Less is More Revisited
Association with Global Multiparty Session Types

Nobuko Yoshida(✉) and Ping Hou

University of Oxford, Oxford, UK
{nobuko.yoshida,ping.hou}@cs.ox.ac.uk

Abstract. Multiparty session types (MPST) [12] provide a type discipline where a programmer or architect specifies a whole view of communications as a *global protocol*, and each distributed program is locally type-checked against its *end-point projection*. After 10 years from the birth of MPST, Scalas and Yoshida [18] discovered that the *proofs* of type safety in the literature which use the end-point projection with *mergeability* are flawed. After this paper, researchers wrongly believed that the end-point projection (with mergeability) was *unsound*. We correct this misunderstanding, proposing a new general proof technique for type soundness of multiparty session π-calculus, which uses an *association* relation between a global type and its end-point projection.

1 Introduction

Today many computer science technologies are centred on *data science* and *machine learning*, which fall within the field of technology used to leverage data for creating and innovating products, services, and infrastructural systems in society. In 1996, C. B. Jones, together with J.R. Gurd, predicted this era of *data*. In their article, *the Global-yet-Personal Information System* (GyP∫IS), in *Computing Tomorrow: future research directions in computer science* edited by Wand and Milner [9], they argued that extracting meaning from data, and understanding and building methods that utilise data to inform predictions will be the central concerns in future computing, proposing the system called GyP∫IS. The thesis of GyP∫IS is that data should be *globally* consistent, *open* for everybody, and yet *personalised* (structured with respect to an individual need). They predicted that future systems would go beyond the scope of the known programming paradigm, and argued for the importance of describing and formally specifying *interactive behaviours*.

Multiparty session types (MPST) [12,13] represent a type discipline which attempts to take a step to meet the GyP∫IS challenge, by offering a programming framework to guarantee a global consistency among interactive agents or processes in the open system [10,19]. It facilitates the description, specification

Work supported by: EPSRC EP/T006544/2, EP/K011715/1, EP/K034413/1, EP/L00058X/1, EP/N027833/2, EP/N028201/1, EP/T014709/2, EP/V000462/1, EP/X015955/1, NCSS/EPSRC VeTSS, and Horizon EU TaRDIS 101093006.

A. Cavalcanti and J. Baxter (Eds.): *The Practice of Formal Methods*, LNCS 14781, pp. 268–291, 2024.
https://doi.org/10.1007/978-3-031-66673-5_14

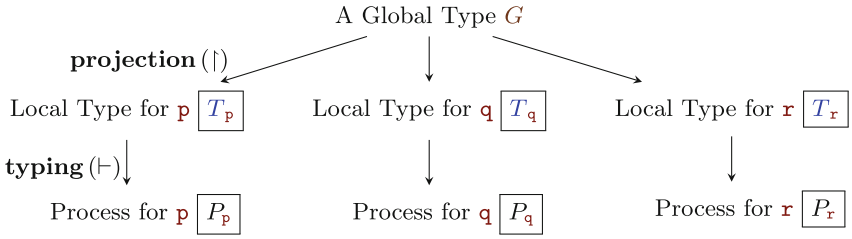


Fig. 1. Top-down methodology of multiparty session types

and verification of communications in concurrent and distributed systems based on *multiparty protocols*.

The design of multiparty protocols begins with a given *global* type (G, top in Fig. 1), and each participant's implementation (process) $P_{\mathtt{p}}$ (bottom) relies on its *local* type $T_{\mathtt{p}}$ (middle), obtained by the end-point projection of the global type. The global and local types reflect the global and local communication behaviours, respectively. Well-typed implementations (processes) that conform to a global type are guaranteed to be *correct by construction*, enjoying safety, deadlock-freedom, and liveness of interactions. In this paper, we demonstrate the rigour of the MPST *top-down* approach, revisiting the work of Scalas and Yoshida [18].

Misunderstanding on the Top-Down MPST. The work [18] has proposed a general MPST framework (called the *bottom-up* approach), which does not require global types. In the bottom-up approach, safety of processes is enforced by directly checking safety of a set of local types. This approach offers more typability, but is computationally more expensive. For worse, in the asynchronous MPST where processes communicating via infinite FIFO queues, type-checking using the bottom-up approach is undecidable, while in the top-down approach, it is decidable (as the projection is decidable).

The work [18] also pointed out that proofs of some of the top-down typing systems, which use the projection with *mergeability*, are flawed. The notion of mergeability was first introduced in [3] for modelling the end-point projection of Web Service Choreography Description Language (WS-CDL) [8]. Its aim is to broaden the typability of processes by allowing more projectable (well-formed) global types than the original limited projection [12]. Mergeability is implemented in the Scribble protocol description language [11,22,23] and other related MPST tools. Without using mergeability, most global protocols are not projectable. We define the projection and mergeability formally in Definition 3.

The statement in [18] has caused misunderstandings among researchers, leading to claims such as "the top-down approach (with mergeability) is *unsound*", and subsequently, assertions that "global types are *problematic*".

This paper proves that we can build a sound typing system using the end-point projection with mergeability, and more importantly, a utilisation of global types leads to a clear proof method for the type soundness.

Outline. Section 2 recalls the multiparty session calculus from [18]; Section 3 defines global and local types, and introduces the *association relation* $G \sqsubseteq_s \Gamma$, where global type G and a set of local types Γ of session name s are related up to *subtyping*. We then prove the sound and complete correspondence between global and local type semantics with respect to $\sqsubseteq$; Section 4 defines the typing system using $\sqsubseteq$ and proves type safety, deadlock-freedom, and liveness of typable processes; and Section 5 concludes with *Questar* by Mario T. Full proofs and auxiliary material are available in the extended version of the paper [21].

2 Multiparty Session π-Calculus

This section presents the syntax of the synchronous multiparty session π-calculus, and provides a formalisation of its operational semantics.

Syntax of Processes in Session π -Calculus. The multiparty session π-calculus models the behaviour of processes that interact using multiparty channels. For simplicity of presentation, our calculus is streamlined to focus on communication; standard extensions, such as with expressions and "if. . . then. . . else" statements, are routine and orthogonal to our formulation.

Definition 1 (Syntax of Multiparty Session π-Calculus). *The* multiparty session π-calculus *syntax is defined as follows:*

$$
\begin{array}{rcll}
c & ::= & x \mid s[\mathtt{p}] & \textit{(variable or channel for session } s \textit{ with role } \mathtt{p}\textit{)} \\
d & ::= & v \mid c & \textit{(basic value, variable, or channel with role)} \\
w & ::= & v \mid s[\mathtt{p}] & \textit{(basic value or channel with role)} \\
P, Q & ::= & \mathbf{0} \mid (\nu s)\, P & \textit{(inaction, restriction)} \\
 & & c[\mathtt{q}] \oplus \mathtt{m}\langle d \rangle . P & \textit{(selection towards role } \mathtt{q}\textit{)} \\
 & & c[\mathtt{q}] \& \{\mathtt{m}_i(x_i).P_i\}_{i \in I} & \textit{(branching from role } \mathtt{q} \textit{ with an index set } I \neq \emptyset\textit{)} \\
 & & \mathbf{def}\ D\ \mathbf{in}\ P \mid X\langle \widetilde{d} \rangle & \textit{(process definition, process call)} \\
 & & P \mid Q \mid \mathtt{err} & \textit{(parallel composition, error)} \\
D & ::= & X(\widetilde{x}) = P & \textit{(declaration of process variable } X\textit{)}
\end{array}
$$

$\mathtt{p}, \mathtt{q}, \ldots$ *denote* roles *belonging to a set* $\mathcal{R}$*;* $s, s', \ldots$ *denote* sessions*;* $x, y, \ldots$ *denote* variables*;* $\mathtt{m}, \mathtt{m}', \ldots$ *denote* message labels*; and* $X, Y, \ldots$ *denote* process variables. *Restriction, branching, and process definitions and declarations act as binders, as expected;* $\mathrm{fc}(P)$ *is the set of* free channels with roles *in* P*, and* $\mathrm{fv}(P)$ *is the set of* free variables *in* P*. We adopt a form of Barendregt convention: bound sessions and process variables are assumed pairwise distinct, and different from free ones. We write* $\Pi_{i \in I} P_i$ *for the parallel composition of processes* P_i*.*

A *session* is a sequence of interactions, including send and receive operations, performed by a set of *roles* (*participants*) in a communication protocol.

The syntax of our calculus (Definition 1) is mostly standard [18]. v is a *basic value* (e.g. unit (), integers, strings). A *channel with role* (a.k.a. *session endpoint*) $s[\mathtt{p}]$ is a multiparty communication endpoint whose user plays role $\mathtt{p}$ in the session

$$\text{[R-}\oplus\&\text{]}\ s[\mathtt{p}][\mathtt{q}]\&\{\mathtt{m}_i(x_i).P_i\}_{i\in I} \mid s[\mathtt{q}][\mathtt{p}]\oplus\mathtt{m}_k\langle w\rangle.Q \ \to\ P_k\{^{w}/_{x_k}\} \mid Q \ \text{ if } \ k\in I$$

$$\text{[R-}X\text{]}\ \mathbf{def}\ X(x_1,\ldots,x_n)=P\ \mathbf{in}\ (X\langle w_1,\ldots,w_n\rangle \mid Q) \to \mathbf{def}\ X(x_1,\ldots,x_n)=P\ \mathbf{in}\ (P\{^{w_1}/_{x_1}\}\cdots\{^{w_n}/_{x_n}\} \mid Q)$$

$$\text{[R-Ctx]}\ P\to P' \ \text{ implies } \ \mathbb{C}[P]\to\mathbb{C}[P']$$

$$\text{[R-Err]}\ s[\mathtt{p}][\mathtt{q}]\&\{\mathtt{m}_i(x_i).P_i\}_{i\in I} \mid s[\mathtt{q}][\mathtt{p}]\oplus\mathtt{m}\langle w\rangle.Q \ \to\ \mathtt{err} \ \text{ if } \ \forall i\in I:\mathtt{m}_i\neq\mathtt{m}$$

$$\text{[R-}\equiv\text{]}\ P'\equiv P\to Q\equiv Q' \ \text{ implies } \ P'\to Q'$$

Fig. 2. Operational semantics of multiparty session π-calculus.

$$P\mid Q\equiv Q\mid P \quad (P\mid Q)\mid R\equiv P\mid(Q\mid R) \quad P\mid\mathbf{0}\equiv P \quad (\nu s)\,\mathbf{0}\equiv\mathbf{0}$$

$$(\nu s)\,(\nu s')\,P\equiv(\nu s')\,(\nu s)\,P \qquad (\nu s)\,(P\mid Q)\equiv P\mid(\nu s)\,Q \ \text{ if } \ s\notin\mathrm{fc}(P)$$

$$\mathbf{def}\ D\ \mathbf{in}\ \mathbf{0}\equiv\mathbf{0} \qquad \mathbf{def}\ D\ \mathbf{in}\ (\nu s)\,P\ \equiv\ (\nu s)\,(\mathbf{def}\ D\ \mathbf{in}\ P) \ \text{ if } \ s\notin\mathrm{fc}(D)$$

$$\mathbf{def}\ D\ \mathbf{in}\ (P\mid Q)\ \equiv\ (\mathbf{def}\ D\ \mathbf{in}\ P)\mid Q \ \text{ if } \ \mathrm{dpv}(D)\cap\mathrm{fpv}(Q)=\emptyset$$

$$\mathbf{def}\ D\ \mathbf{in}\ (\mathbf{def}\ D'\ \mathbf{in}\ P)\ \equiv\ \mathbf{def}\ D'\ \mathbf{in}\ (\mathbf{def}\ D\ \mathbf{in}\ P)$$
$$\text{if } \ (\mathrm{dpv}(D)\cup\mathrm{fpv}(D))\cap\mathrm{dpv}(D')\ =\ (\mathrm{dpv}(D')\cup\mathrm{fpv}(D'))\cap\mathrm{dpv}(D)\ =\ \emptyset$$

Fig. 3. Standard structural congruence rules, where $\mathrm{fpv}(D)$ is the set of *free process variables* in D, and $\mathrm{dpv}(D)$ is the set of *declared process variables* in D.

s. *Inaction* $\mathbf{0}$ represents a terminated process (and is often omitted). *Session restriction* $(\nu s)\,P$ declares a new session s with its scope restricted to the process P. *Selection* (a.k.a. *internal choice*) $c[\mathtt{q}]\oplus\mathtt{m}\langle d\rangle.P$ sends a message $\mathtt{m}$ with payload d to role $\mathtt{q}$ via endpoint c, where c may be a variable or channel with role, while d may also be a basic value. *Branching* (a.k.a. *external choice*) $c[\mathtt{q}]\&\{\mathtt{m}_i(x_i).P_i\}_{i\in I}$ expects to receive a message $\mathtt{m}_i$ (for some $i\in I$) from role $\mathtt{q}$ via endpoint c, and then continues as P_i. *Process definition* $\mathbf{def}\ D\,\mathbf{in}\,P$ and *process call* $X\langle\tilde{d}\rangle$ capture recursion: the call invokes X by expanding it into P, and replacing its formal parameters with the actual ones. *Parallel composition* $P|Q$ denotes two processes capable of concurrent execution and potential communication. Finally, $\mathtt{err}$ represents the *error* process.

