Asynchronous Global Protocols, Precisely

Kai Pischke(✉) and Nobuko Yoshida

University of Oxford, Oxford, UK
kai.pischke@cs.ox.ac.uk

Abstract. Asynchronous multiparty session types are a type-based framework that ensures the compatibility of components in a distributed system by specifying a global protocol. Each component can be independently developed and *refined* locally, before being integrated into a larger system, leading to higher quality distributed software. This paper studies the interplay between global protocols and an asynchronous refinement relation, *precise asynchronous multiparty subtyping*. This subtyping relation locally *optimises* asynchronous messaging, enabling a permutation of two actions in a component while still preserving the safety and liveness of the overall composed system. In this paper, we first define the *asynchronous association* between a global protocol and a set of local (optimised) specifications. We then prove the *soundness* and *completeness* of the operational correspondence of this asynchronous association. We demonstrate that the association acts as an *invariant* to provide type soundness, deadlock-freedom and liveness of a collection of components optimised from the end-point projections of a given global protocol.

Keywords: Multiparty session types · Precise asynchronous multiparty session subtyping · Type-safety · Association · Optimisation

1 Introduction

Concurrent and distributed components, often viewed as multiagents, are an effective abstraction for building flexible concurrent and distributed systems. Jean-Bernard Stefani is a pioneer of *component-based software engineering* (CBSE). He has promoted CBSE to both language and system communities, proposing a number of novel frameworks, systems and models. Two of many examples are a software framework for component-based OS kernels, THINK [11], which enables code-reuse and reduction of development times for building embedded systems; and a modular, extensible and language-independent model for configurable software systems, FRACTAL[1], which was first introduced by France Telecom and

[1] https://fractal.ow2.io/.

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, ARIA and Horizon EU TaRDIS 101093006.

C. A.Mezzina and A. Schmitt(Eds.): Jean-Bernard Stefani Festschrift, LNCS 16065, pp. 116–133, 2026.
https://doi.org/10.1007/978-3-031-99717-4_7

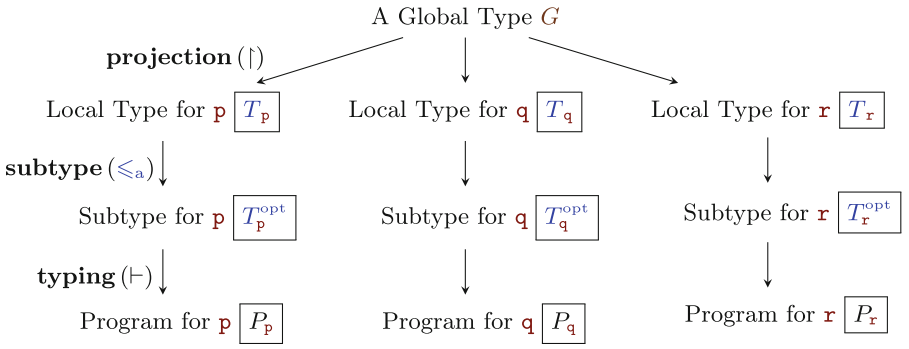


Fig. 1. Top-down methodology of multiparty session types. G denotes a global type, which is projected into the three participants, p, q and r, generating local types T_p, T_q and T_r for each participant. Local types are then refined to T_p^{opt}, T_q^{opt} and T_r^{opt}. Three distributed programs P_p, P_q and P_r follow.

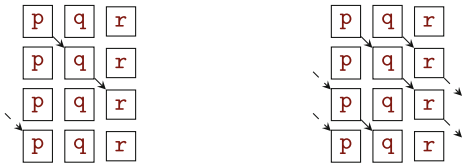
INRIA. THINK has had a significant impact on the embedded systems community, and FRACTAL has been used for developing multiple implementations in different programming languages (such as Java, C, C++, Smalltalk, .Net).

Session types [14,20] are a type discipline for codifying concurrent components. *Multiparty session types* [15,16] (MPST) extend this idea from two-party to multiparty communication, facilitating a programmer in specifying a *global protocol* to coordinate communicating components. Using MPST, we can ensure that typed components interact without type errors or deadlocks *by construction*. Similar to FRACTAL, the MPST framework is *language-agnostic*, and has been adapted into over 20 programming languages [22].

Figure 1 describes the MPST workflow. We assume a set of participants $\mathcal{P}$ in the distributed system. We specify a *global protocol (type)* G, which is projected into a set of *local protocols (types)* $\{T_p\}_{p\in\mathcal{P}}$ from the viewpoint of each participant p. The local type T_p is then *refined* to an optimised local type T_p^{opt} using the *multiparty asynchronous subtyping relation* $\leqslant_a$ [13]. Subtyping $\leqslant_a$ allows for "safe permutations" of actions (explained in § 1), enabling us to type a more optimised program P_p which conforms to T_p^{opt}. Once each program is typed, we can automatically guarantee that a collection of distributed programs $\{P_p\}_{p\in\mathcal{P}}$ satisfy safety, deadlock-freedom and liveness.

This workflow (called *top-down* in [21]) is implemented by the MPST toolchains, SCRIBBLE [25] and νSCR [26], which check whether a given global protocol is well-formed, and if so, generate a corresponding set of local types. Building on this, the Rust toolchain RUMPSTEAK [10] uses νSCR to generate state machines, from which *optimised* APIs are generated using a sound approximation of $\leqslant_a$.

Ring-Choice Example. We explain our workflow by introducing a running example which will be referenced throughout this paper, the ring-choice protocol G_{ring} from [9]:



(a) projection of G_{ring} (b) optimised projection of G_{ring}

Fig. 2. Ring protocol: Projected and optimised interactions (from [9])

$$G_{\text{ring}} = \mu \mathbf{t}.\mathtt{p}{\to}\mathtt{q}{:}\mathsf{add}(\mathsf{int}).\mathtt{q}{\to}\mathtt{r}{:}\left\{\begin{array}{l}\mathsf{add}(\mathsf{int})\,.\,\mathtt{r}{\to}\mathtt{p}{:}\{\mathsf{add}(\mathsf{int})\,.\,\mathbf{t}\}\\ \mathsf{sub}(\mathsf{int})\,.\,\mathtt{r}{\to}\mathtt{p}{:}\{\mathsf{sub}(\mathsf{int})\,.\,\mathbf{t}\}\end{array}\right\}$$

The global type G_{ring} specifies that:

1. p sends an integer n to q labelled by add;
2. q sends an integer m to r labelled by add or sub;
   (a) if add is selected, it sends the integer $m + k$ labelled by add to p, and the protocol restarts from Step 1; and
   (b) if sub is selected, it sends the integer $m - k$ labelled by sub to p, and the protocol restarts from Step 1.

If we assume *synchronous interactions* as illustrated in Fig. 2(a), no data flow would occur from q to r and from r to p *before* q receives data from p. This synchronisation is captured by the local types which are projected from G:

$$\begin{aligned} T_{\mathtt{p}} &= \mu \mathbf{t}.\mathtt{q}{\oplus}\{\mathtt{add}(\mathsf{int}).\mathtt{r}\&\{\mathtt{add}(\mathsf{int}).\mathbf{t}, \mathtt{sub}(\mathsf{int}).\mathbf{t}\}\}\\ T_{\mathtt{q}} &= \mu \mathbf{t}.\mathtt{p}\&\{\mathtt{add}(\mathsf{int}).\mathtt{r}{\oplus}\{\mathtt{add}(\mathsf{int}).\mathbf{t}, \mathtt{sub}(\mathsf{int}).\mathbf{t}\}\}\\ T_{\mathtt{r}} &= \mu \mathbf{t}.\mathtt{q}\&\{\mathtt{add}(\mathsf{int}).\mathtt{p}{\oplus}\{\mathtt{add}(\mathsf{int}).\mathbf{t}\}, \mathtt{sub}(\mathsf{int}).\mathtt{p}{\oplus}\{\mathtt{sub}(\mathsf{int}).\mathbf{t}\}\}\end{aligned}$$

where the notation $\oplus$ is a *selection type* which denotes an internal choice (followed by label and payload), while & denotes a *branching type*, representing an external choice.

Under *asynchronous interactions* illustrated in Fig. 2(b), assuming that each participant begins with its own initial value, q can concurrently choose one of two labels to send the data to r *before* receiving data from p, letting r and p start the next action. By applying asynchronous subtyping ($\leqslant_{\mathrm{a}}$), we can optimise $T_{\mathtt{p}}$ to the following $T_{\mathtt{p}}^{\mathrm{opt}}$, pushing the external choice behind the internal one:

$$T_{\mathtt{q}}^{\mathrm{opt}} = \mu \mathbf{t}.\mathtt{r}{\oplus}\{\mathtt{add}(\mathsf{int}).\mathtt{p}\&\{\mathtt{add}(\mathsf{int}).\mathbf{t}\}, \mathtt{sub}(\mathsf{int}).\mathtt{p}\&\{\mathtt{add}(\mathsf{int}).\mathbf{t}\}\}$$

With process $P_{\mathtt{q}}$ typed by $T_{\mathtt{q}}^{\mathrm{opt}}$, we can run the ring protocol more efficiently (see [10]). An overview of the history of asynchronous subtyping is given in [8], encompassing the theory and applications of the relation.

