A very gentle introduction for synchronous multiparty sessions

Nobuko Yoshida[a]

[a] *Imperial College London, UK*

Keywords: Concurrency, Process Calculi, Multiparty Session Types

1. A Gentle Introduction to Multiparty Session Types

The aim of this section is to give a gentle introduction of multiparty session types for readers who are interested in programming but who are not familiar with session types nor process calculi.

Session types are introduced in a series of papers during the 1990s Honda (1993); Takeuchi et al. (1994); Honda et al. (1998) in the context of pure concurrent processes and programming. Session types have since been studied in many contexts over the last decade—see the surveys of the field Hüttel et al. (2016); Gay and Ravera (2017).

We review multiparty session types, a methodology to enable compositional reasoning about communication.

As a simple example, consider a scenario in which a cart and arm assembly has to fetch objects. We associate a process with each physical component; thus, we model the scenario using a *cart* (Cart) and an *arm* (Arm) attached to the cart. The task involves synchronisation between the cart and the arm. Synchronisation is obtained through the exchange of messages.

Specifically, the protocol works as follows.

1. The cart sends the arm a fold command *fold*. On receiving the command, the arm folds itself. When the arm is completely folded, it sends back a message *ok* to the cart. On receipt of this message, the cart moves.

2. When the cart reaches the object, it stops and sends a *grab* message to the arm to grab the object. While the cart waits, the arm executes the grabbing operation, followed by a folding operation. Then the arm sends a message *ok* to the cart. This sequence may need to be repeated.

3. When all tasks are finished, the cart sends a message *done* to the arm, and the protocol terminates.

The multiparty session types methodology is as follows. First, define a *global type* that gives a shared contract of the allowed pattern of message exchanges in the system. Second, *project* the global type to each end-point participant to get a *local type*: an obligation on the message sends and receipts for each process that together ensure that the pattern of messages are allowed by the global type. Finally, check that the implementation of each process conforms to its local type.

In our protocol, from a global perspective, we expect to see the following pattern of message exchanges, encoded as a *global type* for the communication:

$$\mu \mathbf{t}.\mathsf{Cart} \to \mathsf{Arm}:\{\mathit{fold}.\mathsf{Arm} \to \mathsf{Cart}:\mathit{ok}.\mathsf{Cart} \to \mathsf{Arm}:\mathit{grab}.\mathsf{Arm} \to \mathsf{Cart}:\mathit{ok}.\mathbf{t},\ \mathit{done}.\mathtt{end}\} \tag{1}$$

The type describes the global pattern of communication between Cart and Arm using message exchanges, sequencing, choice, and repetition. The basic pattern $\mathsf{Cart} \to \mathsf{Arm}:m$ indicates a message m sent from the Cart to the Arm. The communication starts with the cart sending either a *fold* or a *done* command to the arm. In case of *done*, the protocol ends (type `end`); otherwise, the communication continues with the sequence *ok*. *grab*. *ok* followed by a repetition of the entire pattern. The operator "." denotes sequencing, and the type $\mu \mathbf{t}.T$ denotes recursion of T.

The global type states what are the valid message sequences allowed in the system. When we implement Cart and Arm separately, we would like to check that their composition conforms to the global type. We can perform this check compositionally as follows.

Since there are only two participants, projecting to each participant is simple. From the perspective of the Cart, the communication can be described by the type:

$$\mu \mathbf{t}.\,((\ !\mathit{fold}.\ ?\mathit{ok}.\ !\mathit{grab}.\ ?\mathit{ok}.\mathbf{t}\ )\quad \oplus \quad (\ !\mathit{done}.\ \mathtt{end}\ )) \tag{2}$$

where $!m$ denotes a message m sent (to the Arm) and $?m$ denotes a message m received from the Arm. and $\oplus$ denotes an (internal) choice. Thus, the type states that Cart repeats actions $!\mathit{fold}.\ ?\mathit{ok}.\ !\mathit{grab}.\ ?\mathit{ok}$ until at some point it sends *done* and exits.

Dually, from the viewpoint of the Arm, the same global session is described by the dual type

$$\mu \mathbf{t}.\,((\ \ ?\mathit{fold}.\ !\mathit{ok}.\ ?\mathit{grab}.\ !\mathit{ok}.\mathbf{t}\ )\quad \&\quad (\ ?\mathit{done}.\ \mathtt{end}\,)) \tag{3}$$

in which & means that a choice is offered externally.

We can now individually check that the implementations of the cart and the arm conform to these local types.

The global type seems overkill if there are only two participants; indeed, the global type is uniquely determined given the local type (2) or its dual (3). However, for applications involving *multiple parties*, the global type and its projection to each participant are essential to provide a shared contract among all participants.

For example, consider a simple ring protocol, where the Arm process above is divided into two parts, Lower and Upper. Now, Cart sends a message *fold* to the lower arm Lower, which forwards the message to Upper. After receiving the message, Upper sends an acknowledgement *ok* to Cart. We start by specifying the global type as:

$$\mathsf{Cart} \to \mathsf{Lower}\colon \mathit{fold}.\mathsf{Lower} \to \mathsf{Upper}\colon \mathit{fold}.\mathsf{Upper} \to \mathsf{Cart}\colon \mathit{ok}.\mathtt{end} \tag{4}$$

As before, we want to check each process locally against a local type such that if each process conforms to its local type then the composition satisfies the global type.

The global type in (4) is *projected* into the three endpoint session types:

Cart's endpoint type:	$\mathsf{Lower}!\mathit{fold}.\mathsf{Upper}?\mathit{ok}.\mathtt{end}$
Lower's endpoint type:	$\mathsf{Cart}?\mathit{fold}.\mathsf{Upper}!\mathit{fold}.\mathtt{end}$
Upper's endpoint type:	$\mathsf{Lower}?\mathit{fold}.\mathsf{Cart}!\mathit{ok}.\mathtt{end}$

where $\mathsf{Lower}!\mathit{fold}$ means "send to Lower a *fold* message," and $\mathsf{Upper}?\mathit{ok}$ means "receive from Upper an *ok* message." Then each process is type-checked against its own endpoint type. When the three processes are executed, their interactions automatically follow the stipulated scenario.

If instead of a global type, we only used three separate binary session types to describe the message exchanges between Cart and Lower, between Lower and Upper, and between Upper and Cart, respectively, without using a global type, then we lose essential sequencing information in this interaction scenario. Consequently, we can no longer guarantee deadlock-freedom among these three parties. Since the three separate binary sessions can be interleaved freely, an implementation of the Cart that conforms to $\mathsf{Upper}?\mathit{ok}.\mathsf{Lower}!\mathit{fold}.\mathtt{end}$ becomes typable. This causes the situation that each of the three parties blocks indefinitely while waiting for a message to be delivered. Thus, we shall use the power of multiparty session types to ensure correct communication patterns.

