

ARTICLE INFO

Keywords:

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

ABSTRACT

Asynchronous multiparty session types are a type-based framework which ensure the compatibility of components in a distributed system by checking compliance against a specified global protocol. We propose a *top-down* approach, starting with the global protocol which is then projected into a set of local specifications. Next, we use an asynchronous refinement relation, *precise asynchronous multiparty subtyping*, to enable local specifications to be optimised by permuting actions within individual asynchronous components. This supports local reasoning, as each component can be independently developed and *refined* in isolation, before being integrated into a larger system. We show that this methodology guarantees both *type soundness* and *liveness* of the collection of optimised components.

In this article, we first propose new operational semantics of global protocols which capture sound optimisations in the context of asynchronous message-passing. Next we define an *asynchronous association* between global protocols and a set of optimised local types. Thirdly, we prove, for the first time, the correctness of the most expressive endpoint projection in the literature, *coinductive full merging projection*. We then show the main theorems of this article: *soundness* and *completeness* of the operational correspondence of the asynchronous association. As a consequence, the association acts as an *invariant* that can be used to transfer key theorems from the bottom-up system to the top-down system. In particular, we used this to prove type soundness, session-fidelity, deadlock-freedom and liveness of the collection of optimised endpoints.

1. Introduction

1.1. Background

Multiparty Session Types (MPST). *Session types* [32, 54] are a type discipline for codifying concurrent components. They give an abstract view of *sessions*, which are structured communications between distributed peers. Session types were first proposed in the context of *binary sessions*, such as client-server protocols or dyadic interactions between two communicating agents. *Multiparty session types* [34, 35] (MPST) extend from two-party to multiparty communications, allowing the programmer to specify a *global protocol* to coordinate communicating components. Using MPST, we can ensure that typed components interact without type errors or deadlocks, entirely *by construction*. The MPST framework is *language-agnostic*, and has been adapted into over 30 programming languages [57].

In MPST, we first start from specifying a global protocol (called a *global type*) which we then project into a set of local protocols (local types). Each local type can then be *refined* to another, compatible local type. Once each endpoint program is type-checked against its refined local type, the set of distributed programs will be guaranteed to be interacting correctly, without type errors or deadlocks.

There are a number of methods to refine or optimise local protocols. In this article, we focus on the **precise multiparty asynchronous subtyping relation** $\leq_a$ [29], as a refinement relation.

Precise Multiparty Asynchronous Subtyping. *Subtyping* enhances the expressiveness of typed programs. The original synchronous subtyping [19, 27] has been extended to accommodate *asynchronous communications* [13], allowing the *anticipation* of *output actions* for message-passing optimisations. We say that a subtyping is *precise* if it is both sound and complete—a subtyping relation is *sound* if no typable program has a type or communication error. It is *complete* if there is no strictly larger sound subtyping relation. It is proven that the synchronous subtyping relation is precise in both binary [13] and multiparty sessions [28].

*Work supported by: EPSRC EP/T006544/2, EP/T014709/2, EP/Z533749/1, ARIA and Horizon EU TaRDIS 101093006 (UKRI number 10066667).

✉ kai.pischke@cs.ox.ac.uk (K. Pischke); jake.masters@cs.ox.ac.uk (J. Masters); nobuko.yoshida@cs.ox.ac.uk (N. Yoshida)

ORCID(s): 0000-0002-6435-9456 (K. Pischke); 0000-0002-3783-6149 (J. Masters); 0000-0002-3925-8557 (N. Yoshida)

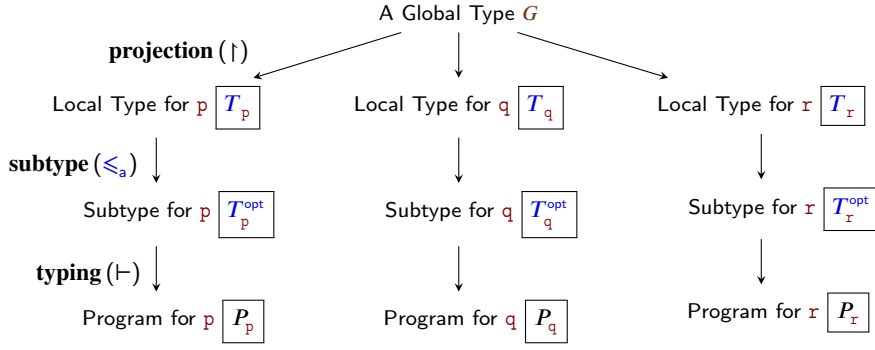


Figure 1: Top-down methodology of multiparty session types. G denotes a global type, which is projected into the three participants, p , q and r , generating local types T_p , T_q and T_r for each participant. Local types are then refined to T_p^{opt} , T_q^{opt} and T_r^{opt} . Three distributed programs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

In this article, we consider *asynchronous communication* which is modelled using infinite FIFO queues: output processes put messages in queues, while input processes read messages from these queues. Hence messages are ordered but non-blocking. In asynchronous FIFO semantics, the types should ensure not only deadlock-freedom, i.e., that input processes will always find messages, but also *liveness*, i.e., that all messages in queues will eventually be consumed.

Chen et al. [13] have proposed the *asynchronous subtyping relation*, where soundness is formulated as ensuring asynchronous liveness. The most technically involved but practically useful subtyping system in this line of research is proposed by Ghilezan et al. [29], which presents the first formalisation of the precise subtyping relation for asynchronous MPST. The relation is defined based on a *session decomposition* technique, from full session types (including internal/external choices) into *single input/output session trees* (without choices). This session decomposition expresses the subtyping relation as a composition of refinement relations between single input/output trees and provides a reasoning principle for optimising the order between asynchronous messages. See [14] for a recent survey on precise subtyping.

Figure 1 describes the MPST workflow. We assume a set of participants (or roles) $\mathcal{R}$ in the distributed system. We specify programs for each participant $\{P_p\}_{p \in \mathcal{R}}$ and a *global protocol (type)* G . To type-check the session $\Pi_{p \in \mathcal{R}} P_p$ (which denotes a parallel composition of P_p) against G , we project G onto a set of *local protocols (types)* $\{T_p\}_{p \in \mathcal{R}}$ from the viewpoint of each participant p , and synthesise *local protocols (types)* $\{T_p^{opt}\}_{p \in \mathcal{R}}$ from each program P_p . We say that G types $\Pi_{p \in \mathcal{R}} P_p$ iff T_p^{opt} is a subtype of T_p i.e. T_p can be refined or optimised to T_p^{opt} .

Asynchronous multiparty subtyping $\leq_a$ allows for “safe permutations” of actions, 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 $\{P_p\}_{p \in \mathcal{R}}$ satisfy safety, deadlock-freedom and liveness.

This top-down workflow is implemented by the MPST toolchains, SCRIBBLE [58] and vSCR [60], which check whether a given global protocol is well-formed, and if so, generate a corresponding set of local types. Building on SCRIBBLE and vSCR, several toolkits have been implemented. For instance, the Rust toolchain RUMPSTEAK [18] uses vSCR to generate state machines, from which *optimised APIs* are generated using a sound approximation of $\leq_a$. We shall discuss the detailed related work on theories and implementations of multiparty asynchronous subtyping relations in § 7. In the next subsection, we explain how we use the asynchronous subtyping relation for communication optimisations, introducing a running example.

1.2. Ring-Choice Example

We explain our workflow by introducing a running example which will be referenced throughout this article, the ring-choice protocol G_{ring} from [17]:

$$G_{\text{ring}} = \mu t. p \rightarrow q : \text{add}(\text{int}). q \rightarrow r : \left\{ \begin{array}{l} \text{add}(\text{int}). r \rightarrow p : \{ \text{add}(\text{int}). t \} \\ \text{sub}(\text{int}). r \rightarrow p : \{ \text{sub}(\text{int}). t \} \end{array} \right\} \quad (1)$$

The global type G_{ring} specifies that:

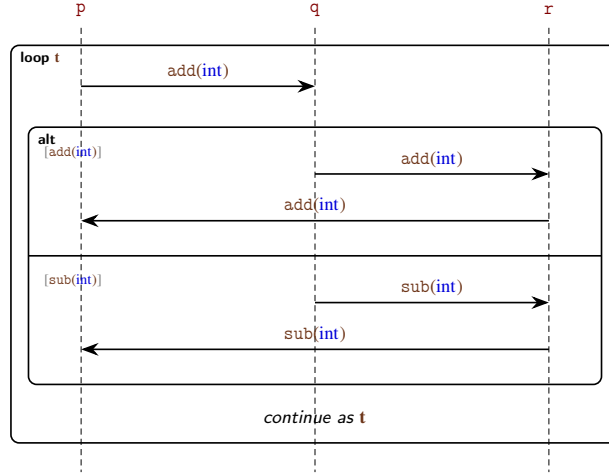
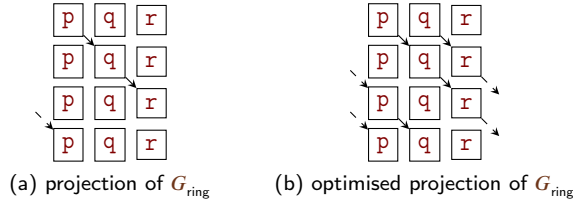

 Figure 2: Message sequence chart for G_{ring} (one unfolding of μt).


Figure 3: Ring protocol: Projected and optimised interactions (from [17])

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.

Global types are semantically similar to more traditional protocol descriptions such as *message sequence charts* (MSCs) [1, 30]. In Figure 2 we provide a MSC for the ring protocol, corresponding to G_{ring} , showing the protocol visually.

If we assume *synchronous interactions* as illustrated in Figure 3(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 :

$$T_p = \mu t. q \oplus \{ \text{add}(\text{int}). r \& \{ \text{add}(\text{int}). t, \text{sub}(\text{int}). t \} \} \quad (2)$$

$$T_q = \mu t. p \& \{ \text{add}(\text{int}). r \oplus \{ \text{add}(\text{int}). t, \text{sub}(\text{int}). t \} \} \quad (3)$$

$$T_r = \mu t. q \& \{ \text{add}(\text{int}). p \oplus \{ \text{add}(\text{int}). t \}, \text{sub}(\text{int}). p \oplus \{ \text{sub}(\text{int}). t \} \} \quad (4)$$

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 Figure 3(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 ($\leq_a$), we can optimise T_q to the following T_q^{opt} , pushing the external choice behind the internal one:

$$T_q^{\text{opt}} = \mu t. r \oplus \{ \text{add}(\text{int}). p \& \{ \text{add}(\text{int}). t \}, \text{sub}(\text{int}). p \& \{ \text{sub}(\text{int}). t \} \} \quad (5)$$

With process P_q typed by T_q^{opt} , we can run the ring protocol more efficiently (see [18]); we make this precise in § 6. An overview of the history of asynchronous subtyping is given by Chen et al. [14], encompassing the theory and applications of the relation.

Associations and Type Soundness We say a set of local types $\{T_p^{\text{opt}}\}_{p \in \mathcal{P}}$ is *associated* to a global type G iff they are refined or optimised by $\leq_a$ from G 's projection $\{T_p\}_{p \in \mathcal{P}}$. This article proves the *sound* and *complete operational correspondence* between behaviours of G and $\{T_p^{\text{opt}}\}_{p \in \mathcal{P}}$. This is the link between optimised programs and global types in the top-down workflow, combining projection and optimisation into one step.

More formally, given a typing context $\Delta = \{p : (\sigma_p, T_p^{\text{opt}})\}_{p \in \text{roles}(G)}$ where $\text{roles}(G)$ is a set of roles in G , σ_p is the type of the queue for participant p , then the *association* between Δ and a global type G is defined as follows:

$$\Delta \sqsubseteq_a G \text{ if } G \vdash_p (\sigma'_p, T_p) \text{ and } T_p^{\text{opt}} \leq_a T_p \quad \text{and } \sigma_p \leq_a \sigma'_p \text{ for all } p \in \text{roles}(G) \quad (6)$$

where $\sigma_p \leq_a \sigma'_p$ extends the asynchronous subtyping to the elements of each queue. Here, $G \vdash_p (\sigma'_p, T_p)$ is the *projection relation*, relating the global type G at participant p to a queue/local-type pair (σ'_p, T_p) that describes p 's local behaviour (formally defined later in Def. 4). 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 [29, 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 [51].