Contributions. This paper proves the *sound* and *complete operational correspondence* between behaviours of global type G and a set of local types $\{T_{\mathtt{p}}^{\mathrm{opt}}\}_{\mathtt{p}\in\mathcal{P}}$, which are refined or optimised by $\leqslant_{\mathrm{a}}$ from G's projection $\{T_{\mathtt{p}}\}_{\mathtt{p}\in\mathcal{P}}$. We say $\{T_{\mathtt{p}}^{\mathrm{opt}}\}_{\mathtt{p}\in\mathcal{P}}$ is *associated* to G.

More formally, given a typing context $\Delta = \{\mathtt{p} : (\sigma_{\mathtt{p}},\ T_{\mathtt{p}}^{\mathrm{opt}})\}_{\mathtt{p}\in\mathrm{roles}(G)}, \Delta_{\mathbf{end}}$ where $\mathrm{roles}(G)$ is a set of roles in G, $\sigma_{\mathtt{p}}$ is the type of the queue for participant $\mathtt{p}$, and $\Delta_{\mathbf{end}}$ is a typing context which contains only termination type **end** (which denotes the participant has completed communications), then the *association* between Δ and a global type G is defined as follows:

$$\Delta \sqsubseteq_a G \qquad \text{if } G \upharpoonright_{\mathtt{p}} (\sigma_{\mathtt{p}}, T_{\mathtt{p}}) \quad \text{and} \quad T_{\mathtt{p}}^{\mathrm{opt}} \leqslant_{\mathrm{a}} T_{\mathtt{p}} \text{ for all } \mathtt{p} \in \mathrm{roles}(G) \tag{1}$$

Once we obtain the soundness and completeness of the association, we can derive the *subject reduction* theorem and *session fidelity* of the top-down approach from the corresponding results of the bottom-up system [13, Theorems 4.11 and 4.13]. The bottom-up system does *not* use global types and their projections, but requires an additional check that the collection of local types (i.e., a typing context) satisfies a *safety* property [19].

More specifically, we divide the steps to derive these results as follows:

Step 1 We define the operational semantics of G (denoted by $G \longrightarrow G'$) and a typing context Δ (denoted by $\Delta \to \Delta'$).

Step 2 We prove **soundness**: if $\Delta \sqsubseteq_a G$ and $G \longrightarrow$, then there exist G' and Δ' such that $G \longrightarrow G'$, $\Delta \to \Delta'$ and $\Delta' \sqsubseteq_a G'$.

Step 3 We prove **completeness**: if $\Delta \sqsubseteq_a G$ and $\Delta \to \Delta'$, then there exists G' such that $G \longrightarrow G'$ and $\Delta' \sqsubseteq_a G'$.

Step 4 We define the typing rule for multiparty session processes using the association:

$$\frac{\forall \mathtt{p} \in \mathrm{dom}(\Delta) \quad \vdash P_{\mathtt{p}} \triangleright T_{\mathtt{p}} \quad \vdash h_{\mathtt{p}} \triangleright \sigma_{\mathtt{p}} \quad \Delta(\mathtt{p}) = (\sigma_{\mathtt{p}}, T_{\mathtt{p}}) \quad \Delta \sqsubseteq_a G}{\vdash^{\mathrm{top}} \Pi_{\mathtt{p}\in\mathrm{dom}(\Delta)}\ (\mathtt{p} \triangleleft P_{\mathtt{p}} \mid \mathtt{p} \triangleleft h_{\mathtt{p}}) \ \triangleright \Delta} \ [\textsc{SessTop}]$$

where $\vdash P \triangleright T$ is a typing judgement to assign type T to process P and $\vdash h \triangleright \sigma$ assigns type σ to a FIFO queue h (defined in [13, Figure 5]). $\mathtt{p} \triangleleft P_{\mathtt{p}}$ means process $P_{\mathtt{p}}$ is acting as participant $\mathtt{p}$, buffering sent messages in its queue $\mathtt{p} \triangleleft h_{\mathtt{p}}$.

Step 5 We prove **the subject reduction theorem of the top-down system** using the completeness of the association with the subject reduction theorem of the bottom-up system [13, Theorem 4.11]; and **the session fidelity theorem of the top-down system** using the soundness and completeness of the association with the session fidelity theorem of the bottom-up system [13, Theorem 4.13]. We can also derive **safety**, **deadlock-freedom** and **liveness** from [19] and [13, Theorem 4.12]. We give detailed explanations in Sect. 5.

$$
\begin{array}{lcll}
B & ::= & \mathsf{int} \mid \mathsf{bool} \mid \mathsf{real} \mid \mathsf{unit} \mid \ldots & \text{Basic types} \\
G & ::= & \mathtt{p}\to\mathtt{q}\colon\{\mathtt{m}_i(B_i).G_i\}_{i\in I} & \text{Transmission} \\
 & \mid & \mathtt{p}\overset{\mathtt{m}_j}{\rightsquigarrow}\mathtt{q}\colon\{\mathtt{m}_i(B_i).G_i\}_{i\in I} & \text{Transmission en route} \\
 & \mid & \mu\mathbf{t}.G \mid \mathbf{t} \mid \mathbf{end} & \text{Recursion, Type variable, Termination} \\
T & ::= & \mathtt{p}\&\{\mathtt{m}_i(B_i).T_i\}_{i\in I} & \text{External choice} \\
 & \mid & \mathtt{p}\oplus\{\mathtt{m}_i(B_i).T_i\}_{i\in I} & \text{Internal choice} \\
 & \mid & \mu\mathbf{t}.T \mid \mathbf{t} \mid \mathbf{end} & \text{Recursion, Type variable, Termination} \\
\sigma & ::= & \varnothing \mid (\mathtt{q},\mathtt{m}(B)) \mid \sigma\cdot\sigma & \text{Empty Queue, Message, Concatenation}
\end{array}
$$

Fig. 3. Syntax of types.

Outline. We provide an extensive exploration of global and local types in Sect.2.1, including syntax, projection, and subtyping. We define operational semantics for both global types (Sect.3.1) and typing contexts (Sect.3.2). We establish the sound and complete operational relationship between these two semantics in Sect.4. Furthermore, we demonstrate that the top-down typing system ensures subject reducution, session fidelity, and liveness in Sect.5. Full proofs are found in [18].

2 Multiparty Session Types

This section introduces *global* and *local* types, together with *queue* types. As in [1], our formulation of global types includes special runtime-specific constructs to allow global types to represent en-route messages which have been sent but not yet received, and we give a novel projection relation (Def. 1) which extends the standard coinductive projection [12, Definition 3.6] to asynchronous semantics by simultaneously projecting onto both local and queue types.

2.1 Global and Local Types

Multiparty Session Type (MPST) theory uses *global types* to provide a comprehensive overview of communications between *roles*, such as $\mathtt{p}, \mathtt{q}, \mathtt{s}, \mathtt{t}, \ldots$, belonging to a set $\mathcal{R}$. 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. 3, where constructs are mostly standard [19].

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.

- $\mathtt{p}\to\mathtt{q}\colon\{\mathtt{m}_\mathtt{i}(B_i).G_i\}_{i\in I}$: a *transmission*, denoting a message from role $\mathtt{p}$ to role $\mathtt{q}$, with a label $\mathtt{m}_i$, a payload of type B_i, 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. $\mathtt{p} \neq \mathtt{q}$).
- $\mathtt{p}\overset{\mathtt{m}_j}{\rightsquigarrow}\mathtt{q}\colon\{\mathtt{m}_\mathtt{i}(B_i).G_i\}_{i\in I}$: a *transmission en route*, representing a *transmission* of the message with index $j \in I$ which has already been sent by role $\mathtt{p}$ but has not been received by role $\mathtt{q}$. Note that since $\mathtt{q}$ has not yet received the message, all branches are still present even though it is predetermined which will be chosen. This type is only meaningful at runtime.
- $\mu\mathbf{t}.G$: a *recursive* global type, where contractive requirements apply [17, §21.8], i.e. each recursion variable $\mathbf{t}$ is bound within a $\mu\mathbf{t}\ldots$ and is guarded.
- **end**: a *terminated* global type (omitted where unambiguous).

Local types (or *session types*) range over $T, T', T_i, \ldots$, and describe the behaviour of a single role. We elucidate each syntactic construct of local types.

An *internal choice* (*selection*), $\mathtt{p}\oplus\{\mathtt{m}_\mathtt{i}(B_i).T_i\}_{i\in I}$, indicates that the *current* role is expected to *send* to role $\mathtt{p}$; an *external choice* (*branching*), $\mathtt{p}\&\{\mathtt{m}_\mathtt{i}(B_i).T_i\}_{i\in I}$, indicates that the *current* role is expected to *receive* from role $\mathtt{p}$; a *recursive* local type, $\mu\mathbf{t}.T$, follows a pattern analogous to $\mu\mathbf{t}.G$; finally, we use **end** for *termination* (omitted where unambiguous). Similar to global types, local types also need pairwise distinct, non-empty labels.

