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.

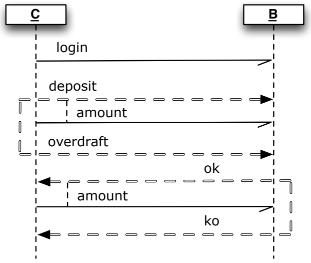
1 Introduction

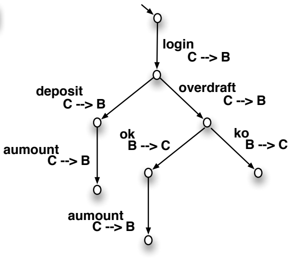
The context A *choreography* describes the expected interactions of a system in terms of the message exchanged between its components (aka *roles*):

> *"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 [15] 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 [12].

[⋆] This work has been partially sponsored by Projects: EU 7FP Grant 295261 (MEALS), ANPCyT 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 [20] (augmented with internal $\_ \oplus \_$ and external $\_ + \_$ choice operators) for roles B and C:

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

In words, T_{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_{C} is the dual of T_{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_{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 [19]. The former interpretation (aka *partial* [19] or weak [23]) admits a realisation if it exhibits a subset of the behaviour. For instance, take

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

then T'_{B} and T_{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 [19]). For instance, T_{B} and T_{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_{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_{B} below (for brevity, each name refers to the interaction of *ATM* with the same initial):

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

The expression $\mathit{check}(c)$ 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_{B} in *partial realisations* of the choreography.[4] (as e.g. in [11]).

[4] For instance, both B_1 and B_2 type-check against T_{B} considered as a session type due to the fact that subtyping for session types [13] is contra-variant with respect to internal choices (and covariant with respect to external choices).

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, $\mathit{check}(c)$ 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. [3, 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 [18] 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 $\mathbf{s}$.

2.1 Types

A *global type term* (GTT, for short) G is derived by the following grammar:

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

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 ($\_^{*^{-}}$), 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 [7], 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 $\mathtt{fst}(\mathtt{G_f}) = \{(\mathtt{p}, \bar{y}), (\mathtt{q}, y), (\mathtt{s}, z)\}$. Formal definitions of such maps are standard and relegated in [3, 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* [14], *knowledge of choice* [7, 14], 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 [3, 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 [3, 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 $\mathtt{r} \in \mathbb{P}$, $\mathtt{G}\lceil\mathtt{r}$ is the *projection* of $\mathtt{G}$ on $\mathtt{r}$ and it is defined homomorphically on $\_\oplus\_$, $\_+\_$, and $\_;\_$ and as follows on the remaining constructs:

$$\mathtt{G}\lceil\mathtt{r} = \begin{cases} y!\mathtt{U} \quad (\text{ resp. } y?\mathtt{U}) & \text{if } \mathtt{G} = \mathtt{r} \to \mathtt{p} : y\langle\mathtt{U}\rangle \quad (\text{ resp. if } \mathtt{G} = \mathtt{p} \to \mathtt{r} : y\langle\mathtt{U}\rangle) \\ \mathtt{G}_i\lceil\mathtt{r} & \text{if } \mathtt{G} = \mathtt{G}_1 \mid \mathtt{G}_2 \text{ and } \mathtt{r} \notin \mathcal{P}(\mathtt{G}_j) \text{ with } j \neq i \in \{1,2\} \\ (\mathtt{G}_1\lceil\mathtt{r})^*;b_1!\mathtt{U}_1;\ldots;b_n!\mathtt{U}_\mathtt{n} & \text{if } \mathtt{G} = \mathtt{G}_1^{*^f},\ \mathrm{cod}(f) = \{b_1\langle\mathtt{U}_1\rangle,\ldots,b_n\langle\mathtt{U}_\mathtt{n}\rangle\}, \text{ and } \mathtt{r} \in \mathtt{rdy}(\mathtt{G}_1) \\ (\mathtt{G}_1\lceil\mathtt{r})^*;b?\mathtt{U} & \text{if } \mathtt{G} = \mathtt{G}_1^{*^f},\ f(\mathtt{r}) = b\langle\mathtt{U}\rangle, \text{ and } \mathtt{r} \notin \mathtt{rdy}(\mathtt{G}_1) \\ \mathtt{end} & \text{if } \mathtt{G} = \mathtt{p} \to \mathtt{q} : y\langle\mathtt{U}\rangle \text{ and } \mathtt{r} \neq \mathtt{p},\mathtt{q} \text{ or if } \mathtt{G} = \mathtt{end} \\ \mathtt{end} & \text{if } \mathtt{G} = \mathtt{G}_1^{*^f} \text{ and } \mathtt{r} \notin \mathcal{P}(\mathtt{G}_1) \text{ or } f(\mathtt{r}) \text{ is undefined} \end{cases}$$