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

Step 1 We define the operational semantics of G (denoted by $G \rightarrow G'$, which states G moves to G' after one communication between two participants) and a typing context Δ (denoted by $\Delta \rightarrow \Delta'$, which states Δ moves to Δ' after one communication).

Step 2 We prove **soundness**: if $\Delta \sqsubseteq_a G$ and G can take a step, then there exist G' and Δ' such that $G \rightarrow G'$, $\Delta \rightarrow \Delta'$ and $\Delta' \sqsubseteq_a G'$. Here, $G \rightarrow G'$ (resp. $\Delta \rightarrow \Delta'$) denotes a single communication step of the global type (resp. typing context), formally introduced in Def. 7. The actual result we prove (Theorem 12) is in fact slightly stronger, but this extra precision is not needed for the subject reduction and session fidelity results.

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

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

$$\frac{\forall p \in \text{dom}(\Delta) \quad \vdash P_p \triangleright T_p \quad \vdash h_p \triangleright \sigma_p \quad \Delta(p) = (\sigma_p, T_p) \quad \Delta \sqsubseteq_a G}{\vdash^{\text{top}} \prod_{p \in \text{dom}(\Delta)} (p \triangleleft P_p \mid p \triangleleft h_p) \triangleright \Delta} \text{ [SESSTOP]}$$

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 [29, Figure 5]). $p \triangleleft P_p$ means process P_p is acting as participant p , buffering sent messages in its queue $p \triangleleft h_p$.

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 [29, 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 [29, Theorem 4.13]. We additionally prove that all projected typing contexts are **safe**, **deadlock-free** and **live**, hence, by [29, Theorem 4.12], the same is true for all associated typing contexts (Thm. 13). We give detailed explanations and proofs in § 6.

1.3. Outline and Contributions

We provide the first sound top-down system of the asynchronous multiparty session types framework, from global types right through to projected local types, their asynchronous subtypes, and all the way down to the processes they type, including the most flexible form of projection. Importantly, we specify the *operational correspondence* between these levels, including in the presence of the additional semantic flexibility allowed by asynchrony. This enables us to prove subject reduction and session fidelity theorems for the top-down system.

Previous works have studied incomplete parts of this system, which are disconnected from each other: the asynchronous subtyping relation [29], extensions to global type syntax to allow for en-route messages [3, 11], the coinductive full projection relation in the synchronous setting [55], and the liveness in the bottom-up system [51].

For completing the top-down system with $\leq_a$, many critical components have been missing, up until now, including (1) the operational correspondence between global types and associated typing contexts (§5.2), (2) correctness of coinductive full projection for asynchronous session types (§2.2), and (3) liveness, deadlock-freedom and safety of local types projected from a global protocol (§5.3). We fill these gaps, and in doing so we introduce several new technical contributions, including a non-deterministic operational semantics for global types (§3.1) that captures the additional behaviours permitted by asynchronous reorderings; the balanced⁺ well-formedness condition and associated depth functions (§4.1); forgetful contexts (Defs. 22 and 24) which handle the non-determinism arising from asynchronous subtyping in the soundness proof; and context projection (Def. 26) which lifts the projection relation to contexts.

Our major contributions are the soundness and completeness theorems (Thms. 11 and 12) which establish that the semantics of global types (§3.1) can faithfully mirror the semantics of any of its associated typing contexts, which includes the projected typing contexts obtained via the coinductive projection (Def. 4). We also prove for the first time that typing contexts projected from a global type under asynchronous subtyping are live (and hence safe and deadlock-free) through the liveness theorem (Thm. 13). No previous work has established liveness for projections in the presence of asynchronous subtyping and coinductive full merging. We use this to derive the subject reduction and session fidelity results in §6.

We provide an extensive exploration of global and local types in §2.1, including syntax, projection, and subtyping. We define operational semantics for both global types (§3.1) and typing contexts (§3.2). We establish the sound and complete operational relationship between these two semantics in §5.

This paper provides a significantly updated and corrected version of the proofs of the theorems from the extended abstract in [49], as well as new examples and detailed explanations. Compared to [49], we have: (1) added full and corrected proofs for all lemmas and theorems; (2) corrected issues with the semantics of global types in [49]; (3) added new technical devices and syntactic functions to simplify the proofs; and (4) added new examples to illustrate our technical results.

2. Multiparty Session Types

This section introduces *global* and *local* types, together with *queue* types. As in the work of Barwell et al. [3], 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. 4) which extends the standard coinductive projection [28, Definition 3.6] to asynchronous semantics by simultaneously projecting onto both local and queue types. We follow this by introducing the *precise asynchronous* subtyping, which will allow us to preemptively send messages.

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 $p, q, s, t, \dots$, 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 below where constructs are mostly standard [51].

B	$::=$	$\text{int} \mid \text{bool} \mid \text{real} \mid \text{unit} \mid \dots$	Basic types
G	$::=$	$p \rightarrow q : \{m_i(B_i).G_i\}_{i \in I}$	Transmission
		$\mid$	
		$p \overset{m}{\rightsquigarrow} q : \{m(B).G\}$	Transmission en route
		$\mid$	
		$\mu t.G \mid t \mid \text{end}$	Recursion, Type variable, Termination
T	$::=$	$p \& \{m_i(B_i).T_i\}_{i \in I} \mid p \oplus \{m_i(B_i).T_i\}_{i \in I}$	External and internal choices
		$\mid$	
		$\mu t.T \mid t \mid \text{end}$	Recursion, Type variable, Termination
σ	$::=$	$\emptyset \mid (q, m(B)) \mid \sigma \cdot \sigma$	Empty Queue, Message, Concatenation

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

- $p \rightarrow q: \{\mathfrak{m}_i(B_i).G_i\}_{i \in I}$: a *transmission*, denoting a message from role p to role q , with a label $\mathfrak{m}_i$, 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 $\mathfrak{m}_i$ be pair-wise distinct, and self receptions be excluded (i.e. $p \neq q$).
- $p \overset{\mathfrak{m}}{\rightarrow} q: \{\mathfrak{m}(B).G\}$: a *transmission en route*, representing a *transmission* of the message $\mathfrak{m}$ which has already been sent by role p but has not been received by role q . This type is only meaningful at runtime. We require that self receptions be excluded (i.e. $p \neq q$).
- $\mu t.G$: a *recursive* global type, where contractive requirements apply [48, §21.8], i.e. each recursion variable t is bound within a $\mu t \dots$ and is guarded.
- **end**: a *terminated* global type (omitted where unambiguous).

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

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

Queue types range over $\sigma, \sigma', \sigma_i, \dots$, and describe the type of queues storing buffered asynchronous messages: $\emptyset$ is the empty queue; $(p, \mathfrak{m}(B))$ is the type of a queued message being sent to participant p with a message label $\mathfrak{m}$ and a payload of type B ; and $\sigma \cdot \sigma'$ is the concatenation of two queues. We consider queue types up-to associativity and we allow queue elements with distinct destinations to commute Def. 1, this simulates the existence of a queue for each potential recipient. We take the convention that $(p, \mathfrak{m}(B)) \cdot \sigma$ is a queue with head $(p, \mathfrak{m}(B))$, which can be dequeued, and tail σ , and that enqueueing $(p, \mathfrak{m}'(B'))$ to σ' gives us $\sigma' \cdot (p, \mathfrak{m}'(B'))$ i.e. queues are read left-to-right.

Definition 1 (Queue Equivalence [29]). We define the relation $\equiv$ inductively on syntactic queues by:

$$\begin{array}{c} \overline{\sigma \equiv \sigma} \quad \overline{\emptyset \cdot \sigma \equiv \sigma} \quad \overline{\sigma \cdot \emptyset \equiv \sigma} \quad \overline{(\sigma \cdot \sigma') \cdot \sigma'' \equiv \sigma \cdot (\sigma' \cdot \sigma'')} \\[10pt] \frac{\sigma_L \equiv \sigma'_L \quad \sigma_R \equiv \sigma'_R}{\sigma_L \cdot \sigma_R \equiv \sigma'_L \cdot \sigma'_R} \quad \frac{p \neq q}{(p, \mathfrak{m}(B)) \cdot (q, \mathfrak{m}'(B')) \equiv (q, \mathfrak{m}'(B')) \cdot (p, \mathfrak{m}(B))} \end{array}$$

Definition 2 (Unfolding). We define the unfolding operator on global types and local types recursively to unfold recursive types. We define $\text{unf}(\mu t.G) = \text{unf}(G[\mu t.G/t])$ and $\text{unf}(G) = G$ otherwise. We define $\text{unf}(\mu t.T) = \text{unf}(T[\mu t.T/t])$ and $\text{unf}(T) = T$ otherwise.

Below we define sets of roles for a given global type.

Definition 3 (Role Functions).

- We define $\text{roles}(G)$ to be the *set of participants* in G by induction on the structure of G : $\text{roles}(\text{end}) = \emptyset$; $\text{roles}(t) = \emptyset$; $\text{roles}(\mu t.G) = \text{roles}(G)$; $\text{roles}(p \rightarrow q: \{\mathfrak{m}_i(B_i).G_i\}_{i \in I}) = \{p, q\} \cup \bigcup_{i \in I} \text{roles}(G_i)$; and $\text{roles}(p \overset{\mathfrak{m}}{\rightarrow} q: \{\mathfrak{m}(B).G\}) = \{p, q\} \cup \text{roles}(G)$.
- We define $\text{sRoles}(G)$ to be the set of participants in G that have sent an en-route transmission by induction on the structure of G : $\text{sRoles}(\text{end}) = \emptyset$; $\text{sRoles}(t) = \emptyset$; $\text{sRoles}(\mu t.G) = \text{sRoles}(G)$; $\text{sRoles}(p \rightarrow q: \{\mathfrak{m}_i(B_i).G_i\}_{i \in I}) = \bigcup_{i \in I} \text{sRoles}(G_i)$; and $\text{sRoles}(p \overset{\mathfrak{m}}{\rightarrow} q: \{\mathfrak{m}(B).G\}) = \{p\} \cup \text{sRoles}(G)$.

- We define $\mathbf{aRoles}(G)$ to be the set of participants in G that may perform actions, i.e. appears either as the receiver of a transmission or as the sender of a non-en-route transmission, by induction on the structure of G : $\mathbf{aRoles}(\mathbf{end}) = \emptyset$; $\mathbf{aRoles}(\mathbf{t}) = \emptyset$; $\mathbf{aRoles}(\mu \mathbf{t}.G) = \mathbf{aRoles}(G)$; $\mathbf{aRoles}(\mathbf{p} \rightarrow \mathbf{q} : \{\mathbf{m}_1(B_i).G_i\}_{i \in I}) = \{\mathbf{p}, \mathbf{q}\} \cup \bigcup_{i \in I} \mathbf{aRoles}(G_i)$; and $\mathbf{aRoles}(\mathbf{p} \xrightarrow{\mathbf{m}} \mathbf{q} : \{\mathbf{m}(B).G\}) = \{\mathbf{q}\} \cup \mathbf{aRoles}(G)$

$\mathbf{sRoles}(G)$ are the participants that have non-empty queues once projected, and $\mathbf{aRoles}(G)$ are the participants that have non-terminated local types once projected.

We will now use the ring protocol global type, and an intermediate global type, to demonstrate unfolding and the role functions.