Queue types range over $\sigma, \sigma', \sigma_i, \ldots$, and describe the type of queues storing buffered asynchronous messages: $\varnothing$ is the empty queue; $(\mathtt{p}, \mathtt{m}(B))$ is the type of a queued message being sent to participant $\mathtt{p}$ with a message label $\mathtt{m}$ and a payload of type B; and $\sigma\cdot\sigma'$ is the concatenation of two queues.

2.2 Projections

Projection. In the top-down approach of MPST, local types are obtained by projecting a global type onto roles. Our definition of *projection*, as given in Def. 1 below, is slightly modified from the traditional presentation of projection as a partial function from global to local types. We define projection coinductively as a relation between global types G and pairs of local types T and queue types σ, allowing the queues to capture buffered messages, projected from en-route transmissions, at the local level.

Definition 1 (Global Type Projection). *The* projection of a global type G onto a role $\mathtt{p}$ *is defined coinductively as a relation* $G\upharpoonright_\mathtt{p}(\sigma, T)$ *by the rules in Fig. 4.*

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

- *If* $\mathrm{unf}(T) = \mathbf{end}$ *and* $\mathrm{unf}(T') = \mathbf{end}$ *then* $T \sqcap T' = \mathbf{end}$.
- *If* $\mathrm{unf}(T) = \mathtt{p}\oplus\{\mathtt{m}_\mathtt{i}(B_i).T_i\}_{i\in I}$ *and* $\mathrm{unf}(T') = \mathtt{p}\oplus\{\mathtt{m}_\mathtt{i}(B_i).T'_i\}_{i\in I}$ *then* $T \sqcap T' = \mathtt{p}\oplus\{\mathtt{m}_\mathtt{i}(B_i).T_i \sqcap T'_i\}_{i\in I}$.

$$\frac{\forall i \in I \quad G_i \upharpoonright_{\mathtt{p}} (\sigma, T_i)}{(\mathtt{q} \to \mathtt{r}\colon \{\mathtt{m}_i(B_i).G_i\}_{i\in I}) \upharpoonright_{\mathtt{p}} (\sigma, \bigsqcap_{i\in I} T_i)}\ [\text{P-}\sqcap]$$

$$\frac{\forall i \in I \quad G_i \upharpoonright_{\mathtt{p}} (\sigma, T_i)}{\left(\mathtt{q} \overset{\mathtt{m}_j}{\rightsquigarrow} \mathtt{r}\colon \{\mathtt{m}_i(B_i).G_i\}_{i\in I}\right) \upharpoonright_{\mathtt{p}} (\sigma, \bigsqcap_{i\in I} T_i)}\ [\text{P-}\sqcap\text{-II}]$$

$$\frac{\forall i \in I \quad G_i \upharpoonright_{\mathtt{p}} (\sigma, T_i)}{(\mathtt{p} \to \mathtt{q}\colon \{\mathtt{m}_i(B_i).G_i\}_{i\in I}) \upharpoonright_{\mathtt{p}} (\sigma, \mathtt{q} \oplus \{\mathtt{m}_i(B_i).T_i\}_{i\in I})}\ [\text{P-}\oplus]$$

$$\frac{G_j \upharpoonright_{\mathtt{p}} (\sigma, T)}{\left(\mathtt{p} \overset{\mathtt{m}_j}{\rightsquigarrow} \mathtt{q}\colon \{\mathtt{m}_i(B_i).G_i\}_{i\in I}\right) \upharpoonright_{\mathtt{p}} (\sigma \cdot (\mathtt{q}, \mathtt{m}_j(B_j)), T)}\ [\text{P-}\oplus\text{-II}]$$

$$\frac{\forall i \in I \quad G_i \upharpoonright_{\mathtt{p}} (\sigma, T_i)}{(\mathtt{q} \to \mathtt{p}\colon \{\mathtt{m}_i(B_i).G_i\}_{i\in I}) \upharpoonright_{\mathtt{p}} (\sigma, \mathtt{q} \& \{\mathtt{m}_i(B_i).T_i\}_{i\in I})}\ [\text{P-}\&]$$

$$\frac{\forall i \in I \quad G_i \upharpoonright_{\mathtt{p}} (\sigma, T_i)}{\left(\mathtt{q} \overset{j}{\rightsquigarrow} \mathtt{r}\colon \{\mathtt{m}_i(B_i).G_i\}_{i\in I}\right) \upharpoonright_{\mathtt{p}} (\sigma, \mathtt{q} \& \{\mathtt{m}_i(B_i).T_i\}_{i\in I})}\ [\text{P-}\&\text{-II}]$$

$$\frac{G\{^{\mu \mathbf{t}.G}/_{\mathbf{t}}\} \upharpoonright_{\mathtt{p}} (\sigma, T)}{\mu \mathbf{t}.G \upharpoonright_{\mathtt{p}} (\sigma, T)}\ [\text{P-L}] \qquad \frac{G \upharpoonright_{\mathtt{p}} (\sigma, T\{^{\mu \mathbf{t}.T}/_{\mathbf{t}}\})}{G \upharpoonright_{\mathtt{p}} (\sigma, \mu \mathbf{t}.T)}\ [\text{P-R}] \qquad \frac{}{\mathbf{end} \upharpoonright_{\mathtt{p}} (\varnothing, \mathbf{end})}\ [\text{P-End}]$$

Fig. 4. Rules for coinductive projection.

– *If* $\mathrm{unf}(T) = \mathtt{p}\&\{\mathtt{m}_i(B_i).T_i\}_{i\in I}$ *and* $\mathrm{unf}(T') = \mathtt{p}\&\{\mathtt{m}_j(B_j).T'_j\}_{j\in J}$ *then* $T \sqcap T' = \mathtt{p}\&\{\mathtt{m}_k(B_k).T''_k\}_{k\in I\cup J}$. *Where* $T''_k = \begin{cases} T_k \sqcap T'_k & \text{if } k \in I \cap J \\ T_k & \text{if } k \in I \setminus J \\ T'_k & \text{if } k \in J \setminus I \end{cases}$

We make use of an unfolding function, defined by $\mathrm{unf}(\mu \mathbf{t}.T) = \mathrm{unf}(T\{^{\mu \mathbf{t}.T}/_{\mathbf{t}}\})$ *when there is a recursion binder at the outermost level otherwise* $\mathrm{unf}(T) = T$.

A new rule, [P-⊕-II], allows an en-route message $(\mathtt{q}, \mathtt{m}_j(B_j))$ to be included in the projected queue of outgoing messages. 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 Def. 1, is used to ensure that the projections of all continuations are "compatible". It is noteworthy that there are global types that cannot be projected onto all of their participants as shown in [21, §4.4].

Recall the ring-choice example: projection $G_{\mathrm{ring}} \upharpoonright_{\mathtt{p}} T_{\mathtt{p}}$ can be derived by applying [P-L], [P-R], [P-⊕], [P-⊓], and merging the results of applying [P-&], to the coinductive hypothesis for each branch.

2.3 Asynchronous Multiparty Subtyping

We introduce a *subtyping* relation $\leqslant_{\mathrm{a}}$ on local types, as defined in Def. 2. This subtyping relation is standard [19], and will be used later when defining local type semantics and establishing the relationship between global and local type semantics.

Given a standard subtyping $<:$ for basic types (e.g. including $\mathsf{int} <: \mathsf{real}$), we give a summary of *asynchronous subtyping* $\leqslant_{\mathrm{a}}$ introduced in [13]. We first consider the tree representation of local type T (denoted by $\mathfrak{T}(T)$).

We write $\mathbb{T}$ for generic trees and additionally define three specific types of tree. *Single-input* trees (denoted by $\mathbb{V}$) are those which have only a singleton choice in all branchings, while *single-output* trees (denoted by $\mathbb{U}$) are those which have only a singleton choice in all selections. Trees which are both single-input and single-output are called *single-input-single-output (SISO) trees* (denoted by $\mathbb{W}$). These can all be defined coinductively by the following equations.

$$\begin{aligned}
\mathbb{T} &= \mathtt{p}\&\{\mathtt{m}_i(B_i).\mathbb{T}_i\}_{i\in I} \mid \mathtt{p}\oplus\{\mathtt{m}_i(B_i).\mathbb{T}_i\}_{i\in I} \mid \mathbf{end}\\
\mathbb{U} &= \mathtt{p}\&\{\mathtt{m}_i(B_i).\mathbb{U}_i\}_{i\in I} \mid \mathtt{p}!\mathtt{m}(B);\mathbb{U} \mid \mathbf{end}\\
\mathbb{V} &= \mathtt{p}?\mathtt{m}(B);\mathbb{V} \mid \mathtt{p}\oplus\{\mathtt{m}_i(B_i).\mathbb{V}_i\}_{i\in I} \mid \mathbf{end}\\
\mathbb{W} &= \mathtt{p}?\mathtt{m}(B);\mathbb{W} \mid \mathtt{p}!\mathtt{m}(B);\mathbb{W} \mid \mathbf{end}
\end{aligned}$$

We will define reorderings of SISO trees, and to do so, we consider non-empty sequences $\mathcal{A}^{(\mathtt{p})}$ of receives not including $\mathtt{p}$ and $\mathcal{B}^{(\mathtt{p})}$ of sends not including $\mathtt{p}$ together with receives from any participant. These sequences are inductively defined (where $\mathtt{p} \neq \mathtt{q}$) by:

$$\mathcal{A}^{(\mathtt{p})} = \mathtt{q}?\mathtt{m}(B) \mid \mathtt{q}?\mathtt{m}(B);\mathcal{A}^{(\mathtt{p})} \qquad \mathcal{B}^{(\mathtt{p})} = \mathtt{r}?\mathtt{m}(B) \mid \mathtt{q}!\mathtt{m}(B) \mid \mathtt{r}?\mathtt{m}(B);\mathcal{B}^{(\mathtt{p})} \mid \mathtt{q}!\mathtt{m}(B);\mathcal{B}^{(\mathtt{p})}$$

We define the set $\mathtt{act}(\mathbb{W})$ of actions of a SISO tree:

$\mathtt{act}(\mathbf{end}) = \emptyset$; $\mathtt{act}(\mathtt{p}?\mathtt{m}(B);\mathbb{W}) = \{\mathtt{p}?\} \cup \mathtt{act}(\mathbb{W})$; and $\mathtt{act}(\mathtt{p}!\mathtt{m}(B);\mathbb{W}) = \{\mathtt{p}!\} \cup \mathtt{act}(\mathbb{W})$.