2. Synchronous Multiparty Session Calculus

This section introduces the syntax and semantics of a synchronous multiparty session calculus. Since our focus is on subtyping, we simplify the calculus in (Kouzapas and Yoshida, 2013) eliminating both shared channels for session initiations and session channels for communications inside sessions — i.e, our calculus is akin to value-passing CCS (Milner, 1989, Chapter 2.8).

$$\mathtt{succ}(\mathsf{n}) \downarrow (\mathsf{n}+1) \qquad \mathtt{neg}(\mathsf{i}) \downarrow (-\mathsf{i}) \qquad \neg\mathsf{true} \downarrow \mathsf{false} \quad \neg\mathsf{false} \downarrow \mathsf{true} \qquad \mathsf{v} \downarrow \mathsf{v}$$

$$(\mathsf{i}_1 > \mathsf{i}_2) \downarrow \begin{cases} \mathsf{true} & \text{if } \mathsf{i}_1 > \mathsf{i}_2, \\ \mathsf{false} & \text{otherwise} \end{cases} \qquad \frac{\mathsf{e}_1 \downarrow \mathsf{v}}{\mathsf{e}_1 \oplus \mathsf{e}_2 \downarrow \mathsf{v}} \qquad \frac{\mathsf{e}_2 \downarrow \mathsf{v}}{\mathsf{e}_1 \oplus \mathsf{e}_2 \downarrow \mathsf{v}} \qquad \frac{\mathsf{e} \downarrow \mathsf{v} \quad \mathcal{E}(\mathsf{v}) \downarrow \mathsf{v}'}{\mathcal{E}(\mathsf{e}) \downarrow \mathsf{v}'}$$

Table 1: Expression evaluation.

$$\text{[S-REC]}\quad \mu X.P \equiv P\{\mu X.P/X\} \qquad \text{[S-MULTI]}\quad P \equiv Q \Rightarrow \mathsf{p} \triangleleft P \mid \mathcal{M} \equiv \mathsf{p} \triangleleft Q \mid \mathcal{M}$$

$$\text{[S-PAR 1]}\quad \mathsf{p} \triangleleft \mathbf{0} \mid \mathcal{M} \equiv \mathcal{M} \qquad \text{[S-PAR 2]}\quad \mathcal{M} \mid \mathcal{M}' \equiv \mathcal{M}' \mid \mathcal{M} \qquad \text{[S-PAR 3]}\quad (\mathcal{M} \mid \mathcal{M}') \mid \mathcal{M}'' \equiv \mathcal{M} \mid (\mathcal{M}' \mid \mathcal{M}'')$$

Table 2: Structural congruence.

Notation 2.1 (Base sets)**.** *We use the following base sets:* values*, ranged over by* $\mathsf{v}, \mathsf{v}', \ldots$*;* expressions*, ranged over by* $\mathsf{e}, \mathsf{e}', \ldots$*;* expression variables*, ranged over by* $x, y, z \ldots$*;* labels*, ranged over by* $\ell, \ell', \ldots$*;* session participants*, ranged over by* $\mathsf{p}, \mathsf{q}, \ldots$*;* process variables*, ranged over by* $X, Y, \ldots$*;* processes*, ranged over by* $P, Q, \ldots$*; and* multiparty sessions*, ranged over by* $\mathcal{M}, \mathcal{M}', \ldots$*.*

Syntax. A value v can be a natural number n, an integer i, or a boolean true / false. An expression e can be a variable, a value, or a term built from expressions by applying the operators $\mathtt{succ}, \mathtt{neg}, \neg, \oplus$, or the relation $>$. An *evaluation context* $\mathcal{E}$ is an expression with exactly one hole. The operator $\oplus$ models non-determinism: $\mathsf{e}_1 \oplus \mathsf{e}_2$ is an expression that might yield either e_1 or e_2.

The processes of the synchronous multiparty session calculus are defined by:

$$P \quad ::= \quad \mathsf{p}!\ell(\mathsf{e}).P \;\mid\; \sum_{i \in I} \mathsf{p}?\ell_i(x_i).P_i \;\mid\; \mathsf{if}\ \mathsf{e}\ \mathsf{then}\ P\ \mathsf{else}\ P \;\mid\; \mu X.P \;\mid\; X \;\mid\; \mathbf{0}$$

The output process $\mathsf{p}!\ell(\mathsf{e}).Q$ sends the value of expression e with label ℓ to participant p. The sum of input processes (external choice) $\sum_{i \in I} \mathsf{p}?\ell_i(x_i).P_i$ is a process that can accept a value with label ℓ_i from participant p, for any $i \in I$.According to the label ℓ_i of the received value, the variable x_i is instantiated with the value in the continuation process P_i. We assume that the set I is always finite and non-empty.

The conditional process $\mathsf{if}\ \mathsf{e}\ \mathsf{then}\ P\ \mathsf{else}\ Q$ represents the internal choice between processes P and Q. Which branch of the conditional process will be taken depends on the evaluation of the expression e. The process $\mu X.P$ is a recursive process. We assume that the recursive processes are *guarded*. For example, $\mu X.\mathsf{p}?\ell(x).X$ is a valid process, while $\mu X.X$ is not. We often omit $\mathbf{0}$ from the tail of processes.

We define a *multiparty session* as a parallel composition of pairs (denoted by $\mathsf{p} \triangleleft P$) of participants and processes:

$$\mathcal{M} \quad ::= \quad \mathsf{p} \triangleleft P \;\mid\; \mathcal{M} \mid \mathcal{M}$$

with the intuition that process P plays the role of participant p, and can interact with other processes playing other roles in $\mathcal{M}$. A multyparty session is *well formed* if all its participants are different. We consider only well-formed multiparty sessions.

Operational semantics. *The value* v *of expression* e (notation $\mathsf{e} \downarrow \mathsf{v}$) is computed as expected, see Table 1. The successor operation $\mathtt{succ}$ is defined only on natural numbers, the negation $\mathtt{neg}$ is defined on integers, and $\neg$ is defined only on boolean values. The internal choice $\mathsf{e}_1 \oplus \mathsf{e}_2$ evaluates either to the value of e_1 or to the value of e_2.

$$\text{[R-COMM]}\quad \frac{j \in I \qquad \mathsf{e} \downarrow \mathsf{v}}{\mathsf{p} \triangleleft \sum_{i \in I} \mathsf{q}?\ell_i(x).P_i \;\mid\; \mathsf{q} \triangleleft \mathsf{p}!\ell_j(\mathsf{e}).Q \;\mid\; \mathcal{M} \longrightarrow \mathsf{p} \triangleleft P_j\{\mathsf{v}/x\} \;\mid\; \mathsf{q} \triangleleft Q \;\mid\; \mathcal{M}}$$

$$\text{[T-CONDITIONAL]}\quad \frac{\mathsf{e} \downarrow \mathsf{true}}{\mathsf{p} \triangleleft \mathsf{if}\ \mathsf{e}\ \mathsf{then}\ P\ \mathsf{else}\ Q \;\mid\; \mathcal{M} \longrightarrow \mathsf{p} \triangleleft P \;\mid\; \mathcal{M}} \qquad \text{[F-CONDITIONAL]}\quad \frac{\mathsf{e} \downarrow \mathsf{false}}{\mathsf{p} \triangleleft \mathsf{if}\ \mathsf{e}\ \mathsf{then}\ P\ \mathsf{else}\ Q \;\mid\; \mathcal{M} \longrightarrow \mathsf{p} \triangleleft Q \;\mid\; \mathcal{M}}$$

$$\text{[R-STRUCT]}\quad \frac{\mathcal{M}_1' \equiv \mathcal{M}_1 \quad \mathcal{M}_1 \longrightarrow \mathcal{M}_2 \quad \mathcal{M}_2 \equiv \mathcal{M}_2'}{\mathcal{M}_1' \longrightarrow \mathcal{M}_2'}$$

Table 3: Reduction rules.

The *computational rules of multiparty sessions* are given in Table 3. They are closed with respect to the structural congruence defined in Table 2. In rule [R-COMM], the participant q sends the value v choosing the label ℓ_j to participant p, who offers inputs on all labels ℓ_i with $i \in I$. In rules [T-CONDITIONAL] and [F-CONDITIONAL], the participant p chooses to continue as P if the condition e evaluates to true and as Q if e evaluates to false. Rule [R-STRUCT] states that the reduction relation is closed with respect to structural congruence. We use $\longrightarrow^*$ with the standard meaning.