Our projection is total on well-formed global types. All but the clauses for the projections of iteration in the definition of $\_\lceil\_$ are straightforward (c.f. [14]). 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* $\mathcal{G}(\mathbf{y})\lceil\mathtt{r}$ of a global type $\mathcal{G}(\mathbf{y}) \overset{\triangle}{=} \mathtt{G}$ with respect to $\mathtt{r}$ is a local type $\mathcal{T}(\mathbf{y}) \overset{\triangle}{=} \mathtt{T}$ where $\mathtt{T} = \mathtt{G}\lceil\mathtt{r}$.

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

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

2.2 Running example

We illustrate our approach on a real yet tractable protocol, the Post Office Protocol - Version 2 (POP2) [5] between a client and a mail server. We describe POP2 with the following choreography where $\mathtt{G}_\mathtt{EXIT}{=}\mathtt{S} \to \mathtt{C} : \mathtt{BYE}\langle\rangle$:

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

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 $\mathtt{S}$ 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 $\mathtt{C}$ repeatedly asks either (a) to enter a folder (sending the folder's name on $\mathtt{FOLD}$) and then receiving back the number of messages in that folder, or (b) to request a message by sending its index along $\mathtt{READ}$ and then receiving back the length of the message. In case (a), the loop is immediately repeated after $\mathtt{S}$'s reply, in case (b) the protocol continues as $\mathtt{G}_{\mathtt{SIZE}}$ where another loop starts with $\mathtt{C}$ either (a) retrieving the message or (b) asking for another message (by interacting again on $\mathtt{READ}$). For (a), $\mathtt{C}$ signals on $\mathtt{RETR}$ that it is ready to receive data that are sent by $\mathtt{S}$ on $\mathtt{MSG}$ (sort $\mathtt{Data}$ abstracts away the format of messages specified in [10]); after these interactions the choreography continues as $\mathtt{G}_{\mathtt{XFER}}$ where the transmission is acknowledged by $\mathtt{C}$ with the interactions in $\mathtt{G}_{\mathtt{XFER}}$: $\mathtt{ACKS}$ keeps the message in the mailbox, $\mathtt{ACKD}$ deletes the message, $\mathtt{NACK}$ notifies that the message has not been received and must be kept in the mailbox; after any acknowledgement, $\mathtt{S}$ sends $\mathtt{C}$ the length of the next message. After some iterations in $\mathtt{G}_{\mathtt{SIZE}}$, $\mathtt{C}$ specifies a different folder and repeats $\mathtt{G}_{\mathtt{NMBR}}$.

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

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

The messages in $\mathtt{T}_{\mathtt{S}}$ are as in $\mathtt{G}_{\mathtt{POP}}$ and $\mathtt{S}$ iterates until either a signal on $\mathtt{QUIT}$ or on $\mathtt{FOLD}$ is sent by $\mathtt{C}$.

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 $\mathtt{S}$ uses a third-party authentication service $\mathtt{A}$:

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

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

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

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:

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

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

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

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

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

Fig. 1. Labelled transitions for specifications

as usual, when writing $\Delta,\mathbf{s}:\mathcal{T}$, $\mathbf{s}\not\in \mathrm{dom}(\Delta)$ 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

$$\alpha ::= \overline{u}^n\mathbf{s} \quad|\quad u_i\mathbf{s} \quad|\quad \bar{s}\mathtt{v} \quad|\quad s\mathtt{v} \quad|\quad \tau \tag{2.3}$$

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:

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

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

[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.