Example 1 (Ring Protocol). Recall the ring protocol (§ 1.2):

$$G_{\text{ring}} = \mu \mathbf{t}. \mathbf{p} \rightarrow \mathbf{q} : \mathbf{add}(\mathbf{int}). \mathbf{q} \rightarrow \mathbf{r} : \left\{ \begin{array}{l} \mathbf{add}(\mathbf{int}) . \mathbf{r} \rightarrow \mathbf{p} : \{\mathbf{add}(\mathbf{int}) . \mathbf{t}\} \\ \mathbf{sub}(\mathbf{int}) . \mathbf{r} \rightarrow \mathbf{p} : \{\mathbf{sub}(\mathbf{int}) . \mathbf{t}\} \end{array} \right\} \quad (7)$$

We have that $\mathbf{roles}(G_{\text{ring}}) = \mathbf{aRoles}(G_{\text{ring}}) = \{\mathbf{p}, \mathbf{q}, \mathbf{r}\}$ and $\mathbf{sRoles}(G_{\text{ring}}) = \emptyset$. We also have that

$$\mathbf{unf}(G_{\text{ring}}) = \mathbf{p} \rightarrow \mathbf{q} : \mathbf{add}(\mathbf{int}). \mathbf{q} \rightarrow \mathbf{r} : \left\{ \begin{array}{l} \mathbf{add}(\mathbf{int}) . \mathbf{r} \rightarrow \mathbf{p} : \left\{ \mathbf{add}(\mathbf{int}) . \mu \mathbf{t}. \mathbf{p} \rightarrow \mathbf{q} : \mathbf{add}(\mathbf{int}). \mathbf{q} \rightarrow \mathbf{r} : \left\{ \begin{array}{l} \mathbf{add}(\mathbf{int}) . \mathbf{r} \rightarrow \mathbf{p} : \{\mathbf{add}(\mathbf{int}) . \mathbf{t}\} \\ \mathbf{sub}(\mathbf{int}) . \mathbf{r} \rightarrow \mathbf{p} : \{\mathbf{sub}(\mathbf{int}) . \mathbf{t}\} \end{array} \right\} \right\} \\ \mathbf{sub}(\mathbf{int}) . \mathbf{r} \rightarrow \mathbf{p} : \left\{ \mathbf{sub}(\mathbf{int}) . \mu \mathbf{t}. \mathbf{p} \rightarrow \mathbf{q} : \mathbf{add}(\mathbf{int}). \mathbf{q} \rightarrow \mathbf{r} : \left\{ \begin{array}{l} \mathbf{add}(\mathbf{int}) . \mathbf{r} \rightarrow \mathbf{p} : \{\mathbf{add}(\mathbf{int}) . \mathbf{t}\} \\ \mathbf{sub}(\mathbf{int}) . \mathbf{r} \rightarrow \mathbf{p} : \{\mathbf{sub}(\mathbf{int}) . \mathbf{t}\} \end{array} \right\} \right\} \end{array} \right\} \quad (8)$$

Example 2 (Ring Protocol II). Consider the global type that occurs during the ring protocol:

$$G = \mathbf{p} \xrightarrow{\mathbf{add}} \mathbf{q} : \mathbf{add}(\mathbf{int}). \mathbf{q} \rightarrow \mathbf{r} : \left\{ \begin{array}{l} \mathbf{add}(\mathbf{int}) . \mathbf{r} \rightarrow \mathbf{p} : \{\mathbf{add}(\mathbf{int}) . G_{\text{ring}}\} \\ \mathbf{sub}(\mathbf{int}) . \mathbf{r} \rightarrow \mathbf{p} : \{\mathbf{sub}(\mathbf{int}) . G_{\text{ring}}\} \end{array} \right\} \quad (9)$$

We have that $\mathbf{roles}(G) = \mathbf{aRoles}(G) = \{\mathbf{p}, \mathbf{q}, \mathbf{r}\}$, $\mathbf{sRoles}(G) = \{\mathbf{p}\}$, and $\mathbf{unf}(G) = G$.

2.2. Projections

In the top-down approach of MPST, local types are obtained by projecting a global type onto roles. Whereas projection is traditionally presented as a *partial function* from a global type and a participant to a local type [35], our Def. 4 below instead defines it as a *coinductive ternary relation* $G \upharpoonright_{\mathbf{p}}(\sigma, T)$ between a global type G , a participant $\mathbf{p}$, and a (queue type, local type) pair. It should be read as a *predicate* which holds when (σ, T) is a valid projection of G at $\mathbf{p}$. The queues σ allow us to capture buffered messages from en-route transmissions at the local level.

The rules are organised by the relationship between the projected role $\mathbf{p}$ and the outermost interaction of G . When $\mathbf{p}$ is the sender, its local type begins with an internal choice ($[P-\oplus]$); when $\mathbf{p}$ is the receiver, with an external choice ($[P-\&]$); when $\mathbf{p}$ is not involved, the continuations from each branch must be reconciled into a single local type using merge ($[P-\sqcap]$). Each of these three cases has a corresponding rule (suffixed -II) that handles the additional presence of an en-route transmission in G , extracting the relevant queue prefix before proceeding. Finally, $[P-\text{END}]$ and $[P-\emptyset]$ cover termination.

Definition 4 (Global Type Projection). The *projection relation* $G \upharpoonright_{\mathbf{p}}(\sigma, T)$ (read “ (σ, T) is a projection of the global type G onto the participant $\mathbf{p}$ ”) is the largest relation such that whenever $G \upharpoonright_{\mathbf{p}}(\sigma, T)$ holds:

$[P-\text{END}]$ If $\mathbf{p} \notin \mathbf{aRoles}(G)$, then $\mathbf{unf}(T) = \mathbf{end}$

$[P-\emptyset]$ If $\mathbf{p} \notin \mathbf{sRoles}(G)$, then $\sigma = \emptyset$

$[P-\sqcap]$ If $\mathbf{unf}(G) = \mathbf{q} \rightarrow \mathbf{r} : \{\mathbf{m}_1(B_i).G_i\}_{i \in I}$, then for all $i \in I$ there exist T_i such that $G_i \upharpoonright_{\mathbf{p}}(\sigma, T_i)$ and $\prod \{T_i\}_{i \in I} \ni T$

$[P-\sqcap\text{-II}]$ If $\mathbf{unf}(G) = \mathbf{q} \xrightarrow{\mathbf{m}} \mathbf{r} : \{\mathbf{m}(B).G'\}$, then $G' \upharpoonright_{\mathbf{p}}(\sigma, T)$

$[P-\oplus]$ If $\mathbf{unf}(G) = \mathbf{p} \rightarrow \mathbf{q} : \{\mathbf{m}_1(B_i).G_i\}_{i \in I}$, then for all $i \in I$ there exist T_i such that $G_i \upharpoonright_{\mathbf{p}}(\sigma, T_i)$ and $\mathbf{unf}(T) = \mathbf{q} \oplus \{\mathbf{m}_1(B_i).T_i\}_{i \in I}$

[P-⊕-II] If $\text{unf}(G) = \mathbf{p} \xrightarrow{\mathbf{m}} \mathbf{q} : \{\mathbf{m}(B).G'\}$, then there exists σ' such that $G' \vdash_{\mathbf{p}}(\sigma', T)$, and $\sigma = (\mathbf{q}, \mathbf{m}_j(B_j)) \cdot \sigma'$

[P-&] If $\text{unf}(G) = \mathbf{q} \rightarrow \mathbf{p} : \{\mathbf{m}_i(B_i).G_i\}_{i \in I}$, then for all $i \in I$ there exist T_i such that $G_i \vdash_{\mathbf{p}}(\sigma, T_i)$ and $\text{unf}(T) = \mathbf{q} \& \{\mathbf{m}_i(B_i).T_i\}_{i \in I}$

[P-&-II] If $\text{unf}(G) = \mathbf{q} \xrightarrow{\mathbf{m}} \mathbf{p} : \{\mathbf{m}(B).G'\}$, then there exists T' such that $G' \vdash_{\mathbf{p}}(\sigma, T')$ and $\text{unf}(T) = \mathbf{q} \& \{\mathbf{m}(B).T'\}$

In rule [P- $\sqcap$], a role not involved in a choice cannot observe which branch was taken. The projection must therefore reconcile the continuations from each branch into a single local type that is compatible with all of them. This is the role of $\sqcap$, the *merge operation for session types (full merging)*. $\sqcap$ is a partial function that takes a finite and non-empty set of local types, $\mathcal{T}$, and returns a set of local types, $\sqcap \mathcal{T}$. We say that T is a merge of $\mathcal{T}$ (equivalently, T is a merge of the types in $\mathcal{T}$) to mean that $T \in \sqcap \mathcal{T}$. We say that $\mathcal{T}$ is mergeable (equivalently, the types in $\mathcal{T}$ are mergeable) if $\sqcap \mathcal{T} \neq \emptyset$. In the case of indexed sets, we may write $\sqcap_{i \in I} T_i$ for $\sqcap \{T_i : i \in I\}$.

We define the merge operation using the coinductive relation *merge* below, by defining $\sqcap \mathcal{T} = \{T : \text{merge} \langle \mathcal{T}, T \rangle\}$. We define *merge* to be the largest relation between non-empty finite sets of local types and local types such that if $\text{merge} \langle \{T_i : i \in I\}, T \rangle$, then:

[$\sqcap$ -end] If $\text{unf}(T) = \mathbf{end}$, then $\text{unf}(T_i) = \mathbf{end}$ for all $i \in I$

[$\sqcap$ -⊕] If $\text{unf}(T) = \mathbf{q} \oplus \{\mathbf{m}_j(S_j).T'_j\}_{j \in J}$, then for all $i \in I$ $\text{unf}(T_i) = \mathbf{q} \oplus \{\mathbf{m}_j(S_j).T_{i,j}\}_{j \in J}$ where for all $j \in J$ $\text{merge} \langle \{T_{i,j} : i \in I\}, T'_j \rangle$

[$\sqcap$ -&] If $\text{unf}(T) = \mathbf{p} \& \{\mathbf{m}_j(S_j).T'_j\}_{j \in J}$, then for all $i \in I$ $\text{unf}(T_i) = \mathbf{p} \& \{\mathbf{m}_j(S_j).T_{i,j}\}_{j \in J_i}$ where $J = \bigcup_{i \in I} J_i$ and for all $j \in J$ $\text{merge} \langle \{T_{i,j} : j \in J_i\}, T'_j \rangle$

Here $\text{unf}(\cdot)$ is the unfolding function of Def. 2.

When the queue is empty, we may leave it implicit i.e. if $G \vdash_{\mathbf{p}}(\emptyset, T)$ then we may instead write $G \vdash_{\mathbf{p}} T$.

The rule [P-END] says that if a global type doesn't specify behaviour for some role, then its projection has no behaviour. The rule [P- $\emptyset$] is similar, it says that if a role doesn't add anything to the queue, then its queue must be empty. The rule [P- $\sqcap$] says that if the head of the global type does not involve a participant and the projections of the continuations are compatible, then the projection of the global type is their merge. The rule [P- $\sqcap$ -II] says that if the head of a global type is an en-route transmission not involving $\mathbf{p}$, then the projection onto $\mathbf{p}$ is the projection of the continuation. The rule [P-⊕] says that if the head of a global type is a communication with sender $\mathbf{p}$ and destination $\mathbf{q}$, then the head of the projection onto $\mathbf{p}$ is a send to $\mathbf{q}$ with the same messages and payloads, and the local continuations are the projections of the global continuations. The rule [P-⊕-II] allows an en-route message $(\mathbf{q}, \mathbf{m}_j(B_j))$ to be included in the projected queue of outgoing messages. If a global type G starts with a transmission from role $\mathbf{p}$ to role $\mathbf{q}$, projecting it onto role $\mathbf{p}$ (resp. $\mathbf{q}$) results in an internal (resp. external) choice, provided that the continuation G is also projectable. The rules [P-&] and [P-&-II] say that if the head of a global type is a communication with sender $\mathbf{q}$ and destination $\mathbf{p}$, then the head of the projection onto $\mathbf{p}$ is a receive from $\mathbf{q}$ with the same messages and payloads, and the local continuations are the projections of the global continuations. When projecting G onto other participants $\mathbf{r}$ ($\mathbf{r} \neq \mathbf{p}$ and $\mathbf{r} \neq \mathbf{q}$), a merge operator, as defined in Def. 4, 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 by Udomsirungruang and Yoshida [55, §4.4].

We now return to our running example to demonstrate a derivation of an instance of a projection and a merge.