Using these definitions, we introduce a *refinement relation* ($\lesssim$) defined coinductively by the following rules:

$$\frac{B' <: B \quad \mathbb{W} \lesssim \mathcal{A}^{(\mathtt{p})};\mathbb{W}' \quad \mathtt{act}(\mathbb{W}) = \mathtt{act}(\mathcal{A}^{(\mathtt{p})};\mathbb{W}')}{\mathtt{p}?\mathtt{m}(B);\mathbb{W} \lesssim \mathcal{A}^{(\mathtt{p})};\mathtt{p}?\mathtt{m}(B');\mathbb{W}'}\ [\textsc{Ref-}\mathcal{A}]$$

$$\frac{B <: B' \quad \mathbb{W} \lesssim \mathcal{B}^{(\mathtt{p})};\mathbb{W}' \quad \mathtt{act}(\mathbb{W}) = \mathtt{act}(\mathcal{B}^{(\mathtt{p})};\mathbb{W}')}{\mathtt{p}!\mathtt{m}(B);\mathbb{W} \lesssim \mathcal{B}^{(\mathtt{p})};\mathtt{p}!\mathtt{m}(B');\mathbb{W}'}\ [\textsc{Ref-}\mathcal{B}]$$

$$\frac{B' <: B \quad \mathbb{W} \lesssim \mathbb{W}'}{\mathtt{p}?\mathtt{m}(B);\mathbb{W} \lesssim \mathtt{p}?\mathtt{m}(B');\mathbb{W}'}\ [\textsc{Ref-In}] \qquad \frac{B <: B' \quad \mathbb{W} \lesssim \mathbb{W}'}{\mathtt{p}!\mathtt{m}(B);\mathbb{W} \lesssim \mathtt{p}!\mathtt{m}(B');\mathbb{W}'}\ [\textsc{Ref-Out}]$$

$$\frac{}{\mathbf{end} \lesssim \mathbf{end}}\ [\textsc{Ref-End}]$$

We can extract sets of single-input and single-output trees from a given tree using the functions $[\![\cdot]\!]_{\mathrm{SI}}$ and $[\![\cdot]\!]_{\mathrm{SO}}$.

$$[\![\mathtt{p}\&\{\mathtt{m}_i(B_i).\mathbb{T}_i\}_{i\in I}]\!]_{\mathrm{SI}} = \bigcup_{i\in I}\{\mathtt{p}?\mathtt{m}_i(B_i);\mathbb{V}_i \mid \mathbb{V}_i \in [\![\mathbb{T}_i]\!]_{\mathrm{SI}}\}$$

$$[\![\mathtt{p}\oplus\{\mathtt{m}_i(B_i).\mathbb{T}_i\}_{i\in I}]\!]_{\mathrm{SI}} = \{\mathtt{p}\oplus\{\mathtt{m}_i(B_i).\mathbb{V}_i\}_{i\in I} \mid \forall i \in I : \mathbb{V}_i \in [\![\mathbb{T}_i]\!]_{\mathrm{SI}}\} \quad [\![\mathbf{end}]\!]_{\mathrm{SI}} = \{\mathbf{end}\}$$

$$[\![\mathtt{p}\oplus\{\mathtt{m}_i(B_i).\mathbb{T}_i\}_{i\in I}]\!]_{\mathrm{SO}} = \bigcup_{i\in I}\{\mathtt{p}!\mathtt{m}_i(B_i);\mathbb{U}_i \mid \mathbb{U}_i \in [\![\mathbb{T}_i]\!]_{\mathrm{SO}}\}$$

$$[\![\mathtt{p}\&\{\mathtt{m}_i(B_i).\mathbb{T}_i\}_{i\in I}]\!]_{\mathrm{SO}} = \{\mathtt{p}\&\{\mathtt{m}_i(B_i).\mathbb{U}_i\}_{i\in I} \mid \forall i \in I : \mathbb{U}_i \in [\![\mathbb{T}_i]\!]_{\mathrm{SO}}\} \quad [\![\mathbf{end}]\!]_{\mathrm{SO}} = \{\mathbf{end}\}$$

Definition 2 (Subtyping). *We consider trees that have only singleton choices in branchings (called* single-input (SI) trees*), or in selections (*single-output (SO) trees*), and we define the session subtyping $\leqslant_a$ over all session types by considering their decomposition into SI, SO, and SISO trees.*

$$\frac{\forall \mathbb{U} \in [\![\mathfrak{T}(T)]\!]_{SO} \quad \forall \mathbb{V}' \in [\![\mathfrak{T}(T')]\!]_{SI} \quad \exists \mathbb{W} \in [\![\mathbb{U}]\!]_{SI} \quad \exists \mathbb{W}' \in [\![\mathbb{V}']\!]_{SO} \quad \mathbb{W} \lesssim \mathbb{W}'}{T \leqslant_a T'} \text{[Sub]}$$

The refinement $\lesssim$ captures safe permutations *of input/output messages, that never cause deadlocks or communication errors under asynchrony; and the subtyping relation $\leqslant_a$ focuses on reconciling refinement $\lesssim$ with the branching structures in session types.*

Example 1 (Subtyping the Ring Protocol Projection). To demonstrate that $T_{\mathtt{q}}^{\mathrm{opt}} \leqslant_a T_{\mathtt{q}}$, we must show that for all $\mathbb{U} \in [\![\mathfrak{T}(T_{\mathtt{q}}^{\mathrm{opt}})]\!]_{\mathrm{SO}}$ and $\mathbb{V}' \in [\![\mathfrak{T}(T_{\mathtt{q}})]\!]_{\mathrm{SI}}$, there exist $\mathbb{W} \in [\![\mathbb{U}]\!]_{\mathrm{SI}}$ and $\mathbb{W}' \in [\![\mathbb{V}']\!]_{\mathrm{SO}}$ such that $\mathbb{W} \lesssim \mathbb{W}'$. Consider the following sets:

$$[\![\mathfrak{T}(T_{\mathtt{q}}^{\mathrm{opt}})]\!]_{\mathrm{SO}} = \left\{\mathtt{r}!\mathsf{add}(\mathsf{int});\mathtt{p}?\mathsf{add}(\mathsf{int});\ldots,\mathtt{r}!\mathsf{sub}(\mathsf{int});\mathtt{p}?\mathsf{add}(\mathsf{int});\ldots,\ldots\right\}$$

$$[\![\mathfrak{T}(T_{\mathtt{q}})]\!]_{\mathrm{SI}} = \left\{\mathtt{p}?\mathsf{add}(\mathsf{int});\mathtt{r}\oplus\left\{\begin{array}{l}\mathsf{add}(\mathsf{int})\ldots.\\ \mathsf{sub}(\mathsf{int})\ldots.\end{array}\right\}\right\}$$

Now, we must find for each $\mathbb{U}$ in the first set and $\mathbb{V}'$ in the second, a pair of SISO trees $(\mathbb{W},\mathbb{W}')$ such that $\mathbb{W} \lesssim \mathbb{W}'$. For instance, if the second $\mathbb{U}$ is chosen, we have $\mathbb{W} = \mathtt{r}!\mathsf{sub}(\mathsf{int});\mathtt{p}?\mathsf{add}(\mathsf{int});\ldots$ and we can pick $\mathbb{W}' = \mathtt{p}?\mathsf{add}(\mathsf{int});\mathtt{p}!\mathsf{sub}(\mathsf{int});\ldots$
Then we can apply rule [Ref-$\mathcal{B}$] to validate that it is safe to reorder the send ahead of the receive in the optimised type. We could form a similar argument in the other cases. Thus we conclude that:$T_{\mathtt{q}}^{\mathrm{opt}} \leqslant_a T_{\mathtt{q}}$.

