathbf{y}).P, \sigma\rangle \xrightarrow{u_i \mathbf{s}} \langle P\{\mathbf{y}/\mathbf{s}\}, \sigma\rangle}[\mathtt{SAcc}] \qquad \frac{e \downarrow \sigma = \mathtt{true} \quad \langle P, \sigma\rangle \xrightarrow{e'\vdash\alpha} \langle P', \sigma'\rangle}{\langle \mathtt{if}\ e:\ P\ \mathtt{else}\ Q, \sigma\rangle \xrightarrow{e\wedge e'\vdash\alpha} \langle P', \sigma'\rangle}[\mathtt{SThen}]$$

$$\langle s(x); P + N, \sigma\rangle \xrightarrow{s\mathtt{v}} \langle P, \sigma[x \mapsto \mathtt{v}]\rangle[\mathtt{SRec}] \qquad \frac{e \downarrow \sigma = \mathtt{false} \quad \langle Q, \sigma\rangle \xrightarrow{e'\vdash\alpha} \langle Q', \sigma'\rangle}{\langle \mathtt{if}\ e:\ P\ \mathtt{else}\ Q, \sigma\rangle \xrightarrow{\neg e\wedge e'\vdash\alpha} \langle Q', \sigma'\rangle}[\mathtt{SElse}]$$

$$\frac{e \downarrow \sigma = \mathtt{v}}{\langle \overline{s}e, \sigma\rangle \xrightarrow{\overline{s}\mathtt{v}} \langle \mathbf{0}, \sigma\rangle}[\mathtt{SSend}] \qquad \langle \mathtt{do}\ P\ \mathtt{until}\ b(x), \sigma\rangle \xrightarrow{b\mathtt{v}} \langle \mathbf{0}, \sigma[x \mapsto \mathtt{v}]\rangle[\mathtt{SLoop}_1]$$

$$\frac{\langle P, \sigma\rangle \xrightarrow{e\vdash\alpha} \langle P', \sigma'\rangle}{\langle P; Q, \sigma\rangle \xrightarrow{e\vdash\alpha} \langle P'; Q, \sigma'\rangle}[\mathtt{SSeq}] \qquad \frac{\langle P, \sigma\rangle \xrightarrow{e\vdash\alpha} \langle P', \sigma'\rangle \quad b \notin \mathtt{fc}(\alpha)}{\langle \mathtt{do}\ P\ \mathtt{until}\ b, \sigma\rangle \xrightarrow{e\vdash\alpha} \langle P'; \mathtt{do}\ P\ \mathtt{until}\ b, \sigma'\rangle}[\mathtt{SLoop}_2]$$

$$\frac{\ell \downarrow \sigma = \varepsilon}{\langle \mathtt{for}\ x\ \mathtt{in}\ \ell\colon P, \sigma\rangle \xrightarrow{\tau} \langle \mathbf{0}, \sigma\rangle}[\mathtt{SFor}_1] \qquad \frac{P \equiv P' \quad \langle P', \sigma\rangle \xrightarrow{e\vdash\alpha} \langle Q', \sigma'\rangle \quad Q' \equiv Q}{\langle P, \sigma\rangle \xrightarrow{e\vdash\alpha} \langle Q, \sigma'\rangle}[\mathtt{SStruct}]$$

$$\frac{\mathbf{s} \notin \mathtt{fc}(P_i) \quad Q_i = P_i\{\mathbf{y}_i/\mathbf{s}\} \text{ for } i = 0, \ldots, n}{\langle \overline{u}^n(\mathbf{y}_0).P_0 \mid u_1(\mathbf{y}_1).P_1 \mid \ldots \mid u_n(\mathbf{y}_n).P_n, \sigma\rangle \xrightarrow{\tau} \langle (\nu\mathbf{s})(Q_0 \mid \ldots \mid Q_n \mid \mathbf{s} : \emptyset), \sigma\rangle}[\mathtt{SInit}]$$

$$\frac{\langle P, \sigma\rangle \xrightarrow{e\vdash\overline{s}\mathtt{v}} \langle P', \sigma'\rangle}{\langle P \mid s : M, \sigma\rangle \xrightarrow{e\vdash\tau} \langle P' \mid s : M \cdot \mathtt{v}, \sigma'\rangle}[\mathtt{SCom}_1] \qquad \frac{\langle P, \sigma\rangle \xrightarrow{e\vdash s\mathtt{v}} \langle P', \sigma'\rangle}{\langle P \mid s : \mathtt{v} \cdot M, \sigma\rangle \xrightarrow{e\vdash\tau} \langle P' \mid s : M, \sigma'\rangle}[\mathtt{SCom}_2]$$

$$\frac{\langle S, \sigma\rangle \xrightarrow{e\vdash\alpha} \langle S', \sigma'\rangle \quad s \notin \mathtt{fc}(\alpha)}{\langle (\nu s)S, \sigma\rangle \xrightarrow{e\vdash\alpha} \langle (\nu s)S', \sigma'\rangle}[\mathtt{SNews}] \qquad \frac{\langle S_1, \sigma\rangle \xrightarrow{e\vdash\alpha} \langle S_1', \sigma'\rangle \quad \mathrm{var}(S_1) \cap \mathrm{var}(S_2) = \emptyset}{\langle S_1 \mid S_2, \sigma\rangle \xrightarrow{e\vdash\alpha} \langle S_1' \mid S_2, \sigma'\rangle}[\mathtt{SPar}]$$

Fig. 2. Labelled transitions for processes (top) and systems (bottom)

semantics relies on a global store σ. However, the condition $\mathrm{var}(S_1) \cap \mathrm{var}(S_2) = \emptyset$ in rule [SPar] ensures that each program has its own local (logical) store (i.e., there is no confusion between local variables of different programs).

Note that, in a sequential composition $P; Q$, the store σ allows us to extend the scope of names bound in P by input prefixes to Q.

3.1 Running examples

In Ex. 3 we give the implementation of $\mathtt{T}_\mathtt{S}$ (i.e., participant S of $\mathtt{G}_\mathtt{POP}$) from § 2.2. To ease the presentation, we use the following auxiliary functions.

- auth : Str → Bool that is used for authenticating clients;
- fn : Str → Int that given a folder name returns the number of messages in that folder (we assume "inbox" to be the default folder);
- mn : Int → Int that given a message number returns its length (in bytes);
- data : void → Data that returns the current message;
- next : void → Int that returns the next message number;
- del : void → Int that returns the next message number and deletes the current message from the folder.

Let s_k denote the name in s corresponding to channel k in $\mathtt{G}_\mathtt{POP}$ and likewise for $\mathtt{G}'_\mathtt{POP}$.

Example 3. The process $P_{\mathtt{INIT}}$ below implements POP2's server.

$$P_{\mathtt{INIT}} = u_{\mathtt{S}}(\mathbf{s}).P_{\mathtt{S}} \qquad P_{\mathtt{S}} = s_{\mathtt{QUIT}}(); P_{\mathtt{EXIT}} + s_{\mathtt{HELO}}(x); P_{\mathtt{MBOX}} \qquad P_{\mathtt{EXIT}} = \overline{s_{\mathtt{BYE}}}$$
$$P_{\mathtt{MBOX}} = \mathtt{if}\ \mathtt{auth}(x)\ :\ \overline{s_{\mathrm{R}}}\mathtt{fn}(\text{``}\mathtt{inbox}\text{''}); P_{\mathtt{NMBR}}\ \mathtt{else}\ \overline{s_{\mathrm{E}}}; P_{\mathtt{EXIT}}$$
$$P_{\mathtt{NMBR}} = \mathtt{do}(s_{\mathtt{FOLD}}(x); \overline{s_{\mathrm{R}}}\mathtt{fn}(x)\ +\ s_{\mathtt{READ}}(x'); \overline{s_{\mathrm{R}}}\mathtt{mn}(x'); P_{\mathtt{SIZE}})\ \mathtt{until}\ s_{\mathtt{QUIT}}(); P_{\mathtt{EXIT}}$$
$$P_{\mathtt{SIZE}} = \mathtt{do}(s_{\mathtt{RETR}}(); \overline{s_{\mathtt{MSG}}}\mathtt{data}(); P_{\mathtt{XFER}}\ +\ s_{\mathtt{READ}}(x'); \overline{s_{\mathrm{R}}}\mathtt{mn}(x'))\ \mathtt{until}\ s_{\mathtt{FOLD}}(x); \overline{s_{\mathrm{R}}}\mathtt{fn}(x)$$
$$P_{\mathtt{XFER}} = s_{\mathtt{ACKS}}(); \overline{s_{\mathrm{R}}}\mathtt{mn}(\mathtt{next}())\ +\ s_{\mathtt{ACKD}}(); \overline{s_{\mathrm{R}}}\mathtt{mn}(\mathtt{del}())\ +\ s_{\mathtt{NACK}}(); \overline{s_{\mathrm{R}}}\mathtt{mn}(x')$$

Firstly, $P_{\mathtt{INIT}}$ initiates a session of type $\mathtt{G}_{\mathtt{POP}}$ as $\mathtt{S}$ then it behaves according to $\mathtt{T}_{\mathtt{S}}$. The non-deterministic choice is resolved in the conditional statement of $P_{\mathtt{MBOX}}$. ♦

Ex. 4 gives an implementation of the server $\mathtt{T}'_{\mathtt{S}}$ of the multiparty variant of POP2.

Example 4. Let $\mathtt{G}'_{\mathtt{POP}}$ be as in Ex. 2 and $P'_{\mathtt{INIT}} = u_{\mathtt{S}}(\mathbf{s}).P'_{\mathtt{S}}$ where

$$P'_{\mathtt{S}} = s_{\mathtt{QUIT}}(); P_{\mathtt{EXIT}} + s_{\mathtt{HELO}}(x); P_{\mathtt{AUTH}}$$
$$P_{\mathtt{AUTH}} = \overline{s_{\mathtt{REQ}}}x; s_{\mathtt{RES}}(y); P'_{\mathtt{MBOX}}$$
$$P'_{\mathtt{MBOX}} = \mathtt{if}\ \mathtt{auth}(x) \wedge y\ :\ \overline{s_{\mathrm{R}}}\mathtt{fn}(\text{``}\mathtt{inbox}\text{''}); P_{\mathtt{NMBR}}\ \mathtt{else}\ \overline{s_{\mathrm{E}}}; P_{\mathtt{EXIT}}$$

Here, $P'_{\mathtt{INIT}}$ resolves the non-deterministic choice in $P'_{\mathtt{MBOX}}$ by taking into account both the value returned by $\mathtt{auth}(\_)$ and the feedback of $\mathtt{A}$ stored in variable y. ♦

4 Whole-Spectrum Implementation

Definition 1 below introduces the notion of candidate implementation of a global type, that is a system consisting of one process for each role in the global type.

Definition 1 (Implementation). *Given $\mathcal{G}(\mathbf{y}) \stackrel{\triangle}{=} \mathtt{G}$ s.t. $\mathcal{P}(\mathcal{G}) = \{\mathtt{p}_1, \ldots, \mathtt{p}_n\}$ and a mapping ι assigning a process to each $\mathtt{p} \in \mathcal{P}(\mathcal{G})$, a* ι-implementation *of $\mathcal{G}$ is a system $\mathcal{I}^{\iota}_{\mathcal{G}}$ such that either (i) $\mathcal{I}^{\iota}_{\mathcal{G}} \equiv \iota(\mathtt{p}_1) \mid \ldots \mid \iota(\mathtt{p}_n)$ and $\mathbf{y} \cap \mathtt{fc}(\iota(\mathtt{p}_1)) = \ldots = \mathbf{y} \cap \mathtt{fc}(\iota(\mathtt{p}_n)) = \emptyset$ or (ii) $\mathcal{I}^{\iota}_{\mathcal{G}} \equiv (\nu\mathbf{y})(\iota(\mathtt{p}_1) \mid \ldots \mid \iota(\mathtt{p}_n) \mid \mathbf{y} : \mathbf{M})$.*

In case (i) the session that implements $\mathcal{G}$ is not initiated. For simplicity, we assume that roles do not use the channels defined by the global type before initiating the corresponding session (i.e., $\mathbf{y} \cap \mathtt{fc}(\iota(\mathtt{p}_i)) = \emptyset$). This is not a limitation since channel names can always be renamed to avoid clashes. Case (ii) captures already initiated sessions; wlog, we assume that the system and the global type use the same session channels $\mathbf{y}$.

We characterise WSI as a relation between the execution traces of a global type $\mathcal{G}$ and its implementations $\mathcal{I}^{\iota}_{\mathcal{G}}$. An execution trace of $\mathcal{I}^{\iota}_{\mathcal{G}}$ is a sequence of input and output actions decorated with the role that performs them (in symbols $\langle \mathtt{p}, s!\mathtt{U} \rangle$ and $\langle \mathtt{p}, s?\mathtt{U} \rangle$).

Definition 2 (Runs of implementations). *Let $\mathcal{I}^{\iota}_{\mathcal{G}}$ be an implementation of $\mathcal{G}(\mathbf{y}) \stackrel{\triangle}{=} \mathtt{G}$. The set $\mathcal{R}_u(\langle \mathcal{I}^{\iota}_{\mathcal{G}}, \sigma \rangle)$ of* runs of $\mathcal{I}^{\iota}_{\mathcal{G}}$ initiated on u with store σ *is the least set closed with respect to the rules in Fig. 3. We write $\mathcal{R}_u(\mathcal{I}^{\iota}_{\mathcal{G}})$ for $\mathcal{R}_u(\langle \mathcal{I}^{\iota}_{\mathcal{G}}, \emptyset \rangle)$. The runs of a set of implementations $\mathbb{I}$ is $\mathcal{R}_u(\mathbb{I}) = \cup_{I \in \mathbb{I}} \mathcal{R}_u(I)$.*

Let $\langle \iota(\mathtt{p}), \sigma\rangle \xrightarrow{e' \vdash \alpha} \langle \iota'(\mathtt{p}), \sigma'\rangle$ stand for $\langle \iota(\mathtt{p}), \sigma\rangle \xrightarrow{e' \vdash \alpha} \langle P, \sigma'\rangle$ and $\iota' = \iota[\mathtt{p} \mapsto P]$

$$\frac{\langle \mathcal{I}^{\iota}_{\mathcal{G}}, \sigma\rangle \xrightarrow{e \vdash \tau} \langle \mathcal{I}^{\iota'}_{\mathcal{G}}, \sigma'\rangle \quad \langle \iota(\mathtt{p}), \sigma\rangle \xrightarrow{e' \vdash \alpha} \langle \iota'(\mathtt{p}), \sigma'\rangle \quad \mathtt{fc}(\alpha) \cap \mathbf{y} \neq \emptyset \quad r \in \mathcal{R}_u(\langle \mathcal{I}^{\iota'}_{\mathcal{G}}, \sigma'\rangle) \quad \mathtt{obj}(\alpha) : \mathtt{U}}{\langle \mathtt{p}, \alpha\{\mathtt{obj}(\alpha)/\mathtt{U}\}\rangle r \in \mathcal{R}_u(\langle \mathcal{I}^{\iota}_{\mathcal{G}}, \sigma\rangle)}[\mathtt{RRInt}] \qquad \frac{\langle \mathcal{I}^{\iota}_{\mathcal{G}}, \sigma\rangle \not\rightarrow}{\epsilon \in \mathcal{R}_u(\langle \mathcal{I}^{\iota}_{\mathcal{G}}, \sigma\rangle)}[\mathtt{RREnd}]$$

$$\frac{\langle \mathcal{I}^{\iota}_{\mathcal{G}}, \sigma\rangle \xrightarrow{e \vdash \alpha} \langle \mathcal{I}^{\iota'}_{\mathcal{G}}, \sigma'\rangle \quad u \notin \mathtt{fc}(\alpha) \quad \langle \iota(\mathtt{p}), \sigma\rangle \xrightarrow{e' \vdash \beta} \langle \iota'(\mathtt{p}), \sigma'\rangle \quad \mathtt{fc}(\beta) \cap \mathbf{y} = \emptyset \quad r \in \mathcal{R}_u(\langle \mathcal{I}^{\iota'}_{\mathcal{G}}, \sigma'\rangle)}{r \in \mathcal{R}_u(\langle \mathcal{I}^{\iota}_{\mathcal{G}}, \sigma\rangle)}[\mathtt{RRExt}]$$

Fig. 3. Runs of implementations

Rules in Fig. 3 rely on the semantics of Fig. 2. In rule [RRInt] (where $\mathtt{obj}(\alpha) = \mathtt{v}$ for $\alpha = \overline{s}\mathtt{v}$ or $\alpha = s\mathtt{v}$), a system reduces when some process $\iota(\mathtt{p})$ in the implementation interacts over a session channel (i.e., α is either $\overline{y}\mathtt{v}$ or $y\mathtt{v}$ with $y \in \mathbf{y}$). Since the action α performed by $\iota(\mathtt{p})$ involves a session channel of the global type, an event α associated to the role $\mathtt{p}$ is added to the trace. Note that the actual value of the message α is substituted by its type, i.e., $\alpha\{\mathtt{obj}(\alpha)/\mathtt{U}\}$ in place of α. Rule [RREnd] is straightforward. Rule [RRExt] accounts for a computation step that does not involve session channels, i.e., an internal transition τ in a role, a communication over a channel not in $\mathbf{y}$, or a session initiation. This rule allows a process to freely initiate sessions over channels different from u (i.e., sessions that do not corresponds to the global type $\mathtt{G}$). On the contrary, when a role attempts to initiate a session over u, rule [RRExt] requires all roles in the implementation to initiate the session (this behaviour is imposed by the premise $u \notin \mathtt{fc}(\alpha)$). We assume that any role in the implementation will execute exactly one action over the channel u which also matches the role assigned by ι. Nested sessions are handled by assuming that all sessions are created over different channels that have the same type. This is just a technical simplification analogous to the possibility of having annotations to indicate the particular instance of the session under analysis.

For global types, we deviate from standard definition of traces Castagna et al. [2012], Chen and Honda [2012] and use, for technical convenience, *annotated traces* that distinguish mandatory from optional actions. We write $[r]$ to denote the optional sequence r. Moreover, we consider an asynchronous communication model *à la* Lamport Lamport [1978] and a trace implicitly denotes the equivalence class of all traces obtained by permuting causally independent actions.

