Project no.: PIRSES-GA-2011-295261

Project full title: Mobility between Europe and Argentina applying Logics to Systems

Project Acronym: MEALS

Deliverable no.: 5.4 / 1

Title of Deliverable: Choice-preserving Multiparty Session Types

Contractual Date of Delivery to the CEC:	**1-Apr-2013**
Actual Date of Delivery to the CEC:	**15-Mar-2013**
Organisation name of lead contractor for this deliverable:	**IMP**
Author(s):	**Laura Bocchi, Hernán Melgratti, Emilio Tuosto**
Participants(s):	**UBA, IMP, ULEIC, UNR, ITBA**
Work package contributing to the deliverable:	**WP5**
Nature:	**R**
Dissemination Level:	**Public**
Total number of pages:	**25**
Start date of project:	1 Oct. 2011 **Duration:** 48 month

Abstract:

In the realisability of choreographies it is crucial how distributed choices are resolved. We introduce a novel notion of realisability for distributed choreographies –called *whole-spectrum realisation*– requiring implementations to resolve non-deterministic choices so that each branch has a context firing it. We represent choreographies as minor variants of global types of Carbone, Honda, and Yoshida and use local types projections to validate processes in a type system that guarantees whole-spectrum realisability.

This project has received funding from the European Union Seventh Framework Programme (FP7 2007-2013) under Grant Agreement Nr. 295261.

Contents

1 Introduction **3**

2 Global and Local Types **5**
2.1 Global types 6
2.2 Local types 8
2.3 Behaviour of types 9

3 Systems **10**

4 Runtime Types **13**

5 Whole-Spectrum Realisation **13**

6 Typing rules **16**

7 Properties of the type system **19**
7.1 Soundness 19
7.2 Whole-spectrum realisation by typing 20

8 Conclusion and related work **21**

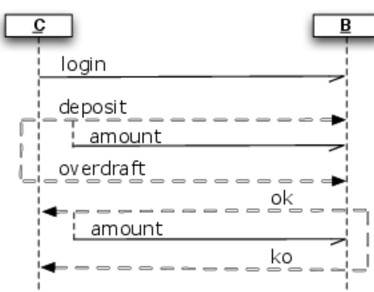
Bibliography **22**

A Typing rules for systems **24**

MEALS Partner Abbreviations **24**

1 Introduction

The context A *choreography* gives a description of the expected behaviour of a composed system in terms of the message exchanged between its component parties. Consider the following simple choreography, *ATM*, involving two parties (aka roles) B and C, where B is the cash machine of a bank that offers a deposit and overdraft service (after a successful authentication) to a client C. If C decides to make a deposit, it indicates the amount of money to be deposited. If C asks to overdraft then B can either grant it or deny it; in the former case C will communicate the amount of money required. *ATM* is depicted as follows:



The doubly stroked lines represent choices and the dashed lines connect interactions with the branches where they occur.

Choreographies are understood to be descriptions of the system from a global perspective, hence they do not have a single execution control. On the contrary, each role needs to be implemented with its own independent control flow. A set of role implementations is a suitable *realisation* of a given choreography when the behaviour emerging from the concurrent execution of the role implementations matches the behaviour prescribed by that choreography. We say that a choreography is *realisable* when it has a realisation.

A possible realisation of the *ATM* choreography can be given using two CCS-like processes augmented with internal (⊕) and external (+) choices:

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

$T_{\mathtt{B}}$ and $T_{\mathtt{C}}$ implement roles B and C, respectively. For instance, $T_{\mathtt{B}}$ specifies that, after C logs in, B waits to interact on ports deposit or overdraft; in the latter case, B non-deterministically decides whether to grant or deny the overdraft. $T_{\mathtt{C}}$ is dual.

As noted in [18], realisability of choreographies requires one to considers several issues. Here, we focus on the interpretation of the prescriptive behaviour of choreographies. In fact, choreographies have been interpreted as either constraints or obligations of distributed interactions [18]. According to the former interpretation, a realisation is admissible if it exhibits a subset of the behaviour (hence such realisations are also referred to as *partial* [18] or weak [20]). For instance, *ATM* is partially realised when considering an implementation for B, $T'_{\mathtt{B}}$, which denies every overdraft request:

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

On the contrary, a realisation is admissible when interpreting choreographies as interaction obligations if it is able to exhibit *all* interaction sequences (hence such realisations are also referred to as *complete* realisations [18]). For instance, $T_{\mathtt{B}}$ and $T_{\mathtt{C}}$ form a complete realisation of *ATM*.

Complete realisations Multiparty session types [1, 15] address the problem of checking whether a distributed implementation realises a choreography but avoids the direct comparison of traces. This is done in two steps: (*i*) *projection*: the choreography (i.e., a global type) is projected onto one local specification (i.e., local type) for each role, (*ii*) *validation*: each role implementation is *type-checked* against its local type. If both steps are successful, then it is guaranteed that the implementation realises the choreography.

While complete realisations have been the predominant interpretation in several research lines (see [20] for a survey), it has been largely neglected in the context of multiparty session types, in which realisation is generally understood as partial (see [12]). In fact, since the pioneer work of [14], session types are aimed at guaranteeing the interaction compatibility among parties (i.e., they express behaviour constraints instead of behaviour obligations).

To the best of our knowledge, the only proposal dealing with complete (i.e., exhaustive) realisations in the context of multiparty session types is the one in [7]. This approach, as others dealing with complete realisations [20], adopt non-deterministic languages (e.g., featuring non-deterministic internal choices as the one in $T_{\mathtt{B}}$).

While being a suitable abstraction for choreographies and their roles, internal non-determinism has to be resolved in concrete implementations using deterministic constructs such as conditional branch statements. Defining complete realisation for deterministic languages poses some difficulties, which we illustrate using the example *ATM*.

The non-deterministic choice in $T_{\mathtt{B}}$ abstracts away from the actual conditions used in a concrete implementation to resolve a choice. This permits e.g. different banks to adopt different policies depending, for instance, on the type of the clients' accounts. Consider the following two possible deterministic implementations, B_1 and B_2, of $T_{\mathtt{B}}$:

$$B_i ::= l(c); (d(); a(x); \ldots + o(); P_i(c)) \quad \text{for } i = 1, 2$$

with

$$P_1(c) ::= \mathtt{if}\ check(c):\ \overline{ok}.a(x)\ \mathtt{else}\ \overline{ko} \qquad P_2(c) ::= \overline{ko}$$

where for brevity, each name in the process refers to the interaction in the diagram above with the same initial and the ellipses are for inconsequential actions. The expression $check(c)$ in P_1 denotes the actual deterministic verification performed by B to decide if the overdraft should be granted.

Clearly both B_1 and B_2 can be used as implementations of $T_{\mathtt{B}}$ in *partial realisations* of the choreography. For instance, both B_1 and B_2 type-check against $T_{\mathtt{B}}$ considered as a session type[1] (as e.g. in [12]).

Contrariwise, neither B_1 nor B_2 can be used as part of any *complete realisation* of the choreography. This observation is straightforward for B_2, which is unable to interact over *ok* after

[1] This is due to the notion of subtyping for session types [13] which is contra-variant wrt internal choices (and covariant wrt external choices).

receiving an overdraft request. The case for B_1 is more subtle. Fixed a deterministic realisation for the client role $T_{\mathtt{C}}$, B_2 will receive a specific account identification within the login request, namely c in the input $l(c)$. Depending on c, $check(c)$ may return either `true` or `false`. Hence, the implementation will be unable to exhibit both alternative traces for all customers. This will be the case for any possible deterministic implementation of the choreography: just one of the alternative traces will be matched.

Contribution We argue that B_1 and B_2 are not equally appealing when interpreting choreographies as interaction obligations; in fact, B_2 consistently precludes the realisation of one alternative while B_1 realises one or the other alternative (provided that *check* is not the constant map) depending on the deterministic implementation of the role $T_{\mathtt{C}}$.

We introduce *whole-spectrum realisations*, a new notion of realisation tailored to the interpretation of choreographies as interaction obligations. A whole-spectrum realisation of a role R guarantees that, whenever the choreography allows R to make an internal choice, there is a context (i.e., a realisation of the remaining roles) for which R chooses such alternative.

Our contributions are a formalisation of whole-spectrum realisation, and a sound type system that guarantees that typable processes form whole-spectrum realisations. For instance, our type system validates B_1 against $T_{\mathtt{B}}$ while it discards B_2. Typing is decidable if the logic used to express internal conditions is decidable (e.g., Presburger arithmetic). As a technical contribution, we give a denotational semantics of global types in which mandatory and optional behaviours are distinguished. Then, we relate it to the operational semantics of local types (c.f. Theorem 2). Finally, the strong connection between local types and processes ensures that well-typed processes enjoy whole-spectrum realisability (c.f. Theorem 3).