Lemma 1 (Merge preserves subtyping). *Given collections of mergeable types T_i and T_i' $(i \in I)$. If for all $i \in I$, $T_i \leqslant_a T_i'$ then $\bigsqcap_{i\in I} T_i \leqslant_a \bigsqcap_{i\in I} T_i'$.*

3 Operational Semantics

3.1 Semantics of Global Types

We now present the Labelled Transition System (LTS) semantics for global types. To begin, we introduce the transition labels in Def. 3, which are also used in the LTS semantics of typing contexts (discussed later in §3.2).

$$\frac{G[\mu \mathtt{t}.G/\mathtt{t}] \xrightarrow{\alpha} G'}{\mu \mathtt{t}.G \xrightarrow{\alpha} G'}\ [\text{GR-}\mu] \qquad \frac{j \in I}{\mathtt{p} \overset{\mathtt{m}_j}{\rightsquigarrow} \mathtt{q}\colon \{\mathtt{m}_i(B_i).G'_i\}_{i\in I} \xrightarrow{\mathtt{q}:\mathtt{p}\&\mathtt{m}_j} G'_j}\ [\text{GR-}\&]$$

$$\frac{j \in I}{\mathtt{p} \to \mathtt{q}\colon \{\mathtt{m}_i(B_i).G'_i\}_{i\in I} \xrightarrow{\mathtt{p}:\mathtt{q}\oplus\mathtt{m}_j} \mathtt{p} \overset{\mathtt{m}_j}{\rightsquigarrow} \mathtt{q}\colon \{\mathtt{m}_i(B_i).G'_i\}_{i\in I}}\ [\text{GR-}\oplus]$$

$$\frac{\forall i \in I : G'_i \xrightarrow{\alpha} G''_i \quad \alpha \neq \mathtt{p}:\mathtt{q}\oplus\mathtt{m}' \quad \alpha \neq \mathtt{p}:\mathtt{r}\&\mathtt{m}''}{\mathtt{p} \to \mathtt{q}\colon \{\mathtt{m}_i(B_i).G'_i\}_{i\in I} \xrightarrow{\alpha} \mathtt{p} \to \mathtt{q}\colon \{\mathtt{m}_i(B_i).G''_i\}_{i\in I}}\ [\text{GR-Ctx-I}]$$

$$\frac{j \in J \quad \forall i \in I : G'_i \xrightarrow{\alpha} G''_i \quad \alpha \neq \mathtt{q}:\mathtt{p}\&\mathtt{m}'}{\mathtt{p} \overset{\mathtt{m}_j}{\rightsquigarrow} \mathtt{q}\colon \{\mathtt{m}_i(B_i).G'_i\}_{i\in I} \xrightarrow{\alpha} \mathtt{p} \overset{\mathtt{m}_j}{\rightsquigarrow} \mathtt{q}\colon \{\mathtt{m}_i(B_i).G''_i\}_{i\in I}}\ [\text{GR-Ctx-II}]$$

Fig. 5. Global type reduction rules.

Definition 3 (Transition Labels). *Let α be a transition label of the form:*

$$\alpha \ ::= \ \mathtt{p}:\mathtt{q}\&\mathtt{m} \ \mid \ \mathtt{p}:\mathtt{q}\oplus\mathtt{m} \quad \textit{(receive or send a message)}$$

Definition 4 (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 $\longrightarrow$.*

The semantics of global types reflect the reorderings permitted by asynchronous subtying, allowing transitions according to specific asynchronous behaviours:

- [GR-μ] permits a valid transition to take place under a recursion binder.
- [GR-&] describes the receiving of asynchronous messages, allowing en-route message to be received.
- [GR-$\oplus$] describes the sending of asynchronous messages, resulting in a standard transmission becoming an en-route one.
- [GR-Ctx-I] allows the semantics to anticipate a deeper transition inside a communication type so long as it is not a send between the same participants or receive by the sending participant. The restriction $\alpha \neq \mathtt{p}:\mathtt{q}\oplus\mathtt{m}'$ corresponds with the fact that $\mathcal{B}^{(\mathtt{p})}$ does not allow sends preempting sends to the same participant, and the restriction $\alpha \neq \mathtt{p}:\mathtt{r}\&\mathtt{m}''$ corresponds with the fact that $\mathcal{B}^{(\mathtt{p})}$ does not allow receives preempting sends to the same participant.
- [GR-Ctx-II] is even more flexible. The only restriction, $\alpha \neq \mathtt{q}:\mathtt{p}\&\mathtt{m}'$ does not place any limits on the sender who has already triggered an en-route message, only requiring the receiver not pre-emptively receive a different message from the same sender, as with $\mathcal{A}^{(\mathtt{p})}$.

In this way, [GR-Ctx-I] and [GR-Ctx-II] enable the semantics to capture the same ideas of safe reorderings which are already present in the existing precise asynchronous subtyping relation. We can safely execute actions pre-emptively exactly when the new behaviour corresponds to a SISO tree which is a refinement of a top level behaviour.

Definition 5 (Balanced$^+$ Global Types). *A global type G is balanced$^+$ iff, for every type G', G'' such that $G \to^* G' \to^* G''$, where $G'' = \mathtt{q}\to\mathtt{r}\colon\{\mathtt{m}_i(B_i).G_i'''\}_{i\in I}$ (or $\mathtt{q}\overset{\mathtt{m}_k}{\rightsquigarrow}\mathtt{r}\colon\{\mathtt{m}_i(B_i).G_i'''\}_{i\in I}$) and for each of the roles $\mathtt{p}\in\{\mathtt{q},\mathtt{r}\}$, there exists a $k\geq 0$ such that all fair paths $G'\to G_1'\to G_2'\to\ldots$ reach a type $G''=\mathtt{s}\to\mathtt{t}\colon\{\mathtt{m}_j(B_j).G_j'''\}_{j\in J}$ (or $\mathtt{q}\overset{\mathtt{m}_k}{\rightsquigarrow}\mathtt{r}\colon\{\mathtt{m}_i(B_i).G_i'''\}_{i\in I}$) in at most k steps with $\mathtt{p}\in\{\mathtt{s},\mathtt{t}\}$. As is standard when defining projectable types, we assume well-formed global types satisfy this condition. For types* without *en-route transmisions, this aligns with the normal definition of balanced types (Def 3.3 in [12] and Def 4.17 in [21]). Given en-route types are only runtime behaviour, we also restrict ourselves to global types G' which are the result of running a global type $G\to^* G'$ where G does not contain en-route transmissions.*

Example 2 (Semantics of Global Type for Ring Protocol). Consider the global type for the ring-choice protocol (§1). The asynchronous semantics enable us to apply both [GR-⊕] reductions, corresponding to sends from p to q and from q to r, before any receive reductions (using [GR-&]) are applied. As we will see later, this particular choice of global reduction path corresponds to behaviour which can only be captured by the optimised local type.

We begin by reducing G_{ring} via a send action from p to q:

$$G_{\text{ring}} \xrightarrow{\mathtt{p}:\mathtt{q}\oplus\mathsf{add}} G^{(1)}_{\text{ring}} = \mathtt{p}\overset{\mathsf{add}}{\rightsquigarrow}\mathtt{q}\colon\mathsf{add}(\mathsf{int}).\mathtt{q}\to\mathtt{r}\colon\left\{\begin{array}{l}\mathsf{add}(\mathsf{int})\,.\,\mathtt{r}\to\mathtt{p}\colon\{\mathsf{add}(\mathsf{int})\,.\,G_{\text{ring}}\}\\ \mathsf{sub}(\mathsf{int})\,.\,\mathtt{r}\to\mathtt{p}\colon\{\mathsf{sub}(\mathsf{int})\,.\,G_{\text{ring}}\}\end{array}\right\}$$

At this point, a message from p to q is in transit. We then perform another [GR-⊕] reduction, using [GR-Ctx-II] to apply the sending from q to r under the existing en-route type:

$$G^{(1)}_{\text{ring}} \xrightarrow{\mathtt{q}:\mathtt{r}\oplus\mathsf{sub}} G^{(2)}_{\text{ring}} = \mathtt{p}\overset{\mathsf{add}}{\rightsquigarrow}\mathtt{q}\colon\mathsf{add}(\mathsf{int}).\mathtt{q}\overset{\mathsf{sub}}{\rightsquigarrow}\mathtt{r}\colon\left\{\begin{array}{l}\mathsf{add}(\mathsf{int})\,.\,\mathtt{r}\to\mathtt{p}\colon\{\mathsf{add}(\mathsf{int})\,.\,G_{\text{ring}}\}\\ \mathsf{sub}(\mathsf{int})\,.\,\mathtt{r}\to\mathtt{p}\colon\{\mathsf{sub}(\mathsf{int})\,.\,G_{\text{ring}}\}\end{array}\right\}$$