Definition 3 (Runs of a global type). *Given a global type term* $\mathtt{G}$*, the set* $\mathcal{R}(\mathtt{G})$ *denotes the runs allowed by* $\mathtt{G}$ *and is defined as the least set closed under the rules in Fig. 4.*

The first three rules are straightforward. Rule [RGPar] considers just the sequential composition of the traces corresponding to the two parallel branches (recall that a trace denotes an equivalence class of executions). The traces of an iterative type $\mathtt{G}^{*^f}$ are given by the rule [RGIter]; the set $\widetilde{\mathcal{R}}(\mathtt{G}^{*^f})$ in the premise contains the traces of the unfolding of $\mathtt{G}^{*^f}$ defined by the rules [$\mathtt{RG}^*\mathtt{1}$] and [$\mathtt{RG}^*\mathtt{2}$]. Optional events are introduced when unfolding an iterative type (rule [$\mathtt{RG}^*\mathtt{2}$]). The main motivation is that an iterative type $\mathtt{G}^{*^f}$ denotes an unbounded number of repetitions $\mathtt{G}$ (i.e., an infinite number of traces).

$$\frac{}{\epsilon \in \mathcal{R}(\mathtt{end})}[\mathtt{RGEnd}] \qquad \frac{}{\langle \mathtt{p}, s!\mathtt{U}\rangle\langle \mathtt{q}, s?\mathtt{U}\rangle \in \mathcal{R}(\mathtt{p} \to \mathtt{q} : s\,\langle \mathtt{U}\rangle)}[\mathtt{RGComm}] \qquad \frac{r \in \mathcal{R}(\mathtt{G}_1) \cup \mathcal{R}(\mathtt{G}_1)}{r \in \mathcal{R}(\mathtt{G}_1 + \mathtt{G}_2)}[\mathtt{RGCh}]$$

$$\frac{r_1 \in \mathcal{R}(\mathtt{G}_1) \quad r_2 \in \mathcal{R}(\mathtt{G}_2)}{r_1 r_2 \in \mathcal{R}(\mathtt{G}_1 ; \mathtt{G}_2)}[\mathtt{RGSeq}] \qquad \frac{r_1 \in \mathcal{R}(\mathtt{G}_1) \quad r_2 \in \mathcal{R}(\mathtt{G}_2)}{r_1 r_2 \in \mathcal{R}(\mathtt{G}_1 \mid \mathtt{G}_2)}[\mathtt{RGPar}]$$

$$\frac{r_1 \in \mathcal{R}(\mathtt{G})}{r_1 \in \widetilde{\mathcal{R}}(\mathtt{G}^{*^f})}[\mathtt{RG}^*\mathtt{1}] \qquad \frac{r_1 \in \widetilde{\mathcal{R}}(\mathtt{G}^{*^f}) \quad r_2 \in \mathcal{R}(\mathtt{G})}{[r_1] r_2 \in \widetilde{\mathcal{R}}(\mathtt{G}^{*^f})}[\mathtt{RG}^*\mathtt{2}]$$

$$\frac{r \in \widetilde{\mathcal{R}}(\mathtt{G}^{*^f}) \quad \mathtt{rdy}(\mathtt{G}) = \{\mathtt{p}\} \quad \mathcal{P}(\mathtt{G}) = \{\mathtt{p}, \mathtt{p}_1, \ldots, \mathtt{p}_n\} \quad \forall 1 \le i \le n : f(\mathtt{p}_i) = s_i\langle \mathtt{U_i}\rangle}{r\langle \mathtt{p}, s_1!\mathtt{U}_1\rangle \ldots \langle \mathtt{p}, s_n!\mathtt{U}_n\rangle\langle \mathtt{p}_1, s_1?\mathtt{U}_1\rangle \ldots \langle \mathtt{p}_n, s_n?\mathtt{U}_n\rangle \quad \in \quad \mathcal{R}(\mathtt{G}^{*^f})}[\mathtt{RGIter}]$$

Fig. 4. Runs of a global type

Note that $\widetilde{\mathcal{R}}(\mathtt{G}^{*^f}) = \{r_1, [r_1]r_2, [[r_1]r_2]r_3, \ldots\}$ with $r_i \in \mathcal{R}(\mathtt{G})$. When implementing an iterative type, we will allow the implementation to perform just a finite number of iterations (but we require at least once iteration). Annotation of optional events are instrumental to the comparison of traces associated with iterative types (which is defined below). Rule $[\mathtt{RGIter}]$ adds the events associated to the termination of an iteration: (i) the ready role $\mathtt{p}$ sends the termination signal to any other role by using the dedicated channels specified by f (i.e., $\langle \mathtt{p}, s_1!\mathtt{U}_1\rangle \ldots \langle \mathtt{p}, s_n!\mathtt{U}_n\rangle$), and (ii) the waiting roles receive the termination message (i.e., $\langle \mathtt{p}_1, s_1?\mathtt{U}_1\rangle \ldots \langle \mathtt{p}_n, s_n?\mathtt{U}_n\rangle$). As for parallel composition, we just consider one of the possible interleavings for the receive events (that can actually happen in any order).

We use the operator $\lessdot$ to compare annotated traces, which is defined as the least preorder satisfying the following rules

$$[r] \lessdot \epsilon \qquad \epsilon \lessdot r \qquad \frac{r \lessdot r'}{[r] \lessdot [r']} \qquad \frac{r_1 \lessdot r_1' \quad r_2 \lessdot r_2'}{r_1 r_2 \lessdot r_1' r_2'}$$

Basically, $r \lessdot r'$ means that r' matches all mandatory actions of r and all optional actions in r' are also optional in r. Let R_1 and R_2 be two sets of annotated traces, we write $R_1 \Subset R_2$ if $r \in R_1$ implies $\exists r' \in R_2$ such that $r \lessdot r'$.

Definition 4 (Whole-spectrum implementation). *A set $\mathbb{I}$ of implementations* covers *a global type* $\mathtt{G}$ *with respect to u iff $\mathcal{R}(\mathtt{G}) \Subset \mathcal{R}_u(\mathbb{I})$. A process P is a* whole-spectrum implementation of $\mathtt{p}_i \in \mathcal{P}(\mathtt{G}) = \{\mathtt{p}_0, \ldots, \mathtt{p}_n\}$ *when there exists a set $\mathbb{I}$ of implementations that covers* $\mathtt{G}$ *with respect to u s.t. $\mathcal{I}^{\iota}_{\mathcal{G}} \in \mathbb{I}$ implies $\iota(\mathtt{p}_i) = P$.*

A whole-spectrum implementation (WSI) of a role $\mathtt{p}_i$ is a process P such that any expected behaviour of the global type can be obtained by putting P into a proper context. For iteration types, the comparison of annotated traces implies that the implementation has to be able to perform the iteration body at least once but possibly many times.

Remark 1. A set of implementations covering a global type $\mathcal{G}$ can exhibit more behaviour than the runs of $\mathcal{G}$. Nonetheless, we use WSI with the usual soundness requirement (given in § 6) to characterise valid implementations.

5 Typing rules

We now give a typing system to guarantee that well-typed systems are a WSI of their global type. Systems are typed by judgements of the form $\mathcal{C} \smile \Gamma \;\vdash\; S \triangleright \Delta \divideontimes \Gamma'$

$$\frac{\Gamma(u)\equiv\mathcal{G}(\mathbf{y}) \qquad \mathcal{C}\smile\Gamma \;\vdash\; P \triangleright \Delta,\mathbf{y}:\mathcal{G}(\mathbf{y})\upharpoonright 0 \;\divideontimes\; \Gamma'}{\mathcal{C}\smile\Gamma \;\vdash\; \overline{u}^{n}(\mathbf{y}).P \triangleright \Delta \;\divideontimes\; \Gamma'}\,[\mathtt{VReq}]$$

$$\frac{\Gamma(u)\equiv\mathcal{G}(\mathbf{y}) \qquad \mathcal{C}\smile\Gamma \;\vdash\; P \triangleright \Delta,\mathbf{y}:\mathcal{G}(\mathbf{y})\upharpoonright i \;\divideontimes\; \Gamma'}{\mathcal{C}\smile\Gamma \;\vdash\; u_i(\mathbf{y}).P \triangleright \Delta \;\divideontimes\; \Gamma'}\,[\mathtt{VAcc}]$$

$$\frac{\begin{array}{c} P=\sum_{i\in I} y_i(x_i);P_i \qquad \forall i: y_i\in\mathbf{y} \text{ and } \mathcal{C}\smile\Gamma,x_i:\mathtt{U}_i \;\vdash\; P_i \triangleright \Delta,\mathbf{y}:\mathcal{T}_i \;\divideontimes\; \Gamma_i \\ \Gamma'=\bigcap_{i\in I}\Gamma_i \qquad \mathtt{fv}(P)\cup\mathtt{fc}(P)\subseteq \mathrm{dom}(\Gamma') \end{array}}{\mathcal{C}\smile\Gamma \;\vdash\; P \triangleright \Delta,\ \mathbf{y}:\sum_{i\in I} y_i?\mathtt{U}_i;\mathcal{T}_i \;\divideontimes\; \Gamma'}\,[\mathtt{VRec}]$$

$$\frac{\Gamma(e)=\mathtt{U} \qquad y\in\mathbf{y}}{\mathcal{C}\smile\Gamma \;\vdash\; \overline{y}e \triangleright \mathbf{y}:y!\mathtt{U} \;\divideontimes\; \Gamma}\,[\mathtt{VSend}] \qquad \frac{\Delta(\mathbf{s})=\mathtt{end} \quad \forall\mathbf{s}\in\mathrm{dom}(\Delta)}{\mathcal{C}\smile\Gamma \;\vdash\; \mathbf{0} \triangleright \Delta \;\divideontimes\; \Gamma}\,[\mathtt{VEnd}]$$

$$\frac{\mathcal{C}\smile\Gamma \;\vdash\; P_1 \triangleright \Delta_1 \;\divideontimes\; \Gamma_1 \qquad \mathcal{C}\smile\Gamma_1 \;\vdash\; P_2 \triangleright \Delta_1 \;\divideontimes\; \Gamma_2}{\mathcal{C}\smile\Gamma \;\vdash\; P_1;P_2 \triangleright \Delta_1;\Delta_2 \;\divideontimes\; \Gamma_2}\,[\mathtt{VSeq}]$$

$$\frac{\Gamma(e)=\mathtt{bool} \qquad \mathcal{C}\wedge e \not\vdash \bot \qquad \mathcal{C}\wedge\neg e \vdash \bot \qquad \mathcal{C}\wedge e\smile\Gamma \;\vdash\; P \triangleright \Delta \;\divideontimes\; \Gamma'}{\mathcal{C}\smile\Gamma \;\vdash\; \mathtt{if}\, e:\, P\, \mathtt{else}\, Q \triangleright \Delta \;\divideontimes\; \Gamma'}\,[\mathtt{VThen}]$$

$$\frac{\Gamma(e)=\mathtt{bool} \qquad \mathcal{C}\wedge e \vdash \bot \qquad \mathcal{C}\wedge\neg e \not\vdash \bot \qquad \mathcal{C}\wedge\neg e\smile\Gamma \;\vdash\; Q \triangleright \Delta \;\divideontimes\; \Gamma'}{\mathcal{C}\smile\Gamma \;\vdash\; \mathtt{if}\, e:\, P\, \mathtt{else}\, Q \triangleright \Delta \;\divideontimes\; \Gamma'}\,[\mathtt{VElse}]$$

$$\frac{\begin{array}{c} \Gamma(e)=\mathtt{bool} \qquad \mathcal{C}\wedge e \not\vdash \bot \qquad \mathcal{C}\wedge\neg e \not\vdash \bot \\ \mathcal{C}\wedge e\smile\Gamma \;\vdash\; P \triangleright \Delta_1 \;\divideontimes\; \Gamma_1 \qquad \mathcal{C}\wedge\neg e\smile\Gamma \;\vdash\; Q \triangleright \Delta_2 \;\divideontimes\; \Gamma_2 \end{array}}{\mathcal{C}\smile\Gamma \;\vdash\; \mathtt{if}\, e:\, P\, \mathtt{else}\, Q \triangleright \Delta_1 \bowtie \Delta_2 \;\divideontimes\; \Gamma_1\cap\Gamma_2}\,[\mathtt{VCond}]$$

$$\frac{\Gamma(\ell)=[\mathtt{U}] \quad \mathcal{C}\vdash \ell\neq\epsilon \quad \mathcal{C}\wedge x\in\ell\smile\Gamma,x:\mathtt{U} \;\vdash\; P \triangleright \mathbf{y}:\mathcal{T} \;\divideontimes\; \Gamma'}{\mathcal{C}\smile\Gamma \;\vdash\; \mathtt{for}\, x\, \mathtt{in}\, \ell\colon P \triangleright \mathbf{y}:\mathcal{T}^{*} \;\divideontimes\; \Gamma'}\,[\mathtt{VFor1}]$$

$$\frac{\mathcal{C}\vdash \ell=\epsilon}{\mathcal{C}\smile\Gamma \;\vdash\; \mathtt{for}\, x\, \mathtt{in}\, \ell\colon P \triangleright \mathbf{y}:\mathtt{end} \;\divideontimes\; \Gamma'}\,[\mathtt{VFor2}]$$

$$\frac{\mathcal{C}\smile\Gamma \;\vdash\; N \triangleright \mathbf{y}:\mathcal{T} \;\divideontimes\; \Gamma'}{\mathcal{C}\smile\Gamma \;\vdash\; \mathtt{do}\, N\, \mathtt{until}\, b(x) \triangleright \mathbf{y}:\mathcal{T}^{*};b?\mathtt{U} \;\divideontimes\; \Gamma',x:\mathtt{U}}\,[\mathtt{VLoop}]$$

Fig. 5. Typing rules for processes

stipulating that, under condition $\mathcal{C}$ and environment Γ, system S is typed as Δ and yields Γ' (where environments Γ, Γ' and Δ are as in § 2.3). Condition $\mathcal{C}$ is called *context assumption*; it is a logical formula derivable by the grammar

$$\mathcal{C} \;::=\; e \;\big|\; \neg\mathcal{C} \;\big|\; \mathcal{C}\wedge\mathcal{C} \qquad \text{where } e \text{ is of type } \mathtt{bool}$$

that identifies the assumptions on variables taken by processes in S. The map Γ' extends Γ with the sorts for the names bound in S. This is needed to correctly type $P;Q$ where in fact a free names of Q could be bound in P.

Due to space limits, Fig. 5 gives only the typing rules to validate processes (the rules for systems are adapted from Honda et al. [2008] and detailed in Appendix C). Condition $\mathcal{C} \not\vdash \bot$ is implicitly assumed among the hypothesis of each rule rule of Fig. 5. Rule [VReq] types session requests of the form $\overline{u}^{n}(\mathbf{y}).P$; its premise checks that P can be typed by extending Δ with the mapping from session names $\mathbf{y}$ to the projection of the global type $\Gamma(u)$ on the 0-th role. Dually, rule [VAcc] types the acceptance of a

$$(\Delta_1 \bowtie \Delta_2)(\mathbf{s}) = \begin{cases} \Delta_1(\mathbf{s}) & \text{if } \mathbf{s} \in \mathrm{dom}(\Delta_1) \setminus \mathrm{dom}(\Delta_2) \\ \Delta_2(\mathbf{s}) & \text{if } \mathbf{s} \in \mathrm{dom}(\Delta_2) \setminus \mathrm{dom}(\Delta_1) \\ \Delta_1(\mathbf{s}) \bowtie \Delta_2(\mathbf{s}) & \text{if } \mathbf{s} \in \mathrm{dom}(\Delta_1) \cap \mathrm{dom}(\Delta_2) \end{cases}$$

$$\mathcal{T}_1 \bowtie \mathcal{T}_2 = \begin{cases} \mathcal{T}_1 \oplus \mathcal{T}_2 & \text{if } \mathcal{T}_1 = y_1!\mathtt{U}_1; \mathcal{T}_1', \;\; \mathcal{T}_2 = y_2!\mathtt{U}_2; \mathcal{T}_2', \;\; y_1 \neq y_2 \\ y!\mathtt{U}; (\mathcal{T}_1' \bowtie \mathcal{T}_2') & \text{if } \mathcal{T}_1 = y!\mathtt{U}; \mathcal{T}_1', \;\; \mathcal{T}_2 = y!\mathtt{U}; \mathcal{T}_2' \\ \bot & \text{otherwise} \end{cases}$$

Fig. 6. Composition of types

session request as i-th role. Rule [VRec] types an external choice $P = \sum_{i \in I} y_i(x_i); P_i$ checking that each branch P_i can be typed against the respective continuation of the type, $\Delta, \mathbf{y} : \mathcal{T}_i$ (once Γ is updated with the type assignment on the bound name x_i); rule [VRec] cannot be applied (making the validation fail) when the names in $\mathtt{fv}(P) \cup \mathtt{fc}(P)$ are not mapped to the same sorts in all environments Γ_i. Rule [VSend] is trivial. Rules [VThen] and [VElse] handle the cases in which the guard of the conditional statement is either a tautology or a contradiction. Rule [VCond] ensures that both branches can be selected by fixing a proper assumption (i.e., both $\mathcal{C} \wedge e$ and $\mathcal{C} \wedge \neg e$ are consistent). Note that $\mathcal{C}$ is augmented with the condition e (resp. $\neg e$) for typing the 'then'-branch (resp. 'else'-branch). The resulting type is $\Delta_1 \bowtie \Delta_2$ defined in Fig. 6. The merge $\Delta_1 \bowtie \Delta_2$ is defined only when Δ_1 and Δ_2 are *compatible*, namely iff

$$\forall \mathbf{s}_1 \in \mathrm{dom}(\Delta_1), \mathbf{s}_2 \in \mathrm{dom}(\Delta_2) : \mathbf{s}_1 \cap \mathbf{s}_2 \neq \emptyset \implies \mathbf{s}_1 = \mathbf{s}_2$$

For $\mathbf{s} \notin \mathrm{dom}(\Delta_1) \cap \mathrm{dom}(\Delta_2)$, the merging behaves as the union of environments Δ_1 and Δ_2, otherwise it returns the merging of the local types $\mathcal{T}_1 = \Delta_1(\mathbf{s})$ and $\mathcal{T}_2 = \Delta_2(\mathbf{s})$; in turn, $\mathcal{T}_1 \bowtie \mathcal{T}_2$ yields an internal choice of $\mathcal{T}_1$ and $\mathcal{T}_2$, but for a common sequence of outputs. Rule [VFor1] assigns the type $\mathcal{T}^*$ to a for loop when its body P has type $\mathcal{T}$ under $\mathcal{C}$ extended with $x \in \ell$, and the environment Γ extended with $x : \mathtt{U}$. Rule [VFor2] is for empty lists. By rule [VLoop], the type of a loop is $\mathcal{T}^*; b?\mathtt{U}$ when its body P has type $\mathcal{T}$ and b is the channel used to receive the termination signal. Notice that the environments of the rules [VFor1] and [VLoop] include only one session (respectively $\mathbf{y} : \mathcal{T}^*$ and $\mathbf{y} : \mathcal{T}^*; b?\mathtt{U}$), hence the body can only perform actions within a single session. Iterations involving messages over multiple sessions could not be checked compositionally since the conformance of a process to a local type would not be sufficient to ensure the correct coordination of a 'for'-iteration with the corresponding 'loop'-iterations. Rule [VEnd] types idle processes with a Δ that maps each session $\mathbf{s}$ to the $\mathtt{end}$ type. Rule [VSeq] checks sequential composition. Here $\Delta_1; \Delta_2$ is the pointwise sequential composition of Δ_1 and Δ_2, i.e., $(\Delta_1; \Delta_2)(\mathbf{s}) = \mathcal{T}_1; \mathcal{T}_2$ where $\mathcal{T}_i = \Delta_i(\mathbf{s})$ if $\mathbf{s} \in \mathrm{dom}(\Delta_i)$ and $\mathcal{T}_i = \mathtt{end}$ otherwise, for $i = 1, 2$. Note that P_2 is typed under the environment Γ_1, which contains the names bound by the input prefixes of P_1.

The following result ensures that type checking is decidable (it follows from the obvious recursive algorithm and decidability of the underlying logic).

