A Very Gentle Introduction to Multiparty Session Types *

Nobuko Yoshida and Lorenzo Gheri

Imperial College London, UK
{n.yoshida,l.gheri}@imperial.ac.uk

Abstract. Multiparty session types (MPST) are a formal specification and verification framework for message-passing protocols without central control: the desired interactions at the scale of the network itself are specified into a session (called *global type*). Global types are then projected onto *local types* (one for each participant), which describe the protocol from a local point of view. These local types are used to validate an application through type-checking, monitoring, and code generation. Theory of session types guarantees that local conformance of all participants induces *global conformance* of the network to the initial global type. This paper provides a very gentle introduction of the simplest version of multiparty session types for readers who are not familiar with session types nor process calculi.

Keywords: Multiparty Session Types · Process Calculi · Distributed Systems · Type Safety · Progress

1 A Gentle Introduction to Multiparty Session Types

Backgrounds. Session types were introduced in a series of papers during the 1990s [8,18,10] 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 [13,4].

A basic observation underlying session types is that a communication-based application often exhibits a highly structured sequence of interactions involving, for example, sequencing, choices and recursion, which as a whole form a natural unit of *session*. The structure of a session is abstracted as a *type* through an intuitive syntax, which is then used for validating programs through associated language primitives.

Multiparty session types generalise the binary session type theory to the multiparty case, preventing deadlock and communication errors in more sophisticated communication protocols involving any number (two or more) of participants. The central idea is to introduce global types, which describe multiparty

* Work partially supported by EPSRC projects EP/K034413/1, EP/K011715/1, EP/L00058X/1, EP/N027833/1 and EP/N028201/1.

conversations from a global perspective and provide a means to check protocol compliance. The theory [12,11] was born by transforming the industry idea developed during designing a protocol description language, Scribble [17], which was presented by Kohei Honda in [9] (see the historical background [20]). This original work was extended in various ways and applied to many different programming languages and tools, some of which use Scribble. The reader who is interested in programming languages can find the tools and papers at *Mobility Reading Group Home Page*.

Binary session types. As a first example, consider a scenario where a customer tries to conclude a deal with a travel agency. We associate a process with the *customer* (`Customer`) and one with the *agency* (`Agency`). The task involves synchronisation in the communication between the customer and the agency. Synchronisation is obtained through the exchange of messages.

Specifically, the protocol works as follows.

1. The customer sends an order to the agency, namely the place they desire to visit in their next travel (let us say *Hawaii*). On receiving the request, the agency looks up the price relative to that travel and sends a quote (*quote*) back to the customer.
2. When the customer receives the *quote*, they make a decision (choice): either to *accept* or to *reject* the offer. The customer communicates their decision to the agency, which is waiting for such a message.
3. In case of acceptance, the agency waits that the customer communicates their address (*address*), then sends a confirmation date (*date*) for the trip back to the customer and the protocol terminates.
4. In case of rejection, the deal is not made and the protocol immediately 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 obtain 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:

$$
\begin{array}{l}
\texttt{Customer} \to \texttt{Agency}\colon \mathit{Hawaii}(\texttt{bool}).\texttt{Agency} \to \texttt{Customer}\colon \mathit{quote}(\texttt{nat}).\\
\texttt{Customer} \to \texttt{Agency} : \{\\
\quad \mathit{accept}(\texttt{bool}).\\
\qquad \texttt{Customer} \to \texttt{Agency}\colon \mathit{address}(\texttt{nat}).\\
\qquad \texttt{Agency} \to \texttt{Customer}\colon \mathit{date}(\texttt{nat}).\texttt{end},\\
\quad \mathit{reject}(\texttt{bool}).\texttt{end}\\
\}
\end{array}
\tag{1}
$$

The type describes the global pattern of communication between `Customer` and `Agency` using message exchanges, sequencing, and choice. The basic pattern $\texttt{Customer} \to \texttt{Agency}\colon m(S)$ indicates a message with label m sent from the `Customer` to the `Agency`, together with an element of sort S. The communication starts with the customer sending the message *Hawaii* to the agency, then the agency sends the *quote* label together with a natural number and at this point the customer either sends an *accept* or a *reject* message. In case of *reject*, the protocol ends (type `end`); otherwise, the communication continues with the sequential exchange of an *address* (from the customer to the agency) and a *date* (from the agency to the customer). The operator "." denotes sequencing, the "," separates possible continuations for the protocol.

The global type states what are the valid message sequences allowed in the system. When we implement `Customer` and `Agency` separately, we would like to check that their composition conforms to the global type. Since there are only two participants, projecting to each participant is simple. From the perspective of the `Customer`, the communication can be described by the type:

$$\begin{array}{l} !\mathit{Hawaii}(\texttt{bool}).?\mathit{quote}(\texttt{nat}). \\ (\,(\,!\mathit{accept}(\texttt{bool}).!\mathit{address}(\texttt{nat}).?\mathit{date}(\texttt{nat}).\texttt{end}\,) \oplus (\,!\mathit{reject}(\texttt{bool}).\texttt{end}\,)) \end{array} \tag{2}$$

where $!m$ denotes a message with label m sent to `Agency`, $?m$ denotes a message with label m received from `Agency`, and $\oplus$ denotes an internal choice. Thus the type states that, `Customer` after sending the message with label *Hawaii*, waits for a natural number for the *quote* and, after receiving it, decides either to send $!\mathit{accept}(\texttt{bool}).\ !\mathit{address}(\texttt{nat})$, wait for a natural number for the *date* and then exit, or to send just $\mathit{reject}(\texttt{bool})$ and exit.

From the viewpoint of `Agency`, the same global session is described by the dual type

$$\begin{array}{l} ?\mathit{Hawaii}(\texttt{bool}).!\mathit{quote}(\texttt{nat}). \\ (\,(\,?\mathit{accept}(\texttt{bool}).?\mathit{address}(\texttt{nat}).!\mathit{date}(\texttt{nat}).\texttt{end}\,) \,\&\, (\,?\mathit{reject}(\texttt{bool}).\texttt{end}\,)) \end{array} \tag{3}$$

in which & means that a choice is offered externally.