We adopt some standard conventions regarding the syntax of processes and sessions. Namely, we will use $\prod_{i \in I} \mathsf{p}_i \triangleleft P_i$ as short for $\mathsf{p}_1 \triangleleft P_1 \mid \ldots \mid \mathsf{p}_n \triangleleft P_n$, where $I = \{1, \ldots, n\}$. We will sometimes use infix notation for external choice process. For example, instead of $\sum_{i \in \{1,2\}} \mathsf{p}?\ell_i(x).P_i$, we will write $\mathsf{p}?\ell_1(x).P_1 + \mathsf{p}?\ell_2(x).P_2$.

Example 2.2. *We now show the operational semantics in action. Consider the following multiparty session with three participants,* `Alice`, `Bob` *and* `Carol` :

$$\mathcal{M} = \mathtt{Alice} \triangleleft P_{\mathtt{Alice}} \mid \mathtt{Bob} \triangleleft P_{\mathtt{Bob}} \mid \mathtt{Carol} \triangleleft P_{\mathtt{Carol}}$$

where

$$\begin{array}{lcl} P_{\mathtt{Alice}} & = & \mathtt{Bob}!\ell_1(50).\mathtt{Carol}?\ell_3(x).\mathbf{0} \\ P_{\mathtt{Bob}} & = & \mathtt{Alice}?\ell_1(x).\mathtt{Carol}!\ell_2(100).\mathbf{0} + \mathtt{Alice}?\ell_4(x).\mathtt{Carol}!\ell_2(2).\mathbf{0} \\ P_{\mathtt{Carol}} & = & \mathtt{Bob}?\ell_2(x).\mathtt{Alice}!\ell_3(\mathtt{succ}(x)).\mathbf{0} \end{array}$$

This mulitiparty session reduces to

$$\mathtt{Alice} \triangleleft \mathbf{0} \mid \mathtt{Bob} \triangleleft \mathbf{0} \mid \mathtt{Carol} \triangleleft \mathbf{0}$$

after three communications occur. First, `Alice` *sends to* `Bob` *natural number* 50 *with the label* ℓ_1. `Bob` *is able to receive values with labels* ℓ_1 *and* ℓ_4. *Next, the only possible communication is between* `Bob` *and* `Carol`. *So,* `Carol` *receives natural number* 100 *from* `Bob`. *The value* 100 *is substituted in the continuation process. Finally, since* $\mathtt{succ}(100) \downarrow 101$, `Carol` *sends* 101 *to* `Alice`. *We can then reduce the session to, for example,* $\mathtt{Alice} \triangleleft \mathbf{0}$, *but not further.*

From the end of Example 2.2, we can see that a session $\mathcal{M}$ always has at least one participant, since we do not have neutral element for the parallel composition. In Section 3, we will introduce a type system ensuring that if a well-typed multiparty session has only one participant, then the corresponding process is $\mathbf{0}$ — hence, the participant's process has no inputs/outputs to perform.

The most crucial property is that when a multiparty session contains communications that will never be executed.

Definition 2.3. *A multiparty session* $\mathcal{M}$ *is* stuck *if* $\mathcal{M} \not\equiv \mathsf{p} \triangleleft \mathbf{0}$ *and there is no multiparty session* $\mathcal{M}'$ *such that* $\mathcal{M} \longrightarrow \mathcal{M}'$. *A multiparty session* $\mathcal{M}$ *gets* stuck, *notation* $\mathtt{stuck}(\mathcal{M})$, *if it reduces to a stuck multiparty session.*

E.g., the multiparty session $\mathcal{M}$ in Example 2.2 is not stuck, and it does not get stuck. A similar multiparty session, where instead of $P_{\texttt{Alice}}$ we take $P'_{\texttt{Alice}} = \texttt{Bob}!\ell_1(50).\texttt{Carol}?\ell_5(x).\mathbf{0}$, gets stuck because of label mismatch.

3. Type System

This section introduces a type system for the calculus presented in Section 2. The formulation is based on Kouzapas and Yoshida (2013, 2015), with adaptations to account for our simplified calculus. We formalise types and projections (Section 3.1), the subtyping relation (Section 3.2), and the typing rules and their properties (Section 3.3).

3.1. Types and Projections

Global types provide global conversation scenarios of multiparty sessions, with a bird's eye view describing the message exchanges between pairs of participants.

Definition 3.1 (Sorts and global types)**.** Sorts*, ranged over by* S*, are defined as:*

$$S \quad ::= \quad \texttt{nat} \mid \texttt{int} \mid \texttt{bool}$$

Global types*, ranged over by* $\mathbb{G}$*, are terms generated by the following grammar:*

$$\mathsf{G} \quad ::= \quad \texttt{end} \mid \mu\mathbf{t}.\mathsf{G} \mid \mathbf{t} \mid \mathtt{p} \to \mathtt{q} : \{\ell_i(S_i).G_i\}_{i\in I}$$

We require that $\mathtt{p} \neq \mathtt{q}$*,* $I \neq \emptyset$*, and* $\ell_i \neq \ell_j$ *whenever* $i \neq j$*, for all* $i, j \in I$*. We postulate that recursion is guarded. Unless otherwise noted, global types are* closed*: a recursion variable* $\mathbf{t}$ *only occurs bounded by* $\mu\mathbf{t}\ldots$.

In Definition 3.1, the type $\mathtt{p} \to \mathtt{q} : \{\ell_i(S_i).\mathsf{G}_i\}_{i\in I}$ formalises a protocol where participant $\mathtt{p}$ must send to $\mathtt{q}$ one message with label ℓ_i and a value of type S_i as payload, for some $i \in I$; then, depending on which ℓ_i was sent by $\mathtt{p}$, the protocol continues as G_i. Value types are restricted to sorts, that can be natural (`nat`), integer (`int`) and boolean (`bool`). The type `end` represents a terminated protocol. Recursive protocol is modelled as $\mu\mathbf{t}.\mathsf{G}$, where recursion variable $\mathbf{t}$ is bound and guarded in G — e.g., $\mu\mathbf{t}.\mathtt{p} \to \mathtt{q} : \ell(\texttt{nat}).\mathbf{t}$ is a valid global type, whereas $\mu\mathbf{t}.\mathbf{t}$ is not. We take the equi-recursive viewpoint, i.e. We identify $\mu\mathbf{t}.\mathsf{G}$ and $\mathsf{G}\{\mu\mathbf{t}.\mathsf{G}/\mathbf{t}\}$.

We define the *set of participants of a global type* G, by structural induction on G, as follows:

$$\mathtt{pt}\{\mu\mathbf{t}.\mathsf{G}\} = \mathtt{pt}\{\mathsf{G}\} \qquad \mathtt{pt}\{\texttt{end}\} = \mathtt{pt}\{\mathbf{t}\} = \emptyset \qquad \mathtt{pt}\{\mathtt{p} \to \mathtt{q} : \{\ell_i(S_i).G_i\}_{i\in I}\} = \{\mathtt{p}, \mathtt{q}\} \cup \mathtt{pt}\{\mathsf{G}_i\}\ (i \in I)$$

We will often write $\mathtt{p} \in \mathsf{G}$ instead of $\mathtt{p} \in \mathtt{pt}\{G\}$.

A *local session type* describes the behaviour of a single participant in a multiparty session.

Definition 3.2 (Local Session Types)**.** *The grammar of session types, ranged over by* T*, is:*

$$\mathsf{T} \quad ::= \quad \texttt{end} \quad \mid \quad \bigwedge_{i\in I} \mathtt{p}?\ell_i(S_i).\mathsf{T}_i \quad \mid \quad \bigvee_{i\in I} \mathtt{q}!\ell_i(S_i).\mathsf{T}_i \quad \mid \quad \mu\mathbf{t}.\mathsf{T} \quad \mid \quad \mathbf{t}$$

We require that $\ell_i \neq \ell_j$ *whenever* $i \neq j$*, for all* $i, j \in I$*. We postulate that recursion is always guarded. Unless otherwise noted, session types are closed.*

Note that, according to the previous definition, labels in a type need to be pairwise different. For example, $\mathtt{p}?\ell(\texttt{int}).\texttt{end} \wedge \mathtt{p}?\ell(\texttt{nat}).\texttt{end}$ is not a type.