Operational Semantics of Session π-Calculus. We provide the operational semantics of our multiparty session π-calculus in Definition 2, using a standard *structural congruence* $\equiv$ defined in Fig. 3.

Definition 2 (Semantics of Multiparty Session π-Calculus). *A* reduction context $\mathbb{C}$ *is defined as:* $\mathbb{C}\ ::=\ \mathbb{C}\mid P\ \big|\ (\nu s)\,\mathbb{C}\ \big|\ \mathbf{def}\ D\,\mathbf{in}\,\mathbb{C}\ \big|\ [\,]$. *The* reduction $\to$ *is inductively defined in Fig. 2. We denote* $\to^*$ *as the reflexive and transitive closure of* $\to$. *We say that* P has an error *iff* $\exists\mathbb{C}$ *with* $P=\mathbb{C}[\mathtt{err}]$.

Our operational semantics is standard [18]. The reduction context $\mathbb{C}$ defines a process with a single placeholder $[]$, serving as a substitute for a specific subterm P. Rule [R-$\oplus\&$] describes a communication on session s involving receiver $\mathtt{p}$ and sender $\mathtt{q}$, provided that the transmitted message $\mathtt{m}_k$ is handled by the receiver ($k\in I$). In case of a message label mismatch, rule [R-Err] is invoked to trigger an $\mathtt{error}$. Rule [R-X] initiates the expansion of process definitions when

called. Additionally, rules [R-Ctx] and [R-≡] facilitate the reduction of processes under reduction contexts and modulo structural congruence, respectively.

Example 1 (Syntax and Semantics of Session π-Calculus, from [1]). Processes P and Q below communicate on a session s: P uses the endpoint $s[\mathtt{p}]$ to send an endpoint $s[\mathtt{r}]$ to role $\mathtt{q}$; Q uses the endpoint $s[\mathtt{q}]$ to receive an endpoint x, then sends a message to role $\mathtt{p}$ via x.

$$P = s[\mathtt{p}][\mathtt{q}]\oplus\mathtt{m}'\langle s[\mathtt{r}]\rangle.s[\mathtt{p}][\mathtt{r}]\&\mathtt{m}(x).\mathbf{0} \qquad Q = s[\mathtt{q}][\mathtt{p}]\&\mathtt{m}'(x).x[\mathtt{p}]\oplus\mathtt{m}\langle 42\rangle.\mathbf{0}$$

By Definition 2, successful reductions yield:

$$\begin{aligned} & (\nu s)\,(P \mid Q) \\ = {} & (\nu s)\,(s[\mathtt{p}][\mathtt{q}]\oplus\mathtt{m}'\langle s[\mathtt{r}]\rangle.s[\mathtt{p}][\mathtt{r}]\&\mathtt{m}(x).\mathbf{0} \mid s[\mathtt{q}][\mathtt{p}]\&\mathtt{m}'(x).x[\mathtt{p}]\oplus\mathtt{m}\langle 42\rangle.\mathbf{0}) \\ [\text{R-}\oplus\&] \to {} & (\nu s)\,(s[\mathtt{p}][\mathtt{r}]\&\mathtt{m}(x).\mathbf{0} \mid s[\mathtt{r}][\mathtt{p}]\oplus\mathtt{m}\langle 42\rangle.\mathbf{0}) \\ [\text{R-}\oplus\&] \to {} & (\nu s)\,\mathbf{0} \equiv \mathbf{0} \end{aligned}$$

Example 2 (OAuth Process, extended from [18]). The following process interacts on session s using channels with role $s[\mathtt{s}]$, $s[\mathtt{c}]$, $s[\mathtt{a}]$, to play resp. roles $\mathtt{s}$, $\mathtt{c}$, $\mathtt{a}$. For brevity, we omit irrelevant message payloads.

$$(\nu s)\,(P_{\mathtt{s}} \mid P_{\mathtt{c}} \mid P_{\mathtt{a}}) \quad \text{where:} \begin{cases} P_{\mathtt{s}} = s[\mathtt{s}][\mathtt{c}]\oplus\mathtt{cancel} \\ P_{\mathtt{c}} = s[\mathtt{c}][\mathtt{s}]\&\left\{\begin{array}{l}\mathtt{login}.s[\mathtt{c}][\mathtt{a}]\oplus\mathtt{passwd}\langle \mathit{"XYZ"}\rangle \\ \mathtt{cancel}.s[\mathtt{c}][\mathtt{a}]\oplus\mathtt{quit},\mathtt{fail}.s[\mathtt{c}][\mathtt{a}]\oplus\mathtt{fatal}\end{array}\right\} \\ P_{\mathtt{a}} = s[\mathtt{a}][\mathtt{c}]\&\{\mathtt{passwd}(y).s[\mathtt{a}][\mathtt{s}]\oplus\mathtt{auth}\langle \mathit{"secret"}\rangle,\mathtt{quit},\mathtt{fatal}\} \end{cases}$$

Here, $(\nu s)\,(P_{\mathtt{s}} \mid P_{\mathtt{c}} \mid P_{\mathtt{a}})$ is the parallel composition of processes $P_{\mathtt{s}}$, $P_{\mathtt{c}}$, $P_{\mathtt{a}}$ in the scope of session s. In $P_{\mathtt{s}}$, "$s[\mathtt{s}][\mathtt{c}]\oplus\mathtt{cancel}$" means: use $s[\mathtt{s}]$ to send `cancel` to $\mathtt{c}$. Process $P_{\mathtt{c}}$ uses $s[\mathtt{c}]$ to receive `login`, `cancel`, or `fatal` from $\mathtt{s}$; then, in the first case, it uses $s[\mathtt{c}]$ to send `passwd` to $\mathtt{a}$; in the second case, it uses $s[\mathtt{c}]$ to send `quit` to $\mathtt{a}$; in the third case, it uses $s[\mathtt{c}]$ to send `fatal` to $\mathtt{a}$. By Definition 2, we have the reductions:

$$(\nu s)\,(P_{\mathtt{s}} \mid P_{\mathtt{c}} \mid P_{\mathtt{a}}) \to (\nu s)\,(\mathbf{0} \mid s[\mathtt{c}][\mathtt{a}]\oplus\mathtt{quit} \mid P_{\mathtt{a}}) \to (\nu s)\,(\mathbf{0}|\mathbf{0}|\mathbf{0}) \equiv \mathbf{0}$$

3 Multiparty Session Types

This section introduces multiparty session types. We provide an extensive exploration Sect. 3.1, including syntax, projection, and subtyping. We establish a Labelled Transition System (LTS) semantics for both global types (Sect. 3.2) and typing contexts (Sect. 3.3). We explain the operational relationship between these two semantics in Sect. 3.4. Furthermore, we demonstrate that a typing context obtained via projection ensures safety, deadlock-freedom, and liveness in Sect. 3.5.

$$
\begin{array}{lcll}
B & ::= & \mathsf{int} \mid \mathsf{bool} \mid \mathsf{real} \mid \mathsf{unit} \mid \ldots & \text{Basic types} \\
S & ::= & B \mid T & \text{Basic type or Session type} \\
G & ::= & \mathtt{p}{\to}\mathtt{q}\colon\{\mathtt{m}_i(S_i).G_i\}_{i\in I} & \text{Transmission} \\
 & \mid & \mu\mathbf{t}.G \mid \mathbf{t} \mid \mathbf{end} & \text{Recursion, Type variable, Termination} \\
T & ::= & \mathtt{p}\&\{\mathtt{m}_i(S_i).T_i\}_{i\in I} & \text{External choice} \\
 & \mid & \mathtt{p}\oplus\{\mathtt{m}_i(S_i).T_i\}_{i\in I} & \text{Internal choice} \\
 & \mid & \mu\mathbf{t}.T \mid \mathbf{t} \mid \mathbf{end} & \text{Recursion, Type variable, Termination}
\end{array}
$$

Fig. 4. Syntax of types.

3.1 Global and Local Types

The Multiparty Session Type (MPST) theory uses *global types* to provide a comprehensive overview of communication between *roles*, such as $\mathtt{p}, \mathtt{q}, \mathtt{s}, \mathtt{t}, \ldots$, belonging to a set $\mathcal{R}$. In contrast, it employs *local types*, which are obtained via *projection* from a global type, to describe how an *individual* role communicates with other roles from a local viewpoint. The syntax of global and local types is presented in Fig. 4, where constructs are mostly standard [18].

Basic types are taken from a set $\mathcal{B}$, and describe types of values, ranging over integers, booleans, real numbers, units, *etc.*

Global types range over $G, G', G_i, \ldots$, and describe the high-level behaviour for all roles. The set of roles in a global type G is denoted by $\mathrm{roles}(G)$. We explain each syntactic construct of global types.

- $\mathtt{p}{\to}\mathtt{q}\colon\{\mathtt{m}_i(S_i).G_i\}_{i\in I}$: a *transmission*, denoting a message from role $\mathtt{p}$ to role $\mathtt{q}$, with a label $\mathtt{m}_i$, a payload of type S_i (either basic or local type), and a continuation G_i, where i is taken from an index set I. We require that the index set be non-empty ($I \neq \emptyset$), labels $\mathtt{m}_i$ be pair-wise distinct, and self receptions be excluded (i.e. $\mathtt{p} \neq \mathtt{q}$).
- $\mu\mathbf{t}.G$: a *recursive* global type, where contractive requirements apply [16, §21.8], i.e. each recursion variable $\mathbf{t}$ is bound within a $\mu\mathbf{t}.\ldots$ and is guarded.
- $\mathbf{end}$: a *terminated* global type (omitted where unambiguous).

Local types (or *session types*) range over $T, U, \ldots$, and describe the behaviour of a single role. We elucidate each syntactic construct of local types.

- $\mathtt{p}\oplus\{\mathtt{m}_i(S_i).T_i\}_{i\in I}$: an *internal choice* (*selection*), indicating that the *current* role is expected to *send* to role $\mathtt{p}$.
- $\mathtt{p}\&\{\mathtt{m}_i(S_i).T_i\}_{i\in I}$: an *external choice* (*branching*), indicating that the *current* role is expected to *receive* from role $\mathtt{p}$.
- $\mu\mathbf{t}.T$: a *recursive* local type, similar to recursive global types.
- $\mathbf{end}$: a *termination* (omitted where unambiguous).

Similar to global types, local types also need pairwise distinct, non-empty labels.

Projection is a *partial* function that provides the local type associated with a participating role in a global type. It takes a global type G and a role $\mathtt{p}$, returning the corresponding local type. Our definition of projection, as given in Definition 3 below, is standard [18].

Definition 3 (Global Type Projection). *The* projection of a global type G *onto a role* $\mathtt{p}$*, written* $G \upharpoonright \mathtt{p}$*, is:*

$$(\mathtt{q}\to\mathtt{r}\colon\{\mathtt{m}_i(S_i).G_i\}_{i\in I})\upharpoonright\mathtt{p} = \begin{cases} \mathtt{r}\oplus\{\mathtt{m}_i(S_i).(G_i\upharpoonright\mathtt{p})\}_{i\in I} & \text{if } \mathtt{p}=\mathtt{q} \\ \mathtt{q}\&\{\mathtt{m}_i(S_i).(G_i\upharpoonright\mathtt{p})\}_{i\in I} & \text{if } \mathtt{p}=\mathtt{r} \\ \bigsqcap_{i\in I} G_i\upharpoonright\mathtt{p} & \text{if } \mathtt{p}\neq\mathtt{q} \text{ and } \mathtt{p}\neq\mathtt{r} \end{cases}$$

$$(\mu\mathbf{t}.G)\upharpoonright\mathtt{p} = \begin{cases} \mu\mathbf{t}.(G\upharpoonright\mathtt{p}) & \text{if } \mathtt{p}\in\mathrm{roles}(G) \text{ or } \mathrm{fv}(\mu\mathbf{t}.G)\neq\emptyset \\ \mathbf{end} & \text{otherwise} \end{cases} \qquad \begin{array}{r}\mathbf{t}\upharpoonright\mathtt{p}=\mathbf{t}\\ \mathbf{end}\upharpoonright\mathtt{p}=\mathbf{end}\end{array}$$

where $\bigsqcap$ *is the* merge operator for session types *(*full merging*), defined as:*

$$\begin{array}{c} \mathtt{p}\oplus\{\mathtt{m}_i(S_i).T_i\}_{i\in I} \sqcap \mathtt{p}\oplus\{\mathtt{m}_i(S_i).T'_i\}_{i\in I} = \mathtt{p}\oplus\{\mathtt{m}_i(S_i).(T_i\sqcap T'_i)\}_{i\in I} \\ \mu\mathbf{t}.T\sqcap\mu\mathbf{t}.T' = \mu\mathbf{t}.(T\sqcap T') \quad \mathbf{t}\sqcap\mathbf{t}=\mathbf{t} \quad \mathbf{end}\sqcap\mathbf{end}=\mathbf{end} \\ \mathtt{p}\&\{\mathtt{m}_i(S_i).T_i\}_{i\in I} \sqcap \mathtt{p}\&\{\mathtt{m}_j(S_j).T'_j\}_{j\in J} = \mathtt{p}\&\{\mathtt{m}_k(S_k).T''_k\}_{k\in I\cup J} \\ \text{where} \quad T''_k = \begin{cases} T_k\sqcap T'_k & \text{if } k\in I\cap J \\ T_k & \text{if } k\in I\setminus J \\ T'_k & \text{if } k\in J\setminus I \end{cases} \end{array}$$

It is noteworthy that there are global types that cannot be projected onto all of their participants. This occurs when certain global types may describe protocols that are inherently meaningless, leading to undefined merging operations, as illustrated in Example 4 following.