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

Multiparty session types. In the case of two parties, the safety can be checked given the local type (2) since its dual (3) is unique (up to unfolding recursions). 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, after the customer's acceptance, the agency needs to forward some details to the hotel, before a direct communication starts between the hotel and the customer. Namely, `Customer` sends a message *details* to `Agency`, which forwards the message to `Hotel`. After receiving the message, `Hotel` sends an acknowledgement *ok* to `Customer`. We

start by specifying the global type as:

$$\texttt{Customer} \to \texttt{Agency}\colon \mathit{details}(\texttt{nat}).\texttt{Agency} \to \texttt{Hotel}\colon \mathit{details}(\texttt{nat}).\ \texttt{Hotel} \to \texttt{Customer}\colon \mathit{ok}(\texttt{bool}).\texttt{end} \tag{4}$$

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 local types:

`Customer`'s endpoint type:	$\texttt{Agency}!\mathit{details}(\texttt{nat}).\texttt{Hotel}?\mathit{ok}(\texttt{bool}).\texttt{end}$
`Agency`'s endpoint type:	$\texttt{Customer}?\mathit{details}(\texttt{nat}).\texttt{Hotel}!\mathit{details}(\texttt{nat}).\texttt{end}$
`Hotel`'s endpoint type:	$\texttt{Agency}?\mathit{details}(\texttt{nat}).\texttt{Customer}!\mathit{ok}(\texttt{bool}).\texttt{end}$

where $\texttt{Agency}!\mathit{details}(\texttt{nat})$ means "send to `Agency` a *details* message," and $\texttt{Hotel}?\mathit{ok}(\texttt{bool})$ means "receive from `Hotel` 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.

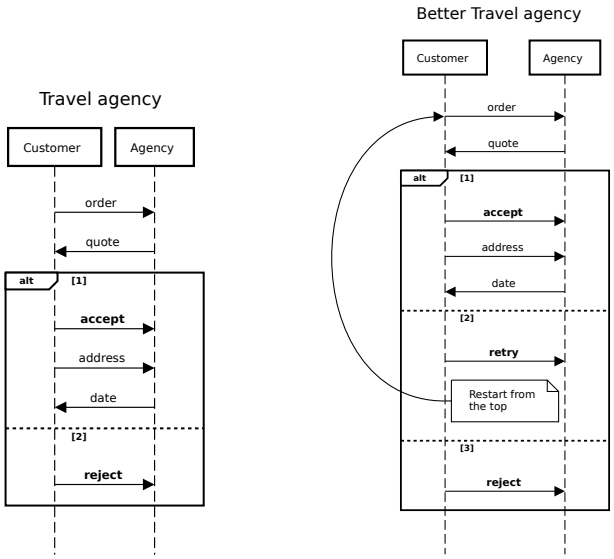
If instead we only used three separate binary session types to describe the message exchanges between `Customer` and `Agency`, between `Agency` and `Hotel`, and between `Hotel` and `Customer`, respectively, without using a global type, then we lose essential sequencing information in this interaction 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 `Customer` that conforms to $\texttt{Hotel}?\mathit{ok}(\texttt{bool}).\texttt{Agency}!\mathit{details}(\texttt{nat}).\texttt{end}$ becomes typable. This causes the situation in which each of the three parties blocks indefinitely while waiting for a message to be delivered.

In what follows we will start from giving more examples in Section 2, which will be used as running examples throughout the whole paper. In Section 3 we will introduce a formal process calculus powerful enough to describe multiparty communication protocols, such as the ones from above. Finally in Section 4, we will introduce a type system; we will show how multiparty session types work by examples and present the key properties of typed processes.

2 Examples, Intuitively

Multiparty session types are aimed at enabling compositional reasoning about communication among different parties. In this section we present informally some essential communication protocols that can be handled with this methodology.

Let us for example consider the basic protocol from the previous section. This simple protocol for a travel agency is syntetically displayed by Figure 1a. A more interesting protocol could be the one shown in Figure 1b. Here the customer is allowed to try *again and again* to obtain the new quote, should they not like the first one. We will see that our calculus (Section 3) is endowed with recursion, which can model an indefinite number of iterations.



(a) A simple protocol (b) A better protocol

Fig. 1: Communication protocols for a travel agency

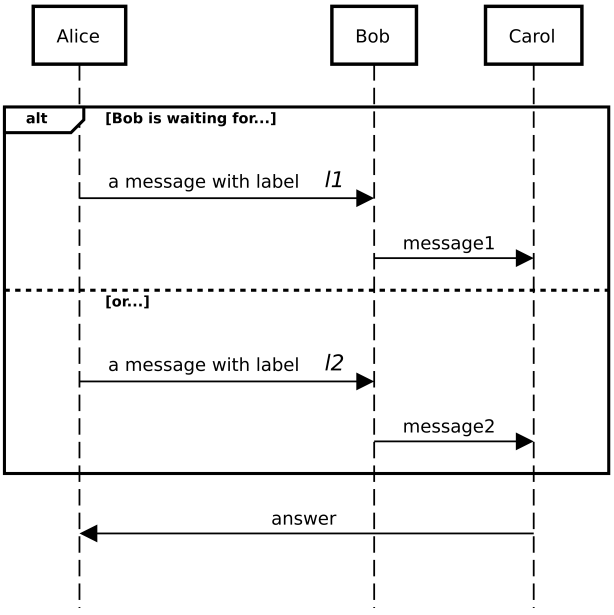


Fig. 2: Multiparty communication protocol

Let us consider a specification in Figure 2, which describes a simple communication protocol among three parties. Bob is waiting for a message from Alice, *non-deterministically* allowing for messages with two different labels (*l1* and *l2*). Alice sends the message with label *l1* and the communication continues with Bob sending a message message1 to Carol and finally Carol returning a message answer to Alice, which depends on the previous communication, including the choice of the label for the original message from Alice to Bob.

In the following sections, we model these protocols by a simple multiparty session calculus, and give types to processes modelling these protocols.

3 Synchronous Multiparty Session Calculus

This section introduces the syntax and semantics of a synchronous multiparty session calculus from [7], which simplifies the calculus in [14] by eliminating both shared channels for session initiations and session channels for communications inside sessions.