Synopsis § 2 - § 4 give the syntax and semantics of types and processes. § 5 gives a denotational semantics of global types and defines whole-spectrum realisation. § 6 presents the typing rules. Typing checks that processes: (1) syntactically conform to types (i.e., they follow the prescribed interactions and message sorting) and (2) are whole-spectrum realisations. § 7 presents the main technical results. § 8 draws some conclusions and discusses related work. The proofs can be found in [3].

2 Global and Local Types

Our *global types* (§ 2.1) and the corresponding *local types* (§ 2.2) are borrowed with some adaptation from [15]. A main difference is that we replace branching constructs with internal and external choices; this increases the uniformity in our theory. We also borrow the generalised form of sequencing of [17] and, as discussed below, we use a less general, but more tractable form of iteration. Finally, we forbid the parallel composition of local types; this simplification is not a major limitation since global types assume single-threaded roles.

Fix a countable infinite set of *(session channel) names* $\mathbb{C}$ ranged over by $u, y, s, \ldots$ and a set of roles ranged over by $\mathtt{p}, \mathtt{q}, \mathtt{r}, \ldots$. Also, we assume basic data types (called *sorts*) such as `bool` (for booleans), `int` (for integers), `str` (for strings), etc.; we let U to range over (tuples of) sorts.

The cardinality of a set X is denoted by $\#X$. Tuples are written in bold font and, abusing

notation, we use them to represent their underlying set; for instance, if $\mathbf{y} = (y_1, y_2, y_3)$, we write $y_2 \in \mathbf{y}$ to mean $y_2 \in \{y_1, y_2, y_3\}$. Substitutions are denoted by $\{_/_\}$ and when writing $\{\mathbf{y}/\mathbf{s}\}$ we mean 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 Global types

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

(a) Set of channel names $\mathtt{ch}(\mathtt{G}) \subseteq \mathbb{C}$

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

$$
\mathtt{rdy}(\mathtt{G}) = \{\mathtt{p} \mid \mathtt{G} \equiv ((\mathtt{p} \to \mathtt{q} : u\langle \mathtt{U} \rangle; \mathtt{G} + \mathtt{G}'); \mathtt{G}_1 \mid \mathtt{G}_2); \mathtt{G}_3\}
$$

(b) Set of *roles* $\mathcal{P}(\mathtt{G})$ and *ready roles* $\mathtt{rdy}(\mathtt{G})$

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

(c) Set of enabled events $\mathtt{fst}(_)$

Figure 1: Auxiliary functions

A *global type term* $\mathtt{G}$ is derived by the following grammar:

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

In words, a global type term can either be a single interaction, the non-deterministic $(_ + _)$, parallel $(_ \mid _)$, or sequential $(_; _)$ composition of two global type terms, the iteration of a global type term $(_^{*^-})$, or the empty term. We opt for a limited (wrt e.g., [15]) form of recursion based on the Kleene-star (as in [7]), because checking for whole-spectrum realisation requires to statically determine if an iteration terminates (see § 5). In $\mathtt{G}^{*^f}$, f injectively maps roles in $\mathtt{G}$ to $\mathbb{C}$; name $f(\mathtt{p})$ is used to notify role $\mathtt{p} \in \mathtt{G}$ when the iteration finishes.

Given a global type term $\mathtt{G}$, Figure 1 defines the sets $\mathtt{ch}(\mathtt{G}) \subseteq \mathbb{C}$ of names used by $\mathtt{G}$, of (ready) roles of $\mathtt{G}$, and of enabled actions $\mathtt{fst}(\mathtt{G})$, i.e., the input and output actions initially enabled in $\mathtt{G}$. For example, let $\mathtt{G}_{fst} = \mathtt{p} \to \mathtt{q} : y\langle \mathtt{U} \rangle; \mathtt{q} \to \mathtt{s} : z\langle \mathtt{U} \rangle$, then $\mathtt{fst}(\mathtt{G}_{fst}) = \{(\mathtt{p}, \overline{y}), (\mathtt{q}, y), (\mathtt{s}, z)\}$.

A *global type* is defined by an equation $\mathcal{G}(\mathbf{y}) \stackrel{\triangle}{=} \mathtt{G}$ such that the names in $\mathbf{y} \subseteq \mathbb{C}$ are pairwise distinct and $\mathtt{ch}(\mathtt{G}) \subseteq \mathbf{y}$. Hereafter, we write $\mathcal{G}(\mathbf{y})$ when the defining equation of a global type is understood or its corresponding term $\mathtt{G}$ is immaterial; also, abusing notation, we may write $\mathcal{G}$ or $\mathtt{G}$ instead of $\mathcal{G}(\mathbf{y})$ when parameters are clear from the context.

Definition 1 (Structural Congruence)**.** The structural congruence over global type terms is the least congruence $\equiv$ such that $_;_$, $_ \mid _$ and $_ + _$ form a monoid with identity $\mathtt{end}$ and both $_ \mid _$

and $_ + _$ are commutative. Global types $\mathcal{G}_i(\mathbf{y}_i) \stackrel{\triangle}{=} \mathtt{G}_i$ for $i = 1,2$ 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 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 $\mathtt{rdy}(\mathcal{G}) = \mathtt{rdy}(\mathtt{G})$.

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

Definition 2 below adopts the syntactic restrictions on global types given in [17], generalising the condition on the branching construct (along the lines of [11]).

Definition 2 (Well-formedness)**.** A global type $\mathcal{G}(\mathbf{y}) \stackrel{\triangle}{=} \mathtt{G}$ is *well-formed* iff one of the following cases apply

1. $\mathtt{G} = \mathtt{end}$ or, if $\mathtt{G} = \mathtt{p} \to \mathtt{q} : y\langle \mathtt{U} \rangle$ then $\mathtt{p} \neq \mathtt{q}$
2. if $\mathtt{G} = \mathtt{G}_1 \mid \mathtt{G}_2$ then $\mathcal{G}_1(\mathbf{y_1}) \stackrel{\triangle}{=} \mathtt{G}_1$ and $\mathcal{G}_2(\mathbf{y_2}) \stackrel{\triangle}{=} \mathtt{G}_2$ with $\mathbf{y_1} \cap \mathbf{y_2} = \emptyset$ are well-formed, and $\mathcal{P}(\mathtt{G}_1) \cap \mathcal{P}(\mathtt{G}_2) = \emptyset$, i.e., parallel branches are disjoint,
3. if $\mathtt{G} = \mathtt{G}_1 + \ldots + \mathtt{G}_n$ then the following conditions hold
   (a) $\#\mathtt{rdy}(\mathtt{G}) = 1$, i.e., one choosing partner,
   (b) for each $j = 1, \ldots, n$, $\mathcal{G}_j(\mathbf{y}) \stackrel{\triangle}{=} \mathtt{G}_j$ is well-formed, $\mathtt{G}_j$ is a guarded by a prefix whose continuation is well-formed, and $\mathcal{P}(\mathcal{G}_j) = \mathcal{P}(\mathcal{G})$,
   (c) for all $i \neq j \in \{1, \ldots, n\}$, $\mathtt{fst}(\mathtt{G}_i) \cap \mathtt{fst}(\mathtt{G}_j) = \emptyset$, i.e., each partner knows the selected branch when performing its first action,
   (d) for all $\mathtt{p} \neq \mathtt{r} \in \mathcal{P}(\mathcal{G})$, for $i, j \in \{1, \ldots, n\}$, if $(\mathtt{p}, y) \in \mathtt{fst}(\mathtt{G}_i)$ and $\mathtt{r}$ is waiting on y in a sub-term of $\mathtt{G}_j$, then $(\mathtt{r}, y) \notin \mathtt{fst}(\mathtt{G}_j)$ and, in $\mathtt{G}_j$, the first input on y of $\mathtt{r}$ follows an input of $\mathtt{p}$, i.e., there is no race in the use of channels.
4. if $\mathtt{G} = \mathtt{G}_0^{*f}$ then $\mathcal{G}_0(\mathbf{y}) \stackrel{\triangle}{=} \mathtt{G}_0$ is well-formed, $\mathrm{cod}(f) \cap \mathtt{ch}(\mathtt{G}_0) = \emptyset$, if $\mathtt{G}_0 \neq \mathtt{end}$ then
   (a) $\#\mathtt{rdy}(\mathtt{G}_0) = 1$ and $\mathrm{dom}(f) = \mathcal{P}(\mathtt{G}_0) \setminus \mathtt{rdy}(\mathtt{G}_0)$,
   (b) for any two different subterms $\mathtt{G}_1^{*f_1}, \mathtt{G}_2^{*f_2}$ of $\mathtt{G}_0$, $\mathrm{cod}(f_1) \cap \mathrm{cod}(f_2) = \emptyset$ and $\mathrm{dom}(f_1) = \mathrm{dom}(f_2) = \mathrm{dom}(f)$.