Example 3 (Ring Protocol Projection). Recall $T_{\mathbf{p}}$ from the ring-choice example (§1.2):

$$T_{\mathbf{p}} = \mu \mathbf{t}. \mathbf{q} \oplus \{\text{add}(\text{int}).\mathbf{r} \& \{\text{add}(\text{int}).\mathbf{t}, \text{sub}(\text{int}).\mathbf{t}\}\} \quad (10)$$

We can derive $G_{\text{ring}} \downarrow_p T_p$ by applying [P- $\oplus$], [P- $\cap$], and merging the results of applying [P- $\&$], to the coinductive hypothesis for each branch.

$$\frac{\frac{\frac{G_{\text{ring}} \downarrow_p T_p}{\text{r} \rightarrow \text{p} : \{ \text{add}(\text{int}) . G_{\text{ring}} \} \downarrow_p \text{r} \& \{ \text{add}(\text{int}) . T_p \}}} \quad \frac{\frac{G_{\text{ring}} \downarrow_p T_p}{\text{r} \rightarrow \text{p} : \{ \text{sub}(\text{int}) . G_{\text{ring}} \} \downarrow_p \text{r} \& \{ \text{sub}(\text{int}) . T_p \}}} \quad \text{[P-}\&\text{]} \quad \text{[P-}\cap\text{]}}{\frac{\text{q} \rightarrow \text{r} : \left\{ \begin{array}{l} \text{add}(\text{int}) . \text{r} \rightarrow \text{p} : \{ \text{add}(\text{int}) . G_{\text{ring}} \} \\ \text{sub}(\text{int}) . \text{r} \rightarrow \text{p} : \{ \text{sub}(\text{int}) . G_{\text{ring}} \} \end{array} \right\} \downarrow_p \text{r} \& \{ \text{add}(\text{int}) . T_p, \text{sub}(\text{int}) . T_p \}}}{G_{\text{ring}} \downarrow_p T_p} \text{[P-}\oplus\text{]} \quad (11)$$

Noting that:

$$\prod \{ \text{r} \& \{ \text{add}(\text{int}) . T_p \}, \text{r} \& \{ \text{sub}(\text{int}) . T_p \} \} \ni \text{r} \& \{ \text{add}(\text{int}) . T_p, \text{sub}(\text{int}) . T_p \} \quad (12)$$

$$\frac{\text{merge} \langle \{ T_p \}, T_p \rangle \quad \text{merge} \langle \{ T_p \}, T_p \rangle}{\text{merge} \langle \{ \text{r} \& \{ \text{add}(\text{int}) . T_p \}, \text{r} \& \{ \text{sub}(\text{int}) . T_p \} \}, \text{r} \& \{ \text{add}(\text{int}) . T_p, \text{sub}(\text{int}) . T_p \} \rangle} \text{[}\cap\text{-}\&\text{]} \quad (13)$$

Of course, this projection is derivable by the full inductive projection, which is a subrelation of the full coinductive projection; this is not always the case. We now provide a more involved example, which makes use of en-route transitions and cannot be projected inductively.

Example 4 (Coinductive Projection). Consider the global type:

$$G = \text{p} \rightarrow \text{r} : \left\{ \begin{array}{l} \text{m}_1 . \text{q} \xrightarrow{\text{m}} \text{p} : \text{m} . \mu \text{t} . \text{p} \rightarrow \text{q} : \text{m}_1 . \text{t} \\ \text{m}_2 . \text{q} \xrightarrow{\text{m}} \text{p} : \text{m} . \text{p} \rightarrow \text{q} : \text{m}_2 . \mu \text{t} . \text{p} \rightarrow \text{q} : \text{m}_1 . \text{t} \end{array} \right\} \quad (14)$$

G projects onto all three roles. The projections onto r and p are obtained directly from the coinductive rules without any merging.

$$\frac{\frac{\frac{\text{q} \xrightarrow{\text{m}} \text{p} : \text{m} . \mu \text{t} . \text{p} \rightarrow \text{q} : \text{m}_1 . \text{t} \downarrow_{\text{r}} \text{end}} \quad \frac{\text{q} \xrightarrow{\text{m}} \text{p} : \text{m} . \text{p} \rightarrow \text{q} : \text{m}_2 . \mu \text{t} . \text{p} \rightarrow \text{q} : \text{m}_1 . \text{t} \downarrow_{\text{r}} \text{end}} \quad \text{[P-END]} \quad \text{[P-END]}}{G \downarrow_{\text{r}} \text{p} \& \{ \text{m}_1, \text{m}_2 \}} \text{[P-}\&\text{]} \quad (15)$$

$$\frac{\frac{\frac{\mu \text{t} . \text{p} \rightarrow \text{q} : \text{m}_1 . \text{t} \downarrow_{\text{p}} \mu \text{t} . \text{q} \oplus \{ \text{m}_1 . \text{t} \}}{\mu \text{t} . \text{p} \rightarrow \text{q} : \text{m}_1 . \text{t} \downarrow_{\text{p}} \mu \text{t} . \text{q} \oplus \{ \text{m}_1 . \text{t} \}} \text{[P-}\oplus\text{]} \quad \frac{\frac{\mu \text{t} . \text{p} \rightarrow \text{q} : \text{m}_1 . \text{t} \downarrow_{\text{p}} \mu \text{t} . \text{q} \oplus \{ \text{m}_1 . \text{t} \}}{\mu \text{t} . \text{p} \rightarrow \text{q} : \text{m}_1 . \text{t} \downarrow_{\text{p}} \mu \text{t} . \text{q} \oplus \{ \text{m}_1 . \text{t} \}} \text{[P-}\oplus\text{]} \quad \frac{\text{p} \rightarrow \text{q} : \text{m}_2 . \mu \text{t} . \text{p} \rightarrow \text{q} : \text{m}_1 . \text{t} \downarrow_{\text{p}} \text{q} \oplus \{ \text{m}_2 . \mu \text{t} . \text{q} \oplus \{ \text{m}_1 . \text{t} \} \}}{\text{q} \xrightarrow{\text{m}} \text{p} : \text{m} . \text{p} \rightarrow \text{q} : \text{m}_2 . \mu \text{t} . \text{p} \rightarrow \text{q} : \text{m}_1 . \text{t} \downarrow_{\text{p}} \text{q} \oplus \{ \text{m}_2 . \mu \text{t} . \text{q} \oplus \{ \text{m}_1 . \text{t} \} \}} \text{[P-}\&\text{-II]}} \quad \text{[P-}\&\text{-II]}}{\frac{\text{q} \xrightarrow{\text{m}} \text{p} : \text{m} . \mu \text{t} . \text{p} \rightarrow \text{q} : \text{m}_1 . \text{t} \downarrow_{\text{p}} \text{q} \oplus \{ \text{m} . \mu \text{t} . \text{q} \oplus \{ \text{m}_1 . \text{t} \} \}} \quad \text{q} \xrightarrow{\text{m}} \text{p} : \text{m} . \text{p} \rightarrow \text{q} : \text{m}_2 . \mu \text{t} . \text{p} \rightarrow \text{q} : \text{m}_1 . \text{t} \downarrow_{\text{p}} \text{q} \oplus \{ \text{m} . \text{q} \oplus \{ \text{m}_2 . \mu \text{t} . \text{q} \oplus \{ \text{m}_1 . \text{t} \} \} \}} \text{[P-}\&\text{-II]}}{G \downarrow_{\text{p}} \text{r} \oplus \left\{ \begin{array}{l} \text{m}_1 . \text{q} \& \{ \text{m} . \mu \text{t} . \text{q} \oplus \{ \text{m}_1 . \text{t} \} \} \\ \text{m}_2 . \text{q} \& \{ \text{m} . \text{q} \oplus \{ \text{m}_2 . \mu \text{t} . \text{q} \oplus \{ \text{m}_1 . \text{t} \} \} \} \end{array} \right\}} \text{[P-}\oplus\text{]} \quad (16)$$

Projecting onto q is more delicate, because q is not involved in the initial $\text{p} \rightarrow \text{r}$ choice and its continuations across the two branches differ. We isolate this case in the following example, in order to explain the merge step that reconciles the two continuations.

Example 5 (Merging in Projection). Continuing from Example 4, we project G onto q . Since q is not a participant in the top-level $\text{p} \rightarrow \text{r}$ communication, the applicable rule is [P- $\cap$]: the projection is obtained by merging the projections of the two branches onto q . In each branch, q is the sender of the en-route message from q to p carrying label m , so [P- $\oplus$ -II] applies and yields a projection of the form $((\text{p}, \text{m}), T_i)$ with common queue-entry prefix (p, m) and residuals

$$T_1 = \mu \text{t} . \text{p} \& \{ \text{m}_1 . \text{t} \} \quad T_2 = \text{p} \& \{ \text{m}_2 . \mu \text{t} . \text{p} \& \{ \text{m}_1 . \text{t} \} \}$$

from the m_1 - and m_2 -branches respectively. The derivation of the projection onto q is:

$$\begin{array}{c}
 \frac{\frac{\mu t.p \rightarrow q : m_1.t \downarrow_q \mu t.p \& \{m_1.t\}}{\mu t.p \rightarrow q : m_1.t \downarrow_q \mu t.p \& \{m_1.t\}} \text{ [P-}\&\text{]}}{\frac{\mu t.p \rightarrow q : m_1.t \downarrow_q \mu t.p \& \{m_1.t\}}{q \rightsquigarrow p : m. \mu t.p \rightarrow q : m_1.t \downarrow_q ((p, m), \mu t.p \& \{m_1.t\})} \text{ [P-}\oplus\text{-II]}} \quad \frac{\frac{\frac{\mu t.p \rightarrow q : m_1.t \downarrow_q \mu t.p \& \{m_1.t\}}{\mu t.p \rightarrow q : m_1.t \downarrow_q \mu t.p \& \{m_1.t\}} \text{ [P-}\&\text{]}}{\mu t.p \rightarrow q : m_1.t \downarrow_q \mu t.p \& \{m_1.t\}} \text{ [P-}\&\text{]}}{\frac{p \rightarrow q : m_2. \mu t.p \rightarrow q : m_1.t \downarrow_q p \& \{m_2. \mu t.p \& \{m_1.t\}\}}{q \rightsquigarrow p : m. p \rightarrow q : m_2. \mu t.p \rightarrow q : m_1.t \downarrow_q ((p, m), p \& \{m_2. \mu t.p \& \{m_1.t\}\})} \text{ [P-}\oplus\text{-II]}} \\
 \hline
 G \downarrow_q ((p, m), p \& \left\{ \begin{array}{l} m_1. \mu t.p \& \{m_1.t\} \\ m_2. \mu t.p \& \{m_1.t\} \end{array} \right\}) \text{ [P-}\sqcap\text{]}
 \end{array} \tag{17}$$

whose top [P- $\sqcap$] step relies on the merge

$$\sqcap \left\{ \mu t.p \& \{m_1.t\}, p \& \{m_2. \mu t.p \& \{m_1.t\}\} \right\} \ni p \& \left\{ \begin{array}{l} m_1. \mu t.p \& \{m_1.t\} \\ m_2. \mu t.p \& \{m_1.t\} \end{array} \right\} \tag{18}$$

which is derivable by a single application of [P- $\&$], with one sub-obligation per label of the resulting external choice:

$$\frac{\text{merge} \left\langle \{ \mu t.p \& \{m_1.t\} \}, \mu t.p \& \{m_1.t\} \right\rangle \quad \text{merge} \left\langle \{ \mu t.p \& \{m_1.t\} \}, \mu t.p \& \{m_1.t\} \right\rangle}{\text{merge} \left\langle \{ \mu t.p \& \{m_1.t\}, p \& \{m_2. \mu t.p \& \{m_1.t\}\} \}, p \& \left\{ \begin{array}{l} m_1. \mu t.p \& \{m_1.t\} \\ m_2. \mu t.p \& \{m_1.t\} \end{array} \right\} \right\rangle} \text{ [P-}\&\text{]} \tag{19}$$

Intuitively, the merged type offers q both labels m_1 and m_2 as an external choice from p : under m_1 it continues as T_1 does after unfolding, immediately re-entering the loop; under m_2 it continues as T_2 does after its m_2 -receive, which in this example also lands in the same loop $\mu t.p \& \{m_1.t\}$. Whichever label p eventually sends to q , the merged local type at q is ready for it, witnessing the compatibility of T_1 and T_2 .