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

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

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

The syntax of systems S and processes P is

$$\begin{array}{rl} P, Q ::= & u_i(\mathbf{y}).P \;\big|\; \overline{u}^n(\mathbf{y}).P \;\big|\; N \;\big|\; \overline{s}e \;\big|\; \mathtt{if}\ e :\ P\ \mathtt{else}\ Q \\ \big| & P; P \;\big|\; \mathtt{for}\ x\ \mathtt{in}\ \ell :\ P \;\big|\; \mathtt{do}\ N\ \mathtt{until}\ b(x) \\ S \ ::= & P \;\big|\; (\nu\mathbf{s})S \;\big|\; S \mid S \;\big|\; s : M \end{array}$$

All constructions but loops are straightforward. In $\mathtt{for}\ x\ \mathtt{in}\ \ell :\ P$, the body P is executed for each element in ℓ, while $\mathtt{do}\ N\ \mathtt{until}\ b(x)$ 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, $\mathtt{fv}(P)$ 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 $\mathtt{fc}(S)$ of free session names of S is defined as expected); a system S is *closed* when $\mathtt{fc}(S) = \emptyset$ 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 $\mathbf{s} \not\subseteq \mathtt{fc}(S')$.

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:

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

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 $\sigma[x \mapsto \mathtt{v}]$ is the update of σ at x with $\mathtt{v}$. 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 $\mathtt{true} \vdash \alpha$ and $e \wedge e' \vdash \alpha$ instead of $e \vdash (e' \vdash \alpha)$.

We comment on the rules in Fig. 2 where $\mathtt{fc}(\alpha)$ is defined as $\mathtt{fc}(\overline{u}^n\mathbf{s}) = \mathtt{fc}(u_i\mathbf{s}) = \{u\}$, $\mathtt{fc}(\overline{s}\mathtt{v}) = \mathtt{fc}(s\mathtt{v}) = \{s\}$, and $\mathtt{fc}(\tau) = \emptyset$. 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 $\mathtt{v}$. 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

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

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

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

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

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

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

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

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

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

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

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 $[\texttt{SInit}]$ 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 $[\texttt{SNews}]$ is standard and allows an action α to pass a restriction that does not involve the names of α. Rule $[\texttt{SInit}]$ 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 semantics relies on a global store σ. However, the condition $\mathrm{var}(S_1) \cap \mathrm{var}(S_2) = \emptyset$ in rule $[\texttt{SPar}]$ 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 $\texttt{T}_\texttt{S}$ (i.e., participant $\texttt{S}$ of $\texttt{G}_\texttt{POP}$) from § 2.2. To ease the presentation, we use the following auxiliary functions.

- $\texttt{auth} : \texttt{Str} \to \texttt{Bool}$ that is used for authenticating clients;

- $\mathtt{fn} : \mathtt{Str} \to \mathtt{Int}$ that given a folder name returns the number of messages in that folder (we assume `inbox` to be the default folder);
- $\mathtt{mn} : \mathtt{Int} \to \mathtt{Int}$ that given a message number returns its length (in bytes);
- $\mathtt{data} : \mathtt{void} \to \mathtt{Data}$ that returns the current message;
- $\mathtt{next} : \mathtt{void} \to \mathtt{Int}$ that returns the next message number;
- $\mathtt{del} : \mathtt{void} \to \mathtt{Int}$ that returns the next message number and deletes the current message from the folder.

Let s_k denote the name in $\mathbf{s}$ 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.

$$P_{\mathtt{INIT}} = u_{\mathtt{S}}(\mathbf{s}).P_{\mathtt{S}} \qquad P_{\mathtt{S}} = s_{\mathtt{QUIT}}();P_{\mathtt{EXIT}} + s_{\mathtt{HELO}}(x);P_{\mathtt{MBOX}} \qquad P_{\mathtt{EXIT}} = \overline{s_{\mathtt{BYE}}}$$