Theorem 1. *Given $\mathcal{C}, \Gamma, \Gamma', S$ and Δ, then the provability of $\mathcal{C} \,\lrcorner\, \Gamma \;\vdash\; S \triangleright \Delta \divideontimes \Gamma'$ is decidable.*

Our proof system discerns between B_1 and B_2 in the introduction (i.e., only B_1 is validated) due to the rules for conditional statements and to the lack of a rule for type refinement. In fact, after a few verification steps on B_1 (resp. B_2) we would reach the following scenario: $P_1(c) = \texttt{if}\ c :\ \overline{s_{\texttt{OK}}}; s_{\texttt{AMOUNT}}(x)\ \texttt{else}\ \overline{s_{\texttt{KO}}}$ (resp. $P_2(c) = \overline{s_{\texttt{KO}}}$) and $\Delta = \mathbf{s} : \texttt{OK}!; \texttt{AMOUNT}? \oplus \texttt{KO}!$. The verification of $P_1(c)$ terminates successfully after an application of $[\texttt{VCond}]$. In the case of $P_2(c)$ the only rule for a sending process, $[\texttt{VSend}]$, cannot be applied against a type with a choice.

5.1 Running examples

We now apply our typing to different implementations of POP2.

Example 5. The first few verification steps of $P_{\texttt{INIT}}$ from Ex. 3 are shown below. By rule $[\texttt{VAcc}]$, the newly created session is added to the session environment, then the verification of the external choice is split by $[\texttt{VRec}]$ into the verification of each branch. As we omit the whole derivation, just assume that $P_{\texttt{INIT}}$ yields $\Gamma' = \Gamma, x : \texttt{Str}$.

$$\dfrac{\Gamma(u) \equiv \texttt{G}_{\texttt{POP}}(\mathbf{s}) \quad \dfrac{\begin{array}{c}\texttt{true} \sqcup \Gamma \ \vdash\ P_{\texttt{EXIT}} \triangleright \mathbf{s} : \texttt{T}_{\texttt{EXIT}} \divideontimes \Gamma' \\ \texttt{true} \sqcup \Gamma, x : \texttt{Str} \ \vdash\ P_{\texttt{MBOX}} \triangleright \mathbf{s} : \texttt{T}_{\texttt{MBOX}} \divideontimes \Gamma'\end{array}}{\texttt{true} \sqcup \Gamma \ \vdash\ s_{\texttt{QUIT}}(); P_{\texttt{EXIT}} + s_{\texttt{HELO}}(x); P_{\texttt{MBOX}} \triangleright \mathbf{s} : \texttt{T}_{\texttt{S}} \divideontimes \Gamma'}\,[\texttt{VRec}]}{\texttt{true} \sqcup \Gamma \ \vdash\ u_{\texttt{S}}(\mathbf{s}).P_{\texttt{S}} \triangleright \emptyset \divideontimes \Gamma'}\,[\texttt{VAcc}]$$

Consider the second branch $P_{\texttt{MBOX}}$; assuming that $\texttt{true} \wedge \texttt{auth}(x)$ is neither a tautology nor a falsum, we apply $[\texttt{VCond}]$ (if it was, the verification would terminate unsuccessfully as the only possible branch would not validate against the choice in $\texttt{T}_{\texttt{MBOX}}$).

$$\dfrac{\begin{array}{c}\texttt{auth}(x) \sqcup \Gamma, x : \texttt{Str} \ \vdash\ \overline{s_{\texttt{R}}}\texttt{mn}(\text{“}\texttt{inbox}\text{”}); P_{\texttt{NMBR}} \triangleright \mathbf{s} : \texttt{R}!\texttt{Int}; \texttt{T}_{\texttt{NMBR}} \divideontimes \Gamma' \\ \neg\texttt{auth}(x) \sqcup \Gamma, x : \texttt{Str} \ \vdash\ \overline{s_{\texttt{E}}}; P_{\texttt{EXIT}} \triangleright \mathbf{s} : \texttt{E}!; \texttt{T}_{\texttt{EXIT}} \divideontimes \Gamma'\end{array}}{\texttt{true} \sqcup \Gamma, x : \texttt{Str} \ \vdash\ P_{\texttt{MBOX}} \triangleright \mathbf{s} : \texttt{T}_{\texttt{MBOX}} \divideontimes \Gamma'}\,[\texttt{VCond}]$$

The rest is trivial observing $\mathbf{s} : \texttt{T}_{\texttt{MBOX}} = \mathbf{s} : \texttt{R}!\texttt{Int}; \texttt{T}_{\texttt{NMBR}} \bowtie \mathbf{s} : \texttt{E}!; \texttt{T}_{\texttt{EXIT}}$. ♦

Ex. 6 types the multiparty variant given in Ex. 4.

Example 6. Assume $\Gamma(u) \equiv \texttt{G}'_{\texttt{POP}}$. The first steps are as in Ex. 5 by rules $[\texttt{VAcc}]$ and $[\texttt{VRec}]$. We focus on the second branch that in this case is $P_{\texttt{AUTH}}$ and apply $[\texttt{VSeq}]$.

$$\dfrac{\begin{array}{c}\texttt{true} \sqcup \Gamma, x : \texttt{Str} \ \vdash\ \overline{s_{\texttt{REQ}}}x \triangleright \mathbf{s} : \texttt{REQ}!\texttt{Str} \divideontimes \Gamma' \\ \texttt{true} \sqcup \Gamma, x : \texttt{Str} \ \vdash\ s_{\texttt{RES}}(y); P'_{\texttt{MBOX}} \triangleright \mathbf{s} : \texttt{RES}?\texttt{Bool}; \texttt{T}_{\texttt{AUTH}} \divideontimes \Gamma'\end{array}}{\texttt{true} \sqcup \Gamma, x : \texttt{Str} \ \vdash\ \overline{s_{\texttt{REQ}}}x; s_{\texttt{RES}}(y); P'_{\texttt{MBOX}} \triangleright \mathbf{s} : \texttt{T}_{\texttt{AUTH}} \divideontimes \Gamma'}\,[\texttt{VSeq}]$$

We show the verification of the first branch

$$\dfrac{\dfrac{-}{\texttt{true} \sqcup \Gamma, x : \texttt{Str} \ \vdash\ \mathbf{0} \triangleright \mathbf{s} : \texttt{end} \divideontimes \Gamma'}\,[\texttt{VEnd}]}{\texttt{true} \sqcup \Gamma, x : \texttt{Str} \ \vdash\ \overline{s_{\texttt{REQ}}}x \triangleright \mathbf{s} : \texttt{REQ}!\texttt{Str} \divideontimes \Gamma'}\,[\texttt{VSend}]$$

and the successive steps for the second branch:

$$\dfrac{\texttt{true} \sqcup \Gamma, x : \texttt{Str} \ \vdash\ P'_{\texttt{MBOX}} \triangleright \mathbf{s} : \texttt{T}'_{\texttt{MBOX}} \divideontimes \Gamma'}{\texttt{true} \sqcup \Gamma, x : \texttt{Str} \ \vdash\ s_{\texttt{RES}}(y); P'_{\texttt{MBOX}} \triangleright \mathbf{s} : \texttt{RES}?\texttt{Bool}; \texttt{T}'_{\texttt{MBOX}} \divideontimes \Gamma'}\,[\texttt{VRec}]$$

The verification of $P'_{\mathtt{MBOX}}$ proceeds with an application of [VCond], [VIf] or [VElse] depending on $\mathtt{auth}() \wedge y$. If $\mathtt{auth}()$ is not a contradiction then [VCond] can be applied as the condition depends from the context (that is the administrator). This leads to a successful validation. In this case, unlike in Ex. 5, the implementation is whole-spectrum even if $\mathtt{auth}()$ is a tautology. If $\mathtt{auth}()$ is a falsum then [VElse] is applied and the process will not validate against the type which has a choice. ♦

Ex. 7 deals with a process implementing two interleaved sessions. Ex. 7 shows that the verification scales to more complex processes that compose different protocols.

Example 7. We give a process that, upon request, engages as a server in a session $\mathtt{G_{POP}}$ (§ 2.2), and as a client in a session $\mathtt{G_{ADMIN}}$ to outsource the authentication. Instead of embedding in the same session this extra interactions with the administrator, as we did in Ex. 2, we represent the multiparty interaction as two interleaved sessions.

$$\mathtt{G_{ADMIN}}{=}\mathtt{C} \to \mathtt{A} : \mathtt{REQ}\,\langle \mathtt{Str} \rangle ; \mathtt{A} \to \mathtt{C} : \mathtt{RES}\,\langle \mathtt{Bool} \rangle \qquad \mathtt{T_C}{=}\mathtt{REQ!Str};\mathtt{RES?Bool}$$

In $\mathtt{G_{ADMIN}}$, the client C sends the administrator A a password and A replies along RES. $\mathtt{T_C}$ is the projection on $\mathtt{G_{ADMIN}}$ on C. We assume $\Gamma(u) \equiv \mathtt{G_{POP}}$ and $\Gamma(v) \equiv \mathtt{G_{ADMIN}}$. Process $P_{\mathtt{INIT}}$ starts, upon request, a session of type $\mathtt{G_{POP}}$ and then requests to start a session of type $\mathtt{G_{ADMIN}}$. We omit the definition of processes $P_{\mathtt{NMBR}}$ and $P_{\mathtt{EXIT}}$ which are as in Ex. 3.

$$\begin{array}{cc} P''_{\mathtt{INIT}}{=}u_{\mathtt{S}}(\mathbf{s}).\overline{v}^{\mathtt{C}}(\mathbf{t}).P''_{\mathtt{S}} & P''_{\mathtt{S}}{=}s_{\mathtt{QUIT}}();P_{\mathtt{EXIT}} + s_{\mathtt{HELO}}(x);P_{\mathtt{AUTH}} \\ P_{\mathtt{AUTH}}{=}\overline{t_{\mathtt{REQ}}}x;t_{\mathtt{RES}}(y);P''_{\mathtt{MBOX}} & P''_{\mathtt{MBOX}}{=}\mathtt{if}\ y:\ \overline{s_{\mathtt{R}}}\mathtt{fn}(\text{“}\mathtt{inbox}\text{”});P_{\mathtt{NMBR}}\ \mathtt{else}\ \overline{s_{\mathtt{E}}};P_{\mathtt{EXIT}} \end{array}$$

The authentication is delegated to the administrator in session $\mathbf{t}$ via the message along $t_{\mathtt{REQ}}$. Session $\mathbf{s}$ continues using the information in y, which stores the last message received in session $\mathbf{t}$. The first verification steps are by rules [VAcc], [VReq] and [VRec].

$$\dfrac{\Gamma(u) \equiv \mathtt{G_{POP}}(\mathbf{s}) \quad \Gamma(v) \equiv \mathtt{G_{POP}}(\mathbf{t}) \quad \dfrac{\begin{array}{c}\mathtt{true} \mathbin{\llcorner} \Gamma \ \vdash\ P_{\mathtt{EXIT}} \rhd \mathbf{s} : \mathtt{T_{EXIT}}, \mathbf{t} : \mathtt{T_S} \circledast \Gamma' \\ \mathtt{true} \mathbin{\llcorner} \Gamma, x : \mathtt{Str} \ \vdash\ P_{\mathtt{AUTH}} \rhd \mathbf{s} : \mathtt{T_{MBOX}}, \mathbf{t} : \mathtt{T_S} \circledast \Gamma'\end{array}}{\mathtt{true} \mathbin{\llcorner} \Gamma \ \vdash\ P''_{\mathtt{S}} \rhd \mathbf{s} : \mathtt{T_S}, \mathbf{t} : \mathtt{T_S} \circledast \Gamma'}\,[\mathtt{VRec}]}{\mathtt{true} \mathbin{\llcorner} \Gamma \ \vdash\ u_{\mathtt{S}}(\mathbf{s}).\overline{v}^{\mathtt{S}}(\mathbf{t}).P''_{\mathtt{S}} \rhd \emptyset \circledast \Gamma'}\,[\mathtt{VAcc}],[\mathtt{VReq}]$$

The verification of the second branch $P_{\mathtt{AUTH}}$ proceeds with one application of [VSeq] where $(\mathbf{s} : \mathtt{T_{MBOX}}, \mathbf{t} : \mathtt{T_S}) \equiv (\mathbf{s} : \mathtt{end}, \mathbf{t} : \mathtt{REQ!Str}); (\mathbf{s} : \mathtt{T_{MBOX}}, \mathbf{t} : \mathtt{RES?Bool})$.

$$\dfrac{\begin{array}{c}\mathtt{true} \mathbin{\llcorner} \Gamma, x : \mathtt{Str} \ \vdash\ \overline{t_{\mathtt{REQ}}}x \rhd \Delta \circledast \mathbf{s} : \mathtt{end}, \mathbf{t} : \mathtt{REQ!Str} \\ \mathtt{true} \mathbin{\llcorner} \Gamma, x : \mathtt{Str} \ \vdash\ t_{\mathtt{RES}}(y);P''_{\mathtt{MBOX}} \rhd \mathbf{s} : \mathtt{T_{MBOX}}, \mathbf{t} : \mathtt{RES?Bool} \circledast \Gamma'\end{array}}{\mathtt{true} \mathbin{\llcorner} \Gamma, x : \mathtt{Str} \ \vdash\ \overline{t_{\mathtt{REQ}}}x;t_{\mathtt{RES}}(y);P''_{\mathtt{MBOX}} \rhd \mathbf{s} : \mathtt{T_{MBOX}}, \mathbf{t} : \mathtt{T_S} \circledast \Gamma'}\,[\mathtt{VSeq}]$$

Focusing on the second branch we apply [VRec]

$$\dfrac{\mathtt{true} \mathbin{\llcorner} \Gamma, x : \mathtt{Str}, y : \mathtt{Bool} \ \vdash\ P''_{\mathtt{MBOX}} \rhd \mathbf{s} : \mathtt{T_{MBOX}}, \mathbf{t} : \mathtt{end} \circledast \Gamma'}{\mathtt{true} \mathbin{\llcorner} \Gamma, x : \mathtt{Str} \ \vdash\ t_{\mathtt{RES}}(y);P''_{\mathtt{MBOX}} \rhd \mathbf{s} : \mathtt{T_{MBOX}}, \mathbf{t} : \mathtt{RES?Bool} \circledast \Gamma'}\,[\mathtt{VRec}]$$

Rule [VCond] can be applied since condition y of the conditional statement in $P''_{\mathtt{MBOX}}$ is neither a tautology nor a contradiction. The rest is as in Ex. 5. ♦

Let $\mathtt{push}([\_], s!\mathtt{v}) = s!\mathtt{v}[\_]$ and $\mathtt{push}(s_1!\mathtt{v}_1; \ldots; s_n!\mathtt{v}_n[\_], s!\mathtt{v}) = s_1!\mathtt{v}_1; \ldots; s_n!\mathtt{v}_n; s!\mathtt{v}[\_]$

$$\Gamma \bullet \Delta, \mathtt{s} : s!\mathtt{v}; \mathbb{M}@\,\mathtt{p} \xrightarrow{\overline{s}\mathtt{v}} \Gamma \bullet \Delta, \mathtt{s} : \mathbb{M}@\,\mathtt{p} \; [\mathtt{TQueue}]$$

$$\frac{\Gamma \bullet \mathtt{s} : \mathcal{T} \xrightarrow{\overline{s}\mathtt{v}} \Gamma \bullet \mathtt{s} : \mathcal{T}' \quad \mathbb{M}'[\_] = \mathtt{push}(\mathbb{M}[\_], s!\mathtt{v})}{\Gamma \bullet \Delta, \mathtt{s} : \mathbb{M}[\mathcal{T}]@\,\mathtt{p} \xrightarrow{\tau} \Gamma \bullet \Delta, \mathtt{s} : \mathbb{M}'[\mathcal{T}']@\,\mathtt{p}} \; [\mathtt{TCom1}]$$

$$\frac{\Gamma \bullet \mathtt{s} : \mathcal{T} \xrightarrow{s\mathtt{v}} \Gamma \bullet \mathtt{s} : \mathcal{T}'}{\Gamma \bullet \mathtt{s} : s!\mathtt{v}; \mathbb{M}_1@\,\mathtt{p},\ \mathbb{M}_2[\mathcal{T}]@\,\mathtt{q} \xrightarrow{\tau} \Gamma \bullet \mathtt{s} : \mathbb{M}_1@\,\mathtt{p},\ \mathbb{M}_2[\mathcal{T}']@\,\mathtt{q}} \; [\mathtt{TCom2}]$$

$$\frac{u \in \mathrm{dom}(\Gamma) \qquad \Gamma(u) \equiv \mathcal{G}(\mathtt{s}) \stackrel{\triangle}{=} \mathtt{G} \qquad \mathcal{P}(\mathtt{G}) = \{\mathtt{p}_1, \ldots, \mathtt{p}_n\}}{\Gamma \bullet \Delta \xrightarrow{\tau} \Gamma \bullet \Delta, \mathtt{s} : (\mathtt{G}\!\upharpoonright\!\mathtt{p}_1)@\,\mathtt{p}_1, \ldots, \mathtt{s} : (\mathtt{G}\!\upharpoonright\!\mathtt{p}_n)@\,\mathtt{p}_n} \; [\mathtt{TInit}]$$

Fig. 7. Additional labelled transitions (to those of Fig. 1) for runtime specifications

6 Properties of the type system

Runtime Types The properties of our type system are stated in terms of the behaviour of local types. As in Honda et al. [2008], *runtime types* extend local types with *message contexts* $\mathbb{M}$ of the form $s_1!\mathtt{v}_1; \cdots; s_n!\mathtt{v}_n[\_]$ with $n \geq 0$, namely $\mathbb{M}$ is a sequence of outputs followed by a hole $[\_]$. To model asynchrony, we stipulate the equality

$$s_1!\mathtt{v}_1; s_2!\mathtt{v}_2; \mathbb{M} \approx s_2!\mathtt{v}_2; s_1!\mathtt{v}_1; \mathbb{M} \qquad \text{if } s_1 \neq s_2$$

A *runtime type* is either a message context $\mathbb{M}$ or a "type in context", that is a term $\mathbb{M}[\mathcal{T}]$. We extend environments so to map session names $\mathtt{s}$ to runtime types of roles in $\mathtt{s}$; we write $\Delta, \mathtt{s} : \mathbb{M}[\mathcal{T}]@\,\mathtt{p}$ to specify that (1) the runtime type of $\mathtt{p} \in \mathbb{P}$ in $\mathtt{s}$ is $\mathbb{M}[\mathcal{T}]$ and (2) that for any $\mathtt{s} : \mathbb{M}'[\mathcal{T}']@\,\mathtt{q}$ in Δ we have $\mathtt{q} \neq \mathtt{p}$.

The semantics of runtime types is obtained by adding the rules in Fig. 7 to those in Fig. 1. Rule [TQueue] removes a message from of a queue. Rules [TCom1] and [TCom2] establish how runtime specifications send and receives messages (the transition in their premises are derived from the rules in Fig. 1). Rule [TInit] initiates a new session by mapping the new session $\mathtt{s}$ to the projections of the global type assigned by Γ.

Soundness The typing rules in § 5 ensure the semantic conformance of processes with the behaviour prescribed by their types. Here, we define conformance in terms of *conditional simulation* that relates states and specifications. Our definition is standard, except for input actions, for which specifications have to simulate only inputs of messages with the expected type (i.e., systems are not responsible when receiving ill-typed messages).