The state $G^{(2)}_{\text{ring}}$ reflects the two en-route messages: one from p to q and one from q to r. We can the proceed with the corresponding receive actions using the [GR-&] rule. First, q receives the message from p:

$$G^{(2)}_{\text{ring}} \xrightarrow{\mathtt{q}:\mathtt{p}\&\mathsf{add}} G^{(3)}_{\text{ring}} = \mathtt{q}\overset{\mathsf{sub}}{\rightsquigarrow}\mathtt{r}\colon\left\{\begin{array}{l}\mathsf{add}(\mathsf{int})\,.\,\mathtt{r}\to\mathtt{p}\colon\{\mathsf{add}(\mathsf{int})\,.\,G_{\text{ring}}\}\\ \mathsf{sub}(\mathsf{int})\,.\,\mathtt{r}\to\mathtt{p}\colon\{\mathsf{sub}(\mathsf{int})\,.\,G_{\text{ring}}\}\end{array}\right\}$$

$$\frac{k \in I}{\mathtt{p}{:}(\sigma, \mathtt{q}\oplus\{\mathtt{m_i}(B_i).T_i\}_{i\in I}) \xrightarrow{\mathtt{p:q}\oplus\mathtt{m_k}(\mathtt{B_k})} \mathtt{p}{:}(\sigma\cdot(\mathtt{q},\mathtt{m}_k(B_k)), T_k)}\ [\Delta\text{-}\oplus]$$

$$\frac{k \in I}{\mathtt{p}{:}(\sigma, \mathtt{q}\&\{\mathtt{m_i}(B_i).T_i\}_{i\in I}), \mathtt{q}{:}(\sigma'\cdot(\mathtt{p},\mathtt{m}_k(B_k)), T) \xrightarrow{\mathtt{p:q}\&\mathtt{m_k}(\mathtt{B_k})} \mathtt{p}{:}(\sigma, T_k), \mathtt{q}{:}(\sigma', T)}\ [\Delta\text{-}\&]$$

$$\frac{\mathtt{p}{:}T\{^{\mu\mathtt{t}.T}/_{\mathtt{t}}\} \xrightarrow{\alpha} \Delta'}{\mathtt{p}{:}\mu\mathtt{t}.T \xrightarrow{\alpha} \Delta'}\ [\Delta\text{-}\mu] \qquad \frac{\Delta \xrightarrow{\alpha} \Delta'}{\Delta, x{:}B \xrightarrow{\alpha} \Delta', x{:}B}\ [\Delta\text{-},B] \qquad \frac{\Delta \xrightarrow{\alpha} \Delta'}{\Delta, c{:}T \xrightarrow{\alpha} \Delta', c{:}T}\ [\Delta\text{-},]$$

Fig. 6. Typing context reduction rules.

Then, r receives the message from q:

$$G^{(3)}_{\text{ring}} \xrightarrow{\mathtt{r:q}\&\mathsf{sub}} G^{(4)}_{\text{ring}} = \mathtt{r}{\to}\mathtt{p}{:}\,\{\mathsf{sub}(\mathsf{int})\,.\,G_{\text{ring}}\}$$

Next, r sends to p:

$$G^{(4)}_{\text{ring}} \xrightarrow{\mathtt{r:p}\oplus\mathsf{sub}} G^{(5)}_{\text{ring}} = \mathtt{r}\overset{\mathsf{sub}}{\rightsquigarrow}\mathtt{p}{:}\,\{\mathsf{sub}(\mathsf{int})\,.\,G_{\text{ring}}\}$$

Finally, p receives this last message, returning us to the original state of the protocol:

$$G^{(5)}_{\text{ring}} \xrightarrow{\mathtt{r:p}\&\mathsf{sub}} G_{\text{ring}}$$

In §3.2, we will show that this reduction sequence corresponds to a behaviour of the optimised local implementation for q.

3.2 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 Def. 6, followed by its reduction rules in Def. 7.

Definition 6 (Typing Contexts). *Δ denotes a partial mapping from participants to queues and types. Their syntax is defined as:*

$$\Delta ::= \emptyset \mid \Delta, \mathtt{p}{:}(\sigma, T)$$

The context composition *Δ_1, Δ_2 is defined iff* $\mathrm{dom}(\Delta_1) \cap \mathrm{dom}(\Delta_2) = \emptyset$.

Definition 7 (Typing Context Reduction). *The* typing context transition *$\xrightarrow{\alpha}$ is inductively defined by the rules in Fig. 6. We write $\Delta \xrightarrow{\alpha}$ if there exists Δ' such that $\Delta \xrightarrow{\alpha} \Delta'$. We write $\Delta \to \Delta'$ iff $\Delta \xrightarrow{\alpha} \Delta'$ for some α and $\Delta \not\to$ for its negation (i.e. there is no Δ' such that $\Delta \to \Delta'$), and we denote $\to^*$ as the reflexive and transitive closure of $\to$.*

Example 3 (Operational Semantics of Optimised Ring Context). As an example, consider the operational semantics of the optimised ring protocol. Each transition captures either a message send or receive, which either enqueues or dequeues a message in the queue of the sending participant.

$$
\begin{aligned}
&\Delta_0 = \mathtt{p}{:}(\varnothing, T_\mathtt{p}), \mathtt{q}{:}(\varnothing, T_\mathtt{q}^{\mathrm{opt}}), \mathtt{r}{:}(\varnothing, T_\mathtt{r}) \\
&\xrightarrow{\mathtt{p}:\mathtt{q}\oplus\mathtt{add}(\mathrm{int})} \Delta_1 = \mathtt{p}{:}(\langle(\mathtt{q}, \mathtt{add}(\mathrm{int}))\rangle, \mathtt{r}\&\left\{\begin{matrix}\mathtt{add}(\mathrm{int}).T_\mathtt{p} \\ \mathtt{sub}(\mathrm{int}).T_\mathtt{p}\end{matrix}\right\}), \mathtt{q}{:}(\varnothing, T_\mathtt{q}^{\mathrm{opt}}), \mathtt{r}{:}(\varnothing, T_\mathtt{r}) \\
&\xrightarrow{\mathtt{q}:\mathtt{r}\oplus\mathtt{sub}(\mathrm{int})} \Delta_2 = \mathtt{p}{:}(\langle(\mathtt{q}, \mathtt{add}(\mathrm{int}))\rangle, \mathtt{r}\&\left\{\begin{matrix}\mathtt{add}(\mathrm{int}).T_\mathtt{p} \\ \mathtt{sub}(\mathrm{int}).T_\mathtt{p}\end{matrix}\right\}), \\
&\qquad\qquad \mathtt{q}{:}(\langle(\mathtt{r}, \mathtt{sub}(\mathrm{int}))\rangle, \mathtt{p}\&\{\mathtt{add}(\mathrm{int}).T_\mathtt{q}^{\mathrm{opt}}\}), \mathtt{r}{:}(\varnothing, T_\mathtt{r}) \\
&\xrightarrow{\mathtt{q}:\mathtt{p}\&\mathtt{add}(\mathrm{int})} \Delta_3 = \mathtt{p}{:}(\varnothing, \mathtt{r}\&\left\{\begin{matrix}\mathtt{add}(\mathrm{int}).T_\mathtt{p} \\ \mathtt{sub}(\mathrm{int}).T_\mathtt{p}\end{matrix}\right\}), \mathtt{q}{:}(\langle(\mathtt{r}, \mathtt{sub}(\mathrm{int}))\rangle, T_\mathtt{q}^{\mathrm{opt}}), \mathtt{r}{:}(\varnothing, T_\mathtt{r}) \\
&\xrightarrow{\mathtt{r}:\mathtt{q}\&\mathtt{sub}(\mathrm{int})} \Delta_4 = \mathtt{p}{:}(\varnothing, \mathtt{r}\&\left\{\begin{matrix}\mathtt{add}(\mathrm{int}).T_\mathtt{p} \\ \mathtt{sub}(\mathrm{int}).T_\mathtt{p}\end{matrix}\right\}), \mathtt{q}{:}(\varnothing, T_\mathtt{q}^{\mathrm{opt}}), \mathtt{r}{:}(\varnothing, \mathtt{p}\oplus\{\mathtt{sub}(\mathrm{int}).T_\mathtt{r}\}) \\
&\xrightarrow{\mathtt{r}:\mathtt{p}\oplus\mathtt{sub}(\mathrm{int})} \Delta_5 = \mathtt{p}{:}(\varnothing, \mathtt{r}\&\left\{\begin{matrix}\mathtt{add}(\mathrm{int}).T_\mathtt{p} \\ \mathtt{sub}(\mathrm{int}).T_\mathtt{p}\end{matrix}\right\}), \mathtt{q}{:}(\varnothing, T_\mathtt{q}^{\mathrm{opt}}), \mathtt{r}{:}(\langle(\mathtt{p}, \mathtt{sub}(\mathrm{int}))\rangle, T_\mathtt{r}) \\
&\xrightarrow{\mathtt{p}:\mathtt{r}\&\mathtt{sub}(\mathrm{int})} \Delta_0
\end{aligned}
$$

4 Global and Local Type Asynchronous Association

Following the introduction of LTS semantics for global types (Def. 4) and typing contexts (Def. 7), we establish a relationship between these two semantics using the projection relation $\upharpoonright_\mathtt{p}$ (Def. 1) and the subtyping relation $\leqslant_a$ (Def. 2).