If a global type G starts with a transmission from role $\mathtt{p}$ to role $\mathtt{q}$, projecting it onto role $\mathtt{p}$ (resp. $\mathtt{q}$) results in an internal (resp. external) choice, provided that the continuation of each branching of G is also projectable. When projecting G onto other participants $\mathtt{r}$ ($\mathtt{r}\neq\mathtt{p}$ and $\mathtt{r}\neq\mathtt{q}$), a merge operator, as defined in Definition 3 and exemplified in Example 4 below, is used to ensure that the projections of all continuations are "compatible".

If a global type G is a termination or type variable, projecting it onto any role results in a termination or type variable, respectively. Finally, the projection of a recursive global type $\mu\mathbf{t}.G$ preserves its recursive construct when projected onto a role in G, or if there are no free type variables in $\mu\mathbf{t}.G$; otherwise, the projection yields termination.

Subtyping. We introduce a *subtyping* relation $\leqslant$ on local types, as defined in Definition 4. This subtyping relation is standard [18], and will be used later when defining local type semantics and establishing the relationship between global and local type semantics.

Definition 4 (Subtyping). *Given a standard subtyping* $<:$ *for basic types (e.g. including* $\mathrm{int} <: \mathrm{real}$*), the* session subtyping relation $\leqslant$ *is coinductively defined:*

$$\frac{B <: B'}{B \leqslant B'}\ [\text{Sub-}B] \qquad \frac{\forall i \in I \quad S_i \leqslant S_i' \quad T_i \leqslant T_i'}{\mathtt{p}\&\{\mathtt{m}_i(S_i).T_i\}_{i\in I} \leqslant \mathtt{p}\&\{\mathtt{m}_i(S_i').T_i'\}_{i\in I\cup J}}\ [\text{Sub-}\&]$$

$$\frac{}{\mathbf{end} \leqslant \mathbf{end}}\ [\text{Sub-}\mathbf{end}] \qquad \frac{\forall i \in I \quad S_i' \leqslant S_i \quad T_i \leqslant T_i'}{\mathtt{p}\oplus\{\mathtt{m}_i(S_i).T_i\}_{i\in I\cup J} \leqslant \mathtt{p}\oplus\{\mathtt{m}_i(S_i').T_i'\}_{i\in I}}\ [\text{Sub-}\oplus]$$

$$\frac{T[\mu\mathbf{t}.T/\mathbf{t}] \leqslant T'}{\mu\mathbf{t}.T \leqslant T'}\ [\text{Sub-}\mu\text{L}] \qquad \frac{T \leqslant T'[\mu\mathbf{t}.T'/\mathbf{t}]}{T \leqslant \mu\mathbf{t}.T'}\ [\text{Sub-}\mu\text{R}]$$

Rule [Sub-B] extends $\leqslant$ to basic types. The remaining rules establish that a subtype describes a more permissive session protocol w.r.t. its supertype. Rule [Sub-$\oplus$] enables the subtype of an internal choice to encompass a broader range of message labels, enabling the sending of more generic payloads. Conversely, rule [Sub-&] dictates that the subtype of an external choice supports a narrower set of input message labels, and less generic payloads. Rule [Sub-**end**] specifies that the type **end** is its own subtype. Finally, rules [Sub-μL] and [Sub-μR] assert the relationship between recursive types, defined up to their unfolding.

Example 3 (Types of OAuth 2.0, from [18]). Consider a protocol from OAuth 2.0 [15]: the service sends to the client *either* a request to `login`, *or* `cancel`; in the first case, `c` continues by sending `passwd` (carrying a string) to the authorisation server, who in turn sends `auth` to `s` (with a boolean, telling whether the client is authorised), and the session **end**s; in the second case, `c` sends `quit` to `a`, and the session **end**s. This protocol can be represented by the global type G_{auth}:

$$G_{\text{auth}} = \mathtt{s}\to\mathtt{c}\colon \left\{ \begin{array}{l} \mathtt{login}.\mathtt{c}\to\mathtt{a}\colon\mathtt{passwd}(\mathtt{str}).\mathtt{a}\to\mathtt{s}\colon\mathtt{auth}(\mathtt{bool}).\mathbf{end} \\ \mathtt{cancel}.\mathtt{c}\to\mathtt{a}\colon\mathtt{quit}.\mathbf{end} \end{array} \right\}$$

Following the MPST top-down methodology, G_{auth} is projected onto three local types (one for each role `s`, `c`, `a`):

$$T_{\mathtt{s}} = \mathtt{c}\oplus\left\{ \begin{array}{l} \mathtt{login}.\mathtt{a}\&\mathtt{auth}(\mathtt{bool}) \\ \mathtt{cancel} \end{array} \right\} \qquad T_{\mathtt{c}} = \mathtt{s}\&\left\{ \begin{array}{l} \mathtt{login}.\mathtt{a}\oplus\mathtt{passwd}(\mathtt{str}) \\ \mathtt{cancel}.\mathtt{a}\oplus\mathtt{quit} \end{array} \right\}$$

$$T_{\mathtt{a}} = \mathtt{c}\&\left\{ \begin{array}{l} \mathtt{passwd}(\mathtt{str}).\mathtt{s}\oplus\mathtt{auth}(\mathtt{bool}) \\ \mathtt{quit} \end{array} \right\}$$

Here, $T_{\mathtt{s}}$ represents the interface of `s` in G_{auth}: it must send ($\oplus$) to `c` either `login` or `cancel`; in the first case, `s` must then receive (&) message `auth`(bool) from `a`, and the session ends; otherwise, in the second case, the session just ends. Types $T_{\mathtt{c}}$ and $T_{\mathtt{a}}$ follow the same intuition.

Example 4 (Merge and Projection, originated from [14]). Two external choice (branching) types from the same role with disjoint labels can be merged into a type carrying both labels, e.g.

$$\mathtt{A}\&\mathtt{greet}(\mathtt{str}) \sqcap \mathtt{A}\&\mathtt{farewell}(\mathtt{bool}) = \mathtt{A}\&\{\mathtt{greet}(\mathtt{str}), \mathtt{farewell}(\mathtt{bool})\}$$

However, this does not apply to internal choices (selections), e.g. $\mathtt{A}\oplus\mathtt{greet}(\mathtt{str}) \sqcap \mathtt{A}\oplus\mathtt{farewell}(\mathtt{bool})$ is undefined.

Two external choices from the same role with same labels but different payloads cannot be merged, e.g. $\mathtt{A}\&\mathtt{greet}(\mathrm{str}) \sqcap \mathtt{A}\&\mathtt{greet}(\mathrm{bool})$ is undefined. This also applies to internal choices.

Two external choices from different roles cannot be merged. Same for internal choices, e.g. $\mathtt{A}\oplus\mathtt{greet}(\mathrm{str}) \sqcap \mathtt{B}\oplus\mathtt{greet}(\mathrm{str})$ is undefined.

Additionally, two local types with same prefixes but unmergable continuations cannot be merged, e.g. $\mathtt{A}\oplus\mathtt{greet}(\mathrm{str}) \sqcap \mathtt{A}\oplus\mathtt{greet}(\mathrm{str}).\mathtt{B}\&\mathtt{farewell}(\mathrm{bool})$ is undefined as $\mathbf{end} \sqcap \mathtt{B}\&\mathtt{farewell}(\mathrm{bool})$ is not mergeable.

Consider a global type:

$$G = \mathtt{A}\to\mathtt{B}\colon \left\{\begin{array}{l}\mathtt{greet}(\mathrm{str}).\mathtt{C}\to\mathtt{A}\colon\mathtt{farewell}(\mathrm{bool})\\ \mathtt{farewell}(\mathrm{bool}).\mathtt{C}\to\mathtt{A}\colon\mathtt{greet}(\mathrm{str})\end{array}\right\}$$

G cannot be projected onto role $\mathtt{C}$ since:

$$\begin{aligned} G\upharpoonright\mathtt{C} &= \mathtt{C}\to\mathtt{A}\colon\mathtt{farewell}(\mathrm{bool})\upharpoonright\mathtt{C} \sqcap \mathtt{C}\to\mathtt{A}\colon\mathtt{greet}(\mathrm{str})\upharpoonright\mathtt{C}\\ &= \mathtt{A}\oplus\mathtt{farewell}(\mathrm{bool}) \sqcap \mathtt{A}\oplus\mathtt{greet}(\mathrm{str}) \quad (\textbf{undefined})\end{aligned}$$

$G\upharpoonright\mathtt{C}$ is undefined because in G, depending on whether $\mathtt{A}$ and $\mathtt{B}$ transmit $\mathtt{greet}$ or $\mathtt{farewell}$, $\mathtt{C}$ is expected to send either $\mathtt{farewell}$ or $\mathtt{greet}$ to $\mathtt{A}$. However, since $\mathtt{C}$ is not involved in the interactions between $\mathtt{A}$ and $\mathtt{B}$, $\mathtt{C}$ is not aware of which message to send; that is G does not provide a valid specification for $\mathtt{C}$.

3.2 Semantics of Global Types

We now present a Labelled Transition System (LTS) semantics for global types. To begin, we introduce the transition labels in Definition 5, which are also used in the LTS semantics of typing contexts (discussed later in Sect. 3.3).

Definition 5 (Transition Labels). *Let α be a transition label of the form:*

$$\begin{array}{rl}\alpha ::= & s[\mathtt{p}]{:}\mathtt{q}\&\mathtt{m}(S) \ \textit{(in session } s\textit{, } \mathtt{p} \textit{ receives } \mathtt{m}(S) \textit{ from } \mathtt{q}\textit{)}\\ \mid & s[\mathtt{p}]{:}\mathtt{q}\oplus\mathtt{m}(S) \ \textit{(in session } s\textit{, } \mathtt{p} \textit{ sends } \mathtt{m}(S) \textit{ to } \mathtt{q}\textit{)}\\ \mid & s[\mathtt{p}][\mathtt{q}]\mathtt{m} \qquad \textit{(in session } s\textit{, message } \mathtt{m} \textit{ is transmitted from } \mathtt{p} \textit{ to } \mathtt{q}\textit{)}\end{array}$$

The subject(s) of a transition label, written $\mathrm{subject}(\alpha)$*, are defined as follows:*

$$\mathrm{subject}(s[\mathtt{p}]{:}\mathtt{q}\&\mathtt{m}(S)) = \mathrm{subject}(s[\mathtt{p}]{:}\mathtt{q}\oplus\mathtt{m}(S)) = \{\mathtt{p}\}$$
$$\mathrm{subject}(s[\mathtt{p}][\mathtt{q}]\mathtt{m}) = \{\mathtt{p},\mathtt{q}\}$$

The label $s[\mathtt{p}][\mathtt{q}]\mathtt{m}$ denotes a synchronising communication between $\mathtt{p}$ and $\mathtt{q}$ via a message label $\mathtt{m}$; the subject of such a label includes *both* roles. The labels $s[\mathtt{p}]{:}\mathtt{q}\oplus\mathtt{m}(S)$ and $s[\mathtt{p}]{:}\mathtt{q}\&\mathtt{m}(S)$ describe sending and receiving actions separately. Since we define a synchronous semantics for global types, these labels are used for the typing context semantics in Sect. 3.3.

We proceed to give a Labelled Transition System (LTS) semantics to a global type G in Definition 6.

$$\frac{G[\mu \mathbf{t}.G/\mathbf{t}] \xrightarrow{\alpha} G'}{\mu \mathbf{t}.G \xrightarrow{\alpha} G'}\ [\text{GR-}\ \mu] \qquad \frac{j \in I}{\mathtt{p}\to\mathtt{q}\colon\{\mathtt{m}_i(S_i).G'_i\}_{i\in I} \xrightarrow{s[\mathtt{p}][\mathtt{q}]\mathtt{m}_j} G'_j}\ [\text{GR-}\ \oplus\&]$$

$$\frac{\forall i \in I : G'_i \xrightarrow{\alpha} G''_i \quad \mathrm{subject}(\alpha) \cap \{\mathtt{p},\mathtt{q}\} = \emptyset}{\mathtt{p}\to\mathtt{q}\colon\{\mathtt{m}_i(S_i).G'_i\}_{i\in I} \xrightarrow{\alpha} \mathtt{p}\to\mathtt{q}\colon\{\mathtt{m}_i(S_i).G''_i\}_{i\in I}}\ [\text{GR-Ctx}]$$

Fig. 5. Global type reduction rules.