Define $\stackrel{\alpha}{\Longrightarrow} = \xrightarrow{\tau}^{*} \xrightarrow{\alpha}$. Let $\Gamma \bullet \Delta \stackrel{s}{\Longrightarrow}$ shorten $\exists \Delta' \exists \mathtt{v} : \Gamma \bullet \Delta \stackrel{s\mathtt{v}}{\Longrightarrow} \Gamma \bullet \Delta'$.

Definition 5 (Conditional simulation). *A relation $\mathbb{R}$ between states and specifications is a* conditional simulation *iff for any $(\langle S, \sigma \rangle, \Gamma \bullet \Delta) \in \mathbb{R}$, if $\langle S, \sigma \rangle \xrightarrow{e \vdash \alpha} \langle S', \sigma' \rangle$ then*

1. *if $\alpha = s\mathtt{v}$ then $\Gamma \bullet \Delta \stackrel{s}{\Longrightarrow}$ and if $\Gamma \bullet \Delta \stackrel{s\mathtt{v}}{\Longrightarrow}$ then there is $\Gamma \bullet \Delta'$ such that $\Gamma \bullet \Delta \stackrel{s\mathtt{v}}{\Longrightarrow} \Gamma \bullet \Delta'$ and $(\langle S', \sigma' \rangle, \Gamma \bullet \Delta') \in \mathbb{R}$*
2. *otherwise, $\Gamma \bullet \Delta \stackrel{\alpha}{\Longrightarrow} \Gamma \bullet \Delta'$ and $(\langle S', \sigma' \rangle, \Gamma \bullet \Delta') \in \mathbb{R}$.*

We write $\langle S, \sigma \rangle \precsim \Gamma \bullet \Delta$ if there is a conditional simulation $\mathbb{R}$ s.t. $(\langle S, \sigma \rangle, \Gamma \bullet \Delta) \in \mathbb{R}$.

By (1), only inputs of S with the expected type have to be matched by $\Gamma \bullet \Delta$ (recall rule [TRec] in Fig. 1), while it is no longer expected to conform to the specification after an ill-typed input (i.e., not allowed by $\Gamma \bullet \Delta$).

Def. 6 establishes consistency for stores in terms of preservation of variables' sorts.

$$\frac{r \in \mathcal{R}_{\mathbf{s}}(\Delta, \mathbf{s} : \mathbb{M}[\mathtt{T}_k]@\,\mathtt{p},\ \mathbb{M}'[\mathtt{T}'_k]@\,\mathtt{q}) \qquad k \in J}{\langle \mathtt{p}, s_k!\mathtt{U}_k\rangle\langle \mathtt{q}, s_k?\mathtt{U}_k\rangle r \in \mathcal{R}_{\mathbf{s}}(\Delta, \mathbf{s} : \mathbb{M}[s_k!\mathtt{U}_k;\mathtt{T}_k]@\,\mathtt{p},\ \mathbb{M}'[\sum_{j\in J} s_j?\mathtt{U}_j;\mathtt{T}'_j]@\,\mathtt{q})}[\mathtt{RTCom}]$$

$$\frac{r \in \mathcal{R}_{\mathbf{s}}(\Delta, \mathbf{s} : \mathtt{T}_i;\mathtt{T}_j@\,\mathtt{p})}{r \in \mathcal{R}_{\mathbf{s}}(\Delta, \mathbf{s} : \mathtt{T}_i^*;\mathtt{T}_j@\,\mathtt{p})}[\mathtt{RTIt1}] \qquad \frac{rr' \in \mathcal{R}_{\mathbf{s}}(\Delta, \mathbf{s} : \mathtt{T}_i;\mathtt{T}_i^*;\mathtt{T}_j@\,\mathtt{p}) \quad r' \in \mathcal{R}_{\mathbf{s}}(\Delta \mathbf{s} : \mathtt{T}_i;\mathtt{T}_j@\,\mathtt{p})}{[r]r' \in \mathcal{R}_{\mathbf{s}}(\Delta, \mathbf{s} : \mathtt{T}_i^*;\mathtt{T}_j@\,\mathtt{p})}[\mathtt{RTIt2}]$$

$$\frac{r \in \mathcal{R}_{\mathbf{s}}(\Delta) \qquad \mathbf{s} \neq \mathbf{r}}{r \in \mathcal{R}_{\mathbf{s}}(\Delta, \mathbf{r} : \mathcal{T})}[\mathtt{RTPar}] \qquad \frac{r \in \mathcal{R}_{\mathbf{s}}(\Delta)}{r \in \mathcal{R}_{\mathbf{s}}(\Delta, \mathbf{s} : \mathtt{end}@\,\mathtt{p})}[\mathtt{RTEnd1}] \qquad \epsilon \in \mathcal{R}_{\mathbf{s}}(\emptyset)[\mathtt{RTEnd2}]$$

$$\frac{r \in \mathcal{R}_{\mathbf{s}}(\Delta, \mathbf{s} : s_j!\mathtt{U}_j;\mathtt{T}_j@\,\mathtt{p}) \quad j \in I}{r \in \mathcal{R}_{\mathbf{s}}(\Delta, \mathbf{s} : \bigoplus_{i\in I} s_i!\mathtt{U}_i;\mathtt{T}_i@\,\mathtt{p})}[\mathtt{RTCh}]$$

Fig. 8. Runs of runtime local types

Definition 6 (Consistent store). *Given an environment Γ, a context assumption $\mathcal{C}$, and a state $\langle S, \sigma\rangle$ with* $\mathrm{var}(S) \subseteq \mathrm{dom}(\sigma)$*, store σ is* consistent for S with respect to Γ and $\mathcal{C}$ *iff $\forall x \in \mathrm{dom}(\sigma)$, $\sigma(x) : \Gamma(x)$, and $\mathcal{C} \downarrow \sigma = \mathtt{true}$.*

Theorem 2 (Subject reduction). *Assume that*

$$\mathcal{C} \,\lrcorner\, \Gamma \ \vdash \ S \triangleright \Delta \ast \Gamma' \quad \text{and} \quad \langle S, \sigma\rangle \xrightarrow{e\vdash\alpha} \langle S', \sigma'\rangle$$

with σ consistent for S with respect to Γ and $\mathcal{C}$. Then

1. *if $\alpha = s\mathtt{v}$ then $\Gamma \bullet \Delta \stackrel{s}{\Longrightarrow}$ and if $\Gamma \bullet \Delta \stackrel{s\mathtt{v}}{\Longrightarrow}$ then there is $\Gamma \bullet \Delta'$ such that $\Gamma \bullet \Delta \stackrel{s\mathtt{v}}{\Longrightarrow} \Gamma \bullet \Delta'$ with $\mathtt{v} : \mathtt{U}$ and $\mathcal{C} \wedge e \,\lrcorner\, \Gamma, x : \mathtt{U} \ \vdash \ S' \triangleright \Delta' \ast \Gamma''$ for some x and some $\Gamma'' \supseteq \Gamma'$*
2. *otherwise $\Gamma \bullet \Delta \stackrel{\alpha}{\Longrightarrow} \Gamma \bullet \Delta'$ and $\mathcal{C} \wedge e \,\lrcorner\, \Gamma \ \vdash \ S' \triangleright \Delta' \ast \Gamma''$ for some $\Gamma'' \supseteq \Gamma'$.*

Corollary 1 (Soundness). *If $\mathcal{C} \,\lrcorner\, \Gamma \ \vdash \ S \triangleright \Delta \ast \Gamma'$ then $\langle S, \sigma\rangle \precsim \Gamma \bullet \Delta$ for all σ consistent store for S with respect to Γ and $\mathcal{C}$.*

WSI by typing We show that well-typed processes are WSIs (Def. 4). First, we relate the runs of a global type with those of its corresponding runtime types. Then, we state the correspondence between the runs of runtime types and well-typed implementations.

Definition 7 (Runs of runtime types). *The set $\mathcal{R}_{\mathbf{s}}(\Delta)$ denotes the runs of events over the channels in* s *generated by Δ, and is inductively defined by the rules in Fig. 8.*

Rule [RTCom] builds the runs for two communicating types. Rules [RTIt1] and [RTIt2] unfold the runs of an iterative type. Note that the mandatory actions of runs associated to recursive types are those requiring at least one execution of the iteration body, while additional executions are optional. The remaining rules are self-explanatory. (The correspondence between the operational and denotational semantics is in Appendix E.)

Thm. 3 ensures that well-formed global types are covered by their projections, while Thm. 4 states that the set of well-typed implementation covers its specification.

Theorem 3. *For any global type $\mathcal{G}(\mathbf{s})$, $\mathcal{R}(\mathcal{G}(\mathbf{s})) \Subset \mathcal{R}_{\mathbf{s}}(\{\mathbf{s} : (\mathcal{G}(\mathbf{s})\upharpoonright \mathtt{p})@\,\mathtt{p}\}_{\mathtt{p}\in\mathcal{P}(\mathcal{G}(\mathbf{s}))})$.*

Theorem 4. *Let* $\mathcal{G}(\mathtt{s}) \stackrel{\triangle}{=} \mathtt{G}$ *be a global type. Fix* $\mathtt{p} \in \mathcal{P}(\mathtt{G})$ *and* P *a well-typed implementation of* $\mathtt{p}$*. Define*

$$\mathbb{I}_{\mathtt{p},P} = \{\mathcal{I}^{\iota}_{\mathcal{G}} | \iota(\mathtt{p}) = P, \forall \mathtt{q} \in \mathcal{P}(\mathtt{G}) : \mathtt{true} \mathrel{\llcorner} \Gamma, u : \mathcal{G}(\mathtt{s}) \ \vdash \ \iota(\mathtt{q}) \triangleright \Delta, \mathtt{s} : \mathcal{G}(\mathtt{s}) \upharpoonright \mathtt{q} \circledast \Gamma'\}$$

then, $\mathcal{R}_{\mathtt{s}}(\{\mathtt{s} : (\mathcal{G}(\mathtt{s}) \upharpoonright \mathtt{p}) @ \mathtt{p}\}_{\mathtt{p} \in \mathcal{P}(\mathcal{G})}) \quad \Subset \quad \mathcal{R}_{u}(\mathbb{I}_{\mathtt{p},P})$.

7 Conclusion and related work

WSI forbids implementations of a role that persistently avoid the execution of some alternative branches in a choreography. Although WSI is defined as a relation between the traces of a global type and those of its candidate implementations, it can be checked by using multiparty session types. Technically, we show that (i) the sets of the projections of a global type $\mathcal{G}$ preserves all the traces in $\mathcal{G}$ (Thm. 3); and (ii) any trace of a local type can be mimicked by a well-typed implementation, if interacting in a proper context (Thm. 4). The soundness of our type system (Corollary 1) ensures that well-typed implementations behave as prescribed by the choreography.

We are currently working on the extension of WSI to other models of choreography as e.g. those based on automata Fu et al. [2005], which poses the classical question about the decidability of the notion of realisability (see Basu et al. [2012]). To the best of our knowledge, the only proposal dealing with complete (i.e., exhaustive) realisations in a behavioural context is Castagna et al. [2012] but this approach focuses on non-deterministic implementation languages. Our type system is more restrictive than Bettini et al. [2008], Bravetti and Zavattaro [2009], Caires and Vieira [2009], Castagna and Padovani [2009], Honda et al. [2008]. We do not consider subtyping because the liberal elimination of internal choices prevents WSI. The investigation of suitable forms of subtyping for WSIs is scope for future work.

WSI coincides with projection realisability Castagna et al. [2012], Lanese et al. [2008], Salaün and Bultan [2009] when implementation languages feature non-deterministic internal choices. On the contrary, WSI provides a finer criterion to distinguish deterministic implementations, as illustrated by the motivating example in the introduction. To some extend our proposal is related to the fair subtyping approach in Padovani [2011], where refinement is studied under the fairness assumption: Fair subtyping differs from usual subtyping when considering infinite computations but WSI differs from partial implementation also when considering finite computations.

The static verification of WSI requires a form of recursion more restrictive than the one in Bettini et al. [2008], Honda et al. [2008], where the number of iterations is limited. This restriction is on the lines of Castagna et al. [2012] that also considers finite traces. The extension of our theory with a more general form of iteration is scope for future work. We argue that this is attainable using annotations Deng and Sangiorgi [2006], Yoshida et al. [2001] to detect loop termination by typing.

Acknowledgements We thank the anonymous reviewers for their insightful and helpful comments on the previous version of this paper.

References

S. Basu, T. Bultan, and M. Ouederni. Deciding choreography realizability. In *POPL*, 2012.

L. Bettini, M. Coppo, L. D'Antoni, M. D. abd Mariangiola Dezani-Ciancaglini, and N. Yoshida. Global progress in dynamically interleaved multiparty sessions. In *CONCUR*, 2008.

M. Bravetti and G. Zavattaro. A theory of contracts for strong service compliance. *MSCS*, 19(3), 2009.

M. Butler, J. Postel, D. Chase, J. Goldberger, and J. Reynoldsa. Post office protocol - version 2. RFC 918, available at `http://tools.ietf.org/html/rfc937`, February 1985.

L. Caires and H. T. Vieira. Conversation types. In *ESOP*, 2009.

G. Castagna and L. Padovani. Contracts for mobile processes. In *CONCUR*, 2009.

G. Castagna, M. Dezani-Ciancaglini, and L. Padovani. On global types and multi-party session. *LMCS*, 8(1), 2012.

T.-C. Chen and K. Honda. Specifying stateful asynchronous properties for distributed programs. In *CONCUR*, 2012.

D. Crocker. Standard for the format of arpa internet text messages. RFC 822, available at `www.ietf.org/rfc/rfc0822.txt`, February 1982.

Y. Deng and D. Sangiorgi. Ensuring termination by typability. *Inf. Comput.*, 204(7), 2006.

M. Dezani-Ciancaglini and U. de' Liguoro. Sessions and session types: An overview. In *WS-FM*, 2009.

X. Fu, T. Bultan, and J. Su. Realizability of conversation protocols with message contents. *Int. J. Web Service Res.*, 2(4):68–93, 2005.

S. Gay and M. Hole. Subtyping for Session Types in the Pi-Calculus. *Acta Inf.*, 42(2/3):191–225, 2005.

K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. In *POPL*, 2008.

N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, and Y. Lafon. `http://www.w3.org/TR/2004/WD-ws-cdl-10-20041217`, 2004.

L. Lamport. Time, clocks, and the ordering of events in a distributed system. *CACM*, 21(7): 558–564, July 1978.

I. Lanese, C. Guidi, F. Montesi, and G. Zavattaro. Bridging the gap between interaction-and process-oriented choreographies. In *SEFM*, 2008.

J. Lange and E. Tuosto. Synthesising choreographies from local session types. In *CONCUR*, 2012.

N. Lohmann and K. Wolf. Decidability results for choreography realization. In *ICSOC*, 2011.

R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

L. Padovani. Fair subtyping for multi-party session types. In *COORDINATION*, 2011.

G. Salaün and T. Bultan. Realizability of choreographies using process algebra encodings. In *Integrated Formal Methods*, 2009.

J. Su, T. Bultan, X. Fu, and X. Zhao. Towards a theory of web service choreographies. In *WS-FM*, 2007.

N. Yoshida, M. Berger, and K. Honda. Strong normalisation in the pi-calculus. In *LICS*, 2001.

A Auxiliary functions

We define some common auxiliary functions. In the following we fix a a global type term $\mathtt{G}$.

- The set of channel names $\mathtt{ch}(\mathtt{G}) \subseteq \mathbb{C}$ of $\mathtt{G}$ is defined as:

$$\begin{aligned} \mathtt{ch}(\mathtt{p} \to \mathtt{q} : y\,\langle \mathtt{U} \rangle) &= \{y\} \\ \mathtt{ch}(\mathtt{G} + \mathtt{G}') &= \mathtt{ch}(\mathtt{G}) \cup \mathtt{ch}(\mathtt{G}') \\ \mathtt{ch}(\mathtt{G} \mid \mathtt{G}') &= \mathtt{ch}(\mathtt{G}) \cup \mathtt{ch}(\mathtt{G}') \\ \mathtt{ch}(\mathtt{G}; \mathtt{G}') &= \mathtt{ch}(\mathtt{G}) \cup \mathtt{ch}(\mathtt{G}') \\ \mathtt{ch}(\mathtt{G}^{*^f}) &= \mathtt{ch}(\mathtt{G}) \cup \mathrm{cod}(f) \\ \mathtt{ch}(\mathtt{end}) &= \emptyset \end{aligned}$$

- The set of *participants* $\mathcal{P}(\mathtt{G})$ of $\mathtt{G}$ is defined as

$$\begin{aligned} \mathcal{P}(\mathtt{p} \to \mathtt{q} : y\,\langle \mathtt{U} \rangle) &= \{\mathtt{p}, \mathtt{q}\} \\ \mathcal{P}(\mathtt{G} + \mathtt{G}') = \mathcal{P}(\mathtt{G} \mid \mathtt{G}') = \mathcal{P}(\mathtt{G}; \mathtt{G}') &= \mathcal{P}(\mathtt{G}) \cup \mathcal{P}(\mathtt{G}') \\ \mathcal{P}(\mathtt{G}^{*^f}) &= \mathcal{P}(\mathtt{G}) \\ \mathcal{P}(\mathtt{end}) &= \emptyset \end{aligned}$$

- The set of enabled events $\mathtt{fst}(\mathtt{G})$ of $\mathtt{G}$ is defined as:

$$\begin{aligned} \mathtt{fst}(\mathtt{p} \to \mathtt{q} : y\,\langle \mathtt{U} \rangle) &= \{(\mathtt{p}, \overline{y}), (\mathtt{q}, y)\} \\ \mathtt{fst}(\mathtt{G} + \mathtt{G}') = \mathtt{fst}(\mathtt{G} \mid \mathtt{G}') &= \mathtt{fst}(\mathtt{G}) \cup \mathtt{fst}(\mathtt{G}') \\ \mathtt{fst}(\mathtt{G}; \mathtt{G}') &= \mathtt{fst}(\mathtt{G}) \cup \{(\mathtt{p}, y) \in \mathtt{fst}(\mathtt{G}') \mid \neg\exists(\mathtt{p}, z) \in \mathtt{fst}(\mathtt{G})\} \\ \mathtt{fst}(\mathtt{G}^{*^f}) &= \mathtt{fst}(\mathtt{G}) \\ \mathtt{fst}(\mathtt{end}) &= \emptyset \end{aligned}$$

- A participant $\mathtt{p}$ is *waiting* in $\mathtt{G}$ when its first enabled actions are only inputs; formally, $\mathtt{p}$ is *waiting in* $\mathtt{G}$ iff $\mathtt{fst}(\mathtt{G}) \cap (\{\mathtt{p}\} \times \overline{\mathbb{C}}) = \emptyset$ and $\mathtt{fst}(\mathtt{G}) \cap (\{\mathtt{p}\} \times \mathbb{C}) \neq \emptyset$. In (2.1) above, $\mathtt{q}$ and $\mathtt{s}$ are waiting $\mathtt{G}_\mathtt{f}$, while $\mathtt{p}$ is not.