Notation 01 (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:

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

The output process $\mathsf{p}!\ell\langle\mathsf{e}\rangle.Q$ 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:

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

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 multiparty session is *well formed* if all its participants are different. We consider only well-formed multiparty sessions.

Example 1. We now show how to encode, in the above calculus, processes respecting the protocols informally presented in Figure 1. Note that we picked one different label for each destination (that can be either Hawaii or France), as well as for each action (as "accept", "reject", ...).

Let us start with processes in Figure 1a.

$$\begin{aligned}
P'_{\texttt{Customer}} = {} & \texttt{Agency}!\mathit{Hawaii}\langle\mathsf{true}\rangle.\texttt{Agency}?\mathit{quote}(x).\\
& \quad \mathsf{if}\ \ (x > 1000)\\
& \quad \mathsf{then}\ \ \texttt{Agency}!\mathit{reject}\langle\mathsf{true}\rangle.\mathbf{0}\\
& \quad \mathsf{else}\ \ \texttt{Agency}!\mathit{accept}\langle\mathsf{true}\rangle.\texttt{Agency}!\mathit{address}\langle 42\rangle.\texttt{Agency}?\mathit{date}(y).\mathbf{0}\\
P'_{\texttt{Agency}} = {} & \texttt{Customer}?\mathit{Hawaii}(x).\texttt{Customer}!\mathit{quote}\langle 5000\rangle.\\
& \quad (\texttt{Customer}?\mathit{accept}(y).\texttt{Customer}?\mathit{address}(z).\\
& \qquad \texttt{Customer}!\mathit{date}\langle 25122019\rangle.\mathbf{0}\\
& \quad +\texttt{Customer}?\mathit{reject}(y).\mathbf{0})\\
& + \texttt{Customer}?\mathit{France}(x).\texttt{Customer}!\mathit{quote}\langle 1000\rangle.\\
& \quad (\texttt{Customer}?\mathit{accept}(y).\texttt{Customer}?\mathit{address}(z).\\
& \qquad \texttt{Customer}!\mathit{date}\langle 25122019\rangle.\mathbf{0}\\
& \quad +\texttt{Customer}?\mathit{reject}(y).\mathbf{0})
\end{aligned}$$

The customer here would ask for Hawaii as a destination, they will receive a quote that will be too expensive ($5000 > 1000$) and, thus, they will end up rejecting the offer (via sending true with label *reject*).

In what follows instead we will extend the above processes, giving to the customer the opportunity to retry, as suggested by the diagram in Figure 1b.

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

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

Table 1: Expression evaluation.

$$\begin{aligned}
P_{\texttt{Customer}} = {} & \texttt{Agency}!\mathit{Hawaii}\langle\mathsf{true}\rangle.\texttt{Agency}?\mathit{quote}(x). \\
& \mathsf{if}\ \ (x > 1000) \\
& \mathsf{then}\ \ \texttt{Agency}!\,\mathit{retry}\langle\mathsf{true}\rangle\,.\texttt{Agency}!\mathit{France}\langle\mathsf{true}\rangle.\texttt{Agency}?\mathit{quote}(y). \\
& \quad \mathsf{if}\ \ (y > 1000) \\
& \quad \mathsf{then}\ \ \texttt{Agency}!\mathit{reject}\langle\mathsf{true}\rangle.\mathbf{0} \\
& \quad \mathsf{else}\ \ \texttt{Agency}!\mathit{accept}\langle\mathsf{true}\rangle.\texttt{Agency}!\mathit{address}\langle 42\rangle.\texttt{Agency}?\mathit{date}(z).\mathbf{0} \\
& \mathsf{else}\ \ \texttt{Agency}!\mathit{accept}\langle\mathsf{true}\rangle.\texttt{Agency}!\mathit{address}\langle 42\rangle.\texttt{Agency}?\mathit{date}(y).\mathbf{0} \\
P_{\texttt{Agency}} = {} & \mu X.\,\texttt{Customer}?\mathit{Hawaii}(x).\texttt{Customer}!\mathit{quote}\langle 5000\rangle. \\
& (\texttt{Customer}?\mathit{accept}(y).\texttt{Customer}?\mathit{address}(z). \\
& \qquad \texttt{Customer}!\mathit{date}\langle 25122019\rangle.\mathbf{0} \\
& \quad + \texttt{Customer}?\mathit{retry}(y).X\ + \texttt{Customer}?\mathit{reject}(y).\mathbf{0}) \\
& + \texttt{Customer}?\mathit{France}(x).\texttt{Customer}!\mathit{quote}\langle 1000\rangle. \\
& (\texttt{Customer}?\mathit{accept}(y).\texttt{Customer}?\mathit{address}(z). \\
& \qquad \texttt{Customer}!\mathit{date}\langle 25122019\rangle.\mathbf{0} \\
& \quad + \texttt{Customer}?\mathit{retry}(y).X\ + \texttt{Customer}?\mathit{reject}(y).\mathbf{0})
\end{aligned}$$

In the example above, we have highlighted the syntactic construct allowing the customer to retry the purchase and obtain a new quote. In $P_{\texttt{Customer}}$ the only occurrence of the label *retry* is within a deterministic choice (if _ then _ else _ construct): if the quote is too high, the customer will communicate to the agency that they would like a different quote for a different destination. More interesting is how $P_{\texttt{Agency}}$ handles the *retry* request, namely by recursion on X: the recursive call is activated each time the agency receives a *retry* request from the customer.

Operational semantics. *The value* v *of expression* e (notation $\mathsf{e} \downarrow \mathsf{v}$) is computed as expected, see Table 1. The successor operation `succ` is defined only on natural numbers, the negation `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.

The *computational rules of multiparty sessions* are given in Table 3. They are closed with respect to structural pre-congruence (a non-symmetric transitive preorder relation) 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

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

$$[\text{S-NIL}]\quad \mathsf{p} \triangleleft \mathbf{0} \mid \mathcal{M} \Rrightarrow \mathcal{M} \qquad [\text{S-COMM}]\quad \mathcal{M} \mid \mathcal{M}' \Rrightarrow \mathcal{M}' \mid \mathcal{M} \qquad [\text{S-ASSOC}]\quad (\mathcal{M} \mid \mathcal{M}') \mid \mathcal{M}'' \Rrightarrow \mathcal{M} \mid (\mathcal{M}' \mid \mathcal{M}'')$$

$$[\text{S-PAR}]\quad \frac{\mathcal{M} \Rrightarrow \mathcal{M}'}{\mathcal{M} \mid \mathcal{M}'' \Rrightarrow \mathcal{M}' \mid \mathcal{M}''}$$

Table 2: Structural pre-congruence.