The session type `end` says that no further communication is possible and the protocol is completed. The *external choice* or *branching type* $\bigwedge_{i\in I} \mathtt{p}?\ell_i(S_i).\mathsf{T}_i$ requires to wait to receive a value of sort S_i (for some $i \in I$) from the participant $\mathtt{p}$, via a message with label ℓ_i; if the received message has label ℓ_i, the protocol will continue as prescribed by T_i. The *internal choice* or *selection type* $\bigvee_{i\in I} \mathtt{q}!\ell_i(S_i).\mathsf{T}_i$ says that the participant implementing the type must choose a labelled message to send to $\mathtt{q}$; if the participant chooses the message ℓ_i, for some $i \in I$, it must include in the message to $\mathtt{q}$ a payload value of sort S_i, and continue as

prescribed by T_i. Recursion is modelled by the session type $\mu \mathbf{t}.\mathsf{T}$. We adopt the following conventions: we do not write branch/selection symbols in case of a singleton choice, we do not write unnecessary parentheses, and we often omit trailing $\mathtt{end}$s.

The set $\mathtt{pt}\{\mathsf{T}\}$ of participants of a session type T is defined inductively as follows

$$\mathtt{pt}\{\bigwedge_{i\in I} \mathsf{p}?\ell_i(S_i).\mathsf{T}_i\} = \mathtt{pt}\{\bigvee_{i\in I} \mathsf{p}!\ell_i(S_i).\mathsf{T}_i\} = \{\mathsf{p}\} \cup \bigcup_{i\in I} \mathtt{pt}\{\mathsf{T}_i\} \quad \mathtt{pt}\{\mu \mathbf{t}.\mathsf{T}\} = \mathtt{pt}\{\mathsf{T}\} \quad \mathtt{pt}\{\mathbf{t}\} = \mathtt{pt}\{\mathtt{end}\} = \emptyset.$$

In Definition 3.3 below, we define the *global type projection* as a coinductive relation $\mathsf{G} \upharpoonright_{\mathsf{r}} \mathsf{T}$ between global and session types. Our definition extends the one originally proposed by Honda et al. (2008, 2016), along the lines of Yoshida et al. (2010) and Deniélou et al. (2012): i.e., it uses a *merging operator* $\bigsqcap$.

Definition 3.3. *The* projection of a global type onto a participant r *is the largest relation* $\upharpoonright_{\mathsf{r}}$ *between global types and session types such that, whenever* $\mathsf{G} \upharpoonright_{\mathsf{r}} \mathsf{T}$*:*

- $\mathsf{r} \notin \mathtt{pt}\{\mathsf{G}\}$ *implies* $\mathsf{T} = \mathtt{end}$*;* [PROJ-END]
- $\mathsf{G} = \mathsf{p} \to \mathsf{r} : \{\ell_i(S_i).\mathsf{G}_i\}_{i\in I}$ *implies* $\mathsf{T} = \bigwedge_{i\in I} \mathsf{p}?\ell_i(S_i).\mathsf{T}_i$*, and* $\mathsf{G}_i \upharpoonright_{\mathsf{r}} \mathsf{T}_i$*,* $\forall i \in I$*;* [PROJ-IN]
- $\mathsf{G} = \mathsf{r} \to \mathsf{q} : \{\ell_i(S_i).\mathsf{G}_i\}_{i\in I}$ *implies* $\mathsf{T} = \bigvee_{i\in I} \mathsf{q}!\ell_i(S_i).\mathsf{T}_i$*, and* $\mathsf{G}_i \upharpoonright_{\mathsf{r}} \mathsf{T}_i$*,* $\forall i \in I$*;* [PROJ-OUT]
- $\mathsf{G} = \mathsf{p} \to \mathsf{q} : \{\ell_i(S_i).\mathsf{G}_i\}_{i\in I}$ *and* $\mathsf{r} \notin \{\mathsf{p}, \mathsf{q}\}$ *implies that there are* T_i*,* $i \in I$ *such that* $\mathsf{T} = \bigsqcap_{i\in I} \mathsf{T}_i$*, and* $\mathsf{G}_i \upharpoonright_{\mathsf{r}} \mathsf{T}_i$*, for every* $i \in I$*.* [PROJ-CONT]

Above, $\bigsqcap$ *is the* merging operator*, that is a partial operation over session types defined as:*

$$\mathsf{T}_1 \bigsqcap \mathsf{T}_2 \;=\; \begin{cases} \mathsf{T}_1 & \text{if } \mathsf{T}_1 = \mathsf{T}_2 \qquad \text{[MRG-ID]} \\ \mathsf{T}_3 & \text{if } \exists I, J : \begin{cases} \mathsf{T}_1 = \bigwedge_{i\in I} \mathsf{p}'?\ell_i(S_i).\mathsf{T}_i & \text{and} \\ \mathsf{T}_2 = \bigwedge_{j\in J} \mathsf{p}'?\ell_j(S_j).\mathsf{T}_j & \text{and} \\ \mathsf{T}_3 = \bigwedge_{k\in I\cup J} \mathsf{p}'?\ell_k(S_k).\mathsf{T}_k \end{cases} \qquad \text{[MRG-BRA]} \\ \textit{undefined} & \textit{otherwise.} \end{cases}$$

Proposition 3.4. *The merging operation is associative, i.e.:* $\mathsf{T} \sqcap (\mathsf{T}' \sqcap \mathsf{T}'') = (\mathsf{T} \sqcap \mathsf{T}') \sqcap \mathsf{T}''$.

By Definition 3.3, merging a type with itself results in itself (rule [MRG-ID]). Moreover, Definition 3.3 allows to combine different external choices (rule [MRG-BRA]) *if and only if* common labels have identical sorts and identical continuations, as formalised in Prop. 3.5 below and illustrated in Examples 3.6, 3.8 and 3.9 .

Proposition 3.5. *For two types* $\mathsf{T}' = \bigwedge_{i\in I} \mathsf{p}'?\ell_i(S_i).\mathsf{T}_i$ *and* $\mathsf{T}'' = \bigwedge_{j\in J} \mathsf{p}''?\ell_j(S_j).\mathsf{T}_j$*, we have that* $\mathsf{T}' \sqcap \mathsf{T}''$ *is defined* if and only if $\mathsf{p}' = \mathsf{p}''$ *and, whenever* $\ell_i = \ell_j$ *(for some* $i \in I$ *and* $j \in J$*),* $S_i = S_j$ *and* $\mathsf{T}_i = \mathsf{T}_j$*.*