Clause (2) requires parallel threads to have disjoint roles and channels, to prevent races on channels. In (3), besides the usual condition on the uniqueness of the selector [17], we allow for a more general form of branching (as in [11]) that does not constraints roles not directly informed from the selector to have the same behaviour in all the branches. In fact, our condition is milder than in [17] as it requires such a role $\mathtt{p}$ to be guarded by inputs on different names in each branch (otherwise $\mathtt{p}$ should not appear in none of the branches). Also, whenever $\mathtt{p}$ is waiting in a branch i on a name y used in input by $\mathtt{r} \neq \mathtt{p}$ in another branch j, there should be an input of $\mathtt{p}$ in the

branch j preceding the one on y of $\mathtt{r}$; this guarantees that $\mathtt{p}$ and $\mathtt{r}$ will not have a race on y (because the former would be aware that either branch j or i had been chosen). This condition and (2) avoid races on channels similarly to linearity in [15] so to rule out choreographies like the one in the following example.

Example 1. Consider $\mathtt{G}_1 + \mathtt{G}_2$ where

$$\mathtt{G}_1 = \mathtt{s} \to \mathtt{q} : x\langle \mathtt{U}\rangle;\ \mathtt{q} \to \mathtt{r} : y\langle \mathtt{U}\rangle;\ \mathtt{q} \to \mathtt{p} : z\langle \mathtt{U}\rangle$$
$$\mathtt{G}_2 = \mathtt{s} \to \mathtt{p} : y\langle \mathtt{U}\rangle;\ \mathtt{p} \to \mathtt{q} : z'\langle \mathtt{U}\rangle;\ \mathtt{q} \to \mathtt{r} : y'\langle \mathtt{U}\rangle$$

If $\mathtt{s}$ chooses $\mathtt{G}_1$, the output on y from $\mathtt{q}$ might be received either by $\mathtt{r}$ or by $\mathtt{p}$ causing, in the latter case, a deadlock. The condition (3)(d) in Definition 2 rules out such choreography since the input of $\mathtt{r}$ in $\mathtt{G}_1$ is not preceded by an input of $\mathtt{p}$. ♦

Clause 4 in Definition 2 is specific to our form of iteration; it requires a unique role to signal the termination of the iteration to any other role $\mathtt{p}$ by using the name $f(\mathtt{p})$. Also, in case of nested iterations, there is no confusion on the names used to signal the termination of each iteration.

2.2 Local types

A *local type term* $\mathtt{T}$ is derived by the following grammar:

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

A local type term is either an internal ($\bigoplus$) or external ($\sum$) guarded choice, the sequential composition of two local type terms $_;_$, the iteration of a term $_^*$, or the empty local type term end. Local type terms cannot be composed in parallel.

The set $\mathsf{ch}(\mathtt{T})$ of channels of $\mathtt{T}$ is defined as follows:

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

A *local type* is defined by an equation $\mathcal{T}(\mathbf{y}) \stackrel{\triangle}{=} \mathtt{T}$ such that the names in $\mathbf{y}$ are pairwise distinct and $\mathsf{ch}(\mathtt{T}) \subseteq \mathbf{y}$. Hereafter, we write $\mathcal{T}(\mathbf{y})$ when the defining equation of a local type is understood or its corresponding term $\mathtt{T}$ is immaterial; also, abusing notation, we may write $\mathcal{T}$ or $\mathtt{T}$ instead of $\mathcal{T}(\mathbf{y})$ when parameters are clear from the context.

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 end as identity, while $_;_$ is associative and has end as identity.

Projection extracts the local types from a global type. We remark that our projection is total on well-formed global types.

Definition 3 (Projection)**.** The projection written $\mathtt{G}\upharpoonright\mathtt{r}$ of a well-formed global type term $\mathtt{G}$ on $\mathtt{r} \in \mathcal{P}(\mathtt{G})$ is a function which returns a local type term as defined in Figure 2. The projection $\mathcal{G}(\mathbf{y})\upharpoonright\mathtt{r}$ of a global type $\mathcal{G}(\mathbf{y}) \stackrel{\triangle}{=} \mathtt{G}$ wrt to a role $\mathtt{r}$ is a local type $\mathcal{T}(\mathbf{y}) \stackrel{\triangle}{=} \mathtt{T}$ where $\mathtt{T} = \mathtt{G}\upharpoonright\mathtt{r}$.

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

Figure 2: Projection of Global Types

All but the clauses for the projections of iteration in Definition 3 are straightforward and inspired by [15]. Each iteration has a unique role $\mathtt{r} \in \mathtt{rdy}(\mathtt{G}_1)$ that decides when to stop the iteration (c.f. clause (4) in Definition 2), and a number of 'passive' roles. Projection sends messages on b_j from $\mathtt{r}$ to each passive role to signal the termination of the iteration.

Example 2. Let $f(\mathtt{q}) = b_1$ and $f(\mathtt{r}) = b_2$. Then

$$
\begin{aligned}
\mathtt{G}\upharpoonright\mathtt{p} &= (y!\mathtt{U})^*; b_1!; b_2!, \\
\mathtt{G}\upharpoonright\mathtt{q} &= (y?\mathtt{U}; z!\mathtt{U})^*; b_1? \\
\mathtt{G}\upharpoonright\mathtt{r} &= (z?\mathtt{U})^*; b_2?
\end{aligned}
$$

are the projections of $\mathtt{G}^{*f}$. ♦

2.3 Behaviour of types

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

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

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

The semantics of specifications abstracts away from the actual values exchanged in communications and is generated by the rules in Figure 3 using the labels

$$
\alpha ::= \overline{u}^n\mathbf{s} \;\mid\; u_i\mathbf{s} \;\mid\; \overline{s}\mathtt{v} \;\mid\; s\mathtt{v} \;\mid\; \tau \tag{4}
$$

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

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

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

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

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

Figure 3: Labelled transitions for specifications

of a value on s, and the silent step. We will use the set $\mathtt{fc}(\alpha)$ of channels of a label α, 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 $\mathtt{fc}(\tau) = \emptyset$. For $\alpha = \overline{s}\mathtt{v}$ or $\alpha = s\mathtt{v}$, we write $\mathtt{obj}(\alpha)$ to denote $\mathtt{v}$.

Intuitively, the rules of Figure 3 dictate how a single role behaves in a session $\mathbf{s}$ and are instrumental for type checking processes as well as for defining the runtime behaviour of specifications (c.f. Figure 5). Rules [TReq] and [TAcc] allow a specification to initiate a new session by projecting (on role $\mathtt{0}$ and $\mathtt{i}$, resp.) the global type associated 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 Systems

Our systems communicate values specified by *expressions* having the following syntax:

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

An expression e is either a variable, or a value, or the composition of expressions (we assume that expressions are implicitly sorted and do not include names). Lists $[e_1, \ldots, e_n]$ and numerical ranges $e_1..e_2$ are used for iteration; in the former case, all the items of a list have the same sort, in the latter case, both expressions are integers and the value of e_1 is smaller or equal than 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).

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

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

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

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

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

The syntax of systems S and processes P is

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

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 $\mathtt{do}\; N\; \mathtt{until}\; b$ repeats N until a signal 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 § 2.2 and § 6). 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 var(_) to systems in the obvious way. The free session names of S, written $\mathtt{fc}(S)$, are defined as:

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

A system S is *closed* when $\mathtt{fc}(S) = \emptyset$ and it is *initial* when S does not contain runtime constructs, namely new session $(\nu s)S'$ and queues $\mathbf{s} : M$. Formally, S is initial iff for each s and S', if $S \equiv (\nu s)S'$ then $s \notin \mathtt{fc}(S')$.