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

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

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

Table 3: Reduction rules.

chooses to continue as P if the condition e evaluates to true and as Q if e evaluates to false. Rule [R-STRUCT] ensures that session reduction respects structural pre-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. 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

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

This mulitiparty session reduces to

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

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 $\texttt{succ}(100) \downarrow 101$, `Carol` sends 101 to `Alice`. We can then reduce the session to, for example, $\texttt{Alice} \triangleleft \mathbf{0}$, but not further.

Exercise 1. Prove the following:

1. $\texttt{Customer} \triangleleft P'_{\texttt{Customer}} \mid \texttt{Agency} \triangleleft P'_{\texttt{Agency}}$ reduces to $\texttt{Customer} \triangleleft \mathbf{0} \mid \texttt{Agency} \triangleleft \mathbf{0}$;
2. $\texttt{Customer} \triangleleft P_{\texttt{Customer}} \mid \texttt{Agency} \triangleleft P_{\texttt{Agency}}$ reduces to $\texttt{Customer} \triangleleft \mathbf{0} \mid \texttt{Agency} \triangleleft \mathbf{0}$.

From the end of Example 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 4, 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 when a multiparty session contains communications that will never be executed.

Definition 1. *A multiparty session $\mathcal{M}$ is* stuck *if $\mathcal{M} \not\Rightarrow \mathtt{p} \triangleleft \mathbf{0}$ and there is no multiparty session $\mathcal{M}'$ such that $\mathcal{M} \longrightarrow \mathcal{M}'$. A multiparty session $\mathcal{M}$ gets* stuck*, notation* $\texttt{stuck}(\mathcal{M})$*, if it reduces to a stuck multiparty session.*

The multiparty session $\mathcal{M}$ in Example 2 does not get stuck. A similar multiparty session, where instead of $P_{\texttt{Alice}}$ we take $P'_{\texttt{Alice}} = \texttt{Bob}!\ell_1\langle 50\rangle.\texttt{Carol}?\ell_5(x).\mathbf{0}$, gets stuck because of label mismatch.

4 Type System

This section introduces a type system for the calculus presented in Section 3 (the formulation is based on [7]). We formalise types and projections (Section 4.1), the subtyping relation (Section 4.2), and the typing rules and their properties (Section 4.3). All stated results in this paper are proved in [7].

4.1 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 2 (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* G*, are terms generated by the following grammar:*

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

We require that $\mathsf{p} \neq \mathsf{q}$*,* $I \neq \emptyset$*, 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 2, the type $\mathsf{p} \to \mathsf{q} : \{\ell_i(S_i).\mathsf{G}_i\}_{i \in I}$ formalises a protocol where participant p must send to 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 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., $\mu \mathbf{t}.\mathsf{p} \to \mathsf{q} : \ell(\mathtt{nat}).\mathbf{t}$ 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:

$$\mathtt{pt}\{\mu \mathbf{t}.\mathsf{G}\} = \mathtt{pt}\{\mathsf{G}\} \qquad \mathtt{pt}\{\mathtt{end}\} = \mathtt{pt}\{\mathbf{t}\} = \emptyset$$

$$\mathtt{pt}\{\mathsf{p} \to \mathsf{q} : \{\ell_i(S_i).G_i\}_{i \in I}\} = \{\mathsf{p}, \mathsf{q}\} \cup \mathtt{pt}\{\mathsf{G}_i\} \ (i \in I)$$

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

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

Definition 3 (Local Session Types). *The grammar of session types, ranged over by* T*, is:*

$$\mathsf{T} ::= \mathtt{end} \quad | \quad \&_{i \in I} \mathsf{p}?\ell_i(S_i).\mathsf{T}_i \quad | \quad \oplus_{i \in I} \mathsf{q}!\ell_i(S_i).\mathsf{T}_i \quad | \quad \mu \mathbf{t}.\mathsf{T} \quad | \quad \mathbf{t}$$

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, $\mathsf{p}?\ell(\mathtt{int}).\mathtt{end} \& \mathsf{p}?\ell(\mathtt{nat}).\mathtt{end}$ 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* $\&_{i \in I} \mathsf{p}?\ell_i(S_i).\mathsf{T}_i$ requires to wait to receive a value of sort S_i (for some $i \in I$) from the participant 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* $\oplus_{i \in I} \mathsf{q}!\ell_i(S_i).\mathsf{T}_i$ says that the participant implementing the type must choose a labelled message to send to q; if the participant chooses the message ℓ_i, for some $i \in I$, it must include in the message to 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

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

In what follows we introduce the concept of *projection* of a global type onto a participant, didactically into two steps: first we give the simpler version of the projection from [11,12], then we extend it with the *merging* operation. The second definition will extend the domain of the projection as a partial function on global types, i.e., the second version of the projection will be well-defined on a wider range of global types.