Example 3.6. *We now give some small examples that illustrate the definition of the merging operator (here,* $i \neq j$ *implies* $\ell_i \neq \ell_j$*):*

$$\begin{array}{rcl} \mathsf{q}!\ell(\mathtt{nat}) \sqcap \mathsf{q}!\ell(\mathtt{nat}) & = & \mathsf{q}!\ell(\mathtt{nat}) \\ \mathsf{p}!\ell(\mathtt{nat}) \sqcap \mathsf{q}!\ell(\mathtt{nat}) & & \textit{undefined: outputs to different participants} \\ \mathsf{q}!\ell_3(\mathtt{nat}) \sqcap \mathsf{q}!\ell_4(\mathtt{nat}) & & \textit{undefined: outputs with different labels} \\ \big(\mathsf{q}?\ell_3(\mathtt{int}) \wedge \mathsf{q}?\ell_5(\mathtt{nat})\big) \sqcap \big(\mathsf{q}?\ell_4(\mathtt{int}) \wedge \mathsf{q}?\ell_5(\mathtt{nat})\big) & = & \mathsf{q}?\ell_3(\mathtt{int}) \wedge \mathsf{q}?\ell_4(\mathtt{int}) \wedge \mathsf{q}?\ell_5(\mathtt{nat}) \\ \mathsf{q}?\ell_3(\mathtt{nat}) \sqcap \mathsf{q}?\ell_3(\mathtt{nat}).\mathsf{q}?\ell_3(\mathtt{nat}) & & \textit{undefined: same prefixes, but different continuations} \\ \mathsf{q}?\ell(\mathtt{nat}) \sqcap \mathsf{q}?\ell(\mathtt{int}) & & \textit{undefined: the payload sorts do not match} \end{array}$$

Proposition 3.7. *The projection relation* $\upharpoonright_{\mathsf{r}}$ *is a partial function.*

We now describe the clauses of Definition 3.3:

- clause [PROJ-END] states that when a global type G is projected onto a participant r who does *not* appear in G (e.g., because $\mathsf{G} = \mathtt{end}$), then the result is the inactive session type $\mathtt{end}$;
- clause [PROJ-IN] (resp. [PROJ-OUT]) states that a global type G starting with a communication from p to r (resp. from r to q) projects onto an external (resp. internal) choice T, provided that the continuations of T are also projections of the corresponding global type continuations.
- clause [PROJ-CONT] states that if G starts with a communication between p and q, and we are projecting G onto a third participant r, then we need to *(1)* skip the initial communication, *(2)* project all the continuations onto r, and *(3) merge* the resulting session types, using the *merging operator* $\sqcap$.

As a result, clause [PROJ-CONT] of Definition 3.3 allows participant r to receive different messages (from a same participant p') in different branches of a global type, as shown in Example 3.8 below.

Example 3.8. *We demonstrate interesting points of Definition 3.3. First, we show some projections of global types. Consider the global type:*

$$\mathsf{G} \;=\; \mathsf{p} \to \mathsf{q} : \{\ell_1(\mathtt{nat}).\mathsf{G}_1, \ell_2(\mathtt{bool}).\mathsf{G}_2\} \qquad \textit{where} \begin{cases} \mathsf{G}_1 \;=\; \mathsf{q} \to \mathsf{r} : \{\ell_3(\mathtt{int}), \ell_5(\mathtt{nat})\} \\ \mathsf{G}_2 \;=\; \mathsf{q} \to \mathsf{r} : \{\ell_4(\mathtt{int}), \ell_5(\mathtt{nat})\} \\ \mathsf{r} \neq \mathsf{p} \end{cases}$$

We have:

$$\mathsf{G}\upharpoonright\mathsf{p} \;=\; \mathsf{q}!\ell_1(\mathtt{nat}).(\mathsf{G}_1\upharpoonright\mathsf{p}) \,\vee\, \mathsf{q}!\ell_2(\mathtt{bool}).(\mathsf{G}_2\upharpoonright\mathsf{p}) \;=\; \mathsf{q}!\ell_1(\mathtt{nat}).\mathtt{end} \,\vee\, \mathsf{q}!\ell_2(\mathtt{bool}).\mathtt{end}$$

$$\begin{aligned} \mathsf{G}\upharpoonright\mathsf{q} \;&=\; \mathsf{p}?\ell_1(\mathtt{nat}).(\mathsf{G}_1\upharpoonright\mathsf{q}) \,\wedge\, \mathsf{p}?\ell_2(\mathtt{bool}).(\mathsf{G}_2\upharpoonright\mathsf{q}) \\ &=\; \mathsf{p}?\ell_1(\mathtt{nat}).\big(\mathsf{r}!\ell_3(\mathtt{int}) \vee \mathsf{r}!\ell_5(\mathtt{nat})\big) \,\wedge\, \mathsf{p}?\ell_2(\mathtt{bool}).\big(\mathsf{r}!\ell_4(\mathtt{int}) \vee \mathsf{r}!\ell_5(\mathtt{nat})\big) \end{aligned}$$

$$\begin{aligned} \mathsf{G}\upharpoonright\mathsf{r} \;&=\; \mathsf{G}_1\upharpoonright\mathsf{r} \sqcap \mathsf{G}_2\upharpoonright\mathsf{r} \;=\; \big(\mathsf{q}?\ell_3(\mathtt{int}) \wedge \mathsf{q}?\ell_5(\mathtt{nat})\big) \sqcap \big(\mathsf{q}?\ell_4(\mathtt{int}) \wedge \mathsf{q}?\ell_5(\mathtt{nat})\big) \\ &=\; \mathsf{q}?\ell_3(\mathtt{int}) \wedge \mathsf{q}?\ell_4(\mathtt{int}) \wedge \mathsf{q}?\ell_5(\mathtt{nat}) \end{aligned}$$

Note that in G, q *could output different messages towards* r*, depending on whether* p *sends* ℓ_1 *or* ℓ_2 *to* q*; therefore, in* $\mathsf{G}\upharpoonright\mathsf{r}$*, the possible inputs of* r *in* G_1 *and* G_2 *are merged into a larger external choice that supports all possible outputs of* q*.*

Importantly, by Definition 3.3, there exist global types that *cannot* be projected onto all their participants. This is because G might describe meaningless protocols, that cause the merging operation $\sqcap$ in clause [PROJ-CONT] to be undefined, as shown in Example 3.9 below.

Example 3.9. *We show two global types that cannot be projected according to the Definition 3.3. Consider the global type* $\mathsf{G} = \mathsf{p} \to \mathsf{q} : \{\ell_1(\mathtt{nat}).\mathsf{G}_1\,,\ \ell_2(\mathtt{bool}).\mathsf{G}_2\}$*, with* $\mathsf{G}_1 = \mathsf{r} \to \mathsf{q} : \ell_3(\mathtt{nat})$ *and* $\mathsf{G}_2 = \mathsf{r} \to \mathsf{q} : \ell_4(\mathtt{nat})$*. Then,*

$$\begin{aligned} \mathsf{G}\upharpoonright\mathsf{p} \;&=\; \mathsf{q}!\ell_1(\mathtt{nat}) \vee \mathsf{q}!\ell_2(\mathtt{bool}) \\ \mathsf{G}\upharpoonright\mathsf{q} \;&=\; \mathsf{p}?\ell_1(\mathtt{nat}).\mathsf{r}?\ell_3(\mathtt{nat}) \,\wedge\, \mathsf{p}?\ell_2(\mathtt{bool}).\mathsf{r}?\ell_4(\mathtt{nat}) \\ \mathsf{G}\upharpoonright\mathsf{r} \;&=\; \mathsf{q}!\ell_3(\mathtt{nat}) \sqcap \mathsf{q}!\ell_4(\mathtt{nat}) \quad \textbf{\textit{(undefined if } } \ell_3 \neq \ell_4\textbf{\textit{)}} \end{aligned}$$

Intuitively, when $\ell_3 \neq \ell_4$*,* $\mathsf{G}\upharpoonright\mathsf{r}$ *is undefined because in* G*, depending on whether* p *and* q *exchange* ℓ_1 *or* ℓ_2*,* r *is supposed to send either* ℓ_3 *or* ℓ_4 *to* q*; however,* r *is not privy to the interactions between* p *and* q*, and thus,* G *provides an invalid specification for* r*. Instead, if* $\ell_3 = \ell_4$*, then by Definition 3.3 we have* $\mathsf{G}\upharpoonright\mathsf{r} = \mathsf{q}!\ell_3(\mathtt{nat}) \sqcap \mathsf{q}!\ell_3(\mathtt{nat}) = \mathsf{q}!\ell_3(\mathtt{nat})$*.*