Definition 6 (Global Type Reductions). *The global type transition* $\xrightarrow{\alpha}$ *is inductively defined by the rules in Fig. 5. We use* $G \longrightarrow G'$ *if there exists* α *such that* $G \xrightarrow{\alpha} G'$*; we write* $G \longrightarrow$ *if there exists* G' *such that* $G \longrightarrow G'$*, and* $G \not\longrightarrow$ *for its negation (i.e. there is no* G' *such that* $G \longrightarrow G'$*). Finally,* $\longrightarrow^*$ *denotes the transitive and reflexive closure of* $\longrightarrow$*.*

Figure 5 depicts the standard global type reduction rules [20]. Rule [GR-⊕&] models the communication between two roles. Rule [GR-μ] handles recursion. Finally, rule [GR-Ctx] allows a reduction of a global type that is causally independent from its prefix, provided that all of the continuations can make the reduction of that label. This causal independence is indicated by the subjects of the label being disjoint from the prefix of the global type. For example, consider the global type $G = \mathtt{p}\to\mathtt{q}\colon\mathtt{m}_1.\mathtt{r}\to\mathtt{u}\colon\mathtt{m}_2.\mathbf{end}$. Given that the subjects of the label $s[\mathtt{r}][\mathtt{u}]\mathtt{m}_2$ are disjoint from $\mathtt{p}$ and $\mathtt{q}$, i.e. $\mathrm{subject}(s[\mathtt{r}][\mathtt{u}]\mathtt{m}_2) \cap \mathtt{p},\mathtt{q} = \emptyset$, and $\mathtt{r}\to\mathtt{u}\colon\mathtt{m}_2.\mathbf{end} \xrightarrow{s[\mathtt{r}][\mathtt{u}]\mathtt{m}_2} \mathbf{end}$, rule [GR-Ctx] allows G to transition via $s[\mathtt{r}][\mathtt{u}]\mathtt{m}_2$: $G \xrightarrow{s[\mathtt{r}][\mathtt{u}]\mathtt{m}_2} \mathtt{p}\to\mathtt{q}\colon\mathtt{m}_1.\mathbf{end}$, which is unrelated to $\mathtt{p}$ and $\mathtt{q}$.

3.3 Semantics of Typing Context

After introducing the semantics of global types, we now present an LTS semantics for *typing contexts*, which are collections of local types. The formal definition of a typing context is provided in Definition 7, followed by its reduction rules in Definition 8.

Definition 7 (Typing Contexts). Θ *denotes a partial mapping from process variables to n-tuples of types, and* Γ *denotes a partial mapping from channels to types. Their syntax is defined as:*

$$\Theta \ ::= \ \emptyset \mid \Theta, X\colon S_1,\ldots,S_n \qquad \Gamma \ ::= \ \emptyset \mid \Gamma, x\colon S \mid \Gamma, s[\mathtt{p}]\colon T$$

The context composition Γ_1, Γ_2 *is defined iff* $\mathrm{dom}(\Gamma_1) \cap \mathrm{dom}(\Gamma_2) = \emptyset$*. We write* $s \in \Gamma$ *iff* $\exists \mathtt{p} : s[\mathtt{p}] \in \mathrm{dom}(\Gamma)$ *(i.e. session* s *occurs in* Γ*). We write* $s \notin \Gamma$ *iff* $\forall \mathtt{p} : s[\mathtt{p}] \notin \mathrm{dom}(\Gamma)$ *(i.e. session* s *does not occur in* Γ*). We write* $\mathrm{dom}(\Gamma) = \{s\}$ *iff* $\forall c \in \mathrm{dom}(\Gamma)$ *there is* $\mathtt{p}$ *such that* $c = s[\mathtt{p}]$ *(i.e.* Γ *only contains session* s*). We write* $\Gamma \leqslant \Gamma'$ *iff* $\mathrm{dom}(\Gamma) = \mathrm{dom}(\Gamma')$ *and* $\forall c \in \mathrm{dom}(\Gamma)\colon \Gamma(c) \leqslant \Gamma'(c)$*. We*

$$\frac{k \in I}{s[\mathtt{p}]:\mathtt{q}\oplus\{\mathtt{m}_i(S_i).T_i\}_{i\in I} \xrightarrow{s[\mathtt{p}]:\mathtt{q}\oplus\mathtt{m}_k(S_k)} s[\mathtt{p}]:T_k}\ [\Gamma\text{-}\oplus] \qquad \frac{k \in I}{s[\mathtt{p}]:\mathtt{q}\&\{\mathtt{m}_i(S_i).T_i\}_{i\in I} \xrightarrow{s[\mathtt{p}]:\mathtt{q}\&\mathtt{m}_k(S_k)} s[\mathtt{p}]:T_k}\ [\Gamma\text{-}\&]$$

$$\frac{\Gamma_1 \xrightarrow{s[\mathtt{p}]:\mathtt{q}\oplus\mathtt{m}(S)} \Gamma_1' \quad \Gamma_2 \xrightarrow{s[\mathtt{q}]:\mathtt{p}\&\mathtt{m}(S')} \Gamma_2' \quad S \leqslant S'}{\Gamma_1, \Gamma_2 \xrightarrow{s[\mathtt{p}][\mathtt{q}]\mathtt{m}} \Gamma_1', \Gamma_2'}\ [\Gamma\text{-}\oplus\&] \qquad \frac{s[\mathtt{p}]:T\{\mu\mathtt{t}.T/\mathtt{t}\} \xrightarrow{\alpha} \Gamma'}{s[\mathtt{p}]:\mu\mathtt{t}.T \xrightarrow{\alpha} \Gamma'}\ [\Gamma\text{-}\mu]$$

$$\frac{\Gamma \xrightarrow{\alpha} \Gamma'}{\Gamma, x{:}B \xrightarrow{\alpha} \Gamma', x{:}B}\ [\Gamma\text{-},B] \qquad \frac{\Gamma \xrightarrow{\alpha} \Gamma'}{\Gamma, c{:}T \xrightarrow{\alpha} \Gamma', c{:}T}\ [\Gamma\text{-},]$$

Fig. 6. Typing context reduction rules.

write Γ_s *iff* $\mathrm{dom}(\Gamma_s) = \{s\}$, $\mathrm{dom}(\Gamma_s) \subseteq \mathrm{dom}(\Gamma)$, *and* $\forall s[\mathtt{p}] \in \mathrm{dom}(\Gamma) : \Gamma(s[\mathtt{p}]) = \Gamma_s(s[\mathtt{p}])$ *(i.e. restriction of* Γ *to session* s*).*

Definition 8 (Typing Context Reduction). *The* typing context transition $\xrightarrow{\alpha}$ *is inductively defined by the rules in Fig. 6. We write* $\Gamma \xrightarrow{\alpha}$ *if there exists* Γ' *such that* $\Gamma \xrightarrow{\alpha} \Gamma'$*. We define two* reductions $\Gamma \rightarrow_s \Gamma'$ *and* $\Gamma \rightarrow \Gamma'$*, as follows:*

- $\Gamma \rightarrow_s \Gamma'$ *holds iff* $\Gamma \xrightarrow{\alpha} \Gamma'$ *with* $\alpha = s[\mathtt{p}][\mathtt{q}]\mathtt{m}$ *for any* $\mathtt{p}, \mathtt{q} \in \mathcal{R}$ *(recall that* $\mathcal{R}$ *is the set of all roles): this means that* Γ *can progress via message transmission on session* s*, involving any roles* $\mathtt{p}$ *and* $\mathtt{q}$*. We write* $\Gamma \rightarrow_s$ *iff* $\Gamma \rightarrow_s \Gamma'$ *for some* Γ'*, and* $\Gamma \not\rightarrow_s$ *for its negation (i.e. there is no* Γ' *such that* $\Gamma \rightarrow_s \Gamma'$*), and we denote* $\rightarrow_s^*$ *as the reflexive and transitive closure of* $\rightarrow_s$*;*
- $\Gamma \rightarrow \Gamma'$ *holds iff* $\Gamma \rightarrow_s \Gamma'$ *for some* s*: this means that* Γ *can progress via message transmission on any session. We write* $\Gamma \rightarrow$ *iff* $\Gamma \rightarrow \Gamma'$ *for some* Γ'*, and* $\Gamma \not\rightarrow$ *for its negation, and we denote* $\rightarrow^*$ *as the reflexive and transitive closure of* $\rightarrow$*.*

Figure 6 contains standard rules for typing context reductions [18]. Rules $[\Gamma\text{-}\oplus]$ and $[\Gamma\text{-}\&]$ specify that a typing context entry can execute an output and input transition, respectively. Rule $[\Gamma\text{-}\oplus\&]$ ensures synchronised matching input/output transitions, with payload compatibility through subtyping; consequently, the context progresses via a message transmission label $s[\mathtt{p}][\mathtt{q}]\mathtt{m}$. Rule $[\Gamma\text{-}\mu]$ pertains to recursion, while $[\Gamma\text{-},B]$ and $[\Gamma\text{-},]$ address reductions in a larger context.

3.4 Relating Semantics Between Global Types and Typing Contexts

Following the introduction of LTS semantics for global types (Definition 6) and typing contexts (definition 8), we establish a relationship between these two semantics using the projection operator $\upharpoonright$ (Definition 3) and the subtyping relation $\leqslant$ (Definition 4).

Definition 9 (Association of Global Types and Typing Contexts). *A typing context Γ is associated with a global type G for a multiparty session s, written $G \sqsubseteq_s \Gamma$, iff Γ can be split into two disjoint (possibly empty) sub-contexts $\Gamma = \Gamma_G, \Gamma_{\mathbf{end}}$ where:*

1. *Γ_G contains projections of G:* $\mathrm{dom}(\Gamma_G) = \{s[\mathtt{p}] \mid \mathtt{p} \in \mathrm{roles}(G)\}$, *and* $\forall \mathtt{p} \in \mathrm{roles}(G) : G \upharpoonright \mathtt{p} \leqslant \Gamma(s[\mathtt{p}])$;
2. *$\Gamma_{\mathbf{end}}$ contains only end endpoints:* $\forall s[\mathtt{p}] \in \mathrm{dom}(\Gamma_{\mathbf{end}}) : \Gamma(s[\mathtt{p}]) = \mathbf{end}$.

The association $\cdot \sqsubseteq_{\cdot} \cdot$ is a binary relation over global types G and typing contexts Γ, parameterised by multiparty sessions s. There are two requirements for the association: (1) the typing context Γ must include an entry for each role in the multiparty session s; and (2) for each role $\mathtt{p}$, the projection of the global type onto this role ($G \upharpoonright \mathtt{p}$) must be a subtype (Definition 4) of its corresponding entry in the typing context ($\Gamma(s[\mathtt{p}])$).

Example 5 (Association). Recall the global type G_{auth} and its projected local types $T_{\mathtt{s}}, T_{\mathtt{c}}, T_{\mathtt{a}}$ in Example 3. Consider the typing context $\Gamma_{\mathrm{auth}} = \Gamma_{\mathrm{auth}_{\mathtt{s}}}, \Gamma_{\mathrm{auth}_{\mathtt{c}}}, \Gamma_{\mathrm{auth}_{\mathtt{a}}}$, where:

$$\Gamma_{\mathrm{auth}_{\mathtt{s}}} = s[\mathtt{s}]{:}\mathtt{c}{\oplus}\mathtt{cancel} \quad \Gamma_{\mathrm{auth}_{\mathtt{c}}} = s[\mathtt{c}]{:}\mathtt{s}\&\left\{\begin{array}{l}\mathtt{login.a}{\oplus}\mathtt{passwd}(\mathrm{str})\\ \mathtt{cancel.a}{\oplus}\mathtt{quit}\\ \mathtt{fail.a}{\oplus}\mathtt{fatal}\end{array}\right\}$$
$$\Gamma_{\mathrm{auth}_{\mathtt{a}}} = s[\mathtt{a}]{:}\mathtt{c}\&\{\mathtt{passwd}(\mathrm{str}).\mathtt{s}{\oplus}\mathtt{auth}(\mathrm{bool}), \mathtt{quit}, \mathtt{fatal}\}$$

Intuitively, Γ_{auth} is associated with G_{auth}, as $\mathtt{s}$ sends only `cancel`, while $\mathtt{c}$ and $\mathtt{a}$ expect to receive additional messages `fail` and `fatal`, respectively. Indeed, the entries in Γ_{auth} adhere to the communication behaviour patterns of each role of G_{auth}, though with fewer output messages and more input ones.

We can formally verify the association of Γ_{auth} with G_{auth} for session s by:

- $\mathrm{roles}(G_{\mathrm{auth}}) = \{\mathtt{s}, \mathtt{c}, \mathtt{a}\}$, and $\mathrm{dom}(\Gamma_{\mathrm{auth}}) = \{s[\mathtt{s}], s[\mathtt{c}], s[\mathtt{a}]\}$
- $G_{\mathrm{auth}} \upharpoonright \mathtt{s} = T_{\mathtt{s}} = \mathtt{c}{\oplus}\left\{\begin{array}{l}\mathtt{login.a}\&\mathtt{auth}(\mathrm{bool})\\ \mathtt{cancel}\end{array}\right\} \leqslant \Gamma_{\mathrm{auth}}(s[\mathtt{s}])$
- $G_{\mathrm{auth}} \upharpoonright \mathtt{c} = T_{\mathtt{c}} = \mathtt{s}\&\left\{\begin{array}{l}\mathtt{login.a}{\oplus}\mathtt{passwd}(\mathrm{str})\\ \mathtt{cancel.a}{\oplus}\mathtt{quit}\end{array}\right\} \leqslant \Gamma_{\mathrm{auth}}(s[\mathtt{c}])$
- $G_{\mathrm{auth}} \upharpoonright \mathtt{a} = T_{\mathtt{a}} = \mathtt{c}\&\left\{\begin{array}{l}\mathtt{passwd}(\mathrm{str}).\mathtt{s}{\oplus}\mathtt{auth}(\mathrm{bool})\\ \mathtt{quit}\end{array}\right\} \leqslant \Gamma_{\mathrm{auth}}(s[\mathtt{a}])$

We demonstrate the *operational correspondence* between a global type and any *associated* typing context through two main theorems: Theorem 1 shows that the reducibility of a global type aligns with that of its associated typing context; while Theorem 2 illustrates that each possible reduction of a typing context is simulated by an action in the reductions of the associated global type.