Given a local type term $\mathtt{T}$, the set $\mathtt{ch}(\mathtt{T})$ is defined as follows:

$$\begin{gathered} \mathtt{ch}(\bigoplus_{i \in I} y_i!\mathtt{U}_i; \mathtt{T}_i) = \mathtt{ch}(\sum_{i \in I} y_i?\mathtt{U}_i; \mathtt{T}_i) = \{y_i \mid i \in I\} \cup \bigcup_{i \in I} \mathtt{ch}(\mathtt{T}_i) \\ \mathtt{ch}(\mathtt{T}_1; \mathtt{T}_2) = \mathtt{ch}(\mathtt{T}_1) \cup \mathtt{ch}(\mathtt{T}_2) \quad \mathtt{ch}(\mathtt{T}^*) = \mathtt{ch}(\mathtt{T}) \quad \mathtt{ch}(\mathtt{end}) = \emptyset \end{gathered}$$

Given an expression e (resp. a list ℓ), the set of *variables* occurring in e (resp. ℓ) are denoted by $\mathrm{var}(e)$ (resp., $\mathrm{var}(\ell)$) and it is defined by:

$$\begin{gathered} \mathrm{var}(x) = \{x\} \quad \mathrm{var}(\mathtt{v}) = \emptyset \quad \mathrm{var}(e_1\ \mathtt{op}\ e_2) = \mathrm{var}(e_1) \cup \mathrm{var}(e_2) \\ \mathrm{var}([e_1, \ldots, e_n]) = \bigcup_{i=1}^{n} \mathrm{var}(e_i) \quad \mathrm{var}(e_1..e_2) = \mathrm{var}(e_1) \cup \mathrm{var}(e_2) \end{gathered}$$

Given a system S, the free session names of S, written $\mathtt{fc}(S)$, are defined as:

$$
\begin{aligned}
&\mathtt{fc}(\sum_{i\in I} y_i(x_i); P_i) \;=\; \bigcup_{i\in I}(\{y_i\} \cup \mathtt{fc}(P_i))\\
&\mathtt{fc}(\overline{s}e) \;=\; \mathtt{fc}(s : M) \;=\; \{s\}\\
&\mathtt{fc}(\mathtt{if}\, e :\; P\, \mathtt{else}\, Q) \;=\; \mathtt{fc}(P; Q) \;=\; \mathtt{fc}(P) \cup \mathtt{fc}(Q)\\
&\mathtt{fc}(\mathtt{for}\, x \in \ell\, \mathtt{in}\, P : \,) \;=\; \mathtt{fc}(P)\\
&\mathtt{fc}(\mathtt{do}\, P\, \mathtt{until}\, b(x)) \;=\; \mathtt{fc}(P) \cup \{b\}\\
&\mathtt{fc}(\overline{u}^n(\mathbf{y}).P) \;=\; \mathtt{fc}(u_i(\mathbf{y}).P) \;=\; \{u\} \cup \mathtt{fc}(P) \setminus \mathbf{y}\\
&\mathtt{fc}((\nu \mathbf{s})S) \;=\; \mathtt{fc}(S) \setminus \mathbf{s}\\
&\mathtt{fc}(S \mid S') \;=\; \mathtt{fc}(S) \cup \mathtt{fc}(S')
\end{aligned}
$$

B Well-formedness

We adopt the usual well-formedness conditions on global types extending them to our framework. In particular, to the usual *linearity* Honda et al. [2008], *single threadness* and *single selector* Honda et al. [2008], and *knowledge of choice* Castagna et al. [2012], Honda et al. [2008] conditions, we add the conditions below specific to our form of iteration.

Definition 8 (Well-formed iteration). *A global type* $\mathcal{G}(\mathbf{y}) \stackrel{\triangle}{=} \mathtt{G}_0^{*^f}$ *is* well-formed *iff* $\mathcal{G}_0(\mathbf{y}) \stackrel{\triangle}{=} \mathtt{G}_0$ *is well-formed,* $\mathrm{cod}(f) \cap \mathtt{ch}(\mathtt{G}_0) = \emptyset$*, and if* $\mathtt{G}_0 \neq \mathtt{end}$ *then*

1. $\#\mathtt{rdy}(\mathtt{G}_0) = 1$ *and* $\mathrm{dom}(f) = \mathcal{P}(\mathtt{G}_0) \setminus \mathtt{rdy}(\mathtt{G}_0)$,
2. *for any two different subterms* $\mathtt{G}_1^{*^{f_1}}, \mathtt{G}_2^{*^{f_2}}$ *of* $\mathtt{G}_0$, $\mathrm{cod}(f_1) \cap \mathrm{cod}(f_2) = \emptyset$ *and* $\mathrm{dom}(f_1) = \mathrm{dom}(f_2) = \mathrm{dom}(f)$.

The single threadness condition requires parallel threads to have disjoint roles and channels, to prevent races on channels. The single selector and knowledge of choice conditions require that in each choice there is a unique participants selecting the branch to execute while all the others are made aware of the choice with suitable input actions.

Finally, the condition in Def. 8 is specific to our form of iteration; it requires a unique role to signal the termination of the iteration to any other role $\mathtt{q}$ by using the name $f(\mathtt{q})$. Also, in case of nested iterations, there is no confusion on the names used to signal the termination of each iteration.

C Typing rules for systems

The typing rules for systems in Fig. 9 extend those for processes, they are borrowed from Honda et al. [2008], and use non-singleton assignments as defined in § 2. Rule [$\mathtt{VPar}$] for parallel composition uses the composition of mappings given below. If Δ_1 and Δ_2 are compatible, their composition is

$$
(\Delta_2 \circ \Delta_2)(\mathbf{s}) = \begin{cases} \Delta_1(\mathbf{s}) & \text{if } \mathbf{s} \in \mathrm{dom}(\Delta_1) \setminus \mathrm{dom}(\Delta_2) \\ \Delta_2(\mathbf{s}) & \text{if } \mathbf{s} \in \mathrm{dom}(\Delta_2) \setminus \mathrm{dom}(\Delta_1) \\ \Delta_1(\mathbf{s}) \circ \Delta_2(\mathbf{s}) & \text{if } \mathbf{s} \in \mathrm{dom}(\Delta_1) \cap \mathrm{dom}(\Delta_2) \end{cases}
$$

$$\frac{\mathcal{C} \,\llcorner\, \Gamma \;\vdash\; S_1 \triangleright \Delta_1 \;\divideontimes\; \Gamma_1 \qquad \mathcal{C} \,\llcorner\, \Gamma \;\vdash\; S_2 \triangleright \Delta_2 \;\divideontimes\; \Gamma_2 \qquad \Delta_1 \text{ and } \Delta_2 \text{ compatible}}{\mathcal{C} \,\llcorner\, \Gamma \;\vdash\; S_1 \mid S_2 \triangleright \Delta_1 \circ \Delta_2 \;\divideontimes\; \Gamma_1, \Gamma_2} \;[\texttt{VPar}]$$

$$\frac{\mathcal{C} \,\llcorner\, \Gamma \;\vdash\; S \triangleright \Delta, \mathbf{s} : \{\mathcal{T}_{\mathtt{p}} @ \mathtt{p}\} \;\divideontimes\; \Gamma_1 \qquad \Gamma(u) \equiv \mathcal{G}(\mathbf{s}) \text{ and } \mathcal{G}\!\upharpoonright\!\mathtt{p} = \mathcal{T}_{\mathtt{p}}}{\mathcal{C} \,\llcorner\, \Gamma \;\vdash\; (\nu \mathbf{s}_u) S \triangleright \Delta \;\divideontimes\; \Gamma_1} \;[\texttt{VNews}]$$

$$\frac{\mathcal{C} \,\llcorner\, \Gamma \;\vdash\; s_i : M \triangleright \Delta, \mathbf{s} : \mathbb{M}[\cdot] @ p \;\divideontimes\; \Gamma_1}{\mathcal{C} \,\llcorner\, \Gamma \;\vdash\; s_i : \mathtt{v} \cdot M \triangleright \Delta, \mathbf{s} : s_i!\mathtt{v}; \mathbb{M}[\cdot] @ p \;\divideontimes\; \Gamma_1} \;[\texttt{VQueue}]$$

$$\mathcal{C} \,\llcorner\, \Gamma \;\vdash\; \mathbf{s} : \emptyset \triangleright \mathbf{s} : \{[\cdot] @ \mathtt{p}\}_{\mathtt{p} \in I} \;\divideontimes\; \Gamma_1 \;[\texttt{VEmpty}]$$

Fig. 9. Typing rules for systems

where, letting $\mathcal{T}_1 \circ \mathcal{T}_2 = \mathbb{M}[\mathcal{T}_2]$ if $\mathcal{T}_1$ is a message context $\mathbb{M}$, and $\mathcal{T}_1 \circ \mathcal{T}_2$ undefined otherwise, we stipulate that

$$\{\mathcal{T}_{\mathtt{p}} @ \mathtt{p}\}_{\mathtt{p} \in I} \circ \{\mathcal{T}'_{\mathtt{q}} @ \mathtt{q}\}_{\mathtt{q} \in J} = \{\mathcal{T}_{\mathtt{p}} @ \mathtt{p} \circ \mathcal{T}'_{\mathtt{p}} @ \mathtt{p}\}_{\mathtt{p} \in I \cap J} \\ \cup \{\mathcal{T}_{\mathtt{p}} @ \mathtt{p}\}_{\mathtt{i} \in I \setminus J} \cup \{\mathcal{T}_{\mathtt{q}} @ \mathtt{q}\}_{\mathtt{q} \in J \setminus I}$$

if $\mathcal{T}_{\mathtt{p}} \circ \mathcal{T}'_{\mathtt{p}}$ is defined for all $\mathtt{p} \in I \cap J$ and it is undefined otherwise. Note that $\mathrm{dom}(\Delta) = \mathrm{dom}(\Delta_1) \cup \mathrm{dom}(\Delta_2)$.

Rule [VNews] uses an annotation u to extend, in the premise, Δ with the correct mapping for session s, namely the projections of $\Gamma(u)$.

Rules [VQueue] and [VEmpty] are for queues.

D Soundness and Subject Reduction

The set of free channels of Δ is defined as

$$\mathtt{ch}(\Delta) = \cup_{\mathbf{s} \in \mathrm{dom}(\Delta)} \{\mathtt{ch}(\mathcal{T}) | \Delta(\mathbf{s}) = \mathcal{T} @ p\}$$

We write $\Gamma' \subseteq \Gamma$ if $\mathrm{dom}(\Gamma') = \mathrm{dom}(\Gamma)$ and $\forall x, u \in \mathrm{dom}(\Gamma')$, $\Gamma(u) = \Gamma'(u)$ (resp. $\Gamma(x) = \Gamma'(x)$).

Lemma 1 (Subject congruence). *If $\mathcal{C} \,\llcorner\, \Gamma \;\vdash\; S \triangleright \Delta \;\divideontimes\; \Gamma'$, $S \equiv S'$, and $\Delta \equiv \Delta'$, then $\mathcal{C} \,\llcorner\, \Gamma \;\vdash\; S' \triangleright \Delta' \;\divideontimes\; \Gamma'$.*

Lemma 2 (Substitution lemma on session channels). *If $\mathcal{C} \,\llcorner\, \Gamma \;\vdash\; S \triangleright \Delta \;\divideontimes\; \Gamma'$ and $\mathbf{s} \notin \mathtt{fc}(S) \cup \mathtt{ch}(\Delta)$ then $\mathcal{C} \,\llcorner\, \Gamma \;\vdash\; S\{\mathbf{s}/\mathbf{y}\} \triangleright \Delta\{\mathbf{s}/\mathbf{y}\} \;\divideontimes\; \Gamma'$ (with $\mathbf{y}$ vector of channels).*

Lemma 3. *If $\mathcal{C} \,\llcorner\, \Gamma \;\vdash\; \mathtt{for}\, x \,\mathtt{in}\, \ell \colon P \triangleright \Delta \;\divideontimes\; \Gamma'$ and $\ell \neq \emptyset$ then $\mathcal{C} \,\llcorner\, \Gamma \;\vdash\; \mathtt{for}\, x \,\mathtt{in}\, \mathtt{tl}(\ell) \colon P \triangleright \Delta \;\divideontimes\; \Gamma'$*

Lemma 4. *If $\mathcal{C} \,\llcorner\, \Gamma \;\vdash\; P \triangleright \Delta \;\divideontimes\; \Gamma'$ and $\Gamma \subseteq \Gamma'$ then $\mathcal{C} \,\llcorner\, \Gamma' \;\vdash\; P \triangleright \Delta \;\divideontimes\; \Gamma'$*

Proof. Observe that if $\mathcal{C} \,\llcorner\, \Gamma \;\vdash\; P \triangleright \Delta \;\divideontimes\; \Gamma'$ then $\mathtt{fv}(P) \subseteq \mathrm{dom}(\Gamma)$. Hence $\mathrm{dom}(\Gamma) \cap \mathrm{dom}(\Gamma')$ can only include names that are bound in P. In this case we can apply a renaming in P.

Lemma 5. *If* $\mathcal{C} \llcorner \Gamma \ \vdash \ S_1 \mid S_2 \triangleright \Delta_1 \circ \Delta_2 \circledast \Gamma'$ *then* $\mathtt{fv}(S_1) \cap \mathtt{fv}(S_2) = \emptyset$.

Lemma 6. *If* $\mathcal{C} \llcorner \Gamma \ \vdash \ S \triangleright \Delta \circledast \Gamma'$ *and* $\mathrm{var}(e) \cap \mathtt{fv}(S) = \emptyset$ *then* $\mathcal{C} \wedge e \llcorner \Gamma \ \vdash \ S \triangleright \Delta \circledast \Gamma'$

Theorem 5 (Subject reduction). *Let* $\mathcal{C} \llcorner \Gamma \ \vdash \ S \triangleright \Delta \circledast \Gamma'$ *and* σ *be a consistent store for* S *wrt* Γ *and* $\mathcal{C}$. *If* $\langle S, \sigma\rangle \xrightarrow{e \vdash \alpha} \langle S', \sigma'\rangle$ *then*

- *if* α *is an input* $s\mathtt{v}$, *then* $\Gamma \bullet \Delta \stackrel{s}{\Longrightarrow}$. *Furthermore, if* $\Gamma \bullet \Delta \stackrel{s\mathtt{v}}{\Longrightarrow} \Gamma \bullet \Delta'$ *with* $\mathtt{v} : \mathtt{U}$ *then* $\mathcal{C} \wedge e \llcorner \Gamma, x : \mathtt{U} \ \vdash \ S' \triangleright \Delta' \circledast \Gamma''$ *for some* x, $\Gamma'' \supseteq \Gamma$
- *otherwise* $\Gamma \bullet \Delta \stackrel{\alpha}{\Longrightarrow} \Gamma \bullet \Delta'$ *and* $\mathcal{C} \wedge e \llcorner \Gamma \ \vdash \ S' \triangleright \Delta' \circledast \Gamma'$

Proof. By induction on the proof (of transition rules for systems).

Base cases - [$\mathtt{SReq}/\mathtt{SAcc}$]. By transition rule [$\mathtt{SReq}$] S is of the form $\overline{u}^n(\mathbf{y}).Q$, $\alpha = \overline{u}^n\mathbf{s}$ and S' is of the form $Q\{\mathbf{s}/\mathbf{y}\}$. By hypothesis (and [$\mathtt{VReq}$]), for some $\mathcal{C}$, Δ, Γ and Γ':

$$\frac{\Gamma(u){=}\mathcal{G}(\mathbf{x}) \qquad \mathcal{C} \llcorner \Gamma \ \vdash \ Q \triangleright \Delta, \mathbf{y} : \mathcal{G}\{\mathbf{y}/\mathbf{x}\}\restriction 0 \circledast \Gamma'}{\mathcal{C} \llcorner \Gamma \ \vdash \ \overline{u}^n(\mathbf{y}).Q \triangleright \Delta \circledast \Gamma'}$$

By transition rule [$\mathtt{TReq}$], and since by premise of (D) $\Gamma(u) = \mathcal{G}(\mathbf{x})$,

$$\Gamma \bullet \Delta \stackrel{\overline{u}^n\mathbf{s}}{\to} \Gamma \bullet \Delta, \mathbf{s} : \mathcal{G}\{\mathbf{s}/\mathbf{x}\}\restriction 0$$

The thesis (i.e., $\mathcal{C} \llcorner \Gamma \ \vdash \ Q\{\mathbf{s}/\mathbf{y}\} \triangleright \Delta, \mathbf{y} : \mathcal{G}\{\mathbf{y}/\mathbf{x}\}\{\mathbf{s}/\mathbf{y}\}\restriction 0 \circledast \Gamma'$) follows from the second premise of (D) applying Lemma 2 (substituting $\mathbf{y}$ with $\mathbf{s}$). The case for [$\mathtt{SAcc}$] is similar.

Base case - [$\mathtt{SSend}$]. By transition rule [$\mathtt{SSsend}$], S is of the form $\overline{s}e$, $\alpha = \overline{s}\mathtt{v}$, and S' is 0, and $e \downarrow \sigma = \mathtt{v}$. By hypothesis (and [$\mathtt{VSend}$])

$$\frac{\Gamma(e){=}\mathtt{U} \qquad s \in \mathbf{s}}{\mathcal{C} \llcorner \Gamma \ \vdash \ \overline{s}e \triangleright \mathbf{s} : s!\mathtt{U} \circledast \Gamma}$$

By consistency of σ with respect to Γ and since $\Gamma \vdash e : \mathtt{U}$ we have $\mathtt{v} : \mathtt{U}$. Hence $\Gamma \bullet \mathbf{s} : s!\mathtt{U}; \mathsf{end} \stackrel{\overline{s}\mathtt{v}}{\to} \Gamma \bullet \mathbf{s} : \mathsf{end}$ by transition rule [$\mathtt{TSend}$]. The thesis (i.e., $\mathcal{C} \llcorner \Gamma \ \vdash \ 0 \triangleright \mathbf{s} : \mathtt{end} \circledast \Gamma$) follows by typing rule [$\mathtt{VEnd}$] (explicitly writing the trailing occurrences of the idle system/type, i.e., $\overline{s}e; \mathbf{0}$ and $\mathbf{s} : s!\mathtt{U}; \mathtt{end}$).

Base case - [$\mathtt{SRec}$]. By transition rule [$\mathtt{SRec}$], S is of the form $s(x).Q + \mathcal{N}$, $\alpha = s\mathtt{v}$, $S' = Q$, and $\sigma' = \sigma[\mathtt{v}/x]$. By hypothesis S can be validated, and it can be only by using rule [$\mathtt{VRec}$]. Hence S we can written as $\sum_{i \in I} s_i(x_i); P_i$ with $s = s_j$, $x = x_j$, $P_i = Q$ for $j \in I$.

$$\frac{\forall i \in I \qquad s_i \in \mathbf{s} \qquad \mathcal{C} \llcorner \Gamma, x_i : \mathtt{U}_i \ \vdash \ P_i \triangleright \Delta, \mathbf{s} : \mathcal{T}_i \circledast \Gamma_i}{\mathcal{C} \llcorner \Gamma \ \vdash \ \sum_{i \in I} s_i(x_i); P_i \triangleright \Delta, \mathbf{s} : \sum_{i \in I} s_i?\mathtt{U}_i; \mathcal{T}_i \circledast \bigcap_i \Gamma_i}$$

By using rules [TPar2] and [TRec] in the premise $\Gamma \bullet \Delta, \mathbf{s} : \sum_{i\in I} s_i?\mathtt{U}_i; \mathcal{T}_i$ can make a corresponding input transition on channel s_j. Furthermore, if $\mathtt{v} : \mathtt{U}_j$ then: $\Gamma \bullet \Delta, \mathbf{s} : \sum_{i\in I} s_i?\mathtt{v}; \mathcal{T}_i \xrightarrow{\bar{s}\mathtt{v}} \Gamma \bullet \Delta, \mathbf{s} : \mathcal{T}_j$. The thesis (i.e., $\mathcal{C} \llcorner \Gamma, x : \mathtt{U} \ \vdash \ Q \triangleright \bowtie_{i\in I} \Delta, \mathbf{s} : \mathcal{T}_j \divideontimes \Gamma_j$) follows by the premise of (D) since $\Gamma_i \supseteq \bigcap_i \Gamma_i$.