$$\begin{aligned}
P_{\mathtt{MBOX}} &= \mathtt{if}\ \mathtt{auth}(x)\ :\ \overline{s_{\mathtt{R}}}\mathtt{fn}(\mathtt{inbox});P_{\mathtt{NMBR}}\ \mathtt{else}\ \overline{s_{\mathtt{E}}};P_{\mathtt{EXIT}}\\
P_{\mathtt{NMBR}} &= \mathtt{do}(s_{\mathtt{FOLD}}(x);\overline{s_{\mathtt{R}}}\mathtt{fn}(x)\ +\ s_{\mathtt{READ}}(x');\overline{s_{\mathtt{R}}}\mathtt{mn}(x');P_{\mathtt{SIZE}})\ \mathtt{until}\ s_{\mathtt{QUIT}}();P_{\mathtt{EXIT}}\\
P_{\mathtt{SIZE}} &= \mathtt{do}(s_{\mathtt{RETR}}();\overline{s_{\mathtt{MSG}}}\mathtt{data}();P_{\mathtt{XFER}}\ +\ s_{\mathtt{READ}}(x');\overline{s_{\mathtt{R}}}\mathtt{mn}(x'))\ \mathtt{until}\ s_{\mathtt{FOLD}}(x);\overline{s_{\mathtt{R}}}\mathtt{fn}(x)\\
P_{\mathtt{XFER}} &= s_{\mathtt{ACKS}}();\overline{s_{\mathtt{R}}}\mathtt{mn}(\mathtt{next}())\ +\ s_{\mathtt{ACKD}}();\overline{s_{\mathtt{R}}}\mathtt{mn}(\mathtt{del}())\ +\ s_{\mathtt{NACK}}();\overline{s_{\mathtt{R}}}\mathtt{mn}(x')
\end{aligned}$$

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

$$\begin{aligned}
P'_{\mathtt{S}} &= s_{\mathtt{QUIT}}();P_{\mathtt{EXIT}} + s_{\mathtt{HELO}}(x);P_{\mathtt{AUTH}}\\
P_{\mathtt{AUTH}} &= \overline{s_{\mathtt{REQ}}}x;s_{\mathtt{RES}}(y);P'_{\mathtt{MBOX}}\\
P'_{\mathtt{MBOX}} &= \mathtt{if}\ \mathtt{auth}(x) \land y\ :\ \overline{s_{\mathtt{R}}}\mathtt{fn}(\mathtt{inbox});P_{\mathtt{NMBR}}\ \mathtt{else}\ \overline{s_{\mathtt{E}}};P_{\mathtt{EXIT}}
\end{aligned}$$

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 $I^{\iota}_{\mathcal{G}}$ such that either (i) $I^{\iota}_{\mathcal{G}} \equiv \iota(\mathtt{p}_1) \mid \ldots \mid \iota(\mathtt{p}_n)$ and $\mathbf{y} \cap \mathtt{fc}(\iota(\mathtt{p}_1)) = \ldots = \mathbf{y} \cap \mathtt{fc}(\iota(\mathtt{p}_n)) = \emptyset$ or (ii) $I^{\iota}_{\mathcal{G}} \equiv (\nu\mathbf{y})(\iota(\mathtt{p}_1) \mid \ldots \mid \iota(\mathtt{p}_n) \mid \mathbf{y} : \mathbf{M})$.*

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 $I^{\iota}_{\mathcal{G}}$. An execution trace of $I^{\iota}_{\mathcal{G}}$ 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 $\langle \mathtt{p}, s?\mathtt{U} \rangle$).

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 $\iota' = \iota[\mathtt{p} \mapsto P]$

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

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

Fig. 3. Runs of implementations

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

Rules in Fig. 3 rely on the semantics of Fig. 2. In rule [$\mathtt{RRInt}$] (where $\mathtt{obj}(\alpha) = \mathtt{v}$ for $\alpha = \bar{s}\mathtt{v}$ 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 $\bar{y}\mathtt{v}$ 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 [$\mathtt{RREnd}$] is straightforward. Rule [$\mathtt{RRExt}$] 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 [$\mathtt{RRExt}$] 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 [7, 9] 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 as in [16] 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 four rules are straightforward. Rule [$\mathtt{RGPar}$] 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

$$\frac{}{\varepsilon \in \mathcal{R}(\mathtt{end})}\,[\mathtt{RGEnd}] \qquad \frac{}{\langle \mathtt{p}, s!\mathtt{U}\rangle\langle \mathtt{q}, s?\mathtt{U}\rangle \in \mathcal{R}(\mathtt{p} \to \mathtt{q} : s\,\langle \mathtt{U}\rangle)}\,[\mathtt{RGComm}]$$