Theorem 1 (Soundness of Association). *Given associated global type G and typing context Γ for session s: $G \sqsubseteq_s \Gamma$. If $G \xrightarrow{\alpha} G'$ where $\alpha = s[\mathtt{p}][\mathtt{q}]\mathtt{m}$, then there exist α', Γ', and G'', such that $\alpha' = s[\mathtt{p}][\mathtt{q}]\mathtt{m}'$, $G \xrightarrow{\alpha'} G''$, $G'' \sqsubseteq_s \Gamma'$, and $\Gamma \xrightarrow{\alpha'} \Gamma'$.*

Theorem 2 (Completeness of Association). *Given associated global type G and typing context Γ for session s: $G \sqsubseteq_s \Gamma$. If $\Gamma \xrightarrow{\alpha} \Gamma'$ where $\alpha = s[\mathtt{p}][\mathtt{q}]\mathtt{m}$, then there exists G' such that $G' \sqsubseteq_s \Gamma'$ and $G \xrightarrow{\alpha} G'$.*

Based on Theorem 1 and Theorem 2, we derive a corollary: a global type G is in operational correspondence with the typing context $\Gamma = \{s[\mathtt{p}]{:}G\upharpoonright \mathtt{p}\}_{\mathtt{p}\in \mathrm{roles}(G)}$, containing the projections of all roles in G.

Example 6 (Soundness and Completeness of Association). Consider the associated global type G_{auth} and typing context Γ_{auth} in Example 5.

We have $G_{\mathrm{auth}} \xrightarrow{s[\mathtt{s}][\mathtt{c}]\mathtt{login}}$, and by the soundness of association (Theorem 1), there exist $\alpha = s[\mathtt{s}][\mathtt{c}]\mathtt{cancel}$, G'_{auth}, and Γ'_{auth} such that

$$G_{\mathrm{auth}} \xrightarrow{s[\mathtt{s}][\mathtt{c}]\mathtt{cancel}} G'_{\mathrm{auth}} = \mathtt{c}{\to}\mathtt{a}{:}\mathtt{quit},\ \Gamma_{\mathrm{auth}} \xrightarrow{s[\mathtt{s}][\mathtt{c}]\mathtt{cancel}} \Gamma'_{\mathrm{auth}} = \Gamma'_{\mathrm{auth}_\mathtt{s}}, \Gamma'_{\mathrm{auth}_\mathtt{c}}, \Gamma'_{\mathrm{auth}_\mathtt{a}}$$

where:

$$\Gamma'_{\mathrm{auth}_\mathtt{s}} = s[\mathtt{s}]{:}\mathbf{end} \qquad \Gamma'_{\mathrm{auth}_\mathtt{c}} = s[\mathtt{c}]{:}\mathtt{a}{\oplus}\mathtt{quit}$$
$$\Gamma'_{\mathrm{auth}_\mathtt{a}} = s[\mathtt{a}]{:}\mathtt{c}\&\{\mathtt{passwd}(\mathrm{str}).\mathtt{s}{\oplus}\mathtt{auth}(\mathrm{bool}), \mathtt{quit}, \mathtt{fatal}\}$$

By Definition 9, it is easy to check $G'_{\mathrm{auth}} \sqsubseteq_s \Gamma'_{\mathrm{auth}}$. Further, we have the reduction for Γ'_{auth}:

$$\Gamma'_{\mathrm{auth}} \xrightarrow{s[\mathtt{c}][\mathtt{a}]\mathtt{quit}} \Gamma''_{\mathrm{auth}} = s[\mathtt{s}]{:}\mathbf{end}, s[\mathtt{c}]{:}\mathbf{end}, s[\mathtt{a}]{:}\mathbf{end}$$

which, by the completeness of association (Theorem 2), relates to the reduction: $G'_{\mathrm{auth}} \xrightarrow{s[\mathtt{c}][\mathtt{a}]\mathtt{quit}} \mathbf{end}$, with $\mathbf{end} \sqsubseteq_s \Gamma''_{\mathrm{auth}}$.

Remark 1 ('Weakness' and 'Adequacy' of Soundness Theorem). Inquisitive readers might question why we opted to formulate a '*weak*' soundness theorem rather than one that mirrors the completeness theorem, as seen in existing literature [5]. This choice arises from our utilisation of the subtyping (Definition 4, notably [Sub-⊕]). A local type in the typing context might offer fewer branches for selection compared to the projected local type, leading to transmission actions in the global type that remain uninhabited.

For example, consider the global type G_{auth} and its associated typing context Γ_{auth}, as shown in Example 5. While the global type G_{auth} might transition through $s[\mathtt{s}][\mathtt{c}]\mathtt{login}$, the associated typing context Γ_{auth} cannot.

Nonetheless, our soundness theorem *adequately* ensures the desired properties guaranteed via association, including safety, deadlock-freedom, and liveness, as demonstrated in Sect. 3.5.

3.5 Typing Context Properties Guaranteed via Association

The design of multiparty session type theory provides a substantial advantage in guaranteeing desirable properties through its methodology. Consequently, processes adhering to the local types obtained from projections are inherently *correct by construction*. This subsection highlights three key properties: *communication*

safety, *deadlock-freedom*, and *liveness*. We demonstrate that these properties are ensured by typing contexts associated with global types.

Communication Safety. We begin by introducing communication safety for typing contexts, a behavioural property that ensures each role can exchange compatible messages, thereby preventing label mismatches.

Definition 10 (Typing Context Safety). *Given a session s, we say that φ is an s-*safety property *of typing contexts iff, whenever $\varphi(\Gamma)$, we have:*

$$\begin{array}{rl} [\text{S-}\oplus\&] & \Gamma \xrightarrow{s[\mathtt{p}]:\mathtt{q}\oplus\mathtt{m}(S)} \text{ and } \Gamma \xrightarrow{s[\mathtt{q}]:\mathtt{p}\&\mathtt{m}'(S')} \text{ implies } \Gamma \xrightarrow{s[\mathtt{p}][\mathtt{q}]\mathtt{m}}; \\ [\text{S-}\mu] & \Gamma = \Gamma',\, s[\mathtt{p}]:\mu\mathbf{t}.T \text{ implies } \varphi(\Gamma',\, s[\mathtt{p}]:T\{\mu\mathbf{t}.T/\mathbf{t}\}); \\ [\text{S-}\to_s] & \Gamma \to_s \Gamma' \text{ implies } \varphi(\Gamma'). \end{array}$$

We say Γ *is s-safe*, written $\text{safe}(s, \Gamma)$, if $\varphi(\Gamma)$ holds for some s-safety property φ. We say Γ *is safe*, written $\text{safe}(\Gamma)$, if $\varphi(\Gamma)$ holds for some property φ which is s-safe for all sessions s occurring in $\text{dom}(\Gamma)$.

Our safety property is derived from a fundamental feature of generalised MPST systems [18, Def. 4.1]. As per Definition 10, safety is a *coinductive* property [17]. This means that for a given a session s, s-safe is the largest s-safety property, including the union of all s-safety properties. To demonstrate the s-safety of a typing context Γ, we must identify a property φ such that $\Gamma \in \varphi$, and subsequently prove that φ qualifies as an s-safety property. More specifically, if such a φ exists, it can be formulated as a set containing Γ and all its reductums (via transition $\to_s^*$). We then verify whether all elements of φ satisfy each clause of Definition 10.

According to clause [S-$\oplus\&$], whenever two roles $\mathtt{p}$ and $\mathtt{q}$ attempt communication, the receiving role $\mathtt{q}$ is required to accommodate all output messages of the sending role $\mathtt{p}$ with compatible payload types, ensuring the feasibility of the communication. Clause [S-μ] unfolds any recursive entry, while clause [S-$\to_s$] specifies that any typing context Γ', to which Γ transitions on session s, must also belong to φ, indicating that Γ' is s-safe as well. Note that any entry $s[\mathtt{p}]:\mathbf{end}$ in Γ vacuously satisfies all clauses.

Example 7 (Typing Context Safety). Consider the typing context Γ_{auth} from Example 5. We know that Γ_{auth} is s-safe by verifying its reductions. For example, we have the reductions $\Gamma_{\text{auth}} \xrightarrow{s[\mathtt{s}][\mathtt{c}]\mathtt{cancel}} \cdot \xrightarrow{s[\mathtt{c}][\mathtt{a}]\mathtt{quit}} \Gamma''_{\text{auth}} = s[\mathtt{s}]:\mathbf{end}, s[\mathtt{c}]:\mathbf{end}, s[\mathtt{a}]:\mathbf{end}$, where each reductum complies with all clauses of Definition 10.

The typing context $\Gamma_A = s[\mathtt{p}]:\mathtt{q}\oplus\mathtt{m}_1.\mathtt{r}\oplus\mathtt{m}_3, s[\mathtt{q}]:\mathtt{p}\&\mathtt{m}_2, s[\mathtt{r}]:\mathtt{p}\&\mathtt{m}_4$ is *not* s-safe. Any property φ containing such a context is *not* a s-safety property, as it violates [S-$\oplus\&$] of Definition 10: $\Gamma_A \xrightarrow{s[\mathtt{p}]:\mathtt{q}\oplus\mathtt{m}_1}$ and $\Gamma_A \xrightarrow{s[\mathtt{q}]:\mathtt{p}\&\mathtt{m}_2}$, but $\Gamma_A \xrightarrow{s[\mathtt{p}][\mathtt{q}]\mathtt{m}_1}$ does not hold.

Finally, let's consider the typing context $\Gamma_B = s[\mathtt{p}]:\mathtt{q}\oplus\mathtt{m}(\text{real}), s[\mathtt{q}]:\mathtt{p}\&\mathtt{m}(\text{int})$. Γ_B is *not* s-safe, as any property φ containing Γ_B contradicts [S-$\oplus\&$]: $\Gamma_B \xrightarrow{s[\mathtt{p}]:\mathtt{q}\oplus\mathtt{m}(\text{real})}$ and $\Gamma_B \xrightarrow{s[\mathtt{q}]:\mathtt{p}\&\mathtt{m}(\text{int})}$, but $\Gamma_B \xrightarrow{s[\mathtt{p}][\mathtt{q}]\mathtt{m}}$ is not uphold due to $\text{real} \not\leqslant \text{int}$.

Deadlock-Freedom. The property of deadlock-freedom determines whether a typing context can keep reducing without getting stuck, unless it reaches a successful terminal state.

Definition 11 (Deadlock-Free Typing Contexts). *Given a session s, a typing context Γ is* s-deadlock-free, *written* $\mathrm{df}(s,\Gamma)$, *iff* $\Gamma \to_s^* \Gamma' \not\to_s$ *implies* $\forall s[\mathtt{p}] \in \mathrm{dom}(\Gamma') : \Gamma'(s[\mathtt{p}]) = \mathbf{end}$.

It is noteworthy that a typing context that reduces infinitely adheres to deadlock-freedom, as it consistently undergoes further reductions. Alternatively, when a terminal typing context is reached, all entries in the typing context must be terminated successfully (**end**). Consequently, a deadlock-free typing context either continues perpetual reduction or terminates.

Example 8 (Typing Context Deadlock-Freedom). The typing context Γ_{auth} in Example 5 is s-deadlock-free, as any terminal typing context Γ'_{auth}, reached from Γ_{auth} via $\Gamma_{\mathrm{auth}} \to_s^* \Gamma'_{\mathrm{auth}} \not\to_s$, only contains **end** entries, which can be easily verified.

The typing context $\Gamma_C = s[\mathtt{p}]{:}\mathtt{q}\oplus\mathtt{m}_1.\mathtt{r}\&\mathtt{m}_3, s[\mathtt{q}]{:}\mathtt{r}\oplus\mathtt{m}_2.\mathtt{p}\&\mathtt{m}_1, s[\mathtt{r}]{:}\mathtt{p}\oplus\mathtt{m}_3.\mathtt{q}\&\mathtt{m}_2$ is s-safe but *not* s-deadlock-free, as its inputs and outputs, despite being dual, are arranged in the incorrect order. Specifically, there are no possible reductions for Γ_C, i.e. $\Gamma_C \not\to_s$, while none of the entries in Γ_C is a termination.