Case - [SLoop1]. By transition rule [SLoop1], S is of the form $\mathtt{do}\ P\ \mathtt{until}\ b(x)$, and $S' = \mathbf{0}$ with $\alpha = b\mathtt{v}$. By hypothesis

$$\frac{\mathcal{C} \llcorner \Gamma \ \vdash \ N \triangleright \mathbf{s} : \mathcal{T} \divideontimes \Gamma'}{\mathcal{C} \llcorner \Gamma \ \vdash \ \mathtt{do}\ N\ \mathtt{until}\ b(x) \triangleright \mathbf{s} : \mathcal{T}^*; b?U \divideontimes \Gamma', x : \mathtt{U}}$$

By using rules [TSeq] and [TLoop] in the premise we have $\Gamma \bullet \mathbf{s} : \mathcal{T}^*; b?\mathtt{U} \xrightarrow{\tau} \Gamma \bullet \mathbf{s} : \mathtt{end}; b?\mathtt{U} \equiv \Gamma \bullet \mathbf{s} : b?\mathtt{U}$. By transition rule [TRec] we then have $\Gamma \bullet \mathbf{s} : b?\mathtt{U} \xrightarrow{b\mathtt{v}} \Gamma \bullet \mathbf{s} : \mathtt{end}$. The thesis follows by typing rule [VEnd] (i.e., $\mathcal{C} \llcorner \Gamma \ \vdash \ \mathbf{0} \triangleright \mathbf{s} : \mathtt{end} \divideontimes \Gamma'$).

Case - [SFor1] By transition rule[SFor1]: $S = \mathtt{for}\ x\ \mathtt{in}\ \ell \colon P$, $\ell \downarrow \sigma = \epsilon$, $S' = \mathbf{0}$ and $\alpha = \tau$. By hypothesis

$$\frac{\Gamma(\ell) {=} [\mathtt{U}] \qquad \mathcal{C} \wedge x \in \ell \llcorner \Gamma, x : \mathtt{U} \ \vdash \ P \triangleright \Delta, \mathbf{s} : \mathcal{T} \divideontimes \Gamma'}{\mathcal{C} \llcorner \Gamma \ \vdash \ \mathtt{for}\ x\ \mathtt{in}\ \ell \colon P \triangleright \Delta, \mathbf{s} : \mathcal{T}^* \divideontimes \Gamma'}$$

We have $\Gamma \bullet \Delta, \mathbf{s} : \mathcal{T}^* \xrightarrow{\tau} \Gamma \bullet \Delta, \mathbf{s} : \mathtt{end}$. By typing rule [VEnd]:

$$\mathcal{C} \llcorner \Gamma \ \vdash \ \mathbf{0} \triangleright \Delta, \mathbf{s} : \mathtt{end} \divideontimes \Gamma'$$

as required.

Case - [SInit] For this case we will use the runtime rules. By transition rule[SInit]: $S = \overline{u}^n(\mathbf{y}_0).P_0 \mid u_1(\mathbf{y}_1).P_1 \mid \ldots \mid u_n(\mathbf{y}_n).P_n$, $S' = (\nu s)(P_0\{\mathbf{s}/\mathbf{y}_0\} \mid P_1\{\mathbf{s}/\mathbf{y}_1\} \mid \ldots \mid P_n\{\mathbf{s}/\mathbf{y}_n\} \mid \mathbf{s} : \emptyset)$, and $\alpha = \tau$. By hypothesis (and rule [VPar2]):

$$\frac{\mathcal{C} \llcorner \Gamma \ \vdash \ \overline{u}^n(\mathbf{y}_0).P_0 \triangleright \Delta_0 \divideontimes \Gamma' \qquad \mathcal{C} \llcorner \Gamma \ \vdash \ u_n(\mathbf{y}_i).P_i \triangleright \Delta_i \divideontimes \Gamma'}{\mathcal{C} \llcorner \Gamma \ \vdash \ \overline{u}^n(\mathbf{y}_0).P_0 \mid u_1(\mathbf{y}_1).P_1 \mid \ldots \mid u_n(\mathbf{y}_n).P_n \triangleright \Delta_0 \circ \Delta_1 \circ .. \circ \Delta_n \divideontimes \Gamma'}$$

By induction each $u_i(\mathbf{y}_n).P_i$ (same for session request) moves to $P_i\{\mathbf{s}/\mathbf{y}_i\}$ and

$$\mathcal{C} \llcorner \Gamma \ \vdash \ u_i(\mathbf{y}_i).P_i \triangleright \Delta_i, \boldsymbol{s} : \mathcal{T} @ i \divideontimes \Gamma'$$

And, by [VEmpty]:

$$\mathcal{C} \llcorner \Gamma \ \vdash \ \mathbf{s} : \emptyset \triangleright \boldsymbol{s} : \{[\cdot] @ i\}_{\mathtt{i}\in\{0,..n\}} \divideontimes \Gamma'$$

By [TInit] we have $\Gamma \bullet \Delta \xrightarrow{\tau} \Gamma \bullet \Delta, \mathbf{s} : \{\mathcal{T}_i @ i\}$.

Since $\Delta, \mathbf{s} : \mathcal{T}_0 @ 0 \circ \mathbf{s} : \mathcal{T}_1 @ 1 \circ \ldots \mathbf{s} : \mathcal{T}_n @ n \circ \boldsymbol{s} : \{[\cdot] @ i\}_{\mathtt{i}\in\{0,..n\}}$ is (by definition of $\circ$) $\Delta, \boldsymbol{s} : \{\mathcal{T}_i @ i\}_{\mathtt{i}\in\{0,..n\}}$ we can use (D) and (D) as premises for the validation rule [VNews] obtaining the thesis:

$$\mathcal{C} \llcorner \Gamma \ \vdash \ (\nu s)(P_0\{\mathbf{s}/\mathbf{y}_0\} \mid P_1\{\mathbf{s}/\mathbf{y}_1\} \mid \ldots \mid P_n\{\mathbf{s}/\mathbf{y}_n\} \mid \mathbf{s} : \emptyset) \triangleright \Delta, \boldsymbol{s} : \{\mathcal{T}_i @ i\}_{\mathtt{i}\in\{0,..n\}} \divideontimes \Gamma'$$

Case - $[\mathtt{SCom_1}/\mathtt{SCom_2}]$ By transition rule [SCom1], S is of the form $P \mid s : M$, $S' = P \mid M \cdot \mathtt{v}$, $\alpha = \tau$, and by the premise of [SCom1]:

$$\langle P, \sigma \rangle \xrightarrow{\bar{s}\mathtt{v}} \langle P', \sigma' \rangle$$

By hypothesis (proof rule[VPar2], noticing that $\mathbb{M}[\mathcal{T}]@p = \mathcal{T}@p \circ \mathbb{M}[\cdot]@p$)

$$\frac{\mathcal{C} \llcorner \Gamma \ \vdash \ P \triangleright \Delta, \mathbf{s} : \mathcal{T}@p \ \divideontimes \ \Gamma' \qquad \mathcal{C} \llcorner \Gamma \ \vdash \ s : M \triangleright \mathbf{s} : \mathbb{M}[\cdot]@p \ \divideontimes \ \Gamma'}{\mathcal{C} \llcorner \Gamma \ \vdash \ P \mid s : M \triangleright \Delta, \mathbf{s} : \mathbb{M}[\mathcal{T}]@p \ \divideontimes \ \Gamma'}$$

From the first premise of (D) and (D), by induction

$$\Gamma \bullet \Delta, \mathbf{s} : \mathcal{T} \xrightarrow{\bar{s}\mathtt{v}} \Gamma \bullet \Delta, \mathbf{s} : \mathcal{T}'$$

and

$$\mathcal{C} \llcorner \Gamma \ \vdash \ P' \triangleright \Delta, \mathbf{s} : \mathcal{T}'@\mathtt{p} \ \divideontimes \ \Gamma'$$

Using (D) as premise for $[\mathtt{TCom_1}]$, which is in turn used as a premise for [TPar]:

$$\Gamma \bullet \Delta, \mathbf{s} : \mathbb{M}[\mathcal{T}]@p \xrightarrow{\tau} \Gamma \bullet \Delta, \mathbf{s} : \mathbb{M}[\mathcal{T}']@p$$

By the second premise of (D), used as a premise for proof rule [VQueue] we obtain

$$\mathcal{C} \llcorner \Gamma \ \vdash \ s : M \cdot \mathtt{v} \triangleright \mathbf{s} : \mathbb{M}[\cdot]@p \ \divideontimes \ \Gamma'$$

The thesis

$$\frac{\mathcal{C} \llcorner \Gamma \ \vdash \ P' \triangleright \Delta, \mathbf{s} : \mathcal{T}'@p \ \divideontimes \ \Gamma' \qquad \mathcal{C} \llcorner \Gamma \ \vdash \ s : M \cdot \mathtt{v} \triangleright \mathbf{s} : \mathbb{M}[\cdot]@p \ \divideontimes \ \Gamma'}{\mathcal{C} \llcorner \Gamma \ \vdash \ P \mid s : M \cdot \mathtt{v} \triangleright \Delta, \mathbf{s} : \mathbb{M}[\mathcal{T}']@p \ \divideontimes \ \Gamma'}$$

holds by proof rule [VPar] where the first premise of [VPar] holds by (D) and the second premise holds by (D). The case for $[\mathtt{SCom_2}]$ is similar.

Case - [SThen/Selse] By transition rule[SThen]: $S = \mathtt{if}\ e :\ P\ \mathtt{else}\ Q$, $S' = P'$, $\alpha = e \vdash \alpha'$, and $e \downarrow \sigma = \mathtt{true}$. By hypothesis (it can be by either [VThen] or [VIf], we consider the latter case, the case for [VThen] is similar):

$$\frac{\Gamma(e){=}\mathtt{bool} \quad \mathcal{C} \wedge e \not\vdash \bot \quad \mathcal{C} \wedge \neg e \not\vdash \bot \\ \mathcal{C} \wedge e \llcorner \Gamma \ \vdash \ P \triangleright \Delta_1 \ \divideontimes \ \Gamma_1 \quad \mathcal{C} \wedge \neg e \llcorner \Gamma \ \vdash \ Q \triangleright \Delta_2 \ \divideontimes \ \Gamma_1}{\mathcal{C} \llcorner \Gamma \ \vdash \ \mathtt{if}\ e :\ P\ \mathtt{else}\ Q \triangleright \Delta_1 \bowtie \Delta_2 \ \divideontimes \ \Gamma_1 \cap \Gamma_2}$$

By induction, from the first premise of (D) and since by premise of [SThen], $P \xrightarrow{e' \vdash \alpha} P'$ we have

$$\Gamma \bullet \Delta_1 \xrightarrow{\alpha} \Gamma \bullet \Delta_1'$$

and

$$\mathcal{C} \wedge e \llcorner \Gamma \ \vdash \ P' \triangleright \Delta_1' \ \divideontimes \ \Gamma_1 \cap \Gamma_2$$

By (D) and definition of $\Delta_1 \bowtie \Delta_2$

$$\Gamma \bullet \Delta_1 \bowtie \Delta_2 \xrightarrow{\alpha} \Gamma \bullet \Delta_1'$$

hence the thesis is straightforward by (D). The case for [`SElse`] is similar.

Case - [`SFor2`] By transition rule[`SFor2`]: $S = \texttt{for}\ x\ \texttt{in}\ \ell\colon P$ and $S' = P'; \texttt{for}\ x\ \texttt{in}\ \texttt{tl}(\ell\downarrow)\colon P$:

$$\frac{\neg \ell \downarrow = \varepsilon \qquad P\{\texttt{hd}(\ell\downarrow)/x\} \xrightarrow{\alpha} P' \qquad \alpha \neq b}{\texttt{for}\ x\ \texttt{in}\ \ell\colon P \xrightarrow{\alpha} P'; \texttt{for}\ x\ \texttt{in}\ \texttt{tl}(\ell\downarrow)\colon P}$$

Also by hypothesis:

$$\frac{\Gamma(\ell) = [\texttt{U}] \qquad \mathcal{C} \wedge x \in \ell \mathbin{\lrcorner} \Gamma, x : \texttt{U} \ \vdash \ P \triangleright \Delta, \mathtt{s} : \mathcal{T} \ \divideontimes \ \Gamma'}{\mathcal{C} \mathbin{\lrcorner} \Gamma \ \vdash \ \texttt{for}\ x\ \texttt{in}\ \ell\colon P \triangleright \Delta, \mathtt{s} : \mathcal{T}^* \ \divideontimes \ \Gamma'}$$

By the second premise of (D) and the substitution lemma:

$$\mathcal{C} \wedge x \in \ell \mathbin{\lrcorner} \Gamma, x : \texttt{U} \ \vdash \ P\{\texttt{hd}(\ell\downarrow)/x\} \triangleright \Delta, \mathtt{s} : \mathcal{T} \ \divideontimes \ \Gamma'$$

By induction, the second premise of (D) and (D) imply

$$\Gamma \bullet \Delta, \mathtt{s} : \mathcal{T} \xrightarrow{\alpha} \Gamma \bullet \Delta, \mathtt{s} : \mathcal{T}'$$

and for some $\mathcal{C}'$ and Γ':

$$\mathcal{C}' \mathbin{\lrcorner} \Gamma' \ \vdash \ P' \triangleright \Delta, \mathtt{s} : \mathcal{T}' \ \divideontimes \ \Gamma'$$

By transition rule [`TLoop2`]

$$\Gamma \bullet \Delta, \mathtt{s} : \mathcal{T}^* \xrightarrow{\tau} \Gamma \bullet \Delta, \mathtt{s} : \mathcal{T}; \mathcal{T}^*$$

which, by transition rule [`TSeq`] with premise (D) gives

$$\Gamma \bullet \Delta, \mathtt{s} : \mathcal{T}; \mathcal{T}^* \xrightarrow{\alpha} \Gamma \bullet \Delta, \mathtt{s} : \mathcal{T}'; \mathcal{T}^*$$

By (D) and Lemma 3

$$\mathcal{C} \mathbin{\lrcorner} \Gamma \ \vdash \ \texttt{for}\ x\ \texttt{in}\ \texttt{tl}(\ell)\colon P \triangleright \Delta, \mathtt{s} : \mathcal{T}^* \ \divideontimes \ \Gamma'$$

Finally, by using (D) and (D) as premise for typing rule [`VFor`]

$$\mathcal{C} \mathbin{\lrcorner} \Gamma \ \vdash \ P'; \texttt{for}\ x\ \texttt{in}\ \texttt{tl}(\ell)\colon P \triangleright \Delta, \mathtt{s} : \mathcal{T}'; \mathcal{T}^* \ \divideontimes \ \Gamma'$$

By (D) and (D) we have the thesis. The case for [`SLoop2`] is similar.

Case - [PSeq] By transition rule [SSeq] S is of the form $P;Q$, and S' is of the form $P';Q$. Furthermore $P \xrightarrow{e\vdash\alpha} P'$. Observe that P appears before Q in a sequential composition. Since we assume that processes do not have a conditional statement as a prefix of a sequential composition, we have $e = \texttt{true}$. By hypothesis

$$\frac{\mathcal{C} \mathbin{\lrcorner} \Gamma \;\vdash\; P_1 \triangleright \Delta_1 \;\circledast\; \Gamma_1 \qquad \mathcal{C} \mathbin{\lrcorner} \Gamma_1 \;\vdash\; P_2 \triangleright \Delta_1 \;\circledast\; \Gamma_2}{\mathcal{C} \mathbin{\lrcorner} \Gamma \;\vdash\; P_1;P_2 \triangleright \Delta_1;\Delta_2 \;\circledast\; \Gamma_2}$$

If α is not an input action, by induction

$$\Gamma \bullet \Delta_1 \xrightarrow{\alpha} \Gamma \bullet \Delta_1'$$

and by transition rule [TSeq] and (D):

$$\Gamma \bullet \Delta_1;\Delta_2 \xrightarrow{\alpha} \Gamma \bullet \Delta_1';\Delta_2$$

with

$$\mathcal{C} \wedge e \mathbin{\lrcorner} \Gamma \;\vdash\; P' \triangleright \Delta_1' \;\circledast\; \Gamma_1 \qquad (e = \texttt{true})$$

The thesis

$$\mathcal{C} \wedge e \mathbin{\lrcorner} \Gamma \;\vdash\; P_1';P_2 \triangleright \Delta_1';\Delta_2 \;\circledast\; \Gamma_2$$

follow from typing rule [VSeq], using (D) as first premise, and the second premise of (D) as second premise where the assumption context can be extended with e since $e = \texttt{true}$.

If α is an input action, by induction we still have (D) and $\Gamma \bullet \Delta_1$ is ready to make a corresponding input action. If $\Gamma \bullet \Delta_1 \xrightarrow{\alpha} \Gamma \bullet \Delta_1'$ then for some x, $\texttt{U}$, $\Gamma_1' \supseteq \Gamma_1$

$$\mathcal{C} \wedge e \mathbin{\lrcorner} \Gamma, x : \texttt{U} \;\vdash\; P' \triangleright \Delta_1' \;\circledast\; \Gamma_1' \qquad (e = \texttt{true})$$

By the second premise of (D) and Lemma 4 observing that $\Gamma_1' \supseteq \Gamma_1$

$$\mathcal{C} \mathbin{\lrcorner} \Gamma_1' \;\vdash\; P_2 \triangleright \Delta_1 \;\circledast\; \Gamma_2$$

The thesis follows by the fact that if $\Gamma \bullet \Delta_1$ is ready to make a corresponding input action then by [TSeq] also $\Gamma \bullet \Delta_1;\Delta_2$ is and if $\Gamma \bullet \Delta_1;\Delta_2 \xrightarrow{\alpha} \Gamma \bullet \Delta_1';\Delta_2$ then

$$\frac{\mathcal{C} \wedge e \mathbin{\lrcorner} \Gamma, x : \texttt{U} \;\vdash\; P_1' \triangleright \Delta_1 \;\circledast\; \Gamma_1' \qquad \mathcal{C} \wedge e \mathbin{\lrcorner} \Gamma_1' \;\vdash\; P_2 \triangleright \Delta_1 \;\circledast\; \Gamma_2}{\mathcal{C} \wedge e \mathbin{\lrcorner} \Gamma \;\vdash\; P_1';P_2 \triangleright \Delta_1;\Delta_2 \;\circledast\; \Gamma_2}$$

follows by applying [VSeq] with premises (D) and (D) where the assumption context can be extended with e because $e = \texttt{true}$.