Now, consider the global type $\mathsf{G}' = \mathsf{p} \to \mathsf{q} : \{\ell_1(\mathtt{nat}).\mathsf{G}'_1\,,\ \ell_2(\mathtt{bool}).\mathsf{G}'_2\}$*, with* $\mathsf{G}'_1 = \mathsf{q} \to \mathsf{r} : \ell_3(\mathtt{nat})$ *and* $\mathsf{G}'_2 = \mathsf{q} \to \mathsf{r} : \ell_3(\mathtt{nat}).\mathsf{q} \to \mathsf{r} : \ell_3(\mathtt{nat})$*. Then,*

$$\begin{aligned} \mathsf{G}'\upharpoonright\mathsf{p} \;&=\; \mathsf{q}!\ell_1(\mathtt{nat}) \vee \mathsf{q}!\ell_2(\mathtt{bool}) \\ \mathsf{G}'\upharpoonright\mathsf{q} \;&=\; \mathsf{p}?\ell_1(\mathtt{nat}).\mathsf{r}!\ell_3(\mathtt{nat}) \,\wedge\, \mathsf{p}?\ell_2(\mathtt{bool}).\mathsf{r}!\ell_3(\mathtt{nat}).\mathsf{r}!\ell_3(\mathtt{nat}) \\ \mathsf{G}'\upharpoonright\mathsf{r} \;&=\; \mathsf{q}?\ell_3(\mathtt{nat}) \sqcap \mathsf{q}?\ell_3(\mathtt{nat}).\mathsf{q}?\ell_3(\mathtt{nat}) \quad \textbf{\textit{(undefined)}} \end{aligned}$$

Here, $\mathsf{G}'\upharpoonright\mathsf{r}$ is undefined because in G', depending on whether p and q exchange ℓ_1 or ℓ_2, r is supposed to receive either one or two messages ℓ_3 from q; however, as in the previous example, r is not aware of the interactions between p and q, and thus, G provides an invalid specification for r. This example could be fixed, e.g., by replacing ℓ_3 with $\ell' \neq \ell_3$ in G'_2, or by letting $\mathsf{G}'_1 = \mathsf{G}'_2$: both fixes would make $\mathsf{G}'\upharpoonright\mathsf{r}$ defined, similarly to Example 3.8.

3.2. Subtyping

The subtyping relation $\leqslant$ is used to augment the flexibility of the type system (introduced in Section 3.3): by determining when a type T is "smaller" than T', it allows to use a process typed by the former whenever a process typed by the latter is required.

Definition 3.10 (Subsorting and subtyping)**.** Subsorting $\leq:$ *is the least reflexive binary relation such that* $\mathtt{nat} \leq: \mathtt{int}$.

Subtyping $\leqslant$ *is the largest relation between session types coinductively defined by the following rules:*

$$[\text{SUB-END}]\quad \mathtt{end} \leqslant \mathtt{end} \qquad [\text{SUB-IN}]\quad \frac{\forall i \in I: \quad S'_i \leq: S_i \quad \mathsf{T}_i \leqslant \mathsf{T}'_i}{\bigwedge_{i\in I\cup J} \mathsf{p}?\ell_i(S_i).\mathsf{T}_i \leqslant \bigwedge_{i\in I} \mathsf{p}?\ell_i(S'_i).\mathsf{T}'_i} \qquad [\text{SUB-OUT}]\quad \frac{\forall i \in I: \quad S_i \leq: S'_i \quad \mathsf{T}_i \leqslant \mathsf{T}'_i}{\bigvee_{i\in I} \mathsf{p}!\ell_i(S_i).\mathsf{T}_i \leqslant \bigvee_{i\in I\cup J} \mathsf{p}!\ell_i(S'_i).\mathsf{T}'_i}$$

Intuitively, the session subtyping $\leqslant$ in Definition 3.10 says that T is smaller than T' when T is "less liberal" than T' — i.e., when T allows for less internal choices, and demands to handle more external choices.[1] A peculiarity of the relation is that, apart from a pair of inactive session types, only inputs and outputs from/to a same participant can be related (with additional conditions to be satisfied). Note that the double line in the subtyping rules indicates that the rules are interpreted *coinductively* (Pierce, 2002, Chapter 21).

- Rule [SUB-END] says that `end` is only subtype of itself.
- Rule [SUB-IN] relates external choices from the same participant p: the subtype must support all the choices of the supertype, and for each common message label, the continuations must be related, too; note that the carried sorts are contravariant: e.g., if the supertype requires to receive a message $\ell_i(\mathtt{nat})$ (for some $i \in I$), then the subtype can support $\ell_i(\mathtt{int})$ or $\ell_i(\mathtt{nat})$, since $\mathtt{nat} \leq: \mathtt{int}$ and $\mathtt{nat} \leq: \mathtt{nat}$.
- Rule [SUB-OUT] relates internal choices towards the same participant p: the subtype must offer a subset of the choices of the supertype, and for each common message label, the continuations must be related, too; note that the carried sorts are covariant: e.g., if the supertype allows to send a message $\ell_i(\mathtt{int})$ (for some $i \in I$), then the subtype can allow to send $\ell_i(\mathtt{int})$ or $\ell_i(\mathtt{nat})$, since $\mathtt{int} \leq: \mathtt{int}$ and $\mathtt{nat} \leq: \mathtt{int}$.

Lemma 3.11. *The subtyping relation $\leqslant$ is reflexive and transitive.*

3.3. Type system

We now introduce a type system for the multiparty session calculus presented in Section 2. We distinguish three kinds of typing judgments:

$$\Gamma \vdash \mathsf{e} : S \qquad \Gamma \vdash P : \mathsf{T} \qquad \vdash \mathcal{M} : \mathsf{G}$$

[1] Readers familiar with the theory of session types might notice that our subtyping relation is inverted w.r.t. the original binary session subtyping, introduced in the works of Gay and Hole (1999, 2005). In such works, smaller types have less internal choices, and more external choices: this is because they formalise a "channel-oriented" notion of subtyping, while we adopt a "process-oriented" view. For a thorough analysis and comparison of the two approaches, see Gay (2016).

$$\Gamma \vdash \mathsf{n} : \mathtt{nat} \qquad \Gamma \vdash \mathsf{i} : \mathtt{int} \qquad \Gamma \vdash \mathsf{true} : \mathtt{bool} \qquad \Gamma \vdash \mathsf{false} : \mathtt{bool} \qquad \Gamma, x : S \vdash x : S$$

$$\frac{\Gamma \vdash \mathsf{e} : \mathtt{nat}}{\Gamma \vdash \mathtt{succ}(\mathsf{e}) : \mathtt{nat}} \qquad \frac{\Gamma \vdash \mathsf{e} : \mathtt{int}}{\Gamma \vdash \mathtt{neg}(\mathsf{e}) : \mathtt{int}} \qquad \frac{\Gamma \vdash \mathsf{e} : \mathtt{bool}}{\Gamma \vdash \neg \mathsf{e} : \mathtt{bool}}$$

$$\frac{\Gamma \vdash \mathsf{e}_1 : S \quad \Gamma \vdash \mathsf{e}_2 : S}{\Gamma \vdash \mathsf{e}_1 \oplus \mathsf{e}_2 : S} \qquad \frac{\Gamma \vdash \mathsf{e}_1 : \mathtt{int} \quad \Gamma \vdash \mathsf{e}_2 : \mathtt{int}}{\Gamma \vdash \mathsf{e}_1 > \mathsf{e}_2 : \mathtt{bool}} \qquad \frac{\Gamma \vdash \mathsf{e} : S \quad S \leq : S'}{\Gamma \vdash \mathsf{e} : S'}$$

Table 4: Typing rules for expressions.

where Γ is the *typing environment*:

$$\Gamma \ ::= \ \emptyset \ \mid \ \Gamma, x : S \ \mid \ \Gamma, X : \mathsf{T}$$

i.e., a mapping that associates expression variables with sorts, and process variables with session types.