$$\frac{r \in \mathcal{R}(\mathtt{G}_1) \cup \mathcal{R}(\mathtt{G}_1)}{r \in \mathcal{R}(\mathtt{G}_1 + \mathtt{G}_2)}\,[\mathtt{RGCh}] \qquad \frac{r_1 \in \mathcal{R}(\mathtt{G}_1) \quad r_2 \in \mathcal{R}(\mathtt{G}_2)}{r_1 r_2 \in \mathcal{R}(\mathtt{G}_1 ; \mathtt{G}_2)}\,[\mathtt{RGSeq}]$$

$$\frac{r_1 \in \mathcal{R}(\mathtt{G}_1) \quad r_2 \in \mathcal{R}(\mathtt{G}_2)}{r_1 r_2 \in \mathcal{R}(\mathtt{G}_1 \mid \mathtt{G}_2)}\,[\mathtt{RGPar}] \qquad \frac{r_1 \in \mathcal{R}(\mathtt{G})}{r_1 \in \widetilde{\mathcal{R}}(\mathtt{G}^{*^f})}\,[\mathtt{RG}^*\mathtt{1}] \quad \frac{r_1 \in \widetilde{\mathcal{R}}(\mathtt{G}^{*^f}) \quad r_2 \in \mathcal{R}(\mathtt{G})}{[r_1] r_2 \in \widetilde{\mathcal{R}}(\mathtt{G}^{*^f})}\,[\mathtt{RG}^*\mathtt{2}]$$

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

Fig. 4. Runs of a global type

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). 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 [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

$$[r] \lessdot \varepsilon \qquad \varepsilon \lessdot r \qquad r \lessdot r' \implies [r] \lessdot [r'] \qquad r_1 \lessdot r_1' \wedge r_2 \lessdot r_2' \implies r_1 r_2 \lessdot r_1' r_2'$$

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.* $I^{\iota}_{\mathcal{G}} \in \mathbb{I}$ *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 $\mathcal{C} \smile \Gamma \ \vdash \ S \rhd \Delta \divideontimes \Gamma'$ 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

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

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 name 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 [14] and detailed in [3, Appendix C]). Condition $\mathcal{C} \not\vdash \bot$ is implicitly assumed among the hypothesis of each 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 session request as i-th role. Rule [VRec] types an external choice $P = \sum\limits_{i \in I} y_i(x_i);P_i$ 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

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

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

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

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

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

$$\frac{\Gamma(e) = \mathtt{bool} \qquad \mathcal{C} \wedge e \nvdash \bot \qquad \mathcal{C} \wedge \neg e \vdash \bot \qquad \mathcal{C} \wedge e \sqcup \Gamma \ \vdash \ P \triangleright \Delta \ \divideontimes \ \Gamma'}{\mathcal{C} \sqcup \Gamma \ \vdash \ \mathtt{if}\ e :\ P\ \mathtt{else}\ Q \triangleright \Delta \ \divideontimes \ \Gamma'}[\texttt{VThen}]$$

$$\frac{\Gamma(e) = \mathtt{bool} \qquad \mathcal{C} \wedge e \vdash \bot \qquad \mathcal{C} \wedge \neg e \nvdash \bot \qquad \mathcal{C} \wedge \neg e \sqcup \Gamma \ \vdash \ Q \triangleright \Delta \ \divideontimes \ \Gamma'}{\mathcal{C} \sqcup \Gamma \ \vdash \ \mathtt{if}\ e :\ P\ \mathtt{else}\ Q \triangleright \Delta \ \divideontimes \ \Gamma'}[\texttt{VElse}]$$

$$\frac{\begin{array}{c} \Gamma(e) = \mathtt{bool} \qquad \mathcal{C} \wedge e \nvdash \bot \qquad \mathcal{C} \wedge \neg e \nvdash \bot \\ \mathcal{C} \wedge e \sqcup \Gamma \ \vdash \ P \triangleright \Delta_1 \ \divideontimes \ \Gamma_1 \qquad \mathcal{C} \wedge \neg e \sqcup \Gamma \ \vdash \ Q \triangleright \Delta_2 \ \divideontimes \ \Gamma_2 \end{array}}{\mathcal{C} \sqcup \Gamma \ \vdash \ \mathtt{if}\ e :\ P\ \mathtt{else}\ Q \triangleright \Delta_1 \bowtie \Delta_2 \ \divideontimes \ \Gamma_1 \cap \Gamma_2}[\texttt{VCond}]$$