Case - [SPar] By transition rule Par S is of the form $S_1 \mid S_2$, and S' is of the form $S_1' \mid S_2$. Furthermore, $S_1 \xrightarrow{e\vdash\alpha} S_1'$. By hypothesis

$$\frac{\mathcal{C} \mathbin{\lrcorner} \Gamma \;\vdash\; S_1 \triangleright \Delta_1 \;\circledast\; \Gamma_1 \qquad \mathcal{C} \mathbin{\lrcorner} \Gamma \;\vdash\; S_2 \triangleright \Delta_2 \;\circledast\; \Gamma_2}{\mathcal{C} \mathbin{\lrcorner} \Gamma \;\vdash\; S_1 \mid S_2 \triangleright \Delta_1 \circ \Delta_2 \;\circledast\; \Gamma_1, \Gamma_2}$$

By induction, if α is not input, from the first premise of (D) and from $S_1 \xrightarrow{e \vdash \alpha} S_1'$ we have $\Gamma \bullet \Delta_1 \xrightarrow{\alpha} \Gamma \bullet \Delta_1'$ and

$$\mathcal{C} \wedge e \mathbin{\lrcorner} \Gamma \ \vdash \ S_1' \triangleright \Delta_1' \ \divideontimes \ \Gamma_1$$

By Lemma 5 $\mathtt{fv}(S_1) \cap \mathtt{fv}(S_2) = \emptyset$.
$\mathtt{fv}(e) \cap \mathtt{fv}(S_2) = \emptyset$ because.
From the second premise of (D) and by Lemma 6

$$\mathcal{C} \wedge e \mathbin{\lrcorner} \Gamma \ \vdash \ S_2 \triangleright \Delta_2 \ \divideontimes \ \Gamma_1$$

The thesis follow from typing rule [VPar], using (D) as first premise, and the second premise of (D) as second premise.

Case - [SNews] By transition rule [SNews], S is of the form $(\nu \mathbf{s})S_1$, and $S' = (\nu \mathbf{s})S_1'$ with

$$\mathtt{s} \notin \mathtt{fc}(S_1)$$

By hypothesis

$$\frac{\mathcal{C} \mathbin{\lrcorner} \Gamma \ \vdash \ S_1 \triangleright \Delta, \mathtt{s} : \{\mathcal{T}_{\mathtt{p}} @ \mathtt{p}\} \ \divideontimes \ \Gamma_1 \qquad \exists a \in \Gamma . \Gamma(a) = \mathcal{G} \text{ and } \mathcal{G} \restriction \mathtt{p} = \mathcal{T}_{\mathtt{p}}}{\mathcal{C} \mathbin{\lrcorner} \Gamma \ \vdash \ (\nu \mathbf{s})S_1 \triangleright \Delta \ \divideontimes \ \Gamma_1}$$

By induction $\Gamma \bullet \Delta, \mathtt{s} : \{\mathcal{T} @ \mathtt{p}\}$ can make a step α and by (D) and applying the transition rule [TPar1] for runtime specifications (Fig. 7), $\Gamma \bullet \Delta, \mathtt{s} : \{\mathcal{T} @ \mathtt{p}\}$ moves to a specification of the form

$$\Gamma \bullet \Delta', \mathtt{s} : \{\mathcal{T} @ \mathtt{p}\}$$

and by induction

$$\mathcal{C} \mathbin{\lrcorner} \Gamma \ \vdash \ S_1' \triangleright \Delta', \mathtt{s} : \{\mathcal{T}_{\mathtt{p}} @ \mathtt{p}\} \ \divideontimes \ \Gamma_1$$

Using (D) as premise of typing rule [VNews] we obtain

$$\mathcal{C} \mathbin{\lrcorner} \Gamma \ \vdash \ (\nu \mathbf{s})S_1' \triangleright \Delta' \ \divideontimes \ \Gamma_1$$

as required.

Soundness follows by straightforward coinduction from Thm. 5.

Corollary 2 (Soundness). *If $\mathcal{C} \mathbin{\lrcorner} \Gamma \ \vdash \ S \triangleright \Delta \ \divideontimes \ \Gamma'$ then $\langle S, \sigma \rangle \precsim \Gamma \bullet \Delta$ for all σ consistent store for S wrt Γ and $\mathcal{C}$.*

Proof. Soundness follows from showing that

$$\mathbb{R} = \{(\langle S, \sigma \rangle, \Gamma \bullet \Delta) \ \big| \ \mathcal{C} \mathbin{\lrcorner} \Gamma \ \vdash \ S \triangleright \Delta \divideontimes \Gamma' \text{ and } \sigma \text{ is a consistent store for } S \text{ wrt } \Gamma \text{ and } \mathcal{C}\}$$

is a conditional simulation, which is straightforward from Thm. 2.

E WSI by typing

The next two results show that the denotational semantics of runtime types coincides with the operational rules given in Fig. 1 and 7.

Lemma 7. *$\Gamma \bullet \Delta \stackrel{\tau}{\Longrightarrow} \Gamma \bullet \Delta'$ then there exists $r \in \mathcal{R}_{\mathbf{s}}(\Delta)$ and $s_k \in \mathtt{s}$ such that $r = \langle \mathtt{p}, s_k!\mathtt{U}_k\rangle\langle \mathtt{q}, s_k?\mathtt{U_k}\rangle r'$ with $r' \in \mathcal{R}_{\mathbf{s}}(\Delta')$.*

Proof. By induction on the proof of the derivation. Then, we proceed by case analysis on the last applied rule. Note that none of the followings [TQueue], [Tcom1], [TInit], [TReq], [TAcc], [TSend], [TRec], [TSeq], [TPar], [TLoop1] and [TLoop2]. Case [TCom2] is as follows.

$$\frac{\Gamma \bullet \mathtt{s} : \mathcal{T} @ \mathtt{q} \stackrel{s\mathtt{v}}{\to} \Gamma \bullet \mathtt{s} : \mathcal{T}' @ \mathtt{q}}{\Gamma \bullet \mathtt{s} : s!\mathtt{v}; \mathbb{M}_1 @ \mathtt{p},\ \mathbb{M}_2[\mathcal{T}] @ \mathtt{q} \stackrel{\tau}{\to} \Gamma \bullet \mathtt{s} : \mathbb{M}_1 @ \mathtt{p},\ \mathbb{M}_2[\mathcal{T}'] @ \mathtt{q}}\,[\mathtt{TCom2}]$$

It is easy to check (by proof induction) that $\Gamma \bullet \mathtt{s} : \mathcal{T} @ \mathtt{q} \stackrel{s\mathtt{v}}{\to} \Gamma \bullet \mathtt{s} : \mathcal{T}' @ \mathtt{q}$ implies either

1. $\mathcal{T} = \mathbb{M}[\mathcal{T}_0; \mathcal{T}_1]$ with $\mathcal{T}_0 = \sum_{i \in I} s_i?\mathtt{U}_i; \mathcal{T}_i$, for some $k \in I : s = s_k$, and $\mathcal{T}' = \mathbb{M}[\mathcal{T}_k; \mathcal{T}_1]$;
2. $\mathcal{T} = \mathbb{M}[\mathcal{T}_0^*; \mathcal{T}_1]$ with $\mathcal{T}_1 = \sum_{i \in I} s_i?\mathtt{U}_i; \mathcal{T}_i$, for some $k \in I : s = s_k$, and $\mathcal{T}' = \mathbb{M}[\mathcal{T}_k; \mathcal{T}_1]$;
3. $\mathcal{T} = \mathbb{M}[\mathcal{T}_0^*; \mathcal{T}_1]$ with $\mathcal{T}_0 = \sum_{i \in I} s_i?\mathtt{U}_i; \mathcal{T}_i$, for some $k \in I : s = s_k$, and $\mathcal{T}' = \mathbb{M}[\mathcal{T}_k; \mathcal{T}_0^*; \mathcal{T}_1]$

Case (1) follows by using rule [RTCom]. Case (2) follows by using first rule [RTIt1] first and then [RTCom] while case (3) follows by using [RTIt2] instead. The proof is completed by noting that all τ reductions that precedes the communication are either initiation of sessions (that do not interfere with the runs) or addition of messages to the queues (that are mimicked by rule [RTCh]. (Note that we can reason analogously to consider reductions after the communication step.)

Lemma 8. *Let $r \in \mathcal{R}_{\mathbf{s}}(\Delta)$ then either (i) $\Gamma \bullet \Delta(\mathtt{s}) \not\to$ or (ii) $\Gamma \bullet \Delta \stackrel{\tau}{\Longrightarrow} \Gamma \bullet \Delta'$, $r = \langle \mathtt{p}, s_k!\mathtt{U}_k\rangle\langle \mathtt{q}, s_k?\mathtt{U_k}\rangle r'$, and $r' \in \mathcal{R}_{\mathbf{s}}(\Delta')$.*

Proof. The proof follows by induction on the proof of the derivation. By straightforward analysis of the last applied rule. The only interesting case is the usage of rule [RTCh] which is mimicked by using rule [TCom1]. We notice here that the traces we consider are generated by keeping outputs in ordered fashion.

We start by stating several auxiliary results about the runs of Runtime Local Types, that will be used for proving main results.

Lemma 9. *Given* $r \in \mathcal{R}_{\mathbf{s}}(\mathbf{s} : \sum_{i \in I} s_i?\mathtt{U}_i;\mathtt{T}_i @\, \mathtt{p}, \mathbb{T})$ *and* $s \in \mathbf{s}$ *and* $s \neq s_i$ *for all* $i \in I$*, then* $r \in \mathcal{R}_{\mathbf{s}}(\mathbf{s} : s : \mathtt{U}.\mathtt{T}' + \sum_{i \in I} s_i?\mathtt{U}_i;\mathtt{T}_i @\, \mathtt{p}, \mathbb{T})$.

Proof. By straightforward induction on the structure of the proof.

It is easy to notice that the converse of the above result does not hold since the new branch can add new traces. Next result characterises some cases in which the other inclusion is also valid. In order to achieve other direction we impose a restriction on the names that can be used as guard of the added branch. This condition mimics the well-formedness condition for the choice operator of global types.

Lemma 10. *Let* $\Delta = \mathbf{s} : \sum_{i \in I} s_i?\mathtt{U}_i;\mathtt{T}_i @\, \mathtt{q}, \{\mathtt{T}_j @\, p_j\}_{j \in J}$ *and* s *s.t.*

1. $\forall i \in I : s \neq s_i$ *and*
2. $\nexists r \in \mathcal{R}_{\mathbf{s}}(\Delta)$ *with* $r = r_1 r_2$ *and* $r_2 \in \mathcal{R}_{\mathbf{s}}(\mathbb{M}[\mathtt{T} \oplus s!\mathtt{U};\mathtt{T}'] @\, \mathtt{p}, \sum_{i \in I} s_i?\mathtt{U}_i;\mathtt{T}_i @\, \mathtt{q}, \mathbb{T})$

Then, $\mathcal{R}_{\mathbf{s}}(\Delta) = \mathbf{s} :\ s?\mathtt{U};\mathtt{T}'' + \sum_{i \in I} s_i?\mathtt{U}_i;\mathtt{T}_i @\, \mathtt{q}, \{\mathtt{T}_j @\, p_j\}_{j \in J}$

Proof. ($\subseteq$) By Lemma 9. ($\supseteq$) By induction on the structure of the proof. Interesting case is when rule [RTCom] is applied, but the case follows because condition (2) avoids the selection of the added branch.

Next result show that projections of a well-formed global type are confusion-free w.r.t. the choices.

Lemma 11. *Let* $\mathcal{G}(\mathbf{s}) = \mathtt{G}_1 + \mathtt{G}_2$ *s.t.* $\mathtt{rdy}(\mathtt{G}) = \{\mathtt{p}\}$, $\Delta = \mathbf{s} : \{\mathcal{G}(\mathbf{s}) \upharpoonright \mathtt{p} @\, p\}_{\mathtt{p} \in \mathcal{P}(\mathcal{G}(\mathbf{s}))}$, $\Delta_i = \mathbf{s} : \{\mathcal{G}_i(\mathbf{s}) \upharpoonright \mathtt{p} @\, p\}_{\mathtt{p} \in \mathcal{P}(\mathcal{G}(\mathbf{s}))}$ *for* $i = 1, 2$*. Then,* $\mathcal{R}_{\mathbf{s}}(\Delta) = \mathcal{R}_{\mathbf{s}}(\Delta_1) \cup \mathcal{R}_{\mathbf{s}}(\Delta_2)$

Proof. If one of the branch $\mathtt{G}_i$ is end we are done. Otherwise, by straightforward induction on the structure of $\mathtt{G}$ we can conclude that $\mathcal{G}(\mathbf{s}) \upharpoonright \mathtt{p} = \mathcal{G}_1 \upharpoonright \mathtt{p} \oplus \mathcal{G}_1 \upharpoonright \mathtt{p}$ with $\mathcal{G}_1 \upharpoonright \mathtt{p} = \bigoplus_{i \in I} s_i!\mathtt{U}_i;\mathtt{T}_i$ and $\mathcal{G}_1 \upharpoonright \mathtt{p} = \bigoplus_{j \in J} s_j!\mathtt{U}_j;\mathtt{T}_j$ with all guards different. Then, the proof proceeds by contradiction. Assume that there exists $r \in \mathcal{R}_{\mathbf{s}}(\Delta)$ and $r \notin \mathcal{R}_{\mathbf{s}}(\Delta_i)$. Note that the last step of the proof for $r \in \mathcal{R}_{\mathbf{s}}(\Delta)$ needs to be by rule [RTCh] because all of the projections $\mathcal{G}(\mathbf{s}) \upharpoonright \mathtt{p}_i$ for $\mathtt{p}_i \neq \mathtt{p}$ are external choices, then assume some $s_i \in I$ is selected. Then, $r \in \mathcal{R}_{\mathbf{s}}(s : s_i!\mathtt{U_i};\mathtt{T}_i @\, p, \{\mathcal{G}(\mathbf{s}) \upharpoonright \mathtt{p}_i @\, p_i\}_{\mathtt{p}_i \in \mathcal{P}(\mathcal{G}(\mathbf{s}) \setminus p\})})$.

Assume $r = r_1\langle \mathtt{p}_k, s_i!\mathtt{U}_i\rangle\langle \mathtt{q}_k, s_i?\mathtt{U_i}\rangle r_2$ where $\langle \mathtt{p}_k, s_i!\mathtt{U}_i\rangle\langle \mathtt{q}_k, s_i?\mathtt{U_i}\rangle$ is the first communication that do not belong to $\mathcal{G}_1$. In this case, it should be that the run is inferred from $\mathcal{R}_{\mathbf{s}}(\mathbf{s} : \mathbb{M}'[s_k!\mathtt{U_k}; \mathtt{T}'_k]@\,\mathtt{p},\ \mathbb{M}[\sum_{j\in J} s_j?\mathtt{U}_j; \mathtt{T}_j]@\,\mathtt{q}, \mathbb{T})$ with the sender in $\mathcal{G}_1$ and the receiver in $\mathcal{G}_2$ this contradicts well formedness, because it should be the case that the choice of the branch has appeared before.

Lemma 12. *Let* $\Delta = s : \mathbb{T}_1, \mathbb{T}_2$ *s.t.* $\mathtt{ch}(\mathbb{T}_1) \cap \mathtt{ch}(\mathbb{T}_2)$. *If* $r_1 \in \mathcal{R}_{\mathbf{s}}(s : \mathbb{T}_1)$ *and* $r_2 \in \mathcal{R}_{\mathbf{s}}(s : \mathbb{T}_2)$ *then* $r_1 r_2 \in \mathcal{R}_{\mathbf{s}}(\mathbf{s} : \Delta)$.

Proof. By straightforward induction on the length of r_1.

Lemma 13. *Let* $\Delta = s : \{\mathtt{T}@\,p_i\}_{i\in I}$ *and* $\Delta' = s : \{\mathtt{T}'@\,p_i\}_{i\in I}$. *If* $r_1 \in \mathcal{R}_{\mathbf{s}}(\Delta_1)$ *and* $r_2 \in \mathcal{R}_{\mathbf{s}}(s : \Delta')$ *then* $r_1 r_2 \in \mathcal{R}_{\mathbf{s}}(\mathbf{s} : \{\mathtt{T}; \mathtt{T}'@\,p_i\}_{i\in I})$.

Proof. By straightforward induction on the length of r_1.

Proof (of Thm. 3). By induction on the derivation of $r \in \mathcal{R}(\mathcal{G}(\mathbf{s}))$ we show $\mathcal{R}(\mathcal{G}(\mathbf{s})) \subseteq \mathcal{R}_{\mathbf{s}}(\Delta)$ implies $\mathcal{R}(\mathcal{G}(\mathbf{s}) \Subset \mathcal{R}_{\mathbf{s}}(\Delta))$. We proceed by analyzing the last applied rule in the derivation of $r \in \mathcal{R}(\mathcal{G}(\mathbf{s}))$.

- Case [RGComm]: Then, $\mathtt{G} = \mathtt{p} \to \mathtt{q} : s\,\langle\mathtt{U}\rangle$ and $r = \langle \mathrm{p}, s!\mathtt{U}\rangle\langle \mathrm{q}, s?\mathtt{U}\rangle$. Moreover, $\Delta = s : s!\mathtt{U}@\,\mathtt{p}; \mathtt{end}, s?\mathtt{U}; \mathtt{end}@\,\mathtt{q}$. It is easy to check that $r \in \mathcal{R}_s(\Delta)$ by using rule [RTCom] (and rules [RTEnd1] and [RTEnd2] to infer that $\epsilon \in \mathcal{R}_s(s : \mathtt{end}@\,\mathtt{p}, \mathtt{end}@\,\mathtt{q})$).
- Case [RGCh]: Then $\mathtt{G} = \mathtt{G}_1 + \mathtt{G}_2$ and $r \in \mathcal{R}(\mathtt{G}_1) \cup \mathcal{R}(\mathtt{G}_1)$. Assume $r \in \mathcal{R}(\mathtt{G}_1)$ (case $r \in \mathcal{R}(\mathtt{G}_2)$ follows analogously). By inductive hypothesis, we know that $r \in \mathcal{R}_s(\mathbf{s} : \{\mathcal{G}(\mathbf{s})\restriction \mathtt{p}@\,\mathtt{p}\}_{\mathtt{p}\in\mathcal{P}(\mathtt{G}_1(\mathbf{s}))})$. W.l.o.g. we can assume that $\mathtt{rdy}(\mathcal{G}) = \mathtt{p}_0$. Then, by rule [RTCh]

  $$r \in \mathcal{R}_s(\mathbf{s} : \mathcal{G}_1(\mathbf{s})\restriction \mathtt{p}_0 \oplus \mathcal{G}_2(\mathbf{s})\restriction \mathtt{p}_0@\,\mathtt{p}_0, \{\mathcal{G}_1(\mathbf{s})\restriction \mathtt{p}@\,\mathtt{p}\}_{\mathtt{p}\in\mathcal{P}(\mathcal{G}_1(\mathbf{s}))\backslash \mathtt{p}_0})$$

  By well-formedness condition, $\mathcal{P}(\mathcal{G}_1(\mathbf{s})) = \mathcal{P}(\mathcal{G}_2(\mathbf{s})) = \mathcal{P}(\mathcal{G}(\mathbf{s}))$ and for any $\mathtt{p} \in \mathcal{P}(\mathcal{G}(\mathbf{s}))\backslash \mathtt{p}_0$, we now that they start with input actions over different names in $\mathcal{G}_1(\mathbf{s})$ and $\mathcal{G}_2(\mathbf{s})$. Therefore, we can repeatedly use Lemma 9 (once for any $\mathtt{p} \in \mathcal{P}(\mathcal{G}(\mathbf{s}))\backslash \mathtt{p}_0$) to conclude that $r \in \mathcal{R}_s(\mathbf{s} : \{\mathcal{G}_1(\mathbf{s})\restriction \mathtt{p} + \mathcal{G}_2(\mathbf{s})\restriction \mathtt{p}@\,\mathtt{p}\}_{\mathtt{p}\in\mathcal{P}(\mathcal{G}(\mathbf{s}))})$.
- Case [RGPar]: Then $\mathtt{G} = \mathtt{G}_1|\mathtt{G}_2$ and $r = r_1 r_2$ with $r_i \in \mathcal{R}(\mathtt{G}_i)$. By well-formedness condition, $\mathcal{P}(\mathtt{G}_1) \cap \mathcal{P}(\mathtt{G}_2)$ and $\mathtt{ch}(\mathtt{G}_1) \cap \mathtt{ch}(\mathtt{G}_2) = \emptyset$. Hence,

  $$\Delta = \mathbf{s} : \{\mathcal{G}_1(\mathbf{s})\restriction \mathtt{p}@\,\mathtt{p}\}_{\mathtt{p}\in\mathcal{P}(\mathcal{G}_1(\mathbf{s}))}, \{\mathcal{G}_2(\mathbf{s})\restriction \mathtt{p}@\,\mathtt{p}\}_{\mathtt{p}\in\mathcal{P}(\mathcal{G}_2(\mathbf{s}))}$$

  By inductive hypothesis, $r_1 \in \mathcal{R}_s(\mathbf{s} : \{\mathcal{G}_1(\mathbf{s})\restriction \mathtt{p}@\,\mathtt{p}\}_{\mathtt{p}\in\mathcal{P}(\mathcal{G}_1(\mathbf{s}))})$ and $r_2 \in \mathcal{R}_s(\mathbf{s} : \{\mathcal{G}_2(\mathbf{s})\restriction \mathtt{p}@\,\mathtt{p}\}_{\mathtt{p}\in\mathcal{P}(\mathcal{G}_2(\mathbf{s}))})$. The proof is completed by using Lemma 12.