Finally, the context $\Gamma_D = s[\mathtt{p}]{:}\mathtt{q}\oplus\{\mathtt{m}_1.\mathtt{r}\oplus\mathtt{m}_2, \mathtt{m}_3\}, s[\mathtt{q}]{:}\mathtt{p}\&\{\mathtt{m}_1, \mathtt{m}_2\}, s[\mathtt{r}]{:}\mathtt{p}\&\mathtt{m}_2$ is s-deadlock-free but *not* s-safe, as it consistently reaches a successful termination by transmitting $\mathtt{m}_1$ between $\mathtt{p}$ and $\mathtt{q}$; while $\mathtt{q}$ is unable to receive the message $\mathtt{m}_3$ when $\mathtt{p}$ intends to send it due to the message mismatch.

Liveness. The liveness property ensures that every pending output/external choice eventually gets triggered through a message transmission. It relies on fairness, specifically based on *strong fairness of components* [7, Fact 2], to guarantee that every enabled message transmission is successfully executed. The definitions of fair and live paths for typing contexts are provided in Definition 12, and these paths are used to formalise the liveness for typing contexts in Definition 13.

Definition 12 (Fair, Live Paths). *A* path *is a possibly infinite sequence of typing contexts $(\Gamma_n)_{n\in N}$, where $N = \{0,1,2,\ldots\}$ is a set of consecutive natural numbers, and, $\forall n \in N$, $\Gamma_n \to \Gamma_{n+1}$. We say that a path $(\Gamma_n)_{n\in N}$ is* fair for session *s iff, $\forall n \in N$: $\Gamma_n \xrightarrow{s[\mathtt{p}][\mathtt{q}]\mathtt{m}}$ implies $\exists k, \mathtt{m}'$ such that $N \ni k \geq n$, and $\Gamma_k \xrightarrow{s[\mathtt{p}][\mathtt{q}]\mathtt{m}'} \Gamma_{k+1}$.*

We say that a path $(\Gamma_n)_{n\in N}$ is live for session *s iff, $\forall n \in N$:*

(L1) $\Gamma_n \xrightarrow{s[\mathtt{p}]:\mathtt{q}\oplus\mathtt{m}(S)}$ implies $\exists k, \mathtt{m}'$ such that $N \ni k \geq n$ and $\Gamma_k \xrightarrow{s[\mathtt{p}][\mathtt{q}]\mathtt{m}'} \Gamma_{k+1}$;
(L2) $\Gamma_n \xrightarrow{s[\mathtt{q}]:\mathtt{p}\&\mathtt{m}(S)}$ implies $\exists k, \mathtt{m}'$ such that $N \ni k \geq n$ and $\Gamma_k \xrightarrow{s[\mathtt{p}][\mathtt{q}]\mathtt{m}'} \Gamma_{k+1}$.

A path is a (possibly infinite) sequence of reductions of a typing context. A path is fair for session s if, along the path, every enabled message transmission is eventually performed on s. Likewise, a path is live for session s if, along the path, every pending internal/external choice is eventually triggered on s.

Definition 13 (Live Typing Contexts). *Given a session s, a typing context Γ is s-*live*, written* $\text{live}(s, \Gamma)$*, iff $\Gamma \to_s^* \Gamma'$ implies all paths starting with Γ' that are fair for session s are also live for s.*

A typing context Γ is s-live if any reductum of Γ (via transition $\to_s^*$) consistently generates a live path for s under fairness.

Example 9 (Fairness and Typing Context Liveness, originated from [6,18]). Consider the typing context:

$$\Gamma_E = s[\mathtt{p}]{:}\mathtt{p}' \oplus \mathtt{m}_1, s[\mathtt{p}']{:}\mathtt{p} \& \mathtt{m}_1, s[\mathtt{q}]{:}\mu \mathbf{t}_\mathtt{q}.\mathtt{q}' \oplus \mathtt{m}_2.\mathbf{t}_\mathtt{q}, s[\mathtt{q}']{:}\mu \mathbf{t}_{\mathtt{q}'}.\mathtt{q} \& \mathtt{m}_2.\mathbf{t}_{\mathtt{q}'}$$

which is s-safe and s-deadlock-free. There is an infinite path starting with Γ_E, where $\mathtt{q}$ and $\mathtt{q}'$ keep communicating in session s, while $\mathtt{p}$ and $\mathtt{p}'$ never trigger a reduction to interact, i.e. $\Gamma_E \xrightarrow{s[\mathtt{q}][\mathtt{q}']\mathtt{m}_2} \Gamma_E \xrightarrow{s[\mathtt{q}][\mathtt{q}']\mathtt{m}_2} \cdots$. Such a path is *not* fair for s because although message transmission between $\mathtt{p}$ and $\mathtt{p}'$ is enabled, it will never occur. Alternatively, along any fair path of Γ_E, all inputs and outputs can eventually be fired on s, indicating that Γ_E is s-live.

Now consider another typing context:

$$\Gamma'_E = s[\mathtt{p}]{:}\mu \mathbf{t}_\mathtt{p}.\mathtt{q} \& \{\mathtt{m}_1.\mathbf{t}_\mathtt{p}, \mathtt{m}_2.\mathbf{t}_\mathtt{p}\}, s[\mathtt{q}]{:}\mu \mathbf{t}_\mathtt{q}.\mathtt{p} \oplus \{\mathtt{m}_1.\mathbf{t}_\mathtt{q}, \mathtt{m}_2.\mathtt{r} \oplus \mathtt{m}_2.\mathbf{t}_\mathtt{q}\}, s[\mathtt{r}]{:}\mu \mathbf{t}_\mathtt{r}.\mathtt{q} \& \mathtt{m}_2.\mathbf{t}_\mathtt{r}$$

which is s-safe and s-deadlock-free. Γ'_E has fair and live paths, where in s, $\mathtt{m}_2$ is transmitted from $\mathtt{q}$ to $\mathtt{p}$, and then to $\mathtt{r}$. However, there is also a fair path, where in s, $\mathtt{m}_1$ is consistently transmitted from $\mathtt{q}$ to $\mathtt{p}$, i.e. $\Gamma'_E \xrightarrow{s[\mathtt{q}][\mathtt{p}]\mathtt{m}_1} \Gamma'_E \xrightarrow{s[\mathtt{q}][\mathtt{p}]\mathtt{m}_1} \cdots$. In this case, $\mathtt{r}$ indefinitely awaits input that will never arrive. Therefore, while this path is fair for s, it is not live, and hence, Γ'_E is not s-live.

Finally, the typing context $\Gamma''_E = s[\mathtt{p}]{:}\mathtt{q} \oplus \{\mathtt{m}_1, \mathtt{m}_2\}, s[\mathtt{q}]{:}\mathtt{p} \& \{\mathtt{m}_1, \mathtt{m}_3\}$ is s-live: there is only one path starting from Γ''_E, i.e. $\Gamma''_E \xrightarrow{s[\mathtt{p}][\mathtt{q}]\mathtt{m}_1} s[\mathtt{p}]{:}\mathbf{end}, s[\mathtt{q}]{:}\mathbf{end}$, which is fair and live for s; while it is not s-safe.

Properties by Association. We conclude by demonstrating, as stated in Thm. 3, that a typing context associated with a global type is constructed to possess the properties of safety, deadlock-freedom, and liveness.

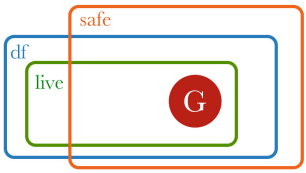
Theorem 3 (Safety, Deadlock-Freedom, and Liveness by Association). *Let G be a global type, Γ a typing context, and s a session. If Γ is associated with G for s: $G \sqsubseteq_s \Gamma$, then Γ is s-safe, s-deadlock-free, and s-live.*

Example 10 (Typing Context Properties Guaranteed by Association). The typing context Γ_{auth} described in Example 5 is associated with G_{auth} for session s, i.e. $G_{\text{auth}} \sqsubseteq_s \Gamma_{\text{auth}}$. Consequently, Γ_{auth} possesses the properties of being s-safe, s-deadlock-free (as also demonstrated in Example 7 and Example 8, respectively), and s-live.

3.6 Relationships Between Typing Context Properties

We now explore both the relationships between typing context properties, and how they relate to association, as formalised in Theorem 4 below.

Theorem 4 *For any typing context Γ and session s, the following statements are valid:*
(1) $\mathrm{df}(s,\Gamma) \not\Longleftarrow \not\Longrightarrow \mathrm{safe}(s,\Gamma)$;
(2) $\mathrm{live}(s,\Gamma) \not\Longleftarrow \not\Longrightarrow \mathrm{safe}(s,\Gamma)$;
(3) $\mathrm{live}(s,\Gamma) \not\Longleftarrow \Longrightarrow \mathrm{df}(s,\Gamma)$;
(4) $\exists G : G \sqsubseteq_s \Gamma \not\Longleftarrow \Longrightarrow \mathrm{safe}(s,\Gamma)$;
(5) $\exists G : G \sqsubseteq_s \Gamma \not\Longleftarrow \Longrightarrow \mathrm{df}(s,\Gamma)$;
(6) $\exists G : G \sqsubseteq_s \Gamma \not\Longleftarrow \Longrightarrow \mathrm{live}(s,\Gamma)$.



In the diagram, the "safe" set (resp. "df" set, "live" set) contains all typing contexts that are s-safe (resp. s-deadlock-free, s-live). The red set $\mathbb{G}$ encompasses all typing contexts associated with some global type for s. The negated implications stated in Theorem 4 are showcased in Example 8, Example 9, and Example 11 (below), respectively.

Example 11 (Non-Associated Typing Context, originated from [2,18]*).* The typing context $\Gamma_F = s[\mathtt{p}]{:}T, s[\mathtt{q}]{:}\oplus\mathtt{m}(T).\mathbf{end}$ with $T = \mu\mathbf{t}.\mathtt{q}\oplus\mathtt{m}(\mathbf{t}).\mathbf{end}$ (from [2, Ex. 1.2]) is s-live (and hence s-deadlock-free and s-safe), but *not* associated with any global type for session s: this is because a recursion variable $\mathbf{t}$ occurs as payload in T, which is not allowed by either Definition 3 or Definition 4.

4 Multiparty Session Typing System

This section presents a type system for the multiparty session π-calculus, as explained in Sect. 2. We introduce the typing rules in Sect. 4.1, and show the main properties of typed processes: subject reduction and session fidelity, in Sect. 4.2. Finally, we demonstrate how process properties such as deadlock-freedom and liveness can be guaranteed by construction in Sect. 4.3.

4.1 Typing Rules

Two kinds of typing contexts, as introduced in Definition 7, are used in our type system: Θ, responsible for assigning an n-tuple of types to each process variable X (one type per argument), and Γ, which maps variables to payload types (basic types or session types), as well as channels with roles (endpoints) to session types. These typing contexts are jointly applied in judgments formulated as:

$$\Theta \cdot \Gamma \vdash P \quad \text{(with } \Theta \text{ omitted when empty)}$$

which interprets as: "considering the process types in Θ, P uses its variables and channels *linearly* based on Γ." This *typing judgement* is defined by the rules in Fig. 7, where, for convenience, we include type annotations for channels bound by process definitions and restrictions.

The main innovation in Fig. 7 lies in rule [T-G-ν], which uses a typing context associated with a global type in some session s (Definition 9) to enforce session

$$\frac{\Theta(X) = S_1, \ldots, S_n}{\Theta \vdash X{:}S_1, \ldots, S_n}\ [\text{T-}\ X] \qquad \frac{\forall i \in 1 \ldots n \quad S_i \text{ is basic or } c_i{:}S_i \vdash c_i{:}\mathbf{end}}{\mathrm{end}(c_1{:}S_1, \ldots, c_n{:}S_n)}\ [\text{T-}\ \mathrm{end}]$$

$$\frac{\mathrm{end}(\Gamma)}{\Theta \cdot \Gamma \vdash \mathbf{0}}\ [\text{T-}\ \mathbf{0}] \qquad \frac{v \in \mathcal{B}}{\emptyset \vdash v{:}B}\ [\text{T-}\ B] \qquad \frac{\Theta \cdot \Gamma_1 \vdash P_1 \qquad \Theta \cdot \Gamma_2 \vdash P_2}{\Theta \cdot \Gamma_1, \Gamma_2 \vdash P_1 \mid P_2}\ [\text{T-}\ |]$$

$$\frac{\Theta \vdash X{:}S_1, \ldots, S_n \quad \mathrm{end}(\Gamma_0) \quad \forall i \in 1 \ldots n \quad \Gamma_i \vdash d_i{:}S_i \quad S_i \not\leqslant \mathbf{end}}{\Theta \cdot \Gamma_0, \Gamma_1, \ldots, \Gamma_n \vdash X\langle d_1, \ldots, d_n\rangle}\ [\text{T-Call}]$$

$$\frac{\Theta, X{:}S_1, \ldots, S_n \cdot x_1{:}S_1, \ldots, x_n{:}S_n \vdash P \qquad \Theta, X{:}S_1, \ldots, S_n \cdot \Gamma \vdash Q}{\Theta \cdot \Gamma \vdash \mathbf{def}\ X(x_1{:}S_1, \ldots, x_n{:}S_n) = P\ \mathbf{in}\ Q}\ [\text{T-}\ \mathbf{def}]$$

$$\frac{\Gamma_1 \vdash c{:}\mathtt{q}\&\{\mathtt{m}_i(S_i).T_i\}_{i\in I} \quad \forall i \in I \quad \Theta \cdot \Gamma, y_i{:}S_i, c{:}T_i \vdash P_i}{\Theta \cdot \Gamma, \Gamma_1 \vdash c[\mathtt{q}]\&\{\mathtt{m}_i(y_i).P_i\}_{i\in I}}\ [\text{T-}\ \&]$$

$$\frac{\Gamma_1 \vdash c{:}\mathtt{q}\oplus\{\mathtt{m}(S).T\} \quad \Gamma_2 \vdash d{:}S \quad S \not\leqslant \mathbf{end} \quad \Theta \cdot \Gamma, c{:}T \vdash P}{\Theta \cdot \Gamma, \Gamma_1, \Gamma_2 \vdash c[\mathtt{q}]\oplus\mathtt{m}\langle d\rangle.P}\ [\text{T-}\ \oplus]$$

$$\frac{S \leqslant S'}{c{:}S \vdash c{:}S'}\ [\text{T-Sub}] \qquad \frac{G \sqsubseteq_s \Gamma' \quad s \notin \Gamma \quad \Theta \cdot \Gamma, \Gamma' \vdash P}{\Theta \cdot \Gamma \vdash (\nu s{:}\Gamma')\,P}\ [\text{T-}\ G\text{-}\nu]$$