Definition 4. *The* projection (no-merging version) of a global type onto a participant r *is defined by recursion on* G*:*

- $\mathtt{end}\upharpoonright\mathsf{r} = \mathtt{end}$*;* [PROJ-END]
- $\mathbf{t}\upharpoonright\mathsf{r} = \mathbf{t}$*;* [PROJ-VAR]
- $(\mu\mathbf{t}.\mathsf{G})\upharpoonright\mathsf{r} = \mu\mathbf{t}.(\mathsf{G}\upharpoonright\mathsf{r})$ *if* $\mathsf{r} \in \mathtt{pt}\{\mathsf{G}\}$ *or* $\mu\mathbf{t}.\mathsf{G}$ *is not closed;* [PROJ-REC-1]
- $(\mu\mathbf{t}.\mathsf{G})\upharpoonright\mathsf{r} = \mathtt{end}$ *; otherwise* [PROJ-REC-2]
- $\mathsf{p} \to \mathsf{r} : \{\ell_i(S_i).\mathsf{G}_i\}_{i\in I}\upharpoonright\mathsf{r} = \&_{i\in I}\mathsf{p}?\ell_i(S_i).\mathsf{G}_i\upharpoonright\mathsf{r}$*;* [PROJ-IN]
- $\mathsf{r} \to \mathsf{q} : \{\ell_i(S_i).\mathsf{G}_i\}_{i\in I}\upharpoonright\mathsf{r} = \oplus_{i\in I}\mathsf{q}!\ell_i(S_i).\mathsf{G}_i\upharpoonright\mathsf{r}$*;* [PROJ-OUT]
- $\mathsf{p} \to \mathsf{q} : \{\ell_i(S_i).\mathsf{G}_i\}_{i\in I}\upharpoonright\mathsf{r} = \mathsf{G}_{i_0}\upharpoonright\mathsf{r}$ *where* $i_0 \in I$*,* *if* $\mathsf{r} \notin \{\mathsf{p},\mathsf{q}\}$ *and, for all* $i,j \in I$*,* $\mathsf{G}_i\upharpoonright\mathsf{r} = \mathsf{G}_j\upharpoonright\mathsf{r}$*;* [PROJ-CONT']
- *undefined otherwise.*

We describe the clauses of Definition 4:

[PROJ-END,PROJ-VAR] give the behaviour of projections on $\mathtt{end}$ and type variables;

[PROJ-REC-1,PROJ-REC-2] give the behaviour of projections on recursive types; in particular [PROJ-REC-2] is needed: if we applied only [PROJ-REC-1] to any recursive type, we will obtain $(\mu\mathbf{t}.\mathsf{p} \to \mathsf{q} : \ell(\mathtt{nat}).\mathbf{t})\upharpoonright\mathsf{r} = \mu\mathbf{t}.\mathbf{t}$, namely we will allow for unguarded occurrences of $\mathbf{t}$, which we do not accept as valid;

[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 $\&_{i\in I}\mathsf{p}?\ell_i(S_i).\mathsf{G}_i\upharpoonright\mathsf{r}$ (resp. $\oplus_{i\in I}\mathsf{q}!\ell_i(S_i).\mathsf{G}_i\upharpoonright\mathsf{r}$), provided that the continuations of $\&_{i\in I}\mathsf{p}?\ell_i(S_i).\mathsf{G}_i\upharpoonright\mathsf{r}$ (resp. $\oplus_{i\in I}\mathsf{q}!\ell_i(S_i).\mathsf{G}_i\upharpoonright\mathsf{r}$) are also projections of the corresponding global type continuations $\mathsf{G}_i\upharpoonright\mathsf{r}$.

[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 make sure that continuation is *the same* on all branches; just below we will see how this restriction can be relaxed.

Example 3. We now show an example of some projections of global types. Consider the global type (where p, q and r are pairwise distinct):

$$\mathsf{G} \;=\; \mathsf{p} \to \mathsf{q} : \{\ell_1(\mathtt{nat}).\mathsf{G}_1, \ell_2(\mathtt{bool}).\mathsf{G}_1\} \text{ where } \mathsf{G}_1 \;=\; \mathsf{q} \to \mathsf{r} : \{\ell_3(\mathtt{int}), \ell_4(\mathtt{nat})\}$$

We have:

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

Note that $\mathsf{G}\upharpoonright\mathsf{r}$ is well defined only because the continuation of the communication after the exchange $\mathsf{p} \to \mathsf{q}$ is equal, namely G_1, in both branches.

In the following Definition 5, we give a more permissive definition of projection, that handles also cases in which the continuation is not the same in all branches, but the types are somehow "compatible", namely they can be merged. Our definition follows [7], which extends [11,12], along the lines of [19] and [2]. i.e., it uses a *merging operator* $\bigsqcap$.

Definition 5. *The* projection of a global type onto a participant r *is defined by recursion on* G*:*

- $\mathtt{end}\upharpoonright\mathsf{r} = \mathtt{end}$*;* [PROJ-END]
- $\mathbf{t}\upharpoonright\mathsf{r} = \mathbf{t}$*;* [PROJ-VAR]
- $(\mu\mathbf{t}.\mathsf{G})\upharpoonright\mathsf{r} = \mu\mathbf{t}.(\mathsf{G}\upharpoonright\mathsf{r})$ *if* $\mathsf{r} \in \mathtt{pt}\{G\}$ *or* $\mu\mathbf{t}.\mathsf{G}$ *is not closed;* [PROJ-REC-1]
- $(\mu\mathbf{t}.\mathsf{G})\upharpoonright\mathsf{r} = \mathtt{end}$ *otherwise* [PROJ-REC-2]
- $\mathsf{p} \to \mathsf{r} : \{\ell_i(S_i).\mathsf{G}_i\}_{i\in I}\upharpoonright\mathsf{r} = \&_{i\in I}\mathsf{p}?\ell_i(S_i).\mathsf{G}_i\upharpoonright\mathsf{r}$*;* [PROJ-IN]
- $\mathsf{r} \to \mathsf{q} : \{\ell_i(S_i).\mathsf{G}_i\}_{i\in I}\upharpoonright\mathsf{r} = \oplus_{i\in I}\mathsf{q}!\ell_i(S_i).\mathsf{G}_i\upharpoonright\mathsf{r}$*;* [PROJ-OUT]
- $\mathsf{p} \to \mathsf{q} : \{\ell_i(S_i).\mathsf{G}_i\}_{i\in I}\upharpoonright\mathsf{r} = \bigsqcap_{i\in I}(\mathsf{G}_i\upharpoonright\mathsf{r})$ *if* $\mathsf{r} \notin \{\mathsf{p}, \mathsf{q}\}$*;* [PROJ-CONT]
- *undefined otherwise.*

Above, $\bigsqcap$ *is the* merging operator*, defined as the extension of the binary operation* $\sqcap$ *to arbitrary arities.* $\sqcap$ *is recursively defined as:*