Definition 4 (Structural congruence)**.** 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:

$$(\nu \mathbf{s})\mathbf{0} \equiv \mathbf{0} \qquad (\nu \mathbf{s})(\nu \mathbf{s}')S \equiv (\nu \mathbf{s}')(\nu \mathbf{s})S \qquad \mathtt{end}^{*f} \equiv \mathtt{end}$$

$$(\nu \mathbf{s})(S \mid S') \equiv S \mid (\nu \mathbf{s})S' \quad \text{when } s \notin \mathtt{fc}(S)$$

The operational semantics of systems is in Figure 4. We use a store σ to record the values assigned to variables. We write $e \downarrow \sigma$ for the result of evaluating e when $\mathrm{var}(e) \subseteq \mathrm{dom}(\sigma)$ (we assume $e \downarrow \sigma$ undefined if $\mathrm{var}(e) \nsubseteq \mathrm{dom}(\sigma)$), and $\sigma[x \mapsto \mathtt{v}]$ for the store obtained by updating x with $\mathtt{v}$ in σ. For simplicity, the store is global. Labels are obtained by extending the grammar (4)

(page 9) 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 $\langle _, \sigma \rangle$ has a α-transition to $\langle S', \sigma' \rangle$ provided that $e \downarrow \sigma$ actually holds. Hereafter 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 Figure 4.

$$\frac{\mathbf{s} \notin \mathtt{fc}(P)}{\langle \overline{u}^n(\mathbf{y}).P, \sigma \rangle \xrightarrow{\overline{u}^n \mathbf{s}} \langle P\{\mathbf{y}/\mathbf{s}\}, \sigma \rangle}\,[\mathtt{SReq}] \qquad \frac{\mathbf{s} \notin \mathtt{fc}(P)}{\langle u_i(\mathbf{y}).P, \sigma \rangle \xrightarrow{u_i \mathbf{s}} \langle P\{\mathbf{y}/\mathbf{s}\}, \sigma \rangle}\,[\mathtt{SAcc}]$$

$$\langle s(x).P + N, \sigma \rangle \xrightarrow{s\mathtt{v}} \langle P, \sigma[x \mapsto \mathtt{v}] \rangle [\mathtt{SRec}] \qquad \frac{e \downarrow \sigma = \mathtt{v}}{\langle \overline{s}e, \sigma \rangle \xrightarrow{\overline{s}\mathtt{v}} \langle \mathbf{0}, \sigma \rangle}\,[\mathtt{SSend}]$$

$$\frac{e \downarrow \sigma = \mathtt{true} \quad \langle P, \sigma \rangle \xrightarrow{e' \vdash \alpha} \langle P', \sigma' \rangle}{\langle \mathtt{if}\ e:\ P\ \mathtt{else}\ Q, \sigma \rangle \xrightarrow{e \wedge e' \vdash \alpha} \langle P', \sigma' \rangle}\,[\mathtt{SThen}] \qquad \frac{e \downarrow \sigma = \mathtt{false} \quad \langle Q, \sigma \rangle \xrightarrow{e' \vdash \alpha} \langle Q', \sigma' \rangle}{\langle \mathtt{if}\ e:\ P\ \mathtt{else}\ Q, \sigma \rangle \xrightarrow{\neg e \wedge e' \vdash \alpha} \langle Q', \sigma' \rangle}\,[\mathtt{SElse}]$$

$$\frac{\ell \downarrow \sigma = \varepsilon}{\langle \mathtt{for}\ x\ \mathtt{in}\ \ell\mathtt{:}\ P, \sigma \rangle \xrightarrow{\tau} \langle \mathbf{0}, \sigma \rangle}\,[\mathtt{SFor}_1] \qquad \frac{\neg \ell \downarrow \sigma = \varepsilon \quad \langle P, \sigma[x \mapsto \mathtt{hd}(\ell \downarrow \sigma)] \rangle \xrightarrow{e \vdash \alpha} \langle P', \sigma' \rangle}{\langle \mathtt{for}\ x\ \mathtt{in}\ \ell\mathtt{:}\ P, \sigma \rangle \xrightarrow{e \vdash \alpha} \langle P'; \mathtt{for}\ x\ \mathtt{in}\ \mathtt{tl}(\ell)\mathtt{:}\ P, \sigma' \rangle}\,[\mathtt{SFor}_2]$$

$$\langle \mathtt{do}\ P\ \mathtt{until}\ b, \sigma \rangle \xrightarrow{b} \langle \mathbf{0}, \sigma \rangle [\mathtt{SLoop}_1] \qquad \frac{\langle P, \sigma \rangle \xrightarrow{e \vdash \alpha} \langle P', \sigma' \rangle \quad \alpha \neq b}{\langle \mathtt{do}\ P\ \mathtt{until}\ b, \sigma \rangle \xrightarrow{e \vdash \alpha} \langle P'; \mathtt{do}\ P\ \mathtt{until}\ b, \sigma' \rangle}\,[\mathtt{SLoop}_2]$$

$$\frac{\langle P, \sigma \rangle \xrightarrow{e \vdash \alpha} \langle P', \sigma' \rangle}{\langle P; Q, \sigma \rangle \xrightarrow{e \vdash \alpha} \langle P'; Q, \sigma' \rangle}\,[\mathtt{SSeq}] \qquad \frac{P \equiv P' \quad \langle P', \sigma \rangle \xrightarrow{e \vdash \alpha} \langle Q', \sigma' \rangle \quad Q' \equiv Q}{\langle P, \sigma \rangle \xrightarrow{e \vdash \alpha} \langle Q, \sigma' \rangle}\,[\mathtt{SStruct}]$$

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

$$\frac{\langle S_1, \sigma \rangle \xrightarrow{e \vdash \alpha} \langle S_1', \sigma' \rangle \quad \mathrm{var}(S_1) \cap \mathrm{var}(S_2) = \emptyset}{\langle S_1 \mid S_2, \sigma \rangle \xrightarrow{e \vdash \alpha} \langle S_1' \mid S_2, \sigma' \rangle}\,[\mathtt{SPar}] \qquad \frac{\langle P, \sigma \rangle \xrightarrow{e \vdash \overline{s}\mathtt{v}} \langle P', \sigma' \rangle}{\langle P \mid s : M, \sigma \rangle \xrightarrow{e \vdash \tau} \langle P' \mid s : M \cdot \mathtt{v}, \sigma' \rangle}\,[\mathtt{SCom}_1]$$

$$\frac{\langle P, \sigma \rangle \xrightarrow{e \vdash s\mathtt{v}} \langle P', \sigma' \rangle}{\langle P \mid s : \mathtt{v} \cdot M, \sigma \rangle \xrightarrow{e \vdash \tau} \langle P' \mid s : M, \sigma' \rangle}\,[\mathtt{SCom}_2] \qquad \frac{\langle S, \sigma \rangle \xrightarrow{e \vdash \alpha} \langle S', \sigma' \rangle \quad s \notin \mathtt{fc}(\alpha)}{\langle (\nu s)S, \sigma \rangle \xrightarrow{e \vdash \alpha} \langle (\nu s)S', \sigma' \rangle}\,[\mathtt{SNews}]$$

Figure 4: Labelled transitions for processes (top) and systems (bottom)

Rules [SReq] and [SAcc] are for requesting and accepting new sessions; in the continuations, they substitute **y** with the session names **s** of the newly created session. 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 $\mathtt{v}$. Rules [SSend] is for sending the result of evaluating an expression in the current store. Rules [SThen] and [SElse] handle 'if' statements as expected; their only peculiarity is that the guard is recorded on the label of the transition: this is instrumental for the correspondence between systems and their types (c.f. § 7.1). The remaining rules are standard but for session initialisation. Rule [SInit] allows n roles to synchronise with $\overline{u}^n(\mathbf{y}_0).P_0$; in the continuation of each role i, the bound names $\mathbf{y}_i$ is 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 [$\mathtt{SCom_1}$] and [$\mathtt{SCom_2}$]. Rule [SInit] requires the synchronisation of all roles. Since processes are single-threaded, this is only possible when each process plays exactly one role in that session.

Note that we use σ to extend the scope of names bound by input prefixes to processes following $_;_$ as in Example 3.