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

$$\frac{\mathcal{C} \vdash \ell = \varepsilon}{\mathcal{C} \sqcup \Gamma \ \vdash \ \mathtt{for}\ x\ \mathtt{in}\ \ell :\ P \triangleright \mathbf{y} : \mathtt{end} \ \divideontimes \ \Gamma'}[\texttt{VFor2}]$$

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

Fig. 5. Typing rules for processes

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 $\mathcal{C} \sqcup \Gamma \ \vdash \ S \triangleright \Delta \ \divideontimes \ \Gamma'$ is decidable.*

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

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

Fig. 6. Composition of types

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: $P_1(c) = \mathtt{if}\ c :\ \overline{s_{\mathtt{OK}}};s_{\mathtt{AMOUNT}}(x)\ \mathtt{else}\ \overline{s_{\mathtt{KO}}}$ (resp. $P_2(c) = \overline{s_{\mathtt{KO}}}$) and $\Delta = \mathbf{s} : \mathtt{OK}!;\mathtt{AMOUNT}? \oplus \mathtt{KO}!$. The verification of $P_1(c)$ terminates successfully after an application of $[\mathtt{VCond}]$. In the case of $P_2(c)$ the only rule for a sending process, $[\mathtt{VSend}]$, 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 $P_{\mathtt{INIT}}$ from Ex. 3 are shown below. By rule $[\mathtt{VAcc}]$, the newly created session is added to the session environment, then the verification of the external choice is split by $[\mathtt{VRec}]$ into the verification of each branch. As we omit the whole derivation, just assume that $P_{\mathtt{INIT}}$ yields $\Gamma' = \Gamma, x : \mathtt{Str}$.

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

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

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

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

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

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

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

We show the verification of the first branch

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

and the successive steps for the second branch:

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

The verification of $P'_{\mathtt{MBOX}}$ proceeds with an application of $[\mathtt{VCond}]$, $[\mathtt{VIf}]$ or $[\mathtt{VElse}]$ depending on $\mathtt{auth}() \land y$. If $\mathtt{auth}()$ is not a contradiction then $[\mathtt{VCond}]$ 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 $[\mathtt{VElse}]$ 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 $\mathtt{G}_{\mathtt{POP}}$ (§ 2.2), and as a client in a session $\mathtt{G}_{\mathtt{ADMIN}}$ 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.

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

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

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

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 $[\mathtt{VAcc}]$, $[\mathtt{VReq}]$ and $[\mathtt{VRec}]$.

$$\dfrac{\dfrac{\begin{array}{c}\mathtt{true}\,\text{\textvisiblespace}\,\Gamma \;\vdash\; P_{\mathtt{EXIT}} \triangleright \mathbf{s} : \mathtt{T}_{\mathtt{EXIT}}, \mathbf{t} : \mathtt{T}_{\mathtt{S}} \divideontimes \Gamma' \\ \mathtt{true}\,\text{\textvisiblespace}\,\Gamma, x : \mathtt{Str} \;\vdash\; P_{\mathtt{AUTH}} \triangleright \mathbf{s} : \mathtt{T}_{\mathtt{MBOX}}, \mathbf{t} : \mathtt{T}_{\mathtt{S}} \divideontimes \Gamma'\end{array}}{\Gamma(u) \equiv \mathtt{G}_{\mathtt{POP}}(\mathbf{s}) \quad \Gamma(v) \equiv \mathtt{G}_{\mathtt{POP}}(\mathbf{t}) \quad \mathtt{true}\,\text{\textvisiblespace}\,\Gamma \;\vdash\; P''_{\mathtt{S}} \triangleright \mathbf{s} : \mathtt{T}_{\mathtt{S}}, \mathbf{t} : \mathtt{T}_{\mathtt{S}} \divideontimes \Gamma'}\,[\mathtt{VRec}]}{\mathtt{true}\,\text{\textvisiblespace}\,\Gamma \;\vdash\; u_{\mathtt{S}}(\mathbf{s}).\bar{v}^{\mathtt{S}}(\mathbf{t}).P''_{\mathtt{S}} \triangleright \emptyset \divideontimes \Gamma'}\,[\mathtt{VAcc}],[\mathtt{VReq}]$$