- $\&_{i\in I}\mathsf{p}?\ell_i(S_i).\mathsf{T}_i \sqcap \&_{i\in J}\mathsf{p}?\ell_i(S_i).\mathsf{T}'_i = \&_{i\in I\cup J}\mathsf{p}?\ell_i(S_i).\mathsf{T}''_i$, *where*

$$\mathsf{T}''_i = \begin{cases} \mathsf{T}_i & i \in I \setminus J \\ \mathsf{T}'_i & i \in J \setminus I \\ \mathsf{T}_i \sqcap \mathsf{T}'_i & i \in I \cup J \end{cases}$$

- $\oplus_{i\in I}\mathsf{p}!\ell_i(S_i).\mathsf{T}_i \sqcap \oplus_{i\in I}\mathsf{p}!\ell_i(S_i).\mathsf{T}'_i = \oplus_{i\in I}\mathsf{p}!\ell_i(S_i).(\mathsf{T}_i \sqcap T'_i)$
- $\mu\mathbf{t}.\mathsf{T} \sqcap \mu\mathbf{t}.\mathsf{T}' = \mu\mathbf{t}.(\mathsf{T} \sqcap \mathsf{T}')$
- $\mathbf{t} \sqcap \mathbf{t} = \mathbf{t}$*;*
- $\mathtt{end} \sqcap \mathtt{end} = \mathtt{end}$*;*
- *undefined otherwise.*

Proposition 1. *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 5, merging a type with itself results in itself (rule [MRG-ID]). Moreover, Definition 5 allows to combine different external choices (rule [MRG-BRA]) *if and only if* common labels have identical sorts and identical continuations, as formalised in Proposition 2 below and illustrated in Examples 4 to 6 .

Proposition 2. *For two types* $\mathsf{T}' = \&_{i\in I}\mathsf{p}'?\ell_i(S_i).\mathsf{T}_i$ *and* $\mathsf{T}'' = \&_{j\in J}\mathsf{p}''?\ell_j(S_j).\mathsf{T}_j$*, 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* $\mathsf{T}_i \sqcap \mathsf{T}_j$ *is defined.*

Example 4. We now give some small examples that illustrate the definition of the merging operator (here, $i \neq j$ implies $\ell_i \neq \ell_j$):

$\mathtt{end} \sqcap \mathsf{q}?\ell_3(\mathtt{nat})$ undefined: the prefixes have different constructors

$\mathsf{q}!\ell(\mathtt{nat}) \sqcap \mathsf{q}!\ell(\mathtt{nat}) = \mathsf{q}!\ell(\mathtt{nat})$

$\mathsf{p}!\ell(\mathtt{nat}) \sqcap \mathsf{q}!\ell(\mathtt{nat})$ undefined: outputs to different participants

$\mathsf{q}!\ell_3(\mathtt{nat}) \sqcap \mathsf{q}!\ell_4(\mathtt{nat})$ undefined: outputs with different labels

$$\big(\mathsf{q}?\ell_3(\mathtt{int})\&\mathsf{q}?\ell_5(\mathtt{nat})\big) \sqcap \big(\mathsf{q}?\ell_4(\mathtt{int})\&\mathsf{q}?\ell_5(\mathtt{nat})\big) = \mathsf{q}?\ell_3(\mathtt{int})\&\mathsf{q}?\ell_4(\mathtt{int})\&\mathsf{q}?\ell_5(\mathtt{nat})$$

$\mathsf{q}?\ell_3(\mathtt{nat}) \sqcap \mathsf{q}?\ell_3(\mathtt{nat}).\mathsf{q}?\ell_3(\mathtt{nat})$
undefined: same prefixes, but $\mathtt{end} \sqcap \mathsf{q}?\ell_3(\mathtt{nat})$ is undefined

$\mathsf{q}?\ell(\mathtt{nat}) \sqcap \mathsf{q}?\ell(\mathtt{int})$ undefined: the payload sorts do not match

Now we understand better how clause [PROJ-CONT] works: it 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 5 allows participant r to receive different messages (from a same participant p') in different branches of a global type, as shown in Example 5 below.

Example 5. We demonstrate interesting points of Definition 5. First, we show some projections of global types. Consider the global type:

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

We have:

$$\begin{aligned} \mathsf{G}\upharpoonright\mathsf{p} \;&=\; \mathsf{q}!\ell_1(\mathtt{nat}).(\mathsf{G}_1\upharpoonright\mathsf{p}) \oplus \mathsf{q}!\ell_2(\mathtt{bool}).(\mathsf{G}_2\upharpoonright\mathsf{p}) \\ &=\; \mathsf{q}!\ell_1(\mathtt{nat}).\mathtt{end} \oplus \mathsf{q}!\ell_2(\mathtt{bool}).\mathtt{end} \end{aligned}$$

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

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

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 5, 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 6 below.

Example 6. We show two global types that cannot be projected according to the Definition 5. 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,

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

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 5 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,

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

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 5.

4.2 Subtyping

The subtyping relation $\leqslant$ is used to augment the flexibility of the type system (introduced in Section 4.3): 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 6 (Subsorting and subtyping). Subsorting $\leq:$ *is a binary relation on sorts such that* $S \leq: S'$ *if and only if* $S = S'$ *or* $(S, S') = (\mathtt{nat}, \mathtt{int})$.

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

$$\text{[SUB-END]} \qquad \mathtt{end} \leqslant \mathtt{end}$$

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

Intuitively, the session subtyping $\leqslant$ in Definition 6 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* [15, Chapter 21]

[SUB-END] says that $\mathtt{end}$ is only subtype of itself.

[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 $\mathtt{nat} \leq: \mathtt{nat}$.

[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 [6,3]. 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 [5].

[SUB-OUT] relates internal choices towards the same participant $\mathtt{p}$: 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 1. *The subtyping relation $\leqslant$ is reflexive and transitive.*

4.3 Type system

We now introduce a type system for the multiparty session calculus presented in Section 3. We distinguish three kinds of typing judgments:

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

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 $\mathtt{q}$, 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);

[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.

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

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

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

Table 4: Typing rules for expressions.