2.3. Asynchronous Multipart Subtyping

Given a standard subtyping $<$ for basic types (e.g. including $\text{int} < \text{int}$ and $\text{int} < \text{real}$), we give a summary of *asynchronous subtyping* $\leq_a$ introduced in [29]. 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.

$$\mathbb{T} = p \& \{m_i(B_i). \mathbb{T}_i\}_{i \in I} \mid p \oplus \{m_i(B_i). \mathbb{T}_i\}_{i \in I} \mid \text{end} \tag{20}$$

$$\mathbb{U} = p \& \{m_i(B_i). \mathbb{U}_i\}_{i \in I} \mid p ! m(B); \mathbb{U} \mid \text{end} \tag{21}$$

$$\mathbb{V} = p ? m(B); \mathbb{V} \mid p \oplus \{m_i(B_i). \mathbb{V}_i\}_{i \in I} \mid \text{end} \tag{22}$$

$$\mathbb{W} = p ? m(B); \mathbb{W} \mid p ! m(B); \mathbb{W} \mid \text{end} \tag{23}$$

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

$$\mathcal{A}^{(p)} = q ? m(B) \mid q ? m(B); \mathcal{A}^{(p)} \quad \mathcal{B}^{(p)} = r ? m(B) \mid q ! m(B) \mid r ? m(B); \mathcal{B}^{(p)} \mid q ! m(B); \mathcal{B}^{(p)} \tag{24}$$

We define the set $\text{act}(\mathbb{W})$ of actions of a SISO tree: $\text{act}(\text{end}) = \emptyset$; $\text{act}(p ? m(B); \mathbb{W}) = \{p ?\} \cup \text{act}(\mathbb{W})$; and $\text{act}(p ! m(B); \mathbb{W}) = \{p !\} \cup \text{act}(\mathbb{W})$. Using these definitions, we introduce a *refinement relation* ($\leq$) defined coinductively by the following rules:

$$\begin{array}{c}
\frac{B' <: B \quad \mathbb{W} \lesssim \mathcal{A}^{(p)}; \mathbb{W}' \quad \text{act}(\mathbb{W}) = \text{act}(\mathcal{A}^{(p)}; \mathbb{W}')}{p?m(B); \mathbb{W} \lesssim \mathcal{A}^{(p)}; p?m(B'); \mathbb{W}'} \text{ [REF-A]} \\
\\
\frac{B <: B' \quad \mathbb{W} \lesssim \mathcal{B}^{(p)}; \mathbb{W}' \quad \text{act}(\mathbb{W}) = \text{act}(\mathcal{B}^{(p)}; \mathbb{W}')}{p!m(B); \mathbb{W} \lesssim \mathcal{B}^{(p)}; p!m(B'); \mathbb{W}'} \text{ [REF-B]} \\
\\
\frac{B' <: B \quad \mathbb{W} \lesssim \mathbb{W}'}{p?m(B); \mathbb{W} \lesssim p?m(B'); \mathbb{W}'} \text{ [REF-IN]} \quad \frac{B <: B' \quad \mathbb{W} \lesssim \mathbb{W}'}{p!m(B); \mathbb{W} \lesssim p!m(B'); \mathbb{W}'} \text{ [REF-OUT]} \quad \frac{}{\text{end} \lesssim \text{end}} \text{ [REF-END]}
\end{array}$$

We can extract sets of single-input and single-output trees from a given tree using the functions $\llbracket \cdot \rrbracket_{\text{SI}}$ and $\llbracket \cdot \rrbracket_{\text{SO}}$.

$$\begin{aligned}
\llbracket p\&\{m_i(B_i).T_i\}_{i \in I} \rrbracket_{\text{SI}} &= \bigcup_{i \in I} \{p?m_i(B_i); V_i \mid V_i \in \llbracket T_i \rrbracket_{\text{SI}}\} \\
\llbracket p\oplus\{m_i(B_i).T_i\}_{i \in I} \rrbracket_{\text{SI}} &= \{p\oplus\{m_i(B_i).V_i\}_{i \in I} \mid \forall i \in I : V_i \in \llbracket T_i \rrbracket_{\text{SI}}\} \quad \llbracket \text{end} \rrbracket_{\text{SI}} = \{\text{end}\} \\
\llbracket p\oplus\{m_i(B_i).T_i\}_{i \in I} \rrbracket_{\text{SO}} &= \bigcup_{i \in I} \{p!m_i(B_i); U_i \mid U_i \in \llbracket T_i \rrbracket_{\text{SO}}\} \\
\llbracket p\&\{m_i(B_i).T_i\}_{i \in I} \rrbracket_{\text{SO}} &= \{p\&\{m_i(B_i).U_i\}_{i \in I} \mid \forall i \in I : U_i \in \llbracket T_i \rrbracket_{\text{SO}}\} \quad \llbracket \text{end} \rrbracket_{\text{SO}} = \{\text{end}\}
\end{aligned}$$

Definition 5 (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 $\leq_a$ over all session types by considering their decomposition into SI, SO, and SISO trees.

$$\frac{\forall \mathbb{U} \in \llbracket \mathfrak{T}(T) \rrbracket_{\text{SO}} \quad \forall \mathbb{V}' \in \llbracket \mathfrak{T}(T') \rrbracket_{\text{SI}} \quad \exists \mathbb{W} \in \llbracket \mathbb{U} \rrbracket_{\text{SI}} \quad \exists \mathbb{W}' \in \llbracket \mathbb{V}' \rrbracket_{\text{SO}} \quad \mathbb{W} \lesssim \mathbb{W}'}{T \leq_a T'} \text{ [SUB]} \quad (25)$$

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

Example 6 (Subtyping the Ring Protocol Projection (§1.2)). Recall the local types:

$$T_q = \mu t. p\&\{\text{add(int)}.r\oplus\{\text{add(int)}.t, \text{sub(int)}.t\}\} \quad (26)$$

$$T_q^{\text{opt}} = \mu t. r\oplus\{\text{add(int)}.p\&\{\text{add(int)}.t, \text{sub(int)}.p\&\{\text{add(int)}.t\}\}\} \quad (27)$$

To demonstrate that $T_q^{\text{opt}} \leq_a T_q$, we must show that for all $\mathbb{U} \in \llbracket \mathfrak{T}(T_q^{\text{opt}}) \rrbracket_{\text{SO}}$ and $\mathbb{V}' \in \llbracket \mathfrak{T}(T_q) \rrbracket_{\text{SI}}$, there exist $\mathbb{W} \in \llbracket \mathbb{U} \rrbracket_{\text{SI}}$ and $\mathbb{W}' \in \llbracket \mathbb{V}' \rrbracket_{\text{SO}}$ such that $\mathbb{W} \lesssim \mathbb{W}'$. Consider the following sets:

$$\llbracket \mathfrak{T}(T_q^{\text{opt}}) \rrbracket_{\text{SO}} = \{r!\text{add(int)}; p?\text{add(int)}; \dots, r!\text{sub(int)}; p?\text{add(int)}; \dots, \dots\} \quad (28)$$

$$\llbracket \mathfrak{T}(T_q) \rrbracket_{\text{SI}} = \left\{ p?\text{add(int)}; r\oplus \left\{ \begin{array}{l} \text{add(int)} \dots \\ \text{sub(int)} \dots \end{array} \right\} \right\} \quad (29)$$

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 $\mathbb{W} = r!\text{sub(int)}; p?\text{add(int)}; \dots$ and we can pick $\mathbb{W}' = p?\text{add(int)}; r!\text{sub(int)}; \dots$

Then we can apply rule [REF-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: $T_q^{\text{opt}} \leq_a T_q$.

Compactness of Subtypes. Beyond enabling message reordering optimisations, subtyping, in both synchronous and asynchronous settings, can also potentially yield *arbitrarily more compact* type representations. This compactness is particularly valuable in practical implementations, where small types can also simplify readability and understanding. For example, consider a type that reads exactly n messages labelled a followed by one b :

$$T_n = \underbrace{p\&\{a.p\&\{a \dots p\&\{a.p\&\{b.\text{end}\}\}\}\}}_{n \text{ nested external choices}} \quad (30)$$

This type has size $O(n)$. However, a subtype that accepts *any* number of a 's before b can be written compactly as:

$$T' = \mu t. p \& \{a.t, b.\text{end}\} \quad (31)$$

We have $T' \leq_a T_n$ for all n , yet T' has constant size $O(1)$. This is in fact the case even for real projected local types, as coinductive projection can in general yield local types superpolynomial in the size of the global type [55].

Properties of Subtyping. While asynchronous subtyping is very flexible, allowing us to delay communications and simplify structure, this can not be done indefinitely. For example, asynchronous subtyping never admits a non-trivial supertype of **end**.

Lemma 1 (End Subtyping). *If $\text{unf}(T') = \text{end}$ and $T' \leq_a T$, then $\text{unf}(T) = \text{end}$.*

Proof. The type **end** is conveniently already a SISO tree. If $T' \leq_a T$ and $\text{unf}(T') = \text{end}$, by inversion we see that $\text{end} \leq T$ can only be derived from [REF-END], and so it must be that $\text{unf}(T) = \text{end}$. $\square$

We also note that, as one might expect from usual properties of subtyping, the precise asynchronous subtyping is a preorder.

Lemma 2 (Reflexivity and Transitivity of Subtyping, Lemma 3.8 in Ghilezan et al. [29]). *For any closed, well-guarded local types T , T' and T'' : (1) $T \leq_a T$ holds, and (2) if $T \leq_a T'$ and $T' \leq_a T''$ then it must be that $T \leq_a T''$ holds.*

However, unlike the synchronous subtyping (Def. 5), the precise asynchronous subtyping does not give us a strong operational correspondence between instances. This is elaborated further in Ex. 15. Regardless, we retain a weak operational correspondence, witnessed by Lem. 16's structural inversion of subtyping.

2.4. Queue Subtyping

As we have the ground subtyping $<:$ and ground types in our queues, we can define a subtyping for queues. This is important as when we subtype local types we may use $<:$ to alter the payload types; we will need to subtype projected queues to identify an operational correspondence after local actions. The queue subtyping is defined to be covariant with B i.e. we define the queue subtyping inductively by:

$$\frac{}{\emptyset \leq_a \emptyset} \quad \frac{\sigma_L \leq_a \sigma'_L \quad \sigma_R \leq_a \sigma'_R}{\sigma_L \cdot \sigma_R \leq_a \sigma'_L \cdot \sigma'_R} \quad \frac{B <: B'}{(p, m(B)) \leq_a (p, m(B'))}$$

The projection of a global type onto a participant produces a pairs containing a queue and a local type. Thus in order to use local type subtyping after projecting as in Eq. (6) (§1), we also require this notion of queue subtyping.

3. Operational Semantics of Global and Local Types

In this section, we will introduce semantics for global types and typing contexts. While the typing context semantics are standard [51], our global type semantics (Def. 7) is new and has been designed to reflect the typing context semantics up-to the precise asynchronous subtyping.

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. 6, which are also used in the LTS semantics of typing contexts (discussed later in §3.2).

Definition 6 (Transition Labels). Let α be a transition label of the form:

$$\alpha ::= p : q \& m(B) \quad \bigg| \quad p : q \oplus m(B) \quad (\text{receive or send a message}) \quad (32)$$

The subject of a transition label, written $\text{subject}(\alpha)$, is defined as follows:

$$\text{subject}(p : q \& m(B)) = \text{subject}(p : q \oplus m(B)) = p \quad (33)$$

If $B <: B'$ then we say that $p : q \& m(B') \leq p : q \& m(B)$ and $p : q \oplus m(B) \leq p : q \oplus m(B')$

The label $p:q \oplus m(B)$ denotes that p has enqueued a message with label m and payload of type B onto its queue for q and the label $p:q \& m(B)$ denotes that p has dequeued/received a message with label m and payload of type B from q 's queue.

Definition 7 (Global Type Transitions). The global type transition $\xrightarrow{\alpha}$ is inductively defined by the following rules.

$$\begin{array}{c}
\frac{G[\mu t.G/t] \xrightarrow{\alpha} G'}{\mu t.G \xrightarrow{\alpha} G'} \quad [\text{GR-}\mu] \qquad \frac{}{p \xrightarrow{m} q: \{m(B).G'\} \xrightarrow{q:p \& m(B)} G'} \quad [\text{GR-}\&] \\
\\
\frac{j \in I}{p \rightarrow q: \{m_i(B_i).G'_i\}_{i \in I} \xrightarrow{p:q \oplus m_j(B_j)} p \xrightarrow{m_j} q: \{m_j(B_j).G'_j\}} \quad [\text{GR-}\oplus] \\
\\
\frac{\forall i \in I : G'_i \xrightarrow{\alpha} G''_i \quad \text{subject}(\alpha) \neq p}{p \rightarrow q: \{m_i(B_i).G'_i\}_{i \in I} \xrightarrow{\alpha} p \rightarrow q: \{m_i(B_i).G''_i\}_{i \in I}} \quad [\text{GR-CTX-I}] \\
\\
\frac{\forall i \in J \subseteq I : G'_i \xrightarrow{\alpha} G''_i \quad \alpha = p:r \oplus m'(B') \quad \text{with } r \neq q}{p \rightarrow q: \{m_i(B_i).G'_i\}_{i \in I} \xrightarrow{\alpha} p \rightarrow q: \{m_i(B_i).G''_i\}_{i \in J}} \quad [\text{GR-CTX-I}'] \\
\\
\frac{G' \xrightarrow{\alpha} G'' \quad \alpha \neq q:p \& m'(B')}{p \xrightarrow{m} q: \{m(B).G'\} \xrightarrow{\alpha} p \xrightarrow{m} q: \{m(B).G''\}} \quad [\text{GR-CTX-II}]
\end{array}$$

We use $G \xrightarrow{\alpha} G'$ if there exists α such that $G \xrightarrow{\alpha} G'$; we write $G \rightarrow$ if there exists G' such that $G \xrightarrow{\alpha} G'$, and $G \not\rightarrow$ for its negation (i.e. there is no G' such that $G \rightarrow G'$). Finally, $\rightarrow^*$ denotes the transitive and reflexive closure of $\rightarrow$.