Example 3. Let $N = s(x); P + s'(x); P'$. The scope of x in $N; Q$ includes Q and from the semantics in Figure 4 we can infer

$$\frac{\langle s(x); P + s'(x); P', \sigma\rangle \xrightarrow{s\mathtt{v}} \langle P, \sigma[x \mapsto \mathtt{v}]\rangle}{\langle N; Q \mid s : \mathtt{v} \cdot M, \sigma\rangle \xrightarrow{\tau} \langle P; Q \mid s : M, \sigma[x \mapsto \mathtt{v}]\rangle}$$

by rules [SRec] and [$\mathtt{SCom_2}$]. ♦

4 Runtime Types

Local types are extended to model the runtime queues in systems. As in [15], this is formalised with *message contexts* and *runtime types*. A message context $\mathbb{M}$ takes the form $s_1!\mathtt{v}_1; \cdots ; s_n!\mathtt{v}_n[_]$ with $n \geq 0$, that is a (possibly empty) sequence of outputs followed by a hole [_]. A runtime type is either a "type in context", that is a term $\mathbb{M}[\mathcal{T}]$, or a message context $\mathbb{M}$. We quotient message contexts with

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

to allow the swapping of messages in different channels to account for asynchrony. We extend environments so to map session names $\mathbf{s}$ to runtime local 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 $\mathtt{p}$ 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 extending the rules of Figure 3 with those in Figure 5. Intuitively, runtime specifications allow roles to asynchronously interact through queues. Rule [TQueue] takes a message out of a queue. Rules [TCom1] and [TCom2] exploit the semantics of local types in order to establish how runtime specifications send and receives messages; in fact, the transition in their premises are derived with the rules in Figure 3. In rule [TCom1] the local type $\mathcal{T} @\, \mathtt{p}$ sends a message to its message context in session $\mathbf{s}$. In rule [TCom2] a local type $\mathcal{T} @\, \mathtt{q}$ receives a message from another local type within session $\mathbf{s}$. Rule [TInit] initiates a new session adding to Δ the map from the new session $\mathbf{s}$ to the projections of the global type assigned by Γ.

5 Whole-Spectrum Realisation

We give a trace semantics for global types [7, 9]. We use *annotated traces* that distinguish mandatory from optional actions. An annotated trace is a sequence of input or output actions (decorated with the name or the role performing them; in symbols $\langle \mathtt{p}, s!\mathtt{U}\rangle$ and $\langle \mathtt{p}, s?\mathtt{U}\rangle$), some of which can

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

$$\frac{\Gamma \bullet \mathbf{s} : \mathcal{T} \xrightarrow{\overline{s}\mathsf{v}} \Gamma \bullet \mathbf{s} : \mathcal{T}'}{\Gamma \bullet \Delta, \mathbf{s} : \mathbb{M}[\mathcal{T}]@\,\mathsf{p} \xrightarrow{\tau} \Gamma \bullet \Delta, \mathbf{s} : s!\mathsf{v};\mathbb{M}[\mathcal{T}']@\,\mathsf{p}}\ [\mathtt{TCom1}]$$

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

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

Figure 5: Additional labelled transitions (to those of Figure 3) for runtime specifications

$$\frac{}{\epsilon \in \mathcal{R}(\mathsf{end})}[\mathtt{RGEnd}] \qquad \frac{}{\langle \mathsf{p}, s!\mathsf{U}\rangle\langle \mathsf{q}, s?\mathsf{U}\rangle \in \mathcal{R}(\mathsf{p} \to \mathsf{q} : s\langle \mathsf{U}\rangle)}[\mathtt{RGComm}] \qquad \frac{r \in \mathcal{R}(\mathsf{G}_1)}{r \in \mathcal{R}(\mathsf{G}_1 + \mathsf{G}_2)}[\mathtt{RGCh1}]$$

$$\frac{r \in \mathcal{R}(\mathsf{G}_2)}{r \in \mathcal{R}(\mathsf{G}_1 + \mathsf{G}_2)}[\mathtt{RGCh2}] \qquad \frac{r_1 \in \mathcal{R}(\mathsf{G}_1) \quad r_2 \in \mathcal{R}(\mathsf{G}_2)}{r_1 r_2 \in \mathcal{R}(\mathsf{G}_1 \mid \mathsf{G}_2)}[\mathtt{RGPar}] \qquad \frac{r_1 \in \mathcal{R}(\mathsf{G})}{r_1 \in \mathcal{R}(\mathsf{G}^*)}[\mathtt{RG^*1}]$$

$$\frac{r_1 \in \mathcal{R}(\mathsf{G}^*) \quad r_2 \in \mathcal{R}(\mathsf{G})}{[r_1]r_2 \in \mathcal{R}(\mathsf{G}^*)}[\mathtt{RG^*2}] \qquad \frac{r \in \mathcal{R}(\mathsf{G}^*) \quad \mathtt{rdy}(\mathsf{G}) = \{\mathsf{p}\} \quad \mathcal{P}(\mathsf{G}) = \{\mathsf{p}, \mathsf{p}_1, \ldots, \mathsf{p}_n\}}{r\langle \mathsf{p}, f(\mathsf{p}_1)!\mathbf{1}\rangle \ldots \langle \mathsf{p}, f(\mathsf{p}_n)!\mathbf{1}\rangle\langle \mathsf{p}_1, f(\mathsf{p}_1)?\mathbf{1}\rangle \ldots \langle \mathsf{p}_n, f(\mathsf{p}_n)?\mathbf{1}\rangle \quad \in \quad \mathcal{R}(\mathsf{G}^{*^f})}[\mathtt{RGIter}]$$

Figure 6: Runs of a global type (where **1** is the empty type)

be optional. We write $[r]$ to denote the optional sequence r. Hereafter, a trace implicitly denotes the equivalence class of all traces obtained by permuting causally independent actions.[2]

Definition 5 (Runs of a global type)**.** Given a global type term G, the set $\mathcal{R}(\mathsf{G})$ denotes the runs allowed by G and is defined as the least set closed under the rules in Figure 6.

The first four rules are straightforward. Rule [RGPar] takes the sequential execution of the branches as the representative trace of all the interleavings. Rules [RG*1] and [RG*2] unfold an iterative type. Note that $\mathcal{R}(\mathsf{G}^*) = \{r_1, [r_1]r_2, [[r_1]r_2]r_3, \ldots\}$ with $r_i \in \mathcal{R}(\mathsf{G})$. Therefore, the mandatory sequences in $\mathcal{R}(\mathsf{G}^*)$ corresponds exactly to those in $\mathcal{R}(\mathsf{G})$. Rule [RGIter] adds to the unfolding of the iterative type the events associated to its termination: (*i*) the ready role p sends the termination signal to any other role by using the dedicated channels specified by f (i.e., $\langle \mathsf{p}, f(\mathsf{p}_1)!\mathbf{1}\rangle \ldots \langle, f(\mathsf{p}_n)!\mathbf{1}\rangle$), and (*ii*) the waiting roles receive the termination message (i.e., $f(\mathsf{p}_1)?\mathbf{1}\rangle \ldots \langle \mathsf{p}_n, f(\mathsf{p}_n)?\mathbf{1}\rangle$). As for parallel, we just consider one of the possible interleavings for the receive events.

We define the candidate implementation of a global type.

[2] We consider an asynchronous communication model *à la* Lamport [16].

$$\frac{\langle \mathcal{I}^{\iota,u}_{\mathcal{G}}, \sigma\rangle \xrightarrow{e\vdash\tau} \langle \mathcal{I}^{\iota',u}_{\mathcal{G}}, \sigma'\rangle \quad \langle \iota(\mathtt{p}_i), \sigma\rangle \xrightarrow{e'\vdash\alpha} \langle \iota'(\mathtt{p}_i), \sigma'\rangle \quad \mathtt{fc}(\alpha)\cap \mathbf{y} \neq \emptyset \quad r \in \mathcal{R}(\langle \mathcal{I}^{\iota',u}_{\mathcal{G}}, \sigma\rangle) \quad \mathtt{obj}(\alpha) : \mathtt{U}}{\langle \mathtt{p}_i, \alpha\{\mathtt{obj}(\alpha)/\mathtt{U}\}\rangle r \in \mathcal{R}(\langle \mathcal{I}^{\iota,u}_{\mathcal{G}}, \sigma\rangle)}\,[\mathtt{RRIn}]$$

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

$$\frac{\langle \mathcal{I}^{\iota,u}_{\mathcal{G}}, \sigma\rangle \not\rightarrow}{\epsilon \in \mathcal{R}(\langle \mathcal{I}^{\iota,u}_{\mathcal{G}}, \sigma\rangle)}\,[\mathtt{RREnd}]$$

Figure 7: Runs of realisations

Definition 6 (Realisation)**.** Let $\mathcal{G}(\mathbf{y}) \triangleq \mathtt{G}$ be a global type with $\mathcal{P}(\mathcal{G}) = \{\mathtt{p}_1, \ldots, \mathtt{p}_n\}$, ι be a map assigning a process to each $\mathtt{p} \in \mathcal{P}(\mathcal{G})$, and u be a name used to initiate a session for $\mathcal{G}$. A *ι-realisation of $\mathcal{G}$* is a system $\mathcal{I}^{\iota,u}_{\mathcal{G}}$ such that either: (*i*) $\mathcal{I}^{\iota,u}_{\mathcal{G}} \equiv \iota(\mathtt{p}_1) \mid \ldots \mid \iota(\mathtt{p}_n)$ and $\mathbf{y} \cap \mathtt{fc}(\iota(\mathtt{p}_i)) = \emptyset$; or (*ii*) $\mathcal{I}^{\iota,u}_{\mathcal{G}} \equiv \nu\mathbf{y}(\iota(\mathtt{p}_1) \mid \ldots \mid \iota(\mathtt{p}_n) \mid \mathbf{y} : \mathbf{M})$.