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

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

$$[\text{T-CHOICE}]\ \frac{\Gamma \vdash \mathsf{e} : \mathtt{bool} \quad \Gamma \vdash P_1 : \mathsf{T} \quad \Gamma \vdash P_2 : \mathsf{T}}{\Gamma \vdash \mathsf{if}\ \mathsf{e}\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2 : \mathsf{T}} \qquad [\text{T-SESS}]\ \frac{\forall i \in I \quad \vdash P_i : \mathsf{G}\upharpoonright \mathsf{p}_i \quad \mathtt{pt}\{\mathsf{G}\} \subseteq \{\mathsf{p}_i \mid i \in I\}}{\vdash \prod_{i \in I} \mathsf{p}_i \triangleleft P_i : \mathsf{G}}$$

Table 5: Typing rules for processes and sessions.

[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 7. We show that the multiparty session $\mathcal{M}$ from Example 2 is well typed. Consider the following global type:

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

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

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

where:

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

Now, let:

$$\mathsf{T}'_{\texttt{Alice}} = \texttt{Bob}!\ell_1(\texttt{nat}).\texttt{Carol}?\ell_3(\texttt{nat}).\texttt{end} \;\oplus\; \texttt{Bob}!\ell_4(\texttt{nat}).\texttt{Carol}?\ell_3(\texttt{nat}).\texttt{end}$$

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

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

we conclude:

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

Example 8. Let us consider processes $P_{\texttt{Customer}}$ and $P_{\texttt{Agency}}$ from Example 1. As a suitable global type for the session

$$\texttt{Customer} \triangleleft P_{\texttt{Customer}} \mid \texttt{Agency} \triangleleft P_{\texttt{Agency}}$$

we pick the following.

$$\begin{array}{l}\mathsf{G} \;=\; \mu\mathbf{t}.\, \texttt{Customer} \to \texttt{Agency} : \{\mathit{Hawaii}(\texttt{bool}).\mathsf{G}_1, \mathit{France}(\texttt{bool}).\mathsf{G}_1\}\\ \text{where}\\ \mathsf{G}_1 = \; \texttt{Agency} \to \texttt{Customer} : \{\; \mathit{quote}(\texttt{nat}).\texttt{Customer} \to \texttt{Agency} : \{\\ \qquad \mathit{accept}(\texttt{bool}).\texttt{Customer} \to \texttt{Agency} : \mathit{address}(\texttt{nat}).\\ \qquad\qquad\qquad\qquad \texttt{Agency} \to \texttt{Customer} : \mathit{date}(\texttt{nat}).\texttt{end},\\ \qquad \mathit{retry}(\texttt{bool}).\mathbf{t},\\ \qquad \mathit{reject}(\texttt{bool}).\texttt{end}\;\}\}\end{array}$$

Observe the recursive behaviour over $\mathbf{t}$. Now, let us project G onto its participant Agency (there is no need to merge types here, namely we can use Definition 4):

$$\begin{array}{lcl}\mathsf{G}{\upharpoonright}\texttt{Agency} &=& \mu\mathbf{t}.\, (\texttt{Customer}?\mathit{Hawaii}(\texttt{bool}).\mathsf{G}_1{\upharpoonright}\texttt{Agency}\\ &\&& \texttt{Customer}?\mathit{France}(\texttt{bool}).\mathsf{G}_1{\upharpoonright}\texttt{Agency})\\ \text{and}&&\\ \mathsf{G}_1{\upharpoonright}\texttt{Agency} &=& \texttt{Customer}!\mathit{quote}(\texttt{nat}).(\\ &&\quad (\texttt{Customer}?\mathit{accept}(\texttt{bool}).\texttt{Customer}?\mathit{address}(\texttt{nat}).\\ &&\qquad\qquad \texttt{Customer}!\mathit{date}(\texttt{nat}).\texttt{end})\\ &\&& \quad (\texttt{Customer}?\mathit{retry}(\texttt{bool}).\mathbf{t})\\ &\&& \quad (\texttt{Customer}?\mathit{reject}(\texttt{bool}).\texttt{end})\;)\end{array}$$

Also this projection presents a recursive behaviour as expected. The reader will now be able to derive:

$$\vdash \texttt{Customer} \triangleleft P_{\texttt{Customer}} \mid \texttt{Agency} \triangleleft P_{\texttt{Agency}} : \mathsf{G}.$$

Exercise 2. This exercise is intended to guide the reader to obtain the final result (using the same notation as in Example 1 and Example 8):

$$\vdash \mathtt{Customer} \triangleleft P_{\mathtt{Customer}} \mid \mathtt{Agency} \triangleleft P_{\mathtt{Agency}} : \mathsf{G}$$

Let us proceed step by step following Example 7.

1. Given the global type G and its projection $\mathsf{G}\upharpoonright\mathtt{Agency}$, derive $\mathsf{G}\upharpoonright\mathtt{Customer}$.
2. Then derive $\vdash P_{\mathtt{Customer}} : \mathsf{G}\upharpoonright\mathtt{Customer}$ and $\vdash P_{\mathtt{Agency}} : \mathsf{G}\upharpoonright\mathtt{Agency}$.

Hint 1. The reader might want to use the identification of $\mu\mathbf{t}.\mathsf{G}$ and $\mathsf{G}\{\mu\mathbf{t}.\mathsf{G}/\mathbf{t}\}$ (this holds both for global and session types), namely the possibility to unfold any recursive construct for types.

Hint 2. In order to prove point 2, the reader might want to proceed in two steps: first, finding appropriate session types $\mathsf{T}_{\mathtt{Customer}}$ and $\mathsf{T}_{\mathtt{Agency}}$ for processes $P_{\mathtt{Customer}}$ and $P_{\mathtt{Agency}}$ respectively, and second, proving $\mathsf{T}_{\mathtt{Customer}} \leqslant \mathsf{G}\upharpoonright\mathtt{Customer}$ and $\mathsf{T}_{\mathtt{Agency}} \leqslant \mathsf{G}\upharpoonright\mathtt{Agency}$.

Exercise 3.
- Prove that: $\vdash \mathtt{Customer} \triangleleft P'_{\mathtt{Customer}} \mid \mathtt{Agency} \triangleleft P_{\mathtt{Agency}} : \mathsf{G}$.
- Does the statement $\vdash \mathtt{Customer} \triangleleft P'_{\mathtt{Customer}} \mid \mathtt{Agency} \triangleleft P'_{\mathtt{Agency}} : \mathsf{G}$ hold? Justify your answer.