We say that *a multiparty session* $\mathcal{M}$ *is well typed* if there is a global type G such that $\vdash \mathcal{M} : \mathsf{G}$. If a multiparty session is well typed, we will sometimes write just $\vdash \mathcal{M}$.

The typing rules for expressions are given in Table 4, and are self-explanatory. The typing rules for processes and multiparty sessions are content of Table 5:

- [T-SUB] is the *subsumption rule*: a process with type T is also typed by the supertype T';
- [T-**0**] says that a terminated process implements the terminated session type;
- [T-REC] types a recursive process $\mu X.P$ with T if P can be typed as T, too, by extending the typing environment with the assumption that X has type T;
- [T-VAR] uses the typing environment assumption that process X has type T;
- [T-INPUT-CHOICE] types a summation of input prefixes as a branching type. It requires that each input prefix targets the same participant q, and that, for all $i \in I$, each continuation process P_i is typed by the continuation type T_i, having the bound variable x_i in the typing environment with sort S_i. Note that the rule implicitly requires the process labels ℓ_i to be pairwise distinct (as per Definition 3.2);
- [T-OUT] types an output prefix with a singleton selection type, provided that the expression in the message payload has the correct sort S, and the process continuation matches the type continuation;
- [T-CHOICE] types a conditional process with T if its sub-processes can be typed by T and expression e is boolean.
- [T-SESS] types multiparty sessions, by associating typed processes to participants. It requires that the processes being composed in parallel can play as participants of a global communication protocol: hence, their types must be projections of a single global type G. The condition $\mathtt{pt}\{\mathsf{G}\} \subseteq \{\mathsf{p}_i \mid i \in I\}$ allows to also type sessions containing $\mathsf{p} \triangleleft \mathbf{0}$: this is needed to assure invariance of typing.

Example 3.12. *We show that the multiparty session* $\mathcal{M}$ *from Example 2.2 is well typed. Consider the following global type:*

$$\begin{aligned} \mathsf{G} \ = \ \mathtt{Alice} \to \mathtt{Bob} : \ & \{\ell_1(\mathsf{nat}).\mathtt{Bob} \to \mathtt{Carol} : \ell_2(\mathsf{nat}).\mathtt{Carol} \to \mathtt{Alice} : \ell_3(\mathsf{nat}).\mathsf{end}, \\ & \ell_4(\mathsf{nat}).\mathtt{Bob} \to \mathtt{Carol} : \ell_2(\mathsf{nat}).\mathtt{Carol} \to \mathtt{Alice} : \ell_3(\mathsf{nat}).\mathsf{end}\}. \end{aligned}$$

We show that participants Alice, Bob *and* Carol *respect the prescribed protocol* G*, by showing that they participate in a well-typed multiparty session. Applying rules from Table 5, we derive*

$$\vdash P_{\mathtt{Alice}} : \mathsf{T}_{\mathtt{Alice}} \qquad \vdash P_{\mathtt{Bob}} : \mathsf{T}_{\mathtt{Bob}} \qquad \vdash P_{\mathtt{Carol}} : \mathsf{T}_{\mathtt{Carol}}$$

$$\frac{\Gamma \vdash P : \mathsf{T} \qquad \mathsf{T} \leqslant \mathsf{T}'}{\Gamma \vdash P : \mathsf{T}'}\ [\text{T-SUB}] \qquad \Gamma \vdash \mathbf{0} : \mathtt{end}\ [\text{T-}\mathbf{0}] \qquad \frac{\Gamma, X : \mathsf{T} \vdash P : \mathsf{T}}{\Gamma \vdash \mu X.P : \mathsf{T}}\ [\text{T-REC}] \qquad \Gamma, X : \mathsf{T} \vdash X : \mathsf{T}\ [\text{T-VAR}]$$

$$\frac{\forall i \in I \quad \Gamma, x_i : S_i \vdash P_i : \mathsf{T}_i}{\Gamma \vdash \sum_{i \in I} \mathsf{q}?\ell_i(x_i).P_i : \bigwedge_{i \in I} \mathsf{q}?\ell_i(S_i).\mathsf{T}_i}\ [\text{T-INPUT-CHOICE}] \qquad \frac{\Gamma \vdash \mathsf{e} : S \quad \Gamma \vdash P : \mathsf{T}}{\Gamma \vdash \mathsf{q}!\ell(\mathsf{e}).P : \mathsf{q}!\ell(S).\mathsf{T}}\ [\text{T-OUT}]$$

$$\frac{\Gamma \vdash \mathsf{e} : \mathtt{bool} \quad \Gamma \vdash P_1 : \mathsf{T} \quad \Gamma \vdash P_2 : \mathsf{T}}{\Gamma \vdash \mathsf{if}\ \mathsf{e}\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2 : \mathsf{T}}\ [\text{T-CHOICE}]$$

$$\frac{\forall i \in I \quad \vdash P_i : \mathsf{G}{\upharpoonright}\mathsf{p}_i \quad \mathtt{pt}\{\mathsf{G}\} \subseteq \{\mathsf{p}_i \mid i \in I\}}{\vdash \prod_{i \in I} \mathsf{p}_i \triangleleft P_i : \mathsf{G}}\ [\text{T-SESS}]$$

Table 5: Typing rules for processes and sessions.

where:

$$\begin{array}{lcl} \mathsf{T}_{\mathtt{Alice}} & = & \mathtt{Bob}!\ell_1(\mathtt{nat}).\mathtt{Carol}?\ell_3(\mathtt{nat}).\mathtt{end} \\ \mathsf{T}_{\mathtt{Bob}} & = & \mathtt{Alice}?\ell_1(\mathtt{nat}).\mathtt{Carol}!\ell_2(\mathtt{nat}).\mathtt{end} \wedge \mathtt{Alice}?\ell_4(\mathtt{nat}).\mathtt{Carol}!\ell_2(\mathtt{nat}).\mathtt{end} \\ \mathsf{T}_{\mathtt{Carol}} & = & \mathtt{Bob}?\ell_2(\mathtt{nat}).\mathtt{Alice}!\ell_3(\mathtt{nat}).\mathtt{end} \end{array}$$

Now, let:

$$\mathsf{T}'_{\mathtt{Alice}} \quad = \quad \mathtt{Bob}!\ell_1(\mathtt{nat}).\mathtt{Carol}?\ell_3(\mathtt{nat}).\mathtt{end} \vee \mathtt{Bob}!\ell_4(\mathtt{nat}).\mathtt{Carol}?\ell_3(\mathtt{nat}).\mathtt{end}$$

Since it holds that $\mathsf{T}_{\mathtt{Alice}} \leqslant \mathsf{T}'_{\mathtt{Alice}}$*, and the projections of* G *to the participants are*

$$\mathsf{G}{\upharpoonright}\mathtt{Alice} = \mathsf{T}'_{\mathtt{Alice}} \qquad \mathsf{G}{\upharpoonright}\mathtt{Bob} = \mathsf{T}_{\mathtt{Bob}} \qquad \mathsf{G}{\upharpoonright}\mathtt{Carol} = \mathsf{T}_{\mathtt{Carol}}$$

we conclude:

$$\vdash \mathtt{Alice} \triangleleft P_{\mathtt{Alice}} \mid \mathtt{Bob} \triangleleft P_{\mathtt{Bob}} \mid \mathtt{Carol} \triangleleft P_{\mathtt{Carol}} : \mathsf{G}.$$

The proposed type system for multiparty sessions enjoys two fundamental properties: typed sessions only reduce to typed sessions (subject reduction), and typed sessions never get stuck. The remaining of this section is devoted to the proof of these properties.