A realisation is a system with n processes, each of them implementing one of the roles in the global type. Definition 6 distinguishes two different cases. Configuration (*i*) stands for a system in which the session that implements $\mathcal{G}$ has not been initiated. To simplify the following definitions we require roles not to 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 in a global type to avoid clashes, e.g., $\mathcal{G}(\mathbf{z}) \triangleq \mathtt{G}\{\mathbf{y}/\mathbf{z}\}$. Case (*ii*) is a system that has already initiated the session implementing $\mathcal{G}$. Again for simplicity, we assume that the system and the global type use the same session channels $\mathbf{y}$.

Definition 7 (Runs of realisations)**.** Let $\mathcal{I}^{\iota,u}_{\mathcal{G}}$ be a realisation of $\mathcal{G}(\mathbf{y}) \triangleq \mathtt{G}$ and σ a store. The sets of *runs* of $\mathcal{I}^{\iota,u}_{\mathcal{G}}$ over σ is the least set $\mathcal{R}(\langle \mathcal{I}^{\iota,u}_{\mathcal{G}}, \sigma\rangle)$ closed with respect to the rules in Figure 7. We write $\mathcal{R}(\mathcal{I}^{\iota,u}_{\mathcal{G}})$ for $\mathcal{R}(\langle \mathcal{I}^{\iota,u}_{\mathcal{G}}, \emptyset\rangle)$. The runs of a sets of realisations $\mathbb{I}$ is $\mathcal{R}(\mathbb{I}) = \cup_{I\in\mathbb{I}}\mathcal{R}(I)$.

The rules in Figure 7 are defined by analysing the enabled transitions of $\mathcal{I}^{\iota,u}_{\mathcal{G}}$. Rule [RRIn] corresponds to the case in which the system reduces because some role $\iota(\mathtt{p}_i)$ in the realisation sends or receives a message over a session channel (i.e., α is either $\overline{s}\mathtt{v}$ or $s\mathtt{v}$ with $s \in \mathbf{y}$). We write $\langle \iota(\mathtt{p}_i), \sigma\rangle \xrightarrow{e'\vdash\alpha} \langle \iota'(\mathtt{p}_i), \sigma'\rangle$ meaning that $\iota' = \iota[\mathtt{p}_i \mapsto P]$ with P the continuation of $\iota(\mathtt{p}_i)$ after executing $e' \vdash \alpha$. Since the action α performed by $\iota(\mathtt{p}_i)$ involves a session channel of the global type, an event α associated to the role $\mathtt{p}_i$ 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 [RRExt] takes into account computations that do not involve session channels, i.e., an internal transition τ in a role, a communication over a channel not in $\mathbf{y}$, or a session initiation. This rule allows a process to freely initiate sessions over channels different from u (i.e., sessions that do not corresponds to the global type $\mathtt{G}$). On the contrary, when a role attempts to initiate

a session over u, rule [RRExt] requires all roles in the realisation to initiate the session (this behaviour is imposed by premise $u \notin \mathtt{fc}(\alpha)$). We assume that any role in the realisation will execute exactly one action over the channel u which also matches the role assigned by ι. Nesting 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. Rule [RREnd] is straightforward.

Whole-spectrum realisation is characterised as a relation on annotated traces. The following rules define the operator $\lessdot$, which is used to compare annotated traces.

$$[r] \lessdot \epsilon \qquad r \lessdot r' r \qquad \frac{r \lessdot r'}{[r] \lessdot r'} \qquad \frac{r \lessdot r'}{[r] \lessdot [r']} \qquad \frac{r_1 \lessdot r'_1 \quad r_2 \lessdot r'_2}{r_1 r_2 \lessdot r'_1 r'_2}$$

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

Definition 8 (Whole-spectrum realisation)**.** A set $\mathbb{I}$ of realisations covers a global type G iff $\mathcal{R}(\mathsf{G}) \Subset \mathcal{R}(\mathbb{I})$. A process P is a *whole-spectrum realisation of* $\mathtt{p}_i \in \mathcal{P}(\mathsf{G})$ when there exists a set $\mathbb{I}$ of realisations that covers G s.t. $\mathcal{I}^{\iota,u}_{\mathcal{G}} \in \mathbb{I}$ implies $\mathcal{I}^{\iota,u}_{\mathcal{G}} = \iota(\mathrm{p}_0) \mid \ldots \mid \iota(\mathrm{p}_n)$ and $\iota(\mathrm{p}_i) = P$.

A whole-spectrum realisation 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 body of the iteration at least once but possibly many times.

Remark 1. A set covering a global type $\mathcal{G}$ can exhibit more behaviour than the runs of $\mathcal{G}$. Nevertheless, we use whole-spectrum realisations with the usual soundness requirement (defined in § 7.1) to characterise valid realisations.

6 Typing rules

Systems are typed by judgements of the form

$$C \llcorner \Gamma \ \vdash \ S \triangleright \Delta \divideontimes \Gamma'$$

that stipulates that S is typed as Δ and yields Γ' under Γ and C. Environments Γ and Δ are as in § 2.3. C is a *context assumption*, that is a logical formula derivable by the grammar

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

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

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

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

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

$$\frac{\Gamma(e){=}\mathtt{U} \qquad y \in \mathbf{y}}{C \smile \Gamma \;\vdash\; \overline{y}e \triangleright \mathbf{y} : y!\mathtt{U} \;\divideontimes\; \Gamma}\,[\mathtt{VSend}]$$

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

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

$$\frac{\Delta(\mathbf{s}) = \mathtt{end} \quad \forall \mathbf{s} \in \mathrm{dom}(()\Delta)}{C \smile \Gamma \;\vdash\; \mathbf{0} \triangleright \Delta \;\divideontimes\; \Gamma}\,[\mathtt{VEnd}]$$

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

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

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

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

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

Figure 8: Typing rules for processes

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

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

Figure 9: Combination of compatible types $\Delta_1 \bowtie \Delta_2$; note that $\Delta \bowtie \Delta = \Delta$ and $\mathrm{dom}(\Delta_1 \bowtie \Delta_2) = \mathrm{dom}(\Delta_1) \cup \mathrm{dom}(\Delta_2)$

We first consider the typing rules for processes in Figure 8. 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 role $\mathtt{0}$. Dually, rule [VAcc] types the acceptance of a session request as role i.

Rule [VRec] types an external choice $P = \sum_{i \in I} y_i(x_i); P_i$ and its premise checks that each branch P_i can be typed against the respective continuation of the type, once Γ is updated with the bound variable x_i. The resulting $\Gamma' = \bigcap_{i \in I} \Gamma_i$ is defined so that $\Gamma'(x) = \mathtt{U}$ iff $\Gamma_i(x) = \mathtt{U}$ for all $i \in I$ and $\Gamma'(u) = \mathcal{G}$ iff $\Gamma_i(u) = \mathcal{G}$ for all $i \in I$ (note that $\mathrm{dom}(\Gamma') = \cap_{i \in I} \mathrm{dom}(\Gamma_{i \in I})$); this ensures that the continuation of P, if any, does not use variables defined only in some of the branches of P, as these variable may be vacuous when the other branches are executed. Finally, if $\mathtt{fv}(P) \cup \mathtt{fc}(P) \not\subseteq \mathrm{dom}(\Gamma')$ then [VRec] cannot be applied and the validation fails.

Rule [VSend] types the send process with a send type in the corresponding session (i.e., $y \in \mathbf{y}$).

Rules [VIf] and [VElse] handle the cases in which the guard of the conditional statement is either a tautology or a contradiction. Rule [VCond] stands for the cases in which both branches can be selected by fixing a proper context assumption (i.e., both $C \wedge e$ and $C \wedge \neg e$ are consistent). Note that the context assumption 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 Figure 9. Notice that the first clause of $\mathcal{T}_1 \bowtie \mathcal{T}_2$ composes the two branches as an internal choice of the actions they perform. We remark that $\bowtie$ is defined only when Δ_1 and Δ_2 are *compatible*, namely iff for all $\mathbf{s}_1 \in \mathrm{dom}(\Delta_1), \mathbf{s}_2 \in \mathrm{dom}(\Delta_2)$, either $\mathbf{s}_1 \cap \mathbf{s}_2 = \emptyset$ or $\mathbf{s}_1 = \mathbf{s}_2$.