Fig. 7. Typing rules for processes.

restrictions. The rest of rules are mostly standard [18]. Rule [T-X] retrieves process variables, while rule [T-B] types a value v if it belongs to a set of basic types $\mathcal{B}$. Rule [T-Sub] applies subtyping within a singleton typing context $c{:}S$ when assigning type S' to a variable or channel c. Additionally, rule [T-end] introduces a predicate $\mathrm{end}(\cdot)$ for typing contexts, denoting the termination of all endpoints. This predicate is used in [T-$\mathbf{0}$] to type an inactive process $\mathbf{0}$. Rules [T-$\oplus$] and [T-&] assign selection and branching types to channels used by selection and branching processes, respectively. Rules [T-**def**] and [T-Call] deal with recursive processes declarations and calls, respectively. Notably, the clauses "$S \not\leqslant \mathbf{end}$" used in rules [T-$\oplus$] and [T-Call] prevent the sending or passing of channels typed as **end**, while permitting the sending/passing of channels and data of any other type. Finally, rule [T-|] *linearly* divides the typing context into two parts, each used to type one sub-process.

Example 12 (Typed Process). Consider the processes P and Q from Example 1. The type context $\Gamma_H = s[\mathtt{p}]{:}T_{H_\mathtt{p}}, s[\mathtt{q}]{:}T_{H_\mathtt{q}}, s[\mathtt{r}]{:}T_{H_\mathtt{r}}$, with $T_{H_\mathtt{p}} = \mathtt{q}\oplus\mathtt{m}'(T_{H_\mathtt{r}}).\ \mathtt{r}\&\mathtt{m}(\mathrm{int}).\mathbf{end}, T_{H_\mathtt{q}} = \mathtt{p}\&\mathtt{m}'(T_{H_\mathtt{r}}).\mathbf{end}, T_{H_\mathtt{r}} = \mathtt{p}\oplus\mathtt{m}(\mathrm{int}).\mathbf{end}$, can type the process $P|Q$ by the following derivation:

$$\dfrac{\dfrac{\Omega \qquad \dfrac{s[\mathtt{p}]{:}\mathtt{r}\&\mathtt{m}(\mathrm{int}).\mathbf{end} \vdash s[\mathtt{p}]{:}\mathtt{r}\&\mathtt{m}(\mathrm{int}).\mathbf{end} \qquad \dfrac{\dfrac{x{:}\mathrm{int}, s[\mathtt{p}]{:}\mathbf{end}}{\mathrm{end}(x{:}\mathrm{int}, s[\mathtt{p}]{:}\mathbf{end})}\ [\text{T-end}]}{x{:}\mathrm{int}, s[\mathtt{p}]{:}\mathbf{end} \vdash \mathbf{0}}\ [\text{T-}\mathbf{0}]}{s[\mathtt{p}]{:}\mathtt{r}\&\mathtt{m}(\mathrm{int}).\mathbf{end} \vdash s[\mathtt{p}]{:}\mathtt{r}\&\mathtt{m}(\mathrm{int}).\mathbf{end}}\ [\text{T-}\&]}{s[\mathtt{p}]{:}\mathtt{r}\&\mathtt{m}(\mathrm{int}).\mathbf{end} \vdash s[\mathtt{p}][\mathtt{r}]\&\mathtt{m}(x).\mathbf{0}}\ [\text{T-}\oplus] \qquad \vdots}{s[\mathtt{p}]{:}T_{H_\mathtt{p}}, s[\mathtt{r}]{:}T_{H_\mathtt{r}} \vdash P \qquad\qquad s[\mathtt{q}]{:}T_{H_\mathtt{q}} \vdash Q}$$

$$\Gamma_H \vdash P \mid Q\ [\text{T-}|]$$

where Ω includes certain trivial conditions used as prerequisites for [T-⊕]: $s[\mathtt{p}]{:}T_{H_\mathtt{p}} \vdash s[\mathtt{p}]{:}T_{H_\mathtt{p}}$, $s[\mathtt{r}]{:}T_{H_\mathtt{r}} \vdash s[\mathtt{r}]{:}T_{H_\mathtt{r}}$, and $T_{H_\mathtt{r}} \not\leqslant \mathbf{end}$. In the omitted part of the derivation, similar to that for $s[\mathtt{p}]{:}T_{H_\mathtt{p}}, s[\mathtt{r}]{:}T_{H_\mathtt{r}} \vdash P$, the process Q is typed by $s[\mathtt{q}]{:}T_{H_\mathtt{q}}$, using rules [T-&], [T-⊕], [T-**0**], and [T-end]. Moreover, as Γ_H is associated with a global type $G_H = \mathtt{p}{\to}\mathtt{q}{:}\mathtt{m}'(\mathtt{p}{\oplus}\mathtt{m}(\mathrm{int})).\mathtt{r}{\to}\mathtt{p}{:}\mathtt{m}(\mathrm{int}).\mathbf{end}$ for session s, i.e. $G_H \sqsubseteq_s \Gamma_H$, following [T-G-$\nu$], the process $P|Q$ is closed under Γ_H, i.e. $\vdash (\nu s{:}\Gamma_H)\, P \mid Q$.

Take the typing contexts $\Gamma_{\mathrm{auth}_\mathtt{s}}, \Gamma_{\mathrm{auth}_\mathtt{c}}, \Gamma_{\mathrm{auth}_\mathtt{a}}$ from Example 5, along with the processes $P_\mathtt{s}, P_\mathtt{c}, P_\mathtt{a}$ from Example 2. These contexts enable the typing of the respective processes. Consequently, the context Γ_{auth} (from Example 5) can be used to type the process $P_\mathtt{s} \mid P_\mathtt{c} \mid P_\mathtt{a}$. Similar to the above example, given Γ_{auth} is associated with the global type G_{auth} (Example 5), according to [T-G-ν], $P_\mathtt{s} \mid P_\mathtt{c} \mid P_\mathtt{a}$ is closed under Γ_{auth}. It is noteworthy that the typing context $s[\mathtt{s}]{:}T_\mathtt{s}$, using the local type $T_\mathtt{s}$ from Example 3, can type the process $P_\mathtt{s}$, due to $T_\mathtt{s} \leqslant \Gamma_{\mathrm{auth}_\mathtt{s}}(s[\mathtt{s}])$ (as demonstrated in Example 5), and $\Gamma_{\mathrm{auth}_\mathtt{s}} \vdash P_\mathtt{s}$. Similarly, $s[\mathtt{c}]{:}T_\mathtt{c}$ and $s[\mathtt{a}]{:}T_\mathtt{a}$ type $P_\mathtt{c}$ and $P_\mathtt{a}$, respectively.

4.2 Subject Reduction and Session Fidelity

We elucidate the key properties of well-typed processes, including *subject reduction* (Theorem 5), *session fidelity* (Theorem 6).

Subject reduction ensures the preservation of types during process reduction. It states that if a well-typed process P reduces to P', then the reduction is simulated by the typing context Γ used to type P. Note that in our subject reduction theorem, P is *not* required to contain only one single session. Instead, we include all restricted sessions in P, ensuring that reductions on these various sessions maintain their respective restrictions. This is enforced by rule [T-G-ν] in Fig. 7. Additionally, the theorem dictates that, in alignment with the MPST top-down methodology, a process initially constructed from global types retains this construction manner even after reductions.

Theorem 5 (Subject Reduction). *Assume* $\Theta \cdot \Gamma \vdash P$ *where* $\forall s \in \Gamma : \exists G_s : G_s \sqsubseteq_s \Gamma_s$. *If* $P \to P'$, *then* $\exists \Gamma'$ *such that* $\Gamma \to^* \Gamma'$, $\Theta \cdot \Gamma' \vdash P'$, *and* $\forall s \in \Gamma' : \exists G'_s : G'_s \sqsubseteq_s \Gamma'_s$.

Corollary 1 (Type Safety). *If* $\emptyset \cdot \emptyset \vdash P$ *and* $P \to^* P'$, *then* P' *has no error.*

Example 13 (Subject Reduction). Take the typed process $P \mid Q$ and the typing context Γ_H from Example 12. After a reduction using [R-⊕&], $P \mid Q$ transitions to $P \mid Q \to s[\mathtt{p}][\mathtt{r}]\&\mathtt{m}(x).\mathbf{0} \mid s[\mathtt{r}][\mathtt{p}]{\oplus}\mathtt{m}\langle 42\rangle.\mathbf{0} = P' \mid Q'$. The typing context Γ_H can reduce to $\Gamma'_H = s[\mathtt{p}]{:}\mathtt{r}\&\mathtt{m}(\mathrm{int}).\mathbf{end}, s[\mathtt{q}]{:}\mathbf{end}, s[\mathtt{r}]{:}\mathtt{p}{\oplus}\mathtt{m}(\mathrm{int}).\mathbf{end}$, via [$\Gamma$-⊕&], which can be used to type $P' \mid Q'$, and is associated with a global type $G'_H = \mathtt{r}{\to}\mathtt{p}{:}\mathtt{m}(\mathrm{int}).\mathbf{end}$. In fact, by applying Theorem 5, we can infer that all process reductions initiated from $P|Q$ maintain well-typedness. Furthermore, based on Corollary 1, we are ensured that they are error-free. Observe that the typing context Γ_{auth} from Examples. 5 and 12 and the process $P_\mathtt{s}|P_\mathtt{c}|P_\mathtt{a}$ from Examples. 2 and 12 also follow subject reduction, ensuring type safety.

Session fidelity asserts the converse implication concerning subject reduction: if a process P is typed by Γ, and Γ can reduce along session s, then P can replicate at least one of the reductions performed by Γ (although not necessarily all such reductions, as Γ over-approximates the behaviour of P). Consequently, we can deduce P's behaviour from Γ's behaviour, as demonstrated in Theorem 7. However, this outcome does *not* hold universally for all well-typed processes: a well-typed process might loop in a recursion, such as $\mathbf{def}\ X(\ldots){=}X\ \mathbf{in}\ X$, or it could deadlock by intricately interleaving communications across multiple sessions [4]. To address this, similarly to [18] and most session type works, we establish session fidelity specifically for processes featuring guarded recursion and implementing a single multiparty session as a parallel composition of one sub-process per role. The formalisation of session fidelity is provided in Theorem 6 below, building upon the concepts introduced in Definition 14.

Definition 14 (from [18]). *Assume $\emptyset \cdot \Gamma \vdash P$. We say that P:*

(1) has guarded definitions *iff in each process definition in P of the form $\mathbf{def}\ X(x_1{:}S_1, \ldots, x_n{:}S_n) = Q\ \mathbf{in}\ P'$, for all $i \in 1..n$, if S_i is a session type, then a call $Y\langle \ldots, x_i, \ldots \rangle$ can only occur in Q as a subterm of $x_i[\mathtt{q}]\&\{\mathtt{m}_j(y_j).P_j\}_{j\in J}$ or $x_i[\mathtt{q}]\oplus\mathtt{m}\langle d\rangle.P''$ (i.e. after using x_i for input or output);*

(2) only plays role $\mathtt{p}$ *in s, by Γ iff:* (i) P has guarded definitions; (ii) $\mathrm{fv}(P) = \emptyset$; (iii) $\Gamma = \Gamma_0, s[\mathtt{p}]{:}T$ with $T \not\leqslant \mathbf{end}$ and $\mathrm{end}(\Gamma_0)$; (iv) for all subterms $(\nu s' {:} \Gamma')\ P'$ in P, $\mathrm{end}(\Gamma')$.

We say "P only plays role $\mathtt{p}$ *in s" iff $\exists \Gamma : \emptyset \cdot \Gamma \vdash P$, and item (2) holds.*

In Definition 14, item (1) formalises guarded recursion for processes, while item (2) identifies a process that plays exactly *one* role on *one* session. It is evident that an ensemble of such processes cannot deadlock by waiting for each other on multiple sessions.