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

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

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

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

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

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

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

$$\frac{\begin{array}{c}\mathtt{true} \text{␣} \Gamma, x : \mathtt{Str} \;\vdash\; \overline{t_{\mathtt{REQ}}}x \triangleright \Delta \circledast \mathbf{s} : \mathtt{end}, \mathbf{t} : \mathtt{REQ!Str} \\ \mathtt{true} \text{␣} \Gamma, x : \mathtt{Str} \;\vdash\; t_{\mathtt{RES}}(y); P''_{\mathtt{MBOX}} \triangleright \mathbf{s} : \mathtt{T}_{\mathtt{MBOX}}, \mathbf{t} : \mathtt{RES?Bool} \circledast \Gamma'\end{array}}{\mathtt{true} \text{␣} \Gamma, x : \mathtt{Str} \;\vdash\; \overline{t_{\mathtt{REQ}}}x; t_{\mathtt{RES}}(y); P''_{\mathtt{MBOX}} \triangleright \mathbf{s} : \mathtt{T}_{\mathtt{MBOX}}, \mathbf{t} : \mathtt{T}_{\mathtt{S}} \circledast \Gamma'} \; [\mathtt{VSeq}]$$

Focusing on the second branch we apply [VRec]

$$\frac{\mathtt{true} \text{␣} \Gamma, x : \mathtt{Str}, y : \mathtt{Bool} \;\vdash\; P''_{\mathtt{MBOX}} \triangleright \mathbf{s} : \mathtt{T}_{\mathtt{MBOX}}, \mathbf{t} : \mathtt{end} \circledast \Gamma'}{\mathtt{true} \text{␣} \Gamma, x : \mathtt{Str} \;\vdash\; t_{\mathtt{RES}}(y); P''_{\mathtt{MBOX}} \triangleright \mathbf{s} : \mathtt{T}_{\mathtt{MBOX}}, \mathbf{t} : \mathtt{RES?Bool} \circledast \Gamma'} \; [\mathtt{VRec}]$$

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. ♦

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 [14], *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 $\mathbf{s}$ to runtime types of roles in $\mathbf{s}$; we write $\Delta, \mathbf{s} : \mathbb{M}[\mathcal{T}] @\, \mathtt{p}$ to specify that (1) the runtime type of $\mathtt{p} \in \mathbb{P}$ in $\mathbf{s}$ is $\mathbb{M}[\mathcal{T}]$ and (2) that for any $\mathbf{s} : \mathbb{M}'[\mathcal{T}'] @\, \mathtt{q}$ 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 $\mathbf{s}$ 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 $\overset{\alpha}{\Longrightarrow} = \xrightarrow{\tau}{}^{*} \xrightarrow{\alpha}$. Let $\Gamma \bullet \Delta \overset{s}{\Longrightarrow}$ shorten $\exists \Delta' \exists \mathtt{v} : \Gamma \bullet \Delta \overset{s\mathtt{v}}{\Longrightarrow} \Gamma \bullet \Delta'$.

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

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*

$$\mathcal{C} \,\textvisiblespace\, \Gamma \;\vdash\; S \rhd \Delta \divideontimes \Gamma' \quad \textit{and} \quad \langle S,\sigma\rangle \xrightarrow{e\vdash\alpha} \langle S',\sigma'\rangle$$

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

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

Corollary 1 (Soundness). *If $\mathcal{C} \,\textvisiblespace\, \Gamma \;\vdash\; S \rhd \Delta \divideontimes \Gamma'$ 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 $\mathbf{s}$ 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 [3, 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 $\mathcal{G}(\mathbf{s})$, $\mathcal{R}(\mathcal{G}(\mathbf{s})) \Subset \mathcal{R}_{\mathbf{s}}(\{\mathbf{s} : (\mathcal{G}(\mathbf{s}) \restriction \mathtt{p})@\,\mathtt{p}\}_{\mathtt{p}\in\mathcal{P}(\mathcal{G}(\mathbf{s}))})$.*