The semantics of global types reflect the behaviours permitted by asynchronous subtyping by allowing specific behaviours to be executed with a type.

- $[\text{GR-}\mu]$ permits a valid transition to take place under a recursion binder.
- $[\text{GR-}\&]$ describes the receiving of asynchronous messages, allowing en-route message to be received.
- $[\text{GR-}\oplus]$ describes the sending of asynchronous messages, resulting in a standard transmission becoming an en-route one.
- $[\text{GR-CTX-I}]$ allows a transition α to be anticipated inside the continuations of a communication $p \rightarrow q$, provided the subject of α is not p . That is, actions by participants other than the sender can proceed independently of p 's pending communication.
- $[\text{GR-CTX-I}']$ handles the case where p itself sends to a *different* recipient $r \neq q$ before completing its communication with q . This mirrors the $B^{(p)}$ reordering in the refinement relation (Def. 5), which permits sends to be moved ahead of sends to other participants. The indexing set may shrink from I to $J \subseteq I$, reflecting that asynchronous subtyping allows some branches to be forgotten when a send is reordered.
- $[\text{GR-CTX-II}]$ applies when a message from p to q is already en route. Here any transition is permitted except q receiving a different message from p , which would violate the ordering of p 's messages to q .

Together, $[\text{GR-CTX-I}]$, $[\text{GR-CTX-I}']$ and $[\text{GR-CTX-II}]$ enable the global type semantics to capture the same safe reorderings that are present in the precise asynchronous subtyping relation.

Example 7 (Non-Determinism in Global Semantics). $[\text{GR-CTX-I}']$ will result in non-determinism i.e. there is G, α , and $G' \neq G''$ with $G \xrightarrow{\alpha} G'$ and $G \xrightarrow{\alpha} G''$.

Consider the following global types:

$$G_{\text{non-det}} = \mu t. p \rightarrow q: \left\{ \begin{array}{l} m_1 . q \rightarrow r: m_1 . t \\ m_2 . q \rightarrow r: m_2 . p \rightarrow r: m \end{array} \right\} \quad (34)$$

$$G' = p \rightarrow q : m_2 . q \rightarrow r : m_2 . p \xrightarrow{m} r : m \quad G'' = \mu t . p \rightarrow q : \left\{ \begin{array}{l} m_1 . q \rightarrow r : m_1 . G' \\ m_2 . q \rightarrow r : m_2 . p \xrightarrow{m} r : m \end{array} \right\} \quad (35)$$

We can derive that $G_{\text{non-det}} \xrightarrow{p:r \oplus m} G'$ and that $G_{\text{non-det}} \xrightarrow{p:r \oplus m} G''$. $G' \neq G''$, so the transition relation is non-deterministic.

This is necessary for $G_{\text{non-det}}$ to capture all behaviours up to asynchronous subtyping. Indeed, suppose we take some deterministic subrelation of the transition relation. We will now be able to associate G with a typing context, which it does not operationally correspond to.

Consider its projections:

$$T_p = \mu t . q \oplus \left\{ \begin{array}{l} m_1 . t \\ m_2 . r \oplus m \end{array} \right\} \quad T_q = \mu t . p \& \left\{ \begin{array}{l} m_1 . r \oplus m_1 . t \\ m_2 . r \oplus m_2 \end{array} \right\} \quad T_r = \mu t . q \& \left\{ \begin{array}{l} m_1 . t \\ m_2 . p \& m \end{array} \right\} \quad (36)$$

We have that

$$r \oplus m . q \oplus m_1 \cdots q \oplus m_1 . q \oplus m_2 \leq_a T_p \quad (37)$$

$\underbrace{\hspace{10em}}_{n+1}$

If $G_{\text{non-det}} \xrightarrow{p:r \oplus m} G'''$, then G''' can perform $p : q \oplus m_1$ at most n times, for some n . This is because the transition has a finite derivation, beyond which we must truncate the possibility for $p : q \oplus m_1$ to occur. If no such transition exists, then operational correspondence fails for this transition relation. As we are assuming that the transition is deterministic, we must take this transition in the simulation of the associated context.

G''' cannot perform the action $p : q \oplus m_1$ $n + 1$ times and so $G_{\text{non-det}}$ does not correspond to the projected context, up to precise asynchronous subtyping.

Therefore, non-determinism is necessary for global types to correspond to projected contexts up to the precise asynchronous subtyping.

There are optimisations of our transition that can limit the non-determinism; we could enforce that maximum $J \subset I$ are taken when they can be shown to exist, but this is not the case in general. For example, $G_{\text{non-det}}$ has no maximum transition for $G_{\text{non-det}} \xrightarrow{p:r \oplus m_1}$, as the redex can be taken to have arbitrary depth.

Example 8 (Semantics of Global Type for Ring Protocol). Consider the global type for the ring-choice protocol (§ 1.2). The asynchronous semantics enable us to apply both $[GR-\oplus]$ transitions, corresponding to sends from p to q and from q to r , before any receive transitions (using $[GR-\&]$) are applied. As we will see later, this particular choice of global transition 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 using $[GR-\oplus]$:

$$G_{\text{ring}} \xrightarrow{p:q \oplus \text{add}} G_{\text{ring}}^{(1)} = p \xrightarrow{\text{add}} q : \text{add}(\text{int}) . q \rightarrow r : \left\{ \begin{array}{l} \text{add}(\text{int}) . r \rightarrow p : \{ \text{add}(\text{int}) . G_{\text{ring}} \} \\ \text{sub}(\text{int}) . r \rightarrow p : \{ \text{sub}(\text{int}) . G_{\text{ring}} \} \end{array} \right\} \quad (38)$$

At this point, a message from p to q is in transit. We then perform another $[GR-\oplus]$ transition, using $[GR-\text{Ctx-II}]$ to apply the sending from q to r under the existing en-route type:

$$G_{\text{ring}}^{(1)} \xrightarrow{q:r \oplus \text{sub}} G_{\text{ring}}^{(2)} = p \xrightarrow{\text{add}} q : \text{add}(\text{int}) . q \xrightarrow{\text{sub}} r : \text{sub}(\text{int}) . r \rightarrow p : \text{sub}(\text{int}) . G_{\text{ring}} \quad (39)$$

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

$$G_{\text{ring}}^{(2)} \xrightarrow{q:p \& \text{add}} G_{\text{ring}}^{(3)} = q \xrightarrow{\text{sub}} r : \text{sub}(\text{int}) . r \rightarrow p : \text{sub}(\text{int}) . G_{\text{ring}} \quad (40)$$

Then, r receives the message from q by $[GR-\&]$:

$$G_{\text{ring}}^{(3)} \xrightarrow{r:q \& \text{sub}} G_{\text{ring}}^{(4)} = r \rightarrow p : \{ \text{sub}(\text{int}) . G_{\text{ring}} \} \quad (41)$$

Next, r sends to p by $[GR-\oplus]$:

$$G_{ring}^{(4)} \xrightarrow{r:p\oplus sub} G_{ring}^{(5)} = r \overset{sub}{\rightsquigarrow} p: \{ \text{sub}(\text{int}) . G_{ring} \} \quad (42)$$

Finally, p receives this last message by $[GR-\&]$, returning us to the original state of the protocol:

$$G_{ring}^{(5)} \xrightarrow{r:p\&sub} G_{ring} \quad (43)$$

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

3.2. Semantics of Typing Contexts

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. 8, followed by its transition rules in Def. 9. We then define safety, freedom, and liveness for contexts in Def. 12.

Definition 8 (Typing Contexts). Δ denotes a partial mapping from participants to queues and types. Their syntax is defined as:

$$\Delta ::= \emptyset \mid \Delta, p: (\sigma, T) \quad (44)$$

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

Definition 9 (Typing Context Transition). The *typing context transition* $\xrightarrow{\alpha}$ is inductively defined by the following rules:

$$\begin{array}{c} \frac{k \in I}{p: (\sigma, q \oplus \{ \mathfrak{m}_i(B_i).T_i \}_{i \in I}) \xrightarrow{p: q \oplus \mathfrak{m}_k(B_k)} p: (\sigma \cdot (q, \mathfrak{m}_k(B_k)), T_k)} \quad [\Delta-\oplus] \\[10pt] \frac{k \in I \quad B' <: B_k}{p: (\sigma, q \& \{ \mathfrak{m}_i(B_i).T_i \}_{i \in I}) \cdot q: ((p, \mathfrak{m}_k(B')) \cdot \sigma', T) \xrightarrow{p: q \& \mathfrak{m}_k(B_k)} p: (\sigma, T_k) \cdot q: (\sigma', T)} \quad [\Delta-\&] \\[10pt] \frac{p: (\sigma, T \{ \mu t.T / t \}) \xrightarrow{\alpha} \Delta'}{p: (\sigma, \mu t.T) \xrightarrow{\alpha} \Delta'} \quad [\Delta-\mu] \quad \frac{\Delta \xrightarrow{\alpha} \Delta'}{\Delta \cdot c: (\sigma, T) \xrightarrow{\alpha} \Delta' \cdot c: (\sigma, T)} \quad [\Delta-\cdot] \end{array}$$

We write $\Delta \xrightarrow{\alpha}$ if there exists Δ' such that $\Delta \xrightarrow{\alpha} \Delta'$. We write $\Delta \rightarrow \Delta'$ iff $\Delta \xrightarrow{\alpha} \Delta'$ for some α and $\Delta \not\rightarrow$ for its negation (i.e. there is no Δ' such that $\Delta \rightarrow \Delta'$), and we denote $\rightarrow^*$ as the reflexive and transitive closure of $\rightarrow$.

Rule $[\Delta-\oplus]$ says that participant p with internal choices I towards q can make the choice $k \in I$ and enqueue the message $\mathfrak{m}_k(B_k)$ for q into its queue, and continue with T_k . Dually, rule $[\Delta-\&]$ says that participant p with external choices I from q can dequeue message $\mathfrak{m}_k(B_k)$ from p 's queue if $k \in I$, and continue with T_k . Rule $[\Delta-\mu]$ unfolds recursive types and rule $[\Delta-\cdot]$ handles enlarging typing contexts.