Definition 8 (Association of Global Types and Typing Contexts). *A typing context Δ is associated with a global type G written $\Delta \sqsubseteq_a G$, iff Δ can be split into two disjoint (possibly empty) sub-contexts $\Delta = \Delta_G, \Delta_\mathbf{end}$ where:*

1. *Δ_G contains projections of G:* $\mathrm{dom}(\Delta_G) = \mathrm{roles}(G)$, *and* $\forall \mathtt{p} \in \mathrm{roles}(G) : \Delta(\mathtt{p}) = (\sigma_\mathtt{p}, T'_\mathtt{p})$ *and* $\exists T_\mathtt{p} : T'_\mathtt{p} \leqslant_a T_\mathtt{p}$ *and* $G\upharpoonright_\mathtt{p}(\sigma_\mathtt{p}, T_\mathtt{p})$*;*
2. *$\Delta_\mathbf{end}$ contains only end endpoints:* $\forall \mathtt{p} \in \mathrm{dom}(\Delta_\mathbf{end}) : \Delta(\mathtt{p}) = (\varnothing, \mathbf{end})$.

The association $\cdot \sqsubseteq_a \cdot$ is a binary relation over typing contexts Δ and global types G. There are two requirements for the association: (1) the typing context Δ must include an entry for each role; and (2) for each role $\mathtt{p}$, its corresponding entry in the typing context ($\Delta(\mathtt{p})$) must be a subtype (Def. 2) of the projection of the global type onto this role.

Looking again at the ring protocol example, we can observe how the reduction of the global type corresponds to updates in the local context. This forms an *operational correspondence* between the global semantics and local process configurations. Each global step is matched by a change in the local context.

$$\begin{array}{ccccccccccc} G_{\text{ring}} & \xrightarrow{\text{p:q}\oplus\text{add}} & G^{(1)}_{\text{ring}} & \xrightarrow{\text{q:r}\oplus\text{sub}} & G^{(2)}_{\text{ring}} & \xrightarrow{\text{q:p\&add}} & G^{(3)}_{\text{ring}} & \xrightarrow{\text{r:q\&sub}} & G^{(4)}_{\text{ring}} & \xrightarrow{\text{r:p}\oplus\text{sub}} & G^{(5)}_{\text{ring}} \\ \sqsubseteq_a & & \sqsubseteq_a & & \sqsubseteq_a & & \sqsubseteq_a & & \sqsubseteq_a & & \sqsubseteq_a \\ \Delta_0 & \xrightarrow{\text{p:q}\oplus\text{add}} & \Delta_1 & \xrightarrow{\text{q:r}\oplus\text{sub}} & \Delta_2 & \xrightarrow{\text{q:p\&add}} & \Delta_3 & \xrightarrow{\text{r:q\&sub}} & \Delta_4 & \xrightarrow{\text{r:p}\oplus\text{sub}} & \Delta_5 \end{array}$$

This idea is illustrated through two main theorems: Thm. 2 shows that the reducibility of a global type aligns with that of its associated typing context; while Thm. 1 illustrates that each possible reduction of a typing context is simulated by an action in the reductions of the associated global type.

Theorem 1 (Completeness of Association). *Given associated global type G and typing context Δ such that $\Delta \sqsubseteq_a G$. If $\Delta \xrightarrow{\alpha} \Delta'$, then there exists G' such that $\Delta' \sqsubseteq_a G'$ and $G \xrightarrow{\alpha} G'$.*

Proof (Sketch). By case analysis on α. For each type of action we consider the possible structure of G permitted by the $\leqslant_{\text{a}}$ relation and find that it must be able to take a corresponding step. See [18] for detailed proofs.

Theorem 2 (Soundness of Association). *Let $\Delta \sqsubseteq_a G$ and assume $G \xrightarrow{\alpha} G'$. Then there exist an action α', a context Δ', and a global type G'' such that*

$$G \xrightarrow{\alpha'} G'', \qquad \Delta \xrightarrow{\alpha'} \Delta', \qquad \Delta' \sqsubseteq_a G''.$$

Proof (Sketch). By induction on the transition $G \xrightarrow{\alpha} G'$. We again consider the possible structure of G permitted by $\leqslant_{\text{a}}$ and conclude that Δ can take a step. We can then use Thm 1 to find a corresponding global type transition which preserves association. See [18] for detailed proofs.

5 Deriving the Main Theorems from Associations

This section demonstrates how to derive the main theorems using soundness and completeness of the associations, together with the corresponding results in [13, Theorems 4.11, 4.12 and 4.13]. Before that, we recall the bottom-up typing system for a multiparty session:

$$\frac{\forall \mathtt{p} \in \text{dom}(\Delta) \quad \vdash P_{\mathtt{p}} \triangleright T_{\mathtt{p}} \quad \vdash h_{\mathtt{p}} \triangleright \sigma_{\mathtt{p}} \quad \Delta(\mathtt{p}) = (\sigma_{\mathtt{p}}, T_{\mathtt{p}}) \quad \varphi(\Delta)}{\vdash^{\text{bot}} \Pi_{\mathtt{p} \in \text{dom}(\Delta)} \ (\mathtt{p} \triangleleft P_{\mathtt{p}} \mid \mathtt{p} \triangleleft h_{\mathtt{p}}) \ \triangleright \Delta} \ [\textsc{SessBot}]$$

where φ is some desired property, which is usually a *safety* property–a selected label is always available at the branching process [19,23]. In [13], a *liveness* property [13, Definition 4.17] is used instead for proving the preciseness of $\leqslant_{\text{a}}$. See [13, § 7.1].

Deriving Subject Reduction Theorem. We prove the subject reduction theorem of the top-down system using the completeness of the association with the following subject reduction theorem of the bottom-up system. We define *asynchronous multiparty session* $(M, M_i, ...)$ as: $M ::= \mathtt{p} \triangleleft P_{\mathtt{p}} \mid \mathtt{p} \triangleleft h_{\mathtt{p}} \mid M \mid M'$.

Theorem 3 (Subject Reduction, Theorem 4.11 [13]). *Assume* $\vdash^{bot} M \triangleright \Delta$ *with* Δ *live and* $M \to^* M'$. *Then there exist live* Δ', Δ'' *such that* $\vdash^{bot} M' \triangleright \Delta''$ *with* $\Delta' \leqslant_a \Delta$ *and* $\Delta' \to^* \Delta''$.

Theorem 4 (Subject Reduction of the Top-Down System). *Assume* $\vdash^{top} M \triangleright \Delta$ *and* $M \to^* M'$. *Then there exists* Δ *such that* $\vdash^{bot} M' \triangleright \Delta'$ *with* $\Delta \to^* \Delta'$.

Proof. Assume $M \equiv \Pi_{\mathtt{p} \in \mathrm{dom}(\Delta)} (\mathtt{p} \triangleleft P_{\mathtt{p}} \mid \mathtt{p} \triangleleft h_{\mathtt{p}})$ and $\vdash^{\mathrm{top}} M \triangleright \Delta$ is derived with

$$\forall \mathtt{p} \in \mathrm{dom}(\Delta) \quad \vdash P_{\mathtt{p}} \triangleright T_{\mathtt{p}} \quad \vdash h_{\mathtt{p}} \triangleright \sigma_{\mathtt{p}} \quad \Delta(\mathtt{p}) = (\sigma_{\mathtt{p}}, T_{\mathtt{p}}) \quad \Delta \sqsubseteq_a G \tag{2}$$

by [SessST]. Suppose $M \to M'$. We need to prove that there exist G' and Δ' such that $\Pi_{\mathtt{p} \in \mathrm{role}(G')} (\mathtt{p} \triangleleft P'_{\mathtt{p}} \mid \mathtt{p} \triangleleft h'_{\mathtt{p}})$ with $\Delta' \sqsubseteq_a G'$.

Note that Δ is live by [19,23]. Hence by Theorem 3, there exist live Δ', Δ'' such that $\vdash^{\mathrm{bot}} \Pi_{\mathtt{p} \in \mathrm{role}(G)} P'_{\mathtt{p}} \triangleright \Delta''$ with $\Delta' \leqslant_a \Delta$ and $\Delta' \to^* \Delta''$. By Definition 8, $\Delta' \sqsubseteq_a G$. Then by Theorem 1, $\Delta' \to^* \Delta''$ implies $G \to^* G'$ and $\Delta'' \sqsubseteq_a G'$. Hence $\vdash^{\mathrm{top}} \Pi_{\mathtt{p} \in \mathrm{dom}(\Delta'')} P'_{\mathtt{p}} \triangleright \Delta''$ as desired.

Deriving Session Fidelity. We derive session fidelity of the top-down system. We use the soundness and completeness of the association with session fidelity of the bottom-up system

Theorem 5 (Session Fidelity, Theorem 4.13 [13]). *Assume* $\vdash^{bot} M \triangleright \Delta$ *with* Δ *live. Assume* $\Delta \to$. *Then there exist* M' *and* Δ' *such that* $M \to^+ M'$, $\Delta \to \Delta'$ *and* $\vdash^{bot} M' \triangleright \Delta'$.

Theorem 6 (Session Fidelity of the Top-Down System). *Assume* $\vdash^{top} M \triangleright \Delta$ *is derived by* $\Delta \sqsubseteq_a G$ *and* $G \to$. *Then there exist* M' *and* Δ' *such that* $M \to^+ M'$, $G \to G'$ *and* $\vdash^{top} M' \triangleright \Delta'$ *with* $\Delta' \sqsubseteq_a G'$.