Example 14 (Playing Only Role). Consider the processes P, Q from Example 1, and the typing context Γ_H from Example 12. Observe that P does not only play either $\mathtt{p}$ or $\mathtt{r}$ in s. This arises from the fact that P can only be typed by a context of the form $s[\mathtt{p}]{:}T_\mathtt{p}, s[\mathtt{r}]{:}T_\mathtt{r}$, where neither $\mathrm{end}(s[\mathtt{p}]{:}T_\mathtt{p})$ nor $\mathrm{end}(s[\mathtt{r}]{:}T_\mathtt{r})$ holds, thus violating item (2) of Definition 14. Conversely, Q only plays role $\mathtt{q}$ in s, by $s[\mathtt{q}]{:}T_{H_\mathtt{q}}$, as all conditions are fulfilled. Note that the process $P_\mathtt{s}$ (resp. $P_\mathtt{c}$, $P_\mathtt{a}$) from Examples. 2 and 12 only plays role $\mathtt{s}$ (resp. $\mathtt{c}$, $\mathtt{a}$) in s, which can be easily verified.

We now formalise our session fidelity result (Theorem 6). Although the statement appears similar to Thm. 5.4 in [18], it utilises a typing context associated with a global type for a specific session s to type the process. This approach guarantees not only the fulfilment of the single-session requirements for processes (Definition 14), but also asserts that a process constructed from a global type maintains this structure even after reductions.

Theorem 6 (Session Fidelity). *Assume* $\emptyset \cdot \Gamma \vdash P$, *with* $G \sqsubseteq_s \Gamma$, $P \equiv \Pi_{\mathtt{p} \in I} P_{\mathtt{p}}$, *and* $\Gamma = \bigcup_{\mathtt{p} \in I} \Gamma_{\mathtt{p}}$ *such that for each* $P_{\mathtt{p}}$: *(1)* $\emptyset \cdot \Gamma_{\mathtt{p}} \vdash P_{\mathtt{p}}$, *and (2) either* $P_{\mathtt{p}} \equiv \mathbf{0}$, *or* $P_{\mathtt{p}}$ *only plays* $\mathtt{p}$ *in* s, *by* $\Gamma_{\mathtt{p}}$. *Then,* $\Gamma \rightarrow_s$ *implies* $\exists \Gamma', G', P'$ *such that* $\Gamma \rightarrow_s \Gamma'$, $P \rightarrow^* P'$, *and* $\emptyset \cdot \Gamma' \vdash P'$, *with* $G' \sqsubseteq_s \Gamma'$, $P' \equiv \Pi_{\mathtt{p} \in I} P'_{\mathtt{p}}$, *and* $\Gamma' = \bigcup_{\mathtt{p} \in I} \Gamma'_{\mathtt{p}}$ *such that for each* $P'_{\mathtt{p}}$: *(1)* $\emptyset \cdot \Gamma'_{\mathtt{p}} \vdash P'_{\mathtt{p}}$, *and (2) either* $P'_{\mathtt{p}} \equiv \mathbf{0}$, *or* $P'_{\mathtt{p}}$ *only plays* $\mathtt{p}$ *in* s, *by* $\Gamma'_{\mathtt{p}}$.

Example 15 (Guarded Definitions in Session Fidelity, from [18]*).* According to rule [T-**def**] in Fig. 7, an unguarded definition $X(x{:}S) = X\langle x \rangle$ can be typed with *any* S. Therefore, for example, we have:

$$\emptyset \cdot s[\mathtt{p}]{:}\mathtt{q} \oplus \mathtt{m}, s[\mathtt{q}]{:}\mathtt{p} \& \mathtt{m} \vdash \mathbf{def}\ X(x{:}\mathtt{q} \oplus \mathtt{m}) = X\langle x \rangle\ \mathbf{in}\ X\langle s[\mathtt{p}] \rangle \mid s[\mathtt{q}][\mathtt{p}] \& \mathtt{m}$$

The above unguarded process reduces trivially by invoking X infinitely, without triggering any typing context reduction. This clarifies the necessity of guarded definitions in Theorem 6.

4.3 Properties of Typed Processes

We showcase that processes constructed from global types guarantee desirable run-time properties, including deadlock-freedom and liveness, as formalised in Definition 15. These properties are relatively straightforward: *deadlock-freedom* indicates that if a process is unable to reduce, then it consists only of inactive sub-processes ($\mathbf{0}$); *liveness* ensures that if a process attempts to perform an input or output, it will eventually succeed.

Definition 15 (Runtime Process Properties). *We say* P *is:*

(1) deadlock-free *iff* $P \rightarrow^* P' \not\rightarrow$ *implies* $P' \equiv \mathbf{0}$;
(2) live *iff* $P \rightarrow^* P' \equiv \mathbb{C}[Q]$ *implies:*
  (a) if $Q = c[\mathtt{q}] \oplus \mathtt{m}\langle w \rangle.Q'$ *then* $\exists \mathbb{C}' : P' \rightarrow^* \mathbb{C}'[Q']$;
  (b) if $Q = c[\mathtt{q}] \& \{\mathtt{m}_i(x_i).Q'_i\}_{i \in I}$ *then* $\exists \mathbb{C}', k \in I, w : P' \rightarrow^* \mathbb{C}'[Q'_k\{^{w}/_{x_k}\}]$.

We conclude by illustrating how a process, typed with a typing context associated to a global type on a session, ensures both deadlock-freedom and liveness.

Theorem 7 (Process Deadlock-Freedom, Liveness). *Assume* $\emptyset \cdot \Gamma \vdash P$, *where* $G \sqsubseteq_s \Gamma$, $P \equiv \Pi_{\mathtt{p} \in I} P_{\mathtt{p}}$, *and* $\Gamma = \bigcup_{\mathtt{p} \in I} \Gamma_{\mathtt{p}}$ *such that for each* $P_{\mathtt{p}}$, *we have* $\emptyset \cdot \Gamma_{\mathtt{p}} \vdash P_{\mathtt{p}}$. *Further, assume that each* $P_{\mathtt{p}}$ *is either* $\mathbf{0}$ *(up to* $\equiv$*), or only plays* $\mathtt{p}$ *in* s, *by* $\Gamma_{\mathtt{p}}$. *Then,* P *is deadlock-free and live.*

Example 16 (Typed Process Properties). The process $P_{\mathtt{s}} \mid P_{\mathtt{c}} \mid P_{\mathtt{a}}$ from Examples 2 and 12 is deadlock-free and live, verified by applying either Definition 15 or Theorem 7.

5 Conclusion

This paper corrects a misunderstanding in multiparty session type theory, proposing a new relation, *association* $G \sqsubseteq_s \Gamma$ between a global type G and a set of local types Γ. The top-down typing system, with the association between global types and their end-point projections given by mergeability, can guarantee *type safety*, *deadlock-freedom*, and *liveness* of session processes *by construction*.

We conclude with song lyrics written by a singer-songwriter, Mario T., who is Cliff's friend and was Nobuko Yoshida's PhD supervisor at the University of Keio. Tokoro-sensei has contributed to many fields of computer science, particularly, concurrent object-oriented languages, with which Cliff had active discussions and communications: for example, Cliff stated that the object-oriented concept proposed in [19] is an important extension from the abstract data type theme for promoting GyP∫ISin [9, § 8.3.5.].

The song for Cliff's Festschrift is composed and selected by Mario T.[1]

Questar

Can't tell what is true. Can't tell what I am.
Can't tell what exists. Can't tell what's illusion.

At every moment I come to myself,
I wonder what is true and what I am.
Decompose things into parts,
people can know something in vain.

Consciousness is a tool to find essences,
putting the whole out of our mind there.

And in the moment when I'm drowsing
I see suddenly its existence,
I don't see it in pieces,
but as a whole, the universe, and you.

Mario T.
(The lyrics were inspired by Keith Jarrett's Questar.)

References

1. Barwell, A., Scalas, A., Yoshida, N., Zhou, F.: Generalised multiparty session types with crash-stop failures. In: 33rd International Conference on Concurrency Theory. LIPIcs, vol. 243, pp. 35:1–35:25. Dagstuhl (2022). https://doi.org/10.4230/LIPIcs.CONCUR.2022.35

[1] Mario T.: https://www.youtube.com/@mariot.3115. Questar: https://www.youtube.com/watch?v=FM6EKa0C88Y.

2. Bernardi, G., Hennessy, M.: Using higher-order contracts to model session types. LMCS **12(2)** (2016). https://doi.org/10.2168/LMCS-12(2:10)2016
3. Carbone, M., Honda, K., Yoshida, N.: Structured Communication-Centred Programming for Web Services. In: De Nicola, R. (eds.) ESOP 2007, LNCS, vol. 4421, pp. 2–17. Springer, Cham (2007). https://doi.org/10.1007/978-3-540-71316-6_2
4. Coppo, M., Dezani-Ciancaglini, M., Yoshida, N., Padovani, L.: Global progress for dynamically interleaved multiparty sessions. MSCS **760** (2015). https://doi.org/10.1017/S0960129514000188
5. Deniélou, P.M., Yoshida, N.: Multiparty Compatibility in Communicating Automata: Characterisation and Synthesis of Global Session Types. In: 40th International Colloquium on Automata, Languages and Programming. LNCS, vol. 7966, pp. 174–186. Springer (2013). https://doi.org/10.1007/978-3-642-39212-2_18
6. Ghilezan, S., Pantović, J., Prokić, I., Scalas, A., Yoshida, N.: Precise subtyping for asynchronous multiparty sessions. ACM Trans. Comput. Logic **24** (2)(14), 1–73 (2023). https://doi.org/10.1145/3568422
7. Glabbeek, R.v., Höfner, P., Horne, R.: Assuming just enough fairness to make session types complete for lock-freedom. In: 2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), pp. 1–13 (2021).https://doi.org/10.1109/LICS52264.2021.9470531
8. Web Services Choreography Working Group (2003). http://w3.org/2002/ws/chor/
9. Gurd, J.R., Jones, C.B.: The global-yet-personal information system. In: Wand, I., Milner, R. (eds.) Computing Tomorrow, pp. 127–157. Cambridge University Press (1996). https://doi.org/10.1017/CBO9780511605611
10. Hewitt, C., de Jong, P.: Open systems. In: Brodie, M.L., Mylopoulos, J., Schmidt, J.W. (eds.) On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases, and Programming Languages, pp. 147–164. Springer (1984). https://doi.org/10.1007/978-1-4612-5196-5_6
11. Honda, K., Mukhamedov, A., Brown, G., Chen, T.C., Yoshida, N.: Scribbling interactions with a formal foundation. In: Natarajan, R., Ojo, A. (eds.) ICDCIT, LNCS, vol. 6536, pp. 55–75. Springer (2011). https://doi.org/10.1007/978-3-642-19056-8_4
12. Honda, K., Yoshida, N., Carbone, M.: Multiparty asynchronous session types. In: POPL (2008). https://doi.org/10.1145/1328438.1328472, full version in [13]
13. Honda, K., Yoshida, N., Carbone, M.: Multiparty asynchronous session types. J. ACM **63**(1) (2016). https://doi.org/10.1145/2827695
14. Miu, A., Ferreira, F., Yoshida, N., Zhou, F.: Communication-safe web programming in typescript with routed multiparty session types. In: International Conference on Compiler Construction, pp. 94–106. CC (2021).https://doi.org/10.1145/3446804.3446854
15. OAuth Working Group: RFC 6749: OAuth 2.0 framework. http://tools.ietf.org/html/rfc6749 (2012)
16. Pierce, B.C.: Types and Programming Languages. The MIT Press, 1st edn. (2002). https://dl.acm.org/doi/abs/10.5555/509043
17. Sangiorgi, D.: Introduction to Bisimulation and Coinduction. Cambridge University Press (2011). https://doi.org/10.1017/CBO9780511777110
18. Scalas, A., Yoshida, N.: Less is more: multiparty session types revisited. Proc. ACM Program. Lang. **3**(POPL), 30:1–30:29 (2019).https://doi.org/10.1145/3290343
19. Tokoro, M.: The society of objects. SIGPLAN OOPS Mess. **5**(2), 3–12 (1993). https://doi.org/10.1145/260304.260305

20. Yoshida, N., Gheri, L.: A very gentle introduction to multiparty session types. In: Distributed Computing and Internet Technology - ICDCIT 2020. LNCS, vol. 11969, pp. 73–93. Springer (2020). https://doi.org/10.1007/978-3-030-36987-3_5
21. Yoshida, N., Hou, P.: Less is More Revisited. CoRR **abs/2402.16741** (2024). https://doi.org/10.48550/ARXIV.2402.16741
22. Yoshida, N., Hu, R., Neykova, R., Ng, N.: The scribble protocol language. In: Abadi, M., Lluch Lafuente, A. (eds.) Trustworthy Global Computing. LNCS, vol. 8358, pp. 22–41. Springer, Cham (2013).https://doi.org/10.1007/978-3-319-05119-2_3
23. Yoshida, N., Zhou, F., Ferreira, F.: Communicating finite state machines and an extensible toolchain for multiparty session types. In: Bampis, E., Pagourtzis, A. (eds.) Fundamentals of Computation Theory. FCT 2021. LNCS, vol. 12867, pp. 18–35. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-86593-1_2