Example 9 (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.

$$\begin{array}{ll} \Delta_0 = p: (\emptyset, T_p), q: (\emptyset, T_q^{\text{opt}}), r: (\emptyset, T_r) \\ \xrightarrow{p: q \oplus \text{add}(\text{int})} \Delta_1 = p: (((q, \text{add}(\text{int}))), r \& \left\{ \begin{array}{l} \text{add}(\text{int}).T_p \\ \text{sub}(\text{int}).T_p \end{array} \right\}), q: (\emptyset, T_q^{\text{opt}}), r: (\emptyset, T_r) & [\Delta-\oplus, \Delta-\mu, \Delta-\cdot] \\ \xrightarrow{q: r \oplus \text{sub}(\text{int})} \Delta_2 = p: (((q, \text{add}(\text{int}))), r \& \left\{ \begin{array}{l} \text{add}(\text{int}).T_p \\ \text{sub}(\text{int}).T_p \end{array} \right\}), q: (((r, \text{sub}(\text{int}))), p \& \{ \text{add}(\text{int}).T_q^{\text{opt}} \}), r: (\emptyset, T_r) & [\Delta-\oplus, \Delta-\mu, \Delta-\cdot] \\ \xrightarrow{q: p \& \text{add}(\text{int})} \Delta_3 = p: (\emptyset, r \& \left\{ \begin{array}{l} \text{add}(\text{int}).T_p \\ \text{sub}(\text{int}).T_p \end{array} \right\}), q: (((r, \text{sub}(\text{int}))), T_q^{\text{opt}}), r: (\emptyset, T_r) & [\Delta-\&, \Delta-\cdot] \\ \xrightarrow{r: q \& \text{sub}(\text{int})} \Delta_4 = p: (\emptyset, r \& \left\{ \begin{array}{l} \text{add}(\text{int}).T_p \\ \text{sub}(\text{int}).T_p \end{array} \right\}), q: (\emptyset, T_q^{\text{opt}}), r: (\emptyset, p \oplus \{ \text{sub}(\text{int}).T_r \}) & [\Delta-\&, \Delta-\mu, \Delta-\cdot] \\ \xrightarrow{r: p \oplus \text{sub}(\text{int})} \Delta_5 = p: (\emptyset, r \& \left\{ \begin{array}{l} \text{add}(\text{int}).T_p \\ \text{sub}(\text{int}).T_p \end{array} \right\}), q: (\emptyset, T_q^{\text{opt}}), r: (((p, \text{sub}(\text{int}))), T_r) & [\Delta-\oplus, \Delta-\cdot] \\ \xrightarrow{p: r \& \text{sub}(\text{int})} \Delta_0 & [\Delta-\&, \Delta-\cdot] \end{array}$$

Not all contexts describe ‘correct’ protocols. The most obvious errors are label mismatches; a message on the queue may possess a label for $\mathbf{p}$ that is excluded in the possible branches of $\mathbf{p}$ ’s type. We call contexts, where this can never occur, *safe* (Def. 10). A context could fail to be correct by terminating with non-empty queues or non-**end** types; these are *deadlocked* contexts (Def. 11). In §6, we shall see similar behaviours for processes and we can define corresponding session correctness properties, which will be implied by the typing context properties. Namely, session deadlock-freedom and session safety (Def. 29) are implied by context deadlock-freedom and context safety, respectively. A more subtle failure is indefinitely delaying participant behaviour, even under *fair scheduling*. Assuming that we reduce a context so that whenever $\mathbf{p}$ can send a message or receive a message on the queue it will do so, we want to know that every message on the queue will eventually be dequeued and that every local receiving type will eventually dequeue a message. This is liveness (Def. 12) and corresponds with session liveness (Def. 30). In this asynchronous setting, context safety and deadlock-freedom are implied by liveness, so we will only need to prove liveness. We establish in §5.3 that every typing context obtained via projection from a balanced global type is live (Thm. 13), and hence safe and deadlock-free.

Definition 10 (Context Safety). We say that a context Δ is safe iff for all $\Delta \rightarrow^* \Delta'$ if $\Delta'(\mathbf{p}) = ((\mathbf{q}, \mathbf{m}(\mathbf{B})).\sigma, T)$ and $\Delta'(\mathbf{q}) = (\sigma', \mathbf{p} \& \{\mathbf{m}_i(\mathbf{B}_i).T_i\}_{i \in I})$ then there is $j \in I$ such that $\mathbf{m} = \mathbf{m}_j$ and $\mathbf{B} <: \mathbf{B}_j$.

Definition 11 (Context Deadlock-freedom). We say that a context Δ is deadlock-free iff for all $\Delta \rightarrow^* \Delta'$ if $\Delta' \not\rightarrow$ then for all $\mathbf{p} \in \text{dom}(\Delta')$ $\text{unf}(\Delta'(\mathbf{p})) = \mathbf{end}$.

Definition 12 (Context Liveness [29]). Let $I = \{0, 1, 2, \dots, n\}$ for some n or $I = \{0, 1, 2, \dots\}$. Let $\{\Delta_i\}_{i \in I}$ be a sequence of typing contexts such that $\Delta_i \rightarrow \Delta_{i+1}$ for $i, i+1 \in I$. We say that the path $\{\Delta_i\}_{i \in I}$ is fair iff for all $i \in I$:

F1 if $\Delta_i \xrightarrow{\mathbf{p}:\mathbf{q} \oplus \mathbf{m}(\mathbf{B})}$, then there are $k, \mathbf{m}'$, and $\mathbf{B}'$ such that $I \ni k+1 > i$ and $\Delta_k \xrightarrow{\mathbf{p}:\mathbf{q} \oplus \mathbf{m}'(\mathbf{B}')} \Delta_{k+1}$

F2 if $\Delta_i \xrightarrow{\mathbf{p}:\mathbf{q} \& \mathbf{m}(\mathbf{B})}$, then there is k such that $I \ni k+1 > i$ and $\Delta_k \xrightarrow{\mathbf{p}:\mathbf{q} \& \mathbf{m}(\mathbf{B})} \Delta_{k+1}$

We say that the path $\{\Delta_i\}_{i \in I}$ is live iff, for all $i \in I$:

L1 If $\Delta_i(\mathbf{p}) = ((\mathbf{q}, \mathbf{m}(\mathbf{B})).\sigma, T)$, then there exist k and $\mathbf{B}'$ such that $I \ni k+1 > i$ and $\Delta_k \xrightarrow{\mathbf{q}:\mathbf{p} \& \mathbf{m}(\mathbf{B}')} \Delta_{k+1}$

L2 If $\Delta_i(\mathbf{p}) = (\sigma, \mathbf{q} \& \{\mathbf{m}_j(\mathbf{B}_j).T_j\}_{j \in J})$, then there exist $k, \mathbf{m}'$, and $\mathbf{B}'$ such that $I \ni k+1 > i$ and $\Delta_k \xrightarrow{\mathbf{p}:\mathbf{q} \& \mathbf{m}'(\mathbf{B}')} \Delta_{k+1}$

We say that a context Δ is live iff all fair paths starting at Δ are live.

The context Δ_0 from Ex. 9 above is live, which we will be able to show by Theorem 13 later.

Finding non-live contexts merely requires us to witness failures of L1 or L2. We now provide three non-live contexts: one that is unsafe; one that is safe but deadlocked; and one that is safe and deadlock-free.

Example 10 (Failures of Liveness).

Unsafe Consider the context $\mathbf{p} : ((\mathbf{q}, \mathbf{m}), \mathbf{end}), \mathbf{q} : (\emptyset, \mathbf{p} \& \{\mathbf{m}'\})$. This context is not live because the message $(\mathbf{q}, \mathbf{m})$ does not match with $\mathbf{p} \& \{\mathbf{m}'\}$ and so no communication can occur, violating L1 and L2. This is also an example of an unsafe context.

Deadlocked Consider the context $\mathbf{p} : (\emptyset, \mathbf{q} \& \{\mathbf{m}\})$. This context is not live because no communication can occur and hence violating L2. This is also an example of a safe but deadlocked context.

Livelocked Consider the context $\mathbf{p} : (\emptyset, \mu \mathbf{t}. \mathbf{q} \oplus \{\mathbf{m}, \mathbf{t}\}), \mathbf{q} : (\emptyset, \mu \mathbf{t}. \mathbf{p} \& \{\mathbf{m}, \mathbf{t}\}), \mathbf{r} : ((\mathbf{p}, \mathbf{m}'), \mathbf{end})$. This context is safe and deadlock-free as it is non-terminating, but it is livelocked as the message on $\mathbf{r}$ ’s queue will never be dequeued, violating L1.

4. Properties of Global Types and Local Types

This section introduces the constraint that well-formed global types must be balanced⁺ (Def. 17). This is an extension of the standard condition, that well-formed global types must be balanced, to our syntax with en-route transitions. We will show that this condition is preserved by transitions (Thm. 3), so the restriction to balanced⁺ types is respected by our semantics. We additionally prove that this allows us to define an assortment of recursive depth functions (Defs. 13, 15 and 18) and understand their interactions with our semantics (Lems. 4 and 5). Following this, we analyse the algebraic properties of the full coinductive merge and the operational correspondence between sets of local types and their merge (§4.2).

4.1. Balancedness

Balanced global types are well-known to correspond to live protocols in the synchronous setting, but in the asynchronous setting we need additional assumptions. Intuitively, a global type is balanced iff every participant occurs at a bounded depth in every reachable global type. To formally define balancedness, we need to define the depth function.

Definition 13 (Depth Function). For a participant $\mathbf{r}$, we define $\text{depth}_{\mathbf{r}}(G)$ to be a partial function on global types by the following recursive definition: if $\mathbf{r} \notin \text{roles}(G)$ and $\text{fv}(G) = \emptyset$ then $\text{depth}_{\mathbf{r}}(G) = 1$, otherwise

$$\begin{aligned} \text{depth}_{\mathbf{r}}(\mu\mathbf{t}.G) &= \text{depth}_{\mathbf{r}}(G) \\ \text{depth}_{\mathbf{r}}(\mathbf{p} \rightarrow \mathbf{q} : \{ \mathbf{m}_i(\mathbf{B}_i).G_i \}_{i \in I}) &= \begin{cases} 1 & \mathbf{r} \in \{\mathbf{p}, \mathbf{q}\} \\ 1 + \max \{ \text{depth}_{\mathbf{r}}(G_i) : i \in I \} & \text{otherwise} \end{cases} \\ \text{depth}_{\mathbf{r}}(\mathbf{p} \xrightarrow{\mathbf{m}} \mathbf{q} : \{ \mathbf{m}(\mathbf{B}).G \}) &= \begin{cases} 1 & \mathbf{r} = \mathbf{q} \\ 1 + \text{depth}_{\mathbf{r}}(G) & \text{otherwise} \end{cases} \end{aligned}$$

Remark 1. The depth of $\mathbf{r}$ in G is the bound on how deep you can go down the type tree of G before either $\mathbf{r}$ occurs or $\mathbf{r}$ is no longer a participant.

Definition 14 (Balanced Global Types [28, Def 3.3] [55, Def. 4.17]). A global type G is balanced iff for all $G \xrightarrow{*} G'$ and $\mathbf{p} \in \text{roles}(G')$, $\text{depth}_{\mathbf{p}}(G)$ exists.

Example 11 (Balanced).

$$G = \mathbf{p} \rightarrow \mathbf{q} : \left\{ \begin{array}{l} \mathbf{m}_0.\mathbf{q} \rightarrow \mathbf{p} : \mathbf{m}' \\ \mathbf{m}_1.\mathbf{q} \rightarrow \mathbf{p} : \mathbf{m}' \end{array} \right\} \quad G' = \mathbf{p} \rightarrow \mathbf{q} : \left\{ \begin{array}{l} \mathbf{m}_0.\mathbf{r} \rightarrow \mathbf{s} : \mathbf{m}' \\ \mathbf{m}_1.\mathbf{r} \rightarrow \mathbf{s} : \mathbf{m}' \end{array} \right\} \quad G'' = \mathbf{p} \rightarrow \mathbf{q} : \left\{ \begin{array}{l} \mathbf{m}_0.\mathbf{r} \rightarrow \mathbf{s} : \mathbf{m}' \\ \mathbf{m}_1.\mathbf{r} \rightarrow \mathbf{s} : \mathbf{m}'' \end{array} \right\} \quad (45)$$

All three of the above types are balanced, their participants all have depth at most 2 for each reachable global type. Note that G'' cannot be projected onto $\mathbf{r}$, since $\sqcap \{ \mathbf{s} \oplus \{ \mathbf{m}' \}, \mathbf{s} \oplus \{ \mathbf{m}'' \} \} = \emptyset$; balancedness does not imply projectability.

The ring protocol, G_{ring} , is balanced as $\mathbf{p}$, $\mathbf{q}$, and $\mathbf{r}$ are the receivers of messages no more than two communications from the head of G' for all $G_{\text{ring}} \xrightarrow{*} G'$.

Example 12 (Unbalanced).

$$G_1 = \mu\mathbf{t}.\mathbf{p} \rightarrow \mathbf{q} : \left\{ \begin{array}{l} \mathbf{m}_0.\mathbf{t} \\ \mathbf{m}_1.\mathbf{p} \rightarrow \mathbf{r} : \mathbf{m} \end{array} \right\} \quad G_2 = \mu\mathbf{t}.\mathbf{p} \rightarrow \mathbf{q} : \left\{ \begin{array}{l} \mathbf{m}_0.\mathbf{t} \\ \mathbf{m}_1.\mathbf{s} \rightarrow \mathbf{r} : \mathbf{m} \end{array} \right\} \quad (46)$$

$$G'_i = \mathbf{p} \rightarrow \mathbf{q} : \mathbf{m}.\mathbf{s} \rightarrow \mathbf{r} : \mathbf{m}.G_i \quad \text{for } i \in \{0, 1, 2\} \quad (47)$$

All of the above types are not balanced. $\text{depth}_{\mathbf{r}}(G_1)$ does not exist, because there is a path down the unfolding of G_1 , avoiding $\mathbf{r}$, so G_1 is unbalanced; this corresponds to the specified protocol not being live. $\text{depth}_{\mathbf{r}}(G_2)$ and $\text{depth}_{\mathbf{s}}(G_2)$ do not exist, because there is a path down the unfolding of G_1 , avoiding $\mathbf{r}$ and $\mathbf{s}$, so G_2 is unbalanced; this corresponds to the specified protocol not allowing commutativity of independent actions i.e.

$$G_2 \xrightarrow{\mathbf{p} : \mathbf{q} \oplus \mathbf{m}_1} . \xrightarrow{\mathbf{s} : \mathbf{r} \oplus \mathbf{m}} . \quad \text{but } G_2 \not\xrightarrow{\mathbf{s} : \mathbf{r} \oplus \mathbf{m}} . \quad (48)$$

G'_i is not balanced for $i \in \{1, 2\}$, since, although all depths exists at G'_i , $G'_i \xrightarrow{*} G_i$, which is not balanced.

G_1 is projectable and operationally corresponds to its projected context, but does not guarantee liveness. G_2 is projectable but does not correspond to its projected context, since contexts always have commutativity for independent actions. Projectability does not imply balancedness and balancedness is necessary to ensure correctness of the projection.

The positions of en-route transmissions in a global type can cause unexpected behaviour. For example, consider the global type

$$G = p \rightarrow q : m . p \xrightarrow{m'} q : m' \quad (49)$$

The projection of G has (q, m') on p 's queue, and q 's local type is $p \& \{m . p \& \{m'\}\}$, so that the projected context is *unsafe*.

We will introduce an extension of balancedness for asynchronous global types, which forbids this behaviour. Suppose that G has no en-route messages and $G \rightarrow^* G'$. An important observation is that for all p and q , every path down G' encounters the same number of en-route transmissions from p to q . We introduce a recursively defined partial function, $\text{count}_{p,q}(G)$, to formalise this.

Definition 15 (Active Roles). We define $\text{mRoles}(G)$ on global types to be the set of pairs of participants that have an en-route transmission from the first to the second, by the recursive definition: $\text{mRoles}(\text{end}) = \emptyset$; $\text{mRoles}(t) = \emptyset$; $\text{mRoles}(\mu t . G) = \text{mRoles}(G)$; $\text{mRoles}(p \rightarrow q : \{m_i(B_i) . G_i\}_{i \in I}) = \bigcup_{i \in I} \text{mRoles}(G_i)$; and $\text{mRoles}(p \xrightarrow{m} q : \{m(B) . G\}) = \{(p, q)\} \cup \text{mRoles}(G)$

Definition 16 (Counting En-Route Messages). We define $\text{count}_{p,q}(G)$ by the following recursive definition:

$$\begin{aligned} \text{count}_{p,q}(\text{end}) &= 0 \\ \text{count}_{p,q}(t) &= 0 \\ \text{count}_{p,q}(\mu t . G) &= \begin{cases} \text{count}_{p,q}(G) & t \notin \text{fv}(G) \text{ or } \text{count}_{p,q}(G) = 0 \\ \text{undefined} & \text{otherwise} \end{cases} \\ \text{count}_{p,q}(p' \rightarrow q' : \{m_i(B_i) . G_i\}_{i \in I}) &= \begin{cases} \text{count}_{p,q}(G_i) & \text{for all } i, j \in I \\ & \text{count}_{p,q}(G_i) = \text{count}_{p,q}(G_j) \\ & (p, q) \neq (p', q') \\ 0 & (p, q) = (p', q') \notin \bigcup_{i \in I} \text{mRoles}(G_i) \\ \text{undefined} & \text{otherwise} \end{cases} \\ \text{count}_{p,q}(p' \xrightarrow{m} q' : \{m(B) . G\}) &= \begin{cases} \text{count}_{p,q}(G) + 1 & p = p' \text{ and } q = q' \\ \text{count}_{p,q}(G) & \text{otherwise} \end{cases} \end{aligned}$$

En-route messages are projected to queue elements. The existence of $\text{count}_{p,q}(G)$ is necessary for this interpretation to be correct: 1. queues have a finite length, which is assured by the conditions on recursion; 2. queues have a known length, which is assured by the conditions on independent communications; and 3. elements cannot be enqueued prematurely, which is ensured by the condition that the count of en-route transmissions from p to q following a transmission from p to q must be 0. The existence of this function is the necessary condition for global types to be asynchronously meaningful.

Definition 17 (Balanced⁺ Global Types). A global type G is balanced⁺ iff, G is balanced (Def. 14) and for all participants p and q , $\text{count}_{p,q}(G)$ exists. Hereafter we assume well-formed global types satisfy this condition.

Given en-route types are only runtime behaviour, we can impose this additional restriction upon balanced types by Theorem 3.

Remark 2. Checking balancedness is decidable [55], by which the existence of $\text{count}_{p,q}(G)$ is decidable. Therefore, by checking all $p, q \in \text{roles}(G)$, balanced^+ is a decidable property.

We now return to our running example (§ 1.2) to briefly check that it satisfies this condition.

Example 13 (Ring Protocol is Balanced^+). G_{ring} has no en-route transmissions and is balanced, so G_{ring} is balanced^+ .

As $G_{\text{ring}} \xrightarrow{*} p \xrightarrow{\text{add}} q : \text{add}(\text{int}).q \rightarrow r : \left\{ \begin{array}{l} \text{add}(\text{int}).r \rightarrow p : \{ \text{add}(\text{int}).G_{\text{ring}} \} \\ \text{sub}(\text{int}).r \rightarrow p : \{ \text{sub}(\text{int}).G_{\text{ring}} \} \end{array} \right\}$, this type is also balanced^+ .

Of course, not all global types are balanced^+ . We now provide examples for each way that balanced^+ can fail to hold.

Example 14 (Non-Balanced⁺ Types). Any non-balanced type is not balanced^+ . For example,

$$G = \mu t. p \rightarrow q : \left\{ \begin{array}{l} m_1 . t \\ m_2 . p \rightarrow r : m \end{array} \right\} \quad (50)$$

G is not balanced as r has unbounded depth, so G is not balanced^+ .

The more interesting violations of balanced^+ can be observed by occurrences of en-route transmissions. None can occur beneath non-trivial recursive binders and they must appear in types uniformly. Consider the following global types:

$$G_1 = \mu t. p \xrightarrow{m} q : m. t \quad G_2 = \mu t. p \xrightarrow{m} q : m \quad G_3 = p \rightarrow q : m'. p \xrightarrow{m} q : m \quad G_4 = p' \rightarrow q' : \left\{ \begin{array}{l} m_1 . p \rightarrow q : m \\ m_2 . p \xrightarrow{m} q : m \end{array} \right\} \quad (51)$$

Only G_2 is balanced^+ , and none of them can occur as a transition from a global type with no en-route transmissions. G_1 cannot occur as it has an en-route transmission under a recursive binder and followed by a recursive variable, so this global type would suggest that infinitely many messages are queued from p to q . G_2 cannot occur for the same reason, however this type is still meaningful as it is equivalent to $p \xrightarrow{m} q : m$. G_3 cannot occur as global type transitions require the top-most message from p to q to be sent before any later messages can be sent. G_4 cannot occur as the en-route transmission only occurs down one branch, whereas global transitions result in en-route transmissions occurring down every path in the result.

Theorem 3 (Transitions preserve En-Route Counting). *If $\text{count}_{p,q}(G)$ exists and $G \xrightarrow{\alpha} G'$, then $\text{count}_{p,q}(G')$ exists. Therefore, if $G \xrightarrow{*} G'$ and G has no en-route transmissions, then $\text{count}_{p,q}(G')$ exists for all p and q .*

Proof. Suppose that $\text{count}_{p,q}(G)$ exists. Suppose that $G \xrightarrow{\alpha} G'$. We now show that: if $\alpha = p : q \oplus m(B)$, then $\text{count}_{p,q}(G') = \text{count}_{p,q}(G) + 1$; if $\alpha = q : p \& m(B)$, then $\text{count}_{p,q}(G') = \text{count}_{p,q}(G) - 1$; and $\text{count}_{p,q}(G') = \text{count}_{p,q}(G)$ otherwise. We proceed by induction on the derivation of $G \xrightarrow{\alpha} G'$.

[GR- μ] $G = \mu t. G''$ and $G''[\mu t. G''/t] \xrightarrow{\alpha} G'$. $t \notin \text{fv}(G'')$ so $G'' = G''[\mu t. G''/t]$ and $\text{count}_{p,q}(G) = \text{count}_{p,q}(G'')$, so apply the I.H.

[GR- $\&$] $G = p' \xrightarrow{m} q' : \{ m(B). G' \}$, and $\alpha = q' : p' \& m(B)$. If $(p', q') = (p, q)$, $\text{count}_{p,q}(G') = \text{count}_{p,q}(G) - 1$ by definition. Otherwise, $\text{count}_{p,q}(G') = \text{count}_{p,q}(G)$ by definition.

[GR- $\oplus$] $G = p' \rightarrow q' : \{ m_i(B_i). G'_i \}_{i \in I}$, $\alpha = p' : q' \oplus m_j(B)$, and $G' = p' \xrightarrow{m_j} q' : \{ m_j(B_j). G'_j \}$. If $(p', q') = (p, q)$, $\text{count}_{p,q}(G') = \text{count}_{p,q}(G) + 1$ by definition. Otherwise, $\text{count}_{p,q}(G') = \text{count}_{p,q}(G)$ by definition.

[GR-CTX-I(*)] $G = p' \rightarrow q' : \{ m_i(B_i). G'_i \}_{i \in I}$, $G' = p' \rightarrow q' : \{ m_i(B_i). G''_i \}_{i \in J}$ and $G'_i \xrightarrow{\alpha} G''_i$ for $i \in J \subseteq I$. If $\alpha = p : q \oplus m(B)$, then $\text{count}_{p,q}(G'_i) + 1 = \text{count}_{p,q}(G''_i)$ for $i \in J$ by the I.H. so $\text{count}_{p,q}(G) + 1 = \text{count}_{p,q}(G')$. If $\alpha = q : p \& m(B)$, then $\text{count}_{p,q}(G'_i) - 1 = \text{count}_{p,q}(G''_i)$ for $i \in J$ by the I.H. so $\text{count}_{p,q}(G) - 1 = \text{count}_{p,q}(G')$. Otherwise, $\text{count}_{p,q}(G'_i) = \text{count}_{p,q}(G''_i)$ for $i \in J$ by the I.H. so $\text{count}_{p,q}(G) = \text{count}_{p,q}(G')$.

[GR-CTX-II] $G = p' \xrightarrow{m} q' : \{m'(B').G''\}$, $G' = p' \xrightarrow{m} q' : \{m'(B').G'''\}$ and $G'' \xrightarrow{\alpha} G'''$. If $\alpha = p:q \oplus m(B)$, then $\text{count}_{p,q}(G'') + 1 = \text{count}_{p,q}(G''')$ by the I.H. so $\text{count}_{p,q}(G) + 1 = \text{count}_{p,q}(G')$. If $\alpha = q:p \& m(B)$, then $\text{count}_{p,q}(G'') - 1 = \text{count}_{p,q}(G''')$ by the I.H. so $\text{count}_{p,q}(G) - 1 = \text{count}_{p,q}(G')$. Otherwise, $\text{count}_{p,q}(G'') = \text{count}_{p,q}(G''')$ by the I.H. so $\text{count}_{p,q}(G) = \text{count}_{p,q}(G')$.

We now note that for G without en-route transmissions, $\text{count}_{p,q}(G) = 0$ for all p, q . Therefore, if $G \xrightarrow{\alpha} G'$, then $\text{count}_{p,q}(G')$ exists for all p, q . $\square$

Protocols begin with empty queues, thus Thm. 3 says that balanced^+ is no more restrictive than the standard balanced requirement.

We can now use balanced^+ to recursively define a helper function in Def. 18. Later on, we make use of this function to structure inductive proofs using Lem. 4.

Definition 18 (Message Depth Function). For participants p and q , we define $\text{mDepth}_{p,q}(G)$, read as ‘en-route message depth of (p, q) ’, to be a partial function on global types by the following recursive definition:

$$\begin{aligned} \text{mDepth}_{p,q}(\mu t.G) &= \text{mDepth}_{p,q}(G) \\ \text{mDepth}_{p,q}(p' \rightarrow q' : \{m_i(B_i).G_i\}_{i \in I}) &= 1 + \max \{ \text{mDepth}_{p,q}(G_i) : i \in I \} \\ \text{mDepth}_{p,q}(p' \xrightarrow{m} q' : \{m(B).