Rule [VFor1] assigns the type $\mathcal{T}^*$ to a for loop when its body P has type $\mathcal{T}$ under the context assumption 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 $\mathcal{T}^*; b?$ 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 (i.e., $\mathbf{y} : \mathcal{T}^*$ and $\mathbf{y} : \mathcal{T}^*; b?$, respectively), 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 [VSeq] checks sequential composition. Here $\Delta_1; \Delta_2$ is the pointwise sequential composition of Δ_1 and Δ_2, i.e., $(\Delta_1; \Delta_2)(\mathbf{s}) = \mathcal{T}_1; \mathcal{T}_2$ where $\mathcal{T}_i = \Delta_i(\mathbf{s})$ if $\mathbf{s} \in \mathrm{dom}(\Delta_i)$ and $\mathcal{T}_i = \mathtt{end}$ otherwise, for $i = 1, 2$. Note that P_2 is typed under the environment Γ_1, which contains the names

bound by the input prefixes of P_1. Rule [VEnd] types idle processes with a Δ that maps each session $\mathbf{s}$ to the $\mathtt{end}$ type.

The typing rules for systems extend those for processes and are borrowed from [15]. We relegate them to Appendix A.

7 Properties of the type system

7.1 Soundness

The typing rules in § 6 ensure the semantic conformance of processes with the behaviour prescribed by their types. Here, we define conformance in terms of *conditional weak 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. Namely, systems are not considered responsible when receiving ill-typed messages. Define $\stackrel{\alpha}{\Longrightarrow} = \stackrel{\tau}{\longrightarrow}^* \stackrel{\alpha}{\longrightarrow}$. Hereafter, $\Gamma \bullet \Delta \stackrel{s}{\Longrightarrow}$ stands for $\Gamma \bullet \Delta \stackrel{s\mathtt{v}}{\Longrightarrow} \Gamma \bullet \Delta'$, for some Δ', $\mathtt{v}'$.

Definition 9 (Conditional simulation)**.** A relation $\mathbb{R}$ between states and specifications is a *(weak) conditional simulation* iff for any $(\langle S, \sigma\rangle, \Gamma \bullet \Delta) \in \mathbb{R}$, if $\langle S, \sigma\rangle \xrightarrow{e \vdash \alpha} \langle S', \sigma'\rangle$ then

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

We write $\langle S, \sigma\rangle \lesssim \Gamma \bullet \Delta$ if there exists a conditional weak simulation $\mathbb{R}$ such that $(\langle S, \sigma\rangle, \Gamma \bullet \Delta) \in \mathbb{R}$.

By (1), only inputs of S with the expected type have to be matched by $\Gamma \bullet \Delta$ (recall rule [TRec] in Figure 3), while it is no longer expected to conform to the specification after an ill-typed input (i.e., not allowed by $\Gamma \bullet \Delta$).

Definition 10 establishes consistency for stores in terms of preservation of variables' sorts.

Definition 10 (Consistent store)**.** Given an environment Γ, a context assumption C, and a state $\langle S, \sigma\rangle$ with $\mathrm{var}(S) \subseteq \mathrm{dom}(\sigma)$, store σ is *consistent for S wrt Γ and C* iff $\forall x \in \mathrm{dom}(\sigma)$, $\sigma(x) : \Gamma(x)$, and $C \downarrow \sigma = \mathtt{true}$.

Theorem 1 (Subject reduction)**.** *Assume that*

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

with σ consistent for S wrt Γ and C. Then

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

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

$$\frac{r \in \mathcal{R}_{\mathbf{s}}(\Delta)}{r \in \mathcal{R}_{\mathbf{s}}(\Delta, \mathbf{s} : \mathtt{end}@\mathtt{p})}[\mathtt{RTEnd1}] \qquad \epsilon \in \mathcal{R}_{\mathbf{s}}(\emptyset)[\mathtt{RTEnd2}] \qquad \frac{r \in \mathcal{R}_{\mathbf{s}}(\Delta, \mathbf{s} : \mathtt{T}_i;\mathtt{T}_j@\mathtt{p})}{r \in \mathcal{R}_{\mathbf{s}}(\Delta, \mathbf{s} : \mathtt{T}_i^*;\mathtt{T}_j@\mathtt{p})}[\mathtt{RTIt1}]$$

$$\frac{rr' \in \mathcal{R}_{\mathbf{s}}(\Delta, \mathbf{s} : \mathtt{T}_i;\mathtt{T}_i^*;\mathtt{T}_j@\mathtt{p}) \qquad r' \in \mathcal{R}_{\mathbf{s}}(\Delta\mathbf{s} : \mathtt{T}_i;\mathtt{T}_j@\mathtt{p})}{[r]r' \in \mathcal{R}_{\mathbf{s}}(\Delta, \mathbf{s} : \mathtt{T}_i^*;\mathtt{T}_j@\mathtt{p})}[\mathtt{RTIt2}] \qquad \frac{r \in \mathcal{R}_{\mathbf{s}}(\Delta, \mathbf{s} : s_j!\mathtt{U}_j;\mathtt{T}_j@\mathtt{p}) \qquad j \in I}{r \in \mathcal{R}_{\mathbf{s}}(\Delta, \mathbf{s} : \bigoplus_{i\in I} s_i!\mathtt{U}_i;\mathtt{T}_i@\mathtt{p})}[\mathtt{RTCh}]$$

Figure 10: Runs of Runtime Local Types.

Proof. By induction on the proof of the judgement. □

Corollary 1 (Soundness)**.** *If* $C \lrcorner \Gamma \ \vdash \ S \triangleright \Delta \divideontimes \Gamma'$ *then* $\langle S, \sigma\rangle \precsim \Gamma \bullet \Delta$ *for all* σ *consistent store for* S *wrt* Γ *and* C*.*

Proof. Soundness follows from showing that

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

is a conditional weak simulation, which is straightforward from Theorem 1. □

7.2 Whole-spectrum realisation by typing

We show that well-typed processes are whole-spectrum realisations (Definition 8). First, we introduce the notion of traces of session environments (as defined in § 2.3).

Definition 11 (Runs of runtime types)**.** Given an environment Δ, the set $\mathcal{R}_{\mathbf{s}}(\Delta)$ denotes the traces of events over the channels in **s** generated by Δ, and its inductively defined by the rules in Figure 10.

Rule [RTCom] builds the traces for two communicating types. Rules [RTIt1] and [RTIt2] unfold the traces of an iterative type. Note that the mandatory actions of traces associated to recursive types are those requiring at least one execution of the iteration body, while additional executions are optional. Remaining rules are self-explanatory.

The next two results show that the denotational semantics coincides with the operational rules given in Figures 3 and 5.

Lemma 1. $\Gamma \bullet \Delta \overset{\tau}{\Longrightarrow} \Gamma \bullet \Delta'$ *(with a communication over* s_k*), then there exists* $r \in \mathcal{R}_{\mathbf{s}}(\Delta)$ *such that* $r = \langle \mathtt{p}, s_k!\mathtt{U}_k\rangle\langle \mathtt{q}, s_k?\mathtt{U}_k\rangle r'$ *with* $r' \in \mathcal{R}_{\mathbf{s}}(\Delta')$*.*

Lemma 2. *Let* $r \in \mathcal{R}_{\mathbf{s}}(\Delta)$ *then either* (*i*) $\Gamma \bullet \Delta(\mathbf{s}) \nrightarrow$ *or* (*ii*) $\Gamma \bullet \Delta \overset{\tau}{\Longrightarrow} \Gamma \bullet \Delta'$ *(by a communication over* s_k*),* $r = \langle \mathtt{p}, s_k!\mathtt{U}_k\rangle\langle \mathtt{q}, s_k?\mathtt{U}_k\rangle r'$*, and* $r' \in \mathcal{R}_{\mathbf{s}}(\Delta')$*.*

Above results follow by induction on the derivation of, respectively, $\Gamma \bullet \Delta \stackrel{\tau}{\Longrightarrow} \Gamma \bullet \Delta'$ and $r \in \mathcal{R}_{\mathbf{s}}(\Delta)$.

We now extend the definition of coverage for local types.