Proof. Assume $\Delta \sqsubseteq_a G$. By the soundness of the association, $G \to$ implies $\Delta \to$. Suppose $M \equiv \Pi_{\mathtt{p} \in \mathrm{dom}(\Delta)} (\mathtt{p} \triangleleft P_{\mathtt{p}} \mid \mathtt{p} \triangleleft h_{\mathtt{p}})$ and $\vdash^{\mathrm{top}} M' \triangleright \Delta$ is derived with (2) above. By Theorem 5, there exist M' and Δ' such that $M \to^+ M'$ and $\Delta \to \Delta'$. Hence by the completeness of the association, and Theorem 4, $G \to G'$ and $\Delta' \sqsubseteq_a G'$ with $\vdash^{\mathrm{top}} M' \triangleright \Delta'$, as desired.

Next we show that typed multiparty sessions are live (defined in [13, § 2.3]).

Theorem 7 (Liveness of the Top-Down System). *Assume* $\vdash^{top} M \triangleright \Delta$. *Then for all* M' *such that* $M \to^* M'$, M' *is safe, deadlock-free and live.*

Proof. We first note that if M is live, then M is safe and deadlock-free. If $\Delta \sqsubseteq_a G$, then Δ is live, hence we have $\vdash^{\mathrm{bot}} M \triangleright \Delta$. Then by Theorem 4.12 in [13], M is live.

6 Conclusion

We have proposed an asynchronous association relation and proved its sound and complete operational correspondence. This work is the first to prove these results based on (1) asynchronous precise subtyping and (2) projection with co-inductive full merging. We introduced a new operational semantics for global types, which captures more behaviours allowed by permuting actions than the previous asynchronous global type semantics in [2,16]. We developed a new projection relation which associates global types with a pair of a local type and a queue type for each participant. Using this correspondence, we derived the subject reduction theorem and the session fidelity theorem of the top-down system from the corresponding theorems of the bottom-up system [13, Theorem 4.11 and 4.13]. Since the projection Δ of G is known to be safe, deadlock-free and live [19,24], we can derive that asynchronous multiparty session processes typed by the top-down typing system ([SessTop]) are also safe, deadlock-free and live (Theorem 7). While [13] has proved the subject reduction theorem and session fidelity theorem under the subsumption rule of $\leqslant_{\mathrm{a}}$, it does not use the top-down typing system. On the other hand, [12] has shown that multiparty synchronous subtyping is precise in the synchronous multiparty session calculus using the top-down system. None of the previous work has defined association with respect to asynchronous subtyping or co-inductive projection. An interesting open question is whether the association theorems hold for the sound decidable asynchronous subtyping relations [7,10] (and [4,5] by extending binary to multiparty session types) so that we can derive the subject reduction theorems under those relations.

We have demonstrated the usefulness of association in deriving the main theorems of the top-down system, by *reusing* the theorems in [13]. We have not yet reached a stage to claim that MPST is a theoretical framework for building component-based software systems as Jean-Bernard Stefani has defined. There still needs to be more effort applied to developing practical applications of MPST for testing and maintaining compositionality and reusability of protocols. The most challenging topic is to type individual *components*, each being written in a different programming language or running on a different platform, while ensuring their type-safety and deadlock-freedom, assuming they conform to a shared global protocol. Implementing such a component-based architecture requires significant engineering effort such as defining system requirements, identifying components, splitting the system into components, integrating these components, and designing the interfaces for components. We plan to conduct a serious study along these lines in the near future to make MPST a true *theory of CBSE*.

References

1. Barwell, A.D., Scalas, A., Yoshida, N., Zhou, F.: Generalised Multiparty Session Types with Crash-Stop Failures Technical Report (2022)
2. Barwell, A.D., Scalas, A., Yoshida, N., Zhou, F.: Generalised multiparty session types with crash-stop failures (2022). To appear in LMCS

3. Blair, G., Coupaye, T., Stefani, J.-B.: Component-based architecture: the fractal initiative. Ann. Telecommun. **64**(1–2), 1–4 (2009)
4. Bocchi, L., King, A., Murgia, M.: Asynchronous subtyping by trace relaxation. In: Finkbeiner, B., Kovács, L. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 30th International Conference, TACAS 2024, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2024, Luxembourg City, Luxembourg, 6–11 April 2024, Proceedings, Part I, vol. 14570 of Lecture Notes in Computer Science, pp. 207–226. Springer, Heidelberg (2024)
5. Bravetti, M., Carbone, M., Lange, J., Yoshida, N., Zavattaro, G.: A sound algorithm for asynchronous session subtyping and its implementation. Log. Methods Comput. Sci. **17**(1) (2021)
6. Bruneton, E., Coupaye, T., Leclercq, M., Quéma, V., Stefani, J.B.: The fractal component model and its support in java. Softw. Pract. Exp. **36**(11–12), 1257–1284 (2006)
7. Castro-Perez, D., Yoshida, N.: CAMP: cost-aware multiparty session protocols. Proc. ACM Program. Lang. **4**(OOPSLA), 155:1–155:30 (2020)
8. Chen, T.C., Dezani-Ciancaglini, M., Yoshida, N.: On the preciseness of subtyping in session types: 10 years later. In: Proceedings of the 26th International Symposium on Principles and Practice of Declarative Programming, PPDP '24. Association for Computing Machinery, New York (2024)
9. Cutner, Z., Yoshida, N.: Safe session-based asynchronous coordination in rust. In: Damiani, F., Dardha, O. (eds.) COORDINATION 2021. LNCS, vol. 12717, pp. 80–89. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-78142-2_5
10. Cutner, Z., Yoshida, N., Vassor, M.: Deadlock-free asynchronous message reordering in rust with multiparty session types. In: 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP '22, pp. 261–246. ACM (2022). arxiv:2112.12693
11. Fassino, J.P., Stefani, J.B., Lawall, J.L., Muller, G.: Think: a software framework for component-based operating system kernels. In: Proceedings of the General Track of the Annual Conference on USENIX Annual Technical Conference, ATEC'02, pp. 73–86. USENIX Association (2002)
12. Ghilezan, S., Jakšić, S., Pantović, J., Scalas, A., Yoshida, N.: Precise subtyping for synchronous multiparty sessions. J. Logical Algebr. Methods Program. **104**, 127–173 (2019)
13. Ghilezan, S., Pantović, J., Prokić, I., Scalas, A., Yoshida, N.: Precise subtyping for asynchronous multiparty sessions. ACM Trans. Comput. Logic **24**(2), 1–73 (2023)
14. Honda, K., Vasconcelos, V.T., Kubo, M.: Language primitives and type disciplines for structured communication-based programming. In: Proceedings of ESOP 1998, vol. 1381 of LNCS, pp. 22–138. Springer, Heidelberg (1998)
15. Honda, K., Yoshida, N., Carbone, M.: Multiparty asynchronous session types. In: Proceedings of POPL 2008, pp. 273–284. ACM (2008)
16. Honda, K., Yoshida, N., Carbone, M.: Multiparty asynchronous session types. J. ACM **63**(1), 9:1–9:67 (2016)
17. Pierce, B.: Types and Programming Languages. MIT Press, Cambridge (2002)
18. Pischke, K., Yoshida, N.: Asynchronous global protocols, precisely: Full proofs. https://arxiv.org/abs/2505.17676 (2025)
19. Scalas, A., Yoshida, N.: Less is more: multiparty session types revisited. Proc. ACM Program. Lang. **3**(POPL), 1–29 (2019)

20. Takeuchi, K., Honda, K., Kubo, M.: An interaction-based language and its typing system. In: Halatsis, C., Maritsas, D., Philokyprou, G., Theodoridis, S. (eds.) PARLE 1994. LNCS, vol. 817, pp. 398–413. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-58184-7_118
21. Udomsrirungruang, T., Yoshida, N.: Top-down or bottom-up? Complexity analyses of synchronous multiparty session types. Proc. ACM Program. Lang. **9**(POPL), 1040–1071 (2025)
22. Yoshida, N.: Programming language implementations with multiparty session types. In: de Boer, F.S., Damiani, F., Hähnle, R., Johnsen, E.B., Kamburjan, E. (eds.) Active Object Languages: Current Research Trends, vol. 14360 of Lecture Notes in Computer Science, pp. 147–165. Springer, Heidelberg (2024)
23. Yoshida, N., Hou, P.: Less is More Revisit (2024)
24. Yoshida, N., Hou, P.: Less is more revisited, 2024. Accepted by Cliff B. Jones Festschrift Proceeding (2004)
25. Yoshida, N., Hu, R., Neykova, R., Ng, N.: The scribble protocol language. In: Abadi, M., Lluch Lafuente, A. (eds.) TGC 2013. LNCS, vol. 8358, pp. 22–41. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-05119-2_3
26. Yoshida, N., Zhou, F., Ferreira, F.: Communicating finite state machines and an extensible toolchain for multiparty session types. In: Bampis, E., Pagourtzis, A. (eds.) FCT 2021. LNCS, vol. 12867, pp. 18–35. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-86593-1_2