- Case [RGSeq]: Then $\mathtt{G} = \mathtt{G}_1; \mathtt{G}_2$ and $r = r_1 r_2$ with $r_i \in \mathcal{R}(\mathtt{G}_i)$. By well-formedness condition,

$$\Delta = \mathbf{s} : \{\mathcal{G}_1(\mathbf{s})\restriction \mathtt{p}; \mathcal{G}_2(\mathbf{s})\restriction \mathtt{p} @ \mathtt{p}\}_{\mathtt{p}\in\mathcal{P}(\mathcal{G}(\mathbf{s}))}$$

By inductive hypothesis, $r_1 \in \mathcal{R}_s(\mathbf{s} : \{\mathcal{G}_1(\mathbf{s})\restriction \mathtt{p} @ \mathtt{p}\}_{\mathtt{p}\in\mathcal{P}(\mathcal{G}_1(\mathbf{s}))})$ and $r_2 \in \mathcal{R}_s(\mathbf{s} : \{\mathcal{G}_2(\mathbf{s})\restriction \mathtt{p} @ \mathtt{p}\}_{\mathtt{p}\in\mathcal{P}(\mathcal{G}_2(\mathbf{s}))})$. The proof is completed by using Lemma 13.
- Case [RGIter]: Then, $\mathtt{G} = \mathtt{G}_1^{*^f}$ with $\mathcal{P}(\mathtt{G}) = \{\mathtt{p}, \mathtt{p}_1, \ldots, \mathtt{p}_n\}$, $\mathtt{rdy}(\mathtt{G}) = \mathtt{p}$, $\forall 1 \leq i \leq n : f(\mathtt{p}_i) = s_i\langle \mathtt{U_i}\rangle$, and $r = r'\langle \mathtt{p}, s_1!\mathtt{U}_1\rangle \ldots \langle \mathtt{p}, s_n!\mathtt{U}_n\rangle\langle \mathtt{p}_1, s_1?\mathtt{U}_1\rangle \ldots \langle \mathtt{p}_n, s_n?\mathtt{U}_n\rangle$ with $r' \in \widetilde{\mathcal{R}}(\mathtt{G})$. Therefore,

$$\Delta = \mathbf{s} : (\mathcal{G}_1(\mathbf{s})\restriction \mathtt{p})^*; b_1!\mathtt{U}_1; \ldots; b_n!\mathtt{U_n} @ \mathtt{p}, \{(\mathcal{G}_1(\mathbf{s})\restriction \mathtt{p}_i)^*; b_i?\mathtt{U_i} @ \mathtt{p}_i\}_{i\in 1..n}$$

By induction on the structure of the derivation, it can be proved that

$$r_1 = [\ldots[[r'_k]r'_{k-1}]\ldots r'_1]r'_0$$

with $r'_i \in \mathcal{R}(\mathtt{G}_1)$. Then, by induction on k we show that $r \in \mathcal{R}_s(\Delta)$. Case $k = 0$. Then, $r'_0 \in \mathcal{R}(\mathtt{G}_1)$ and

$$\Delta' = \mathbf{s} : (\mathcal{G}_1(\mathbf{s})\restriction \mathtt{p}); b_1!\mathtt{U}_1; \ldots; b_n!\mathtt{U_n} @ \mathtt{p}, \{(\mathcal{G}_1(\mathbf{s})\restriction \mathtt{p}_i); b_i?\mathtt{U_i} @ \mathtt{p}_i\}_{i\in 1..n}$$

It is easy to check that $r'_0 \in \mathcal{R}_s(\Delta')$ (by using first Lemma 13, inductive hypothesis and repeatedly rule [RTCom]). By rule [RTIt1], $r \in \mathcal{R}_s(\Delta)$. Case $k = n+1$ follows immediately by using inductive hypothesis and rule [RTIt2].
- Case [RGEnd]: Follows straightforwardly since all projections are end. Hence, the derivation is built by using [RTEnd2] and repeated application of [RTEnd1] (once for each participant).

Proof (of Thm. 4). We proceed by induction on the derivation of $r \in \mathcal{R}_{\mathbf{s}}(\{\mathbf{s} : (\mathcal{G}(\mathbf{s})\restriction \mathtt{p}) @ \mathtt{p}\}_{\mathtt{p}\in\mathcal{P}(\mathcal{G})})$ to show that for all $\Delta = \mathbf{s} : \{\mathtt{T_p} @ \mathtt{p}\}_{\mathtt{p}\in\mathcal{P}(\mathcal{G})}$,

$$\mathcal{R}_{\mathbf{s}}(\Delta) \quad \Subset \quad \mathcal{R}_u(\{\langle \mathcal{I}^{\iota}_{\mathcal{G}}, \sigma\rangle \mid \mathcal{I}^{\iota}_{\mathcal{G}} \in \mathbb{I}^{\Delta,\mathcal{C}}_{\mathtt{p},P} \wedge \sigma \text{ consistent with } \mathcal{C}\})$$

with

$$\mathbb{I}^{\Delta,\mathcal{C}}_{\mathtt{p},P} = \{\mathcal{I}^{\iota}_{\mathcal{G}} \mid \iota(\mathtt{p}) = P,\ \mathcal{C} \smile \Gamma \ \vdash\ \mathcal{I}^{\iota}_{\mathcal{G}} \triangleright \Delta \divideontimes \Gamma'\}$$

We proceed by case analysis on the last rule applied to derive r and write . If last rule is:

- Case [RTCom]: Then, $r = \langle \mathtt{p}, s_k!\mathtt{U}_k\rangle\langle \mathtt{q}, s_k?\mathtt{U_k}\rangle r'$ with $\mathtt{T_p} = \mathbb{M}[s_k!\mathtt{U_k}; \mathtt{T}_k]$, $\mathtt{T_q} = \mathbb{M}'[\sum_{j\in J} s_j?\mathtt{U}_j; \mathtt{T}'_j]$ with $k \in J$, and $r' \in \mathcal{R}_{\mathbf{s}}(\Delta, \mathbf{s} : \mathbb{M}[\mathtt{T}_k] @ \mathtt{p},\ \mathbb{M}'[\mathtt{T}'_k] @ \mathtt{q})$.
Consider $\mathtt{T_p}$. By inspecting the typing rules for systems, we can conclude that $\mathbb{M}$ capture the types of the messages in the session queues of $\mathcal{I}^{\iota}_{\mathcal{G}}$ (applying the rule [VQueue]), while $s_k!\mathtt{U_k}; \mathtt{T}_k$ is the type associated to $\iota(\mathtt{p})$. Then, by inspecting the typing rule for processes (Fig. 5), we can conclude that $\iota(\mathtt{p})$ is a process having, possibly after conditional, iteration operators or actions over different sessions, an

output prefix on top (this can be shown by induction on the derivation of the type). In what follows, we assume wlog that the output prefix is on top. Consequently, for all σ consistent with $\mathcal{C}$ and , we assume

$$\langle \iota(\mathtt{p}), \sigma\rangle \xrightarrow{e \vdash \overline{s_k}\mathtt{v}} \langle \iota''(\mathtt{p}), \sigma\rangle$$

for some ι'' (s.t. $\iota''(\mathtt{p}_i) = \iota(\mathtt{p}_i)$ for all $\mathtt{p}_i \neq \mathtt{p}$. Therefore

$$\langle \mathcal{I}^{\iota}_{\mathcal{G}}, \sigma\rangle \xrightarrow{e \vdash \overline{s_k}\mathtt{v}} \langle \mathcal{I}^{\iota''}_{\mathcal{G}}, \sigma\rangle \qquad (1)$$

Now, take $\mathtt{T}_\mathtt{q}$. Reasoning as for $\mathtt{T}_\mathtt{p}$, we can conclude that $\sum_{j \in J} s_j?\mathtt{U}_j; \mathtt{T}'_j$ is the type of $\iota(\mathtt{q})$ and also that $\iota(\mathtt{q})$ is a process having an input prefix on top. Analogously, for all σ consistent with $\mathcal{C}$,

$$\langle \iota''(\mathtt{q}), \sigma\rangle \xrightarrow{e' \vdash s_k\mathtt{v}} \langle \iota'(\mathtt{q}), \sigma'\rangle$$

for some ι' for some ι' (s.t. $\iota'(\mathtt{p}_i) = \iota''(\mathtt{p}_i)$ for all $\mathtt{p}_i \neq \mathtt{p}$. Hence,

$$\langle \mathcal{I}^{\iota''}_{\mathcal{G}}, \sigma\rangle \xrightarrow{e' \vdash s_k\mathtt{v}} \langle \mathcal{I}^{\iota'}_{\mathcal{G}}, \sigma\rangle \qquad (2)$$

By subject reduction on applied twice (on Eq.(1) and Eq.(2)), there exist Δ_1, Δ_2 s.t.

- $\Gamma \bullet \Delta \overset{\overline{s_k}\mathtt{v}}{\Longrightarrow} \Gamma \bullet \Delta_1 \overset{s_k\mathtt{v}}{\Longrightarrow} \Gamma \bullet \Delta_2$
- $\mathcal{C} \wedge e \wedge e' \mathrel{\llcorner\!\lrcorner} \Gamma, x : \mathtt{U} \;\vdash\; \mathcal{I}^{\iota'}_{\mathcal{G}} \rhd \Delta_2 \divideontimes \Gamma'$ for some x and some $\Gamma' \supseteq \Gamma$.

Let $\mathcal{C}' = \mathcal{C} \wedge e \wedge e'$. Note that $\mathcal{C}'$ is consistent and σ' is consistent with $\mathcal{C} \wedge e \wedge e'$. From $\Gamma \bullet \Delta \overset{\overline{s_k}\mathtt{v}}{\Longrightarrow} \Gamma \bullet \Delta_1 \overset{s_k\mathtt{v}}{\Longrightarrow} \Gamma \bullet \Delta_2$ we can conclude $\Gamma \bullet \Delta \overset{\tau}{\Longrightarrow} \Gamma \bullet \Delta_2$ (by combining both proofs with the rules [TQueue], [TCom1] and [TCom2]). By Lemma 8, $r = \langle \mathtt{p}, s_k!\mathtt{U}_k\rangle\langle \mathtt{q}, s_k?\mathtt{U}_\mathtt{k}\rangle r'$, and $r' \in \mathcal{R}_\mathbf{s}(\Delta_2)$.
By inductive hypothesis,

$$r' \in \mathcal{R}_u(\{\langle \mathcal{I}^{\iota}_{\mathcal{G}}, \sigma\rangle \mid \mathcal{I}^{\iota}_{\mathcal{G}} \in \mathbb{I}^{\Delta_2, \mathcal{C}'}_{\mathtt{p}, \iota(P)} \wedge \sigma \text{ consistent with } \mathcal{C}\}) \qquad (3)$$

The proof is completed by combining Eq. (1-3) using the rules [RRExt] and [RRInt] to conclude that $r \in \mathcal{R}_u(\{\langle \mathcal{I}^{\iota}_{\mathcal{G}}, \sigma\rangle \mid \mathcal{I}^{\iota}_{\mathcal{G}} \in \mathbb{I}^{\Delta, \mathcal{C}}_{\mathtt{p}, P} \wedge \sigma \text{ consistent with } \mathcal{C}\})$.

– Case [RTCh]: By inspecting typing rules for systems and processes, we conclude that $\iota(\mathtt{p}) = \mathtt{if}\ e :\ P\ \mathtt{else}\ Q$ (cases in which branch is after actions over other session or sequential composition are handled analogously). Then, the proof follows from the fact that both $\mathcal{C} \wedge e \not\vdash \bot$ and $\mathcal{C} \wedge \neg e \not\vdash \bot$ because $\iota(\mathtt{p}) = \mathtt{if}\ e :\ P\ \mathtt{else}\ Q$ is typed as a sum. Hence, there is at least one σ consistent with $\mathcal{C}$, such that $\langle \iota(\mathtt{p}), \sigma\rangle \xrightarrow{\tau} \langle \iota'(\mathtt{p}), \sigma\rangle$ and $\mathcal{C} \mathrel{\llcorner\!\lrcorner} \Gamma \;\vdash\; \iota'(\mathtt{p}) \rhd \mathtt{s} : s_j!\mathtt{U}_\mathtt{j}; \mathtt{T}_j @ \mathtt{p}, \Delta' \divideontimes \Gamma'$. Then, the proof is completed by using inductive hypothesis and rule [RRExt].
– Case [RTPar] holds trivially.
– Case [RTEnd1] and [RTEnd2]: Follows by using rule [RREnd].

- Case [RTIt1]: Wlog assume that $\iota(\mathtt{p}) = \mathtt{for}\, x \,\mathtt{in}\, \ell\colon P; Q$ or $\iota(\mathtt{p}) = \mathtt{do}\, N \,\mathtt{until}\, b(x); Q$ (we remark that by typing rules [VThen] and [VElse] iteration can appear under $\mathtt{if}\,\_\colon\,\_\mathtt{else}\,\_$ but the proof follows analogously). Consider $\iota(\mathtt{p}) = \mathtt{for}\, x \,\mathtt{in}\, \ell\colon P; Q$. As for [RTCh], it should be the case that there is at least one σ consistent with $\mathcal{C}$ and $\mathcal{C} \vdash \ell \neq \epsilon$. Then, assume ℓ to have length k. Then, the iteration will be equivalent to $\iota(\mathtt{p}) \equiv P\{\ell_1/x\}; \ldots; P\{\ell_k/x\}; Q$. Therefore, the operational semantics of processes ensures that $r' \in \mathcal{R}_u(\langle \mathcal{I}^{\iota}_{\mathcal{G}}, \sigma\rangle)$ implies $r' = r_1 r_2$ with $r_2 \in \mathcal{R}_u(\langle \mathcal{I}^{\iota'}_{\mathcal{G}}, \sigma\rangle)$ and $\iota'(\mathtt{p}) = P; Q$, $\iota'(\mathtt{q}) = \iota(\mathtt{q})$ for all $\mathtt{p} \neq \mathtt{q}$ (this can be checked by structural induction on processes). By typing rule we know that $\mathcal{C} \,\lrcorner\, \Gamma \vdash P; Q \triangleright \mathtt{s} : \mathtt{T}_i; \mathtt{T}_j @ \mathtt{p}, \Delta' \divideontimes \Gamma'$. By inductive hypothesis, $r_2 \in \mathcal{R}_{\mathtt{s}}(\mathtt{s} : \{\mathtt{T}'_{\mathtt{p}} @ \mathtt{p}\}_{\mathtt{p} \in \mathcal{P}(\mathcal{G})})$ with $\mathcal{C} \,\lrcorner\, \Gamma \vdash \iota'(\mathtt{p}) \triangleright \Delta' \divideontimes \mathtt{s} : \mathtt{T}_{\mathtt{p}}, \Gamma$. Then, the proof is completed by noting that the run r of the specification is matched by a potentially longer trace r', but they coincide on the suffix r_2 (condition established by second inference rule of $\lessdot$). Case $\iota(\mathtt{p}) = \mathtt{do}\, N \,\mathtt{until}\, b(x); Q$ follows analogously.
- Case [RTIt2]: Analogous to the previous case.

F Whole-spectrum implementations and Guarded Automata

In this section we briefly discuss how our notion of whole-spectrum implementation (WSI) can be defined when specifications and implementations are defined as Guarded Automata.

We first recall some basic definitions from Fu et al. [2005]: (P, M) is a composition schema where $P = \{\mathtt{p}_1, \ldots, \mathtt{p}_n\}$ is a set of participants and M are the messages (i.e., the alphabet), $R = \langle (P, M), A\rangle$ is a conversation protocol where A is a guarded automaton, $W = \langle (P, M), A_1, \ldots, A_n\rangle$ is a web service composition, $L(R) = L(A)$ is the language of a conversation protocol. For a web service composition $W = \langle (P, M), A_1, \ldots, A_n\rangle$ we have runs, send sequences and conversations;

1. a run of W is a sequence of configurations $\gamma = c_0, c_1, \ldots, c_n$ where:
   - c_0 is an initial configuration
   - $c_i \to c_{i+1} \quad (i = 0 \ldots n-1)$
   - c_n is a final configuration
2. a send sequence γ on a run γ is the sequence messages, one for each send action in γ, recorded in the order in which they are sent,
3. a conversation is a word w over M for which there is a run γ of W such that $w = \gamma$,
4. the conversations of a web service W, written $C(W)$, is the set of all the conversations for W.

We are now ready to introduce a notion of WSI for guarded automata.

Definition 9 (Whole-spectrum realisation of a guarded automaton). *Let P be a set of participants defined as $\{\mathtt{p}_1, \ldots, \mathtt{p}_n\}$. A_i is a whole-spectrum realisation of $\mathtt{p}_i \in P$ in conversation protocol $R = \langle (P, M), A\rangle$ if for all $w \in L(R)$ there exist $\{A_j\}_{j \in \{1,\ldots,n\}\setminus\{i\}}$ such that $w \in C(< (P, M), A_1, \ldots, A_n >)$.*

Definition 10 (WSI of guarded automaton). A_i *is a WSI of* $\mathtt{p}_i$ *in conversation protocol* $R = \langle (P, M), A \rangle$ *if: (1)* A_i *is a deterministic guarded automaton, and (2)* A_i *is a whole-spectrum realisation of* $\mathtt{p}_i$ *in* $R = \langle (P, M), A \rangle$.