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 7 (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:*

$$\big(\mathsf{p} \to \mathsf{q} : \{\ell_i(S_i).\mathsf{G}_i\}_{i\in I}\big) \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q} \;=\; \mathsf{G}_k \qquad \text{if } \exists k \in I : \ell = \ell_k$$

$$\big(\mathsf{r} \to \mathsf{s} : \{\ell_i(S_i).\mathsf{G}_i\}_{i\in I}\big) \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q} \;=\; \mathsf{r} \to \mathsf{s} : \{\ell_i(S_i).\mathsf{G}_i \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}\}_{i\in I} \quad \text{if } \begin{cases} \{\mathsf{r},\mathsf{s}\} \cap \{\mathsf{p},\mathsf{q}\} = \emptyset \\ \forall i \in I : \{\mathsf{p},\mathsf{q}\} \subseteq \mathsf{G}_i \end{cases}$$

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

Example 9. 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 5, and their types denoted as G and G_1, respectively. Also take the projection:

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

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:

$$\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell_1} \mathsf{q} = \ \mathsf{G}_1 \ = \mathsf{q} \to \mathsf{r} : \{\ell_3(\mathtt{int}), \ell_5(\mathtt{nat})\}$$

$$(\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell_1} \mathsf{q})\upharpoonright\mathsf{r} = \mathsf{G}_1\upharpoonright\mathsf{r} = \mathsf{q}?\ell_3(\mathtt{int})\&\mathsf{q}?\ell_5(\mathtt{nat})$$

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 1 (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 2 (Progress). *If* $\vdash \mathcal{M} : \mathsf{G}$, *then either* $\mathcal{M} \Rrightarrow \mathsf{p} \triangleleft \mathbf{0}$ *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 (Type Safety). *If* $\vdash \mathcal{M} : \mathsf{G}$, *then it does not hold* $\mathtt{stuck}(\mathcal{M})$.

Proof. Direct consequence of Theorem 1, Theorem 2, and Definition 1.

Finally the reader wishes to learn more about the full asynchronous multiparty session types (which includes channel passing, asynchrony (FIFO queues), shared names and parameterised recursion) can proceed to [1]. The article [16] also discusses type soundness of various MPST calculi.

References

1. Coppo, M., Dezani-Ciancaglini, M., Padovani, L., Yoshida, N.: A gentle introduction to multiparty asynchronous session types. In: SFM. LNCS, vol. 9104, pp. 146–178. Springer (2015)
2. Deniélou, P., Yoshida, N., Bejleri, A., Hu, R.: Parameterised multiparty session types. Logical Methods in Computer Science **8**(4) (2012). https://doi.org/10.2168/LMCS-8(4:6)2012
3. Gay, S., Hole, M.: Subtyping for session types in the pi calculus. Acta Informatica **42**(2/3), 191–225 (2005). https://doi.org/10.1007/s00236-005-0177-z
4. Gay, S., Ravera, A. (eds.): Behavioural Types: from Theory to Tools. River Publishers (2017)

5. Gay, S.J.: 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. LNCS, vol. 9600, pp. 95–108. Springer International Publishing (2016). https://doi.org/10.1007/978-3-319-30936-1_5
6. Gay, S.J., Hole, M.: Types and subtypes for client-server interactions. In: ESOP. pp. 74–90 (1999). https://doi.org/10.1007/3-540-49099-X_6
7. Ghilezan, S., Jaksic, S., Pantovic, J., Scalas, A., Yoshida, N.: Precise subtyping for synchronous multiparty sessions. J. Log. Algebr. Meth. Program. **104**, 127–173 (2019). https://doi.org/10.1016/j.jlamp.2018.12.002
8. Honda, K.: Types for dyadic interaction. In: CONCUR'93. pp. 509–523 (1993)
9. Honda, K., Mukhamedov, A., Brown, G., Chen, T.C., Yoshida, N.: Scribbling interactions with a formal foundation. In: ICDCIT. LNCS, vol. 6536, pp. 55–75. Springer (2011). https://doi.org/10.1007/978-3-642-19056-8_4
10. Honda, K., Vasconcelos, V.T., Kubo, M.: Language primitives and type disciplines for structured communication-based programming. In: ESOP. LNCS, vol. 1381, pp. 22–138. Springer (1998). https://doi.org/10.1007/BFb0053567
11. Honda, K., Yoshida, N., Carbone, M.: Multiparty asynchronous session types. In: POPL. pp. 273–284. ACM Press (2008). https://doi.org/10.1145/1328438.1328472
12. Honda, K., Yoshida, N., Carbone, M.: Multiparty asynchronous session types. Journal of ACM **63**, 1–67 (2016)
13. 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.: Foundations of session types and behavioural contracts. ACM Comput. Surv. **49**(1) (2016). https://doi.org/10.1145/2873052
14. Kouzapas, D., Yoshida, N.: Globally governed session semantics. In: CONCUR. LNCS, vol. 8052, pp. 395–409. Springer (2013). https://doi.org/10.1145/1328438.1328472
15. Pierce, B.C.: Types and Programming Languages. MIT Press (2002)
16. Scalas, A., Yoshida, N.: Less is more: Multiparty session types revisited. ACM Program. Lang. **POPL** (2019). https://doi.org/10.1145/3290343
17. Scribble home page, http://www.scribble.org
18. Takeuchi, K., Honda, K., Kubo, M.: An Interaction-based Language and its Typing System. In: PARLE'94. LNCS, vol. 817, pp. 398–413 (1994). https://doi.org/10.1007/3-540-58184-7_118
19. Yoshida, N., Deniélou, P., Bejleri, A., Hu, R.: Parameterised multiparty session types. In: FOSSACS. LNCS, vol. 6014, pp. 128–145. Springer (2010). https://doi.org/10.1007/978-3-642-12032-9_10
20. Yoshida, N., Hu, R., Neykova, R., Ng, N.: The scribble protocol language. In: TGC. LNCS, vol. 8358, pp. 22–41. Springer (2013)