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

$$\mathbb{I}_{\mathtt{p},P} = \{\, I^{\iota}_{\mathcal{G}} | \iota(\mathtt{p}) = P, \forall \mathtt{q} \in \mathcal{P}(\mathtt{G}) : \mathtt{true} \,\textvisiblespace\, \Gamma, u : \mathcal{G}(\mathbf{s}) \;\vdash\; \iota(\mathtt{q}) \rhd \Delta, \mathbf{s} : \mathcal{G}(\mathbf{s}) \restriction \mathtt{q} \divideontimes \Gamma' \}$$

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

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

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

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

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

Fig. 8. Runs of runtime local types

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 [12], which poses the classical question about the decidability of the notion of realisability (see [1]). To the best of our knowledge, the only proposal dealing with complete (i.e., exhaustive) realisations in a behavioural context is [7] but this approach focuses on non-deterministic implementation languages. Our type system is more restrictive than [2, 4, 6, 8, 14]. 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 [7, 17, 22] 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 extent our proposal is related to the fair subtyping approach in [21], 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 [2, 14], where the number of iterations is limited. This restriction is on the lines of [7] that also considers finite traces. The extension of our theory with a more general form of iteration is scope for future work.

Acknowledgements We thank the reviewers for their insightful and helpful comments.

References

1. S. Basu, T. Bultan, and M. Ouederni. Deciding choreography realizability. In *POPL*, 2012.
2. L. Bettini, M. Coppo, L. D'Antoni, M. De Luca, M. Dezani-Ciancaglini, and N. Yoshida. Global progress in dynamically interleaved multiparty sessions. In *CONCUR*, 2008.
3. L. Bocchi, H. Melgratti, and E. Tuosto. Extended version of this paper. Available at `http://publicaciones.dc.uba.ar/Publications/2014/BMT14c/`, 2014.
4. M. Bravetti and G. Zavattaro. A theory of contracts for strong service compliance. *MSCS*, 19(3), 2009.
5. 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.
6. L. Caires and H. T. Vieira. Conversation types. In *ESOP*, 2009.
7. G. Castagna, M. Dezani-Ciancaglini, and L. Padovani. On global types and multi-party session. *LMCS*, 8(1), 2012.
8. G. Castagna and L. Padovani. Contracts for mobile processes. In *CONCUR*, 2009.
9. T.-C. Chen and K. Honda. Specifying stateful asynchronous properties for distributed programs. In *CONCUR*, 2012.
10. D. Crocker. Standard for the format of arpa internet text messages. RFC 822, available at `www.ietf.org/rfc/rfc0822.txt`, February 1982.
11. M. Dezani-Ciancaglini and U. de' Liguoro. Sessions and session types: An overview. In *WS-FM*, 2009.
12. X. Fu, T. Bultan, and J. Su. Realizability of conversation protocols with message contents. *Int. J. Web Service Res.*, 2(4):68–93, 2005.
13. S. Gay and M. Hole. Subtyping for Session Types in the Pi-Calculus. *Acta Inf.*, 42(2/3):191–225, 2005.
14. K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. In *POPL*, 2008.
15. N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, and Y. Lafon. `http://www.w3.org/TR/2004/WD-ws-cdl-10-20041217`, 2004.
16. L. Lamport. Time, clocks, and the ordering of events in a distributed system. *CACM*, 21(7):558–564, July 1978.
17. I. Lanese, C. Guidi, F. Montesi, and G. Zavattaro. Bridging the gap between interaction-and process-oriented choreographies. In *SEFM*, 2008.
18. J. Lange and E. Tuosto. Synthesising choreographies from local session types. In *CONCUR*, 2012.
19. N. Lohmann and K. Wolf. Decidability results for choreography realization. In *ICSOC*, 2011.
20. R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
21. L. Padovani. Fair subtyping for multi-party session types. In *COORDINATION*, 2011.
22. G. Salaün and T. Bultan. Realizability of choreographies using process algebra encodings. In *Integrated Formal Methods*, 2009.
23. J. Su, T. Bultan, X. Fu, and X. Zhao. Towards a theory of web service choreographies. In *WS-FM*, 2007.