Definition 12. A map Δ covers a global type $\mathcal{G}(\mathbf{s})$ if $\mathcal{R}(\mathcal{G}(\mathbf{s})) \Subset \mathcal{R}_{\mathbf{s}}(\Delta)$.

The next result ensures that any well-formed global type is covered by the set of its projections.

Theorem 2. *Given a global type $\mathcal{G}(\mathbf{s})$ then the environment $\Delta = \{\mathbf{s} : (\mathcal{G}(\mathbf{s}) \upharpoonright \mathtt{p}) @ \mathtt{p}\}_{\mathtt{p} \in \mathcal{P}(\mathcal{G}(\mathbf{s}))}$ covers $\mathcal{G}(\mathbf{s})$.*

Proof. It follows by induction on the derivation of $r \in \mathcal{R}(\mathcal{G})$. When last applied rule is [RGCh1] or [RGCh2] we use an auxiliary result stating that a projection of a well-formed global choice can only execute the guard corresponding to the branch chosen by the selector. For iteration, we show that the annotated traces are in one to one correspondence. □

We now show that any well-typed realisation covers its specification.

Theorem 3. *Let $\mathcal{G}(\mathbf{s}) \stackrel{\triangle}{=} \mathtt{G}$ be a global type and $\mathcal{I}^{\iota,u}_{\mathcal{G}}$ a realisation such that $\mathtt{true} \mathbin{\llcorner\!\lrcorner} \Gamma, u : \mathcal{G}(\mathbf{s}) \vdash \iota(\mathtt{p}) \triangleright \Delta, \mathbf{s} : \mathcal{G}(\mathbf{s}) \upharpoonright \mathtt{p} \divideontimes \Gamma'$ for all $\mathtt{p} \in \mathcal{P}(\mathtt{G})$. Then,*

$$\mathcal{R}_{\mathbf{s}}(\{\mathbf{s} : (\mathcal{G}(\mathbf{s}) \upharpoonright \mathtt{p}) @ \mathtt{p}\}_{\mathtt{p} \in \mathcal{P}(\mathcal{G})}) \quad \Subset \quad \mathcal{R}(\mathcal{I}^{\iota,u}_{\mathcal{G}})$$

Proof. By induction on the derivation of $r \in \mathcal{R}_{\mathbf{s}}(\{\mathbf{s} : (\mathcal{G}(\mathbf{s}) \upharpoonright \mathtt{p}) @ \mathtt{p}\}_{\mathtt{p} \in \mathcal{P}(\mathcal{G})})$. Interesting cases are: (*i*) internal choice, but in this case the typing rules for conditional processes ensure that all branches of the internal choice can be selected; and (*ii*) iteration, where we show that each annotated trace of the type is mimicked by a bounded unfolding of the corresponding iterative process. □

8 Conclusion and related work

We introduced the notion of *whole-spectrum realisation* according to which the deterministic implementation of a role of a choreography cannot persistently avoid the execution of a branch of an internal choice specified by that choreography. Although whole-spectrum realisation is defined as a relation between the execution 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 $\mathcal{G}$ (Theorem 2); and (*ii*) any trace of a local type can be mimicked by a well-typed implementation, if interacting in a proper context (Theorem 3). In addition, the soundness of our type system (Corollary 1) ensures that well-typed realisations behave as prescribed by the choreography.

Existing theories on contracts and behavioural types [1, 4, 6, 8, 15] feature subtyping in which a subtype may add (i.e., more external choices) and/or remove behaviour (i.e., less internal choices). We do not consider subtyping because the liberal elimination of internal choices prevents complete realisations. Hence, our type system is more restrictive than, e.g., [15]. Nevertheless, it may enjoy the same safety and progress properties (details are omitted due to space limitation).

Our treatment of choices is similar to the abstraction relation in [5], where valid refinements of specifications are characterised by a *simulation-based relation* that preserves choices. The formal study of whether our type system preserves the abstraction relation in [5] is left as a future work. To some extend our proposal is related to the fair subtyping approach in [19], where refinement is studied under the fairness assumption. This approach differs from ours since fair subtyping differs from usual subtyping only when considering infinite computations. Reversely, whole-spectrum realisation differs from usual realisation also when considering finite processes.

The static verification of whole-spectrum realisation requires a more restrictive form of recursion than the one in [1, 15], 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. We argue that this is attainable using annotations [2, 10, 21] to detect loop termination by typing.

Bibliography

[1] 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*, LNCS 5201. Springer, 2008.

[2] L. Bocchi, K. Honda, E. Tuosto, and N. Yoshida. A theory of design-by-contract for distributed multiparty interactions. In *CONCUR*, LNCS 6269, 2010.

[3] L. Bocchi, H. Melgratti, and E. Tuosto. Choice-preserving multiparty session types. http://www.cs.le.ac.uk/people/lb148/WSR.html.

[4] M. Bravetti and G. Zavattaro. A theory of contracts for strong service compliance. *MSCS*, 19(3):601–638, 2009.

[5] M. G. Buscemi and H. C. Melgratti. Abstract processes in orchestration languages. In *ESOP*, LNCS 5502. Springer, 2009.

[6] L. Caires and H. T. Vieira. Conversation types. In *ESOP*, LNCS 5502. Springer, 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*, LNCS 5710, 2009.

[9] T.-C. Chen and K. Honda. Specifying stateful asynchronous properties for distributed programs. In *CONCUR*, LNCS 7454, 2012.

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

[11] P.-M. Deniélou and N. Yoshida. Dynamic multirole session types. In *POPL*. ACM, 2011.

[12] M. Dezani-Ciancaglini and U. de'Liguoro. Sessions and session types: An overview. In *WS-FM*, LNCS 6194, pages 1–28, 2009.

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

[14] K. Honda, V. T. Vasconcelos, and M. Kubo. Language primitives and type discipline for structured communication-based programming. In *ESOP*, volume 1381 of *LNCS*, pages 122–138. Springer, 1998.

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

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

[17] J. Lange and E. Tuosto. Synthesising choreographies from local session types. In *CONCUR*, LNCS 7454, 2012.

[18] N. Lohmann and K. Wolf. Decidability results for choreography realization. In *ICSOC*, LNCS 7084. Springer, 2011.

[19] L. Padovani. Fair subtyping for multi-party session types. In *COORDINATION*, LNCS 6721. Springer, 2011.

[20] J. Su, T. Bultan, X. Fu, and X. Zhao. Towards a theory of web service choreographies. In *WS-FM*, LNCS 4937, pages 1–16. Springer, 2007.

[21] N. Yoshida, M. Berger, and K. Honda. Strong normalisation in the pi-calculus. In *LICS*. IEEE Computer Society, 2001.

A Typing rules for systems

The typing rules for systems in Figure 11 extend those for processes, they are borrowed from [15], and use non-singleton assignments as defined in § 2. Rule [VPar] for parallel composition uses the composition of mappings given below. If Δ_1 and Δ_2 are compatible, their composition is

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

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

$$\{\mathcal{R}_\mathtt{p} @ \mathtt{p}\}_{\mathtt{p} \in I} \circ \{\mathcal{R}'_\mathtt{q} @ \mathtt{q}\}_{\mathtt{q} \in J} = \begin{array}{l} \{\mathcal{R}_\mathtt{p} @ \mathtt{p} \circ \mathcal{R}'_\mathtt{p} @ \mathtt{p}\}_{\mathtt{p} \in I \cap J} \\ \cup \{\mathcal{R}_\mathtt{p} @ \mathtt{p}\}_{\mathtt{i} \in I \setminus J} \cup \{\mathcal{R}_\mathtt{q} @ \mathtt{q}\}_{\mathtt{q} \in J \setminus I} \end{array}$$

if $\mathcal{R}_\mathtt{p} \circ \mathcal{R}'_\mathtt{p}$ is defined for all $\mathtt{p} \in I \cap J$ and it is undefined otherwise. Note that $\mathrm{dom}(\Delta) = \mathrm{dom}(\Delta_1) \cup \mathrm{dom}(\Delta_2)$.

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

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

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

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

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

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

Figure 11: Typing rules for systems

MEALS Partner Abbreviations

SAU: Saarland University, D

RWT: RWTH Aachen University, D

TUD: Technische Universität Dresden, D

INR: Institut National de Recherche en Informatique et en Automatique, FR

IMP: Imperial College of Science, Technology and Medicine, UK

ULEIC: University of Leicester, UK

TUE: Technische Universiteit Eindhoven, NL

UNC: Universidad Nacional de Córdoba, AR

UBA: Universidad de Buenos Aires, AR

UNR: Universidad Nacional de Río Cuarto, AR

ITBA: Instituto Técnológico Buenos Aires, AR