In order to state subject reduction, we need to formalise how global types are modified when multiparty sessions reduce and evolve.

Definition 3.13 (Global types consumption and reduction)**.** *The* consumption *of the communication* $\mathsf{p} \xrightarrow{\ell} \mathsf{q}$ *for the global type* G *(notation* $\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}$*) is the global type coinductively defined as follows:*

$$\begin{array}{lcll} \big(\mathsf{p} \to \mathsf{q} : \{\ell_i(S_i).\mathsf{G}_i\}_{i \in I}\big) \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q} & = & \mathsf{G}_k & \text{if } \exists k \in I : \ell = \ell_k \\ \big(\mathsf{r} \to \mathsf{s} : \{\ell_i(S_i).\mathsf{G}_i\}_{i \in I}\big) \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q} & = & \mathsf{r} \to \mathsf{s} : \{\ell_i(S_i).\mathsf{G}_i \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}\}_{i \in I} & \text{if } \begin{cases} \{\mathsf{r},\mathsf{s}\} \cap \{\mathsf{p},\mathsf{q}\} = \emptyset & \text{and} \\ \forall i \in I : \{\mathsf{p},\mathsf{q}\} \subseteq \mathsf{G}_i \end{cases} \end{array}$$

The reduction of global types *is the smallest pre-order relation closed under the rule:* $\mathsf{G} \Longrightarrow \mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}$

Example 3.14. *We show that a projection of a global type before the consumption might require to support more external choices than the projection after the consumption. Take* G*, its subterm* G_1*, from Example 3.8, and their types denoted as* G *and* G_1*, respectively. Also take the projection:*

$$\mathsf{G}{\upharpoonright}\mathsf{r} \ = \ \mathsf{q}?\ell_3(\mathtt{int}) \wedge \mathsf{q}?\ell_4(\mathtt{int}) \wedge \mathsf{q}?\ell_5(\mathtt{nat})$$

and recall the explanation on how $\mathsf{G}\upharpoonright\mathsf{r}$ *above merges all the possible inputs that* r *might receive from* q*, depending on whether* p *first sends* ℓ_1 *or* ℓ_2 *to* q*. We have:*

$$\begin{array}{rcccl} \mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell_1} \mathsf{q} & = & \mathsf{G}_1 & = & \mathsf{q} \to \mathsf{r} : \{\ell_3(\mathtt{int}), \ell_5(\mathtt{nat})\} \\ (\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell_1} \mathsf{q})\upharpoonright\mathsf{r} & = & \mathsf{G}_1\upharpoonright\mathsf{r} & = & \mathsf{q}?\ell_3(\mathtt{int}) \wedge \mathsf{q}?\ell_5(\mathtt{nat}) \end{array}$$

and we obtain $\mathsf{G}\upharpoonright\mathsf{r} \leqslant (\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell_1} \mathsf{q})\upharpoonright\mathsf{r}$*. The reason is that, after the transition from* G *to* G_1*, there is no possibility for* q *to send* ℓ_4 *to* r*, hence* r *does not need to support such a message in its projection.*

Note that a process that plays the role of r *in* G*, and is therefore typed by* $\mathsf{G}\upharpoonright\mathsf{r}$*, has to support the input of* ℓ_4 *from* q*, by rule* [T-INPUT-CHOICE] *in Table 5. After the transition from* G *to* G_1*, the same process is also typed by* $\mathsf{G}_1\upharpoonright\mathsf{r}$*, by rule* [T-SUB] *— but will never receive a message* ℓ_4 *from* q*.*

We can now prove subject reduction.

Theorem 3.15 (Subject Reduction)**.** *Let* $\vdash \mathcal{M} : \mathsf{G}$*. For all* $\mathcal{M}'$*, if* $\mathcal{M} \longrightarrow \mathcal{M}'$*, then* $\vdash \mathcal{M}' : \mathsf{G}'$ *for some* G' *such that* $\mathsf{G} \Longrightarrow \mathsf{G}'$*.*

Theorem 3.16 (Progress)**.** *If* $\vdash \mathcal{M} : \mathsf{G}$*, then either* $\mathcal{M} \equiv \mathsf{p} \triangleleft \mathbf{0}$ *or there is* $\mathcal{M}'$ *such that* $\mathcal{M} \longrightarrow \mathcal{M}'$*.*

As a consequence of subject reduction and progress, we get the safety property stating that a typed multiparty session will never get stuck.

Theorem 3.17 (Type Safety)**.** *If* $\vdash \mathcal{M} : \mathsf{G}$*, then it does not hold* $\mathtt{stuck}(\mathcal{M})$*.*

Proof. Direct consequence of Theorem 3.15, Theorem 3.16, and Definition 2.3. □

References

Deniélou, P., Yoshida, N., Bejleri, A., Hu, R., 2012. Parameterised multiparty session types. Logical Methods in Computer Science 8. doi:10.2168/LMCS-8(4:6)2012.

Gay, S., Hole, M., 2005. Subtyping for session types in the pi calculus. Acta Informatica 42, 191–225. doi:10.1007/s00236-005-0177-z.

Gay, S., Ravera, A. (Eds.), 2017. Behavioural Types: from Theory to Tools. River Publishers.

Gay, S.J., 2016. Subtyping supports safe session substitution, in: A List of Successes That Can Change the World: Essays Dedicated to Philip Wadler on the Occasion of His 60th Birthday, Springer International Publishing. pp. 95–108. doi:10.1007/978-3-319-30936-1_5.

Gay, S.J., Hole, M., 1999. Types and subtypes for client-server interactions, in: ESOP, pp. 74–90. doi:10.1007/3-540-49099-X_6.

Honda, K., 1993. Types for dyadic interaction, in: CONCUR'93, pp. 509–523.

Honda, K., Vasconcelos, V.T., Kubo, M., 1998. Language primitives and type disciplines for structured communication-based programming, in: ESOP, Springer. pp. 22–138. doi:10.1007/BFb0053567.

Honda, K., Yoshida, N., Carbone, M., 2008. Multiparty asynchronous session types, in: POPL, ACM Press. pp. 273–284. doi:10.1145/1328438.1328472.

Honda, K., Yoshida, N., Carbone, M., 2016. Multiparty asynchronous session types. Journal of ACM 63, 1–67.

Hüttel, H., Lanese, I., Vasconcelos, V.T., Caires, L., Carbone, M., Deniélou, P.M., Mostrous, D., Padovani, L., Ravara, A., Tuosto, E., Vieira, H.T., Zavattaro, G., 2016. Foundations of session types and behavioural contracts. ACM Comput. Surv. 49. doi:10.1145/2873052.

Kouzapas, D., Yoshida, N., 2013. Globally governed session semantics, in: CONCUR, Springer. pp. 395–409. doi:10.1145/1328438.1328472.

Kouzapas, D., Yoshida, N., 2015. Globally governed session semantics. Logical Methods in Computer Science 10.

Milner, R., 1989. Communication and Concurrency. Prentice-Hall, Inc.

Pierce, B.C., 2002. Types and Programming Languages. MIT Press.

Takeuchi, K., Honda, K., Kubo, M., 1994. An Interaction-based Language and its Typing System, in: PARLE'94, pp. 398–413. doi:10.1007/3-540-58184-7_118.

Yoshida, N., Deniélou, P., Bejleri, A., Hu, R., 2010. Parameterised multiparty session types, in: FOSSACS, Springer. pp. 128–145. doi:10.1007/978-3-642-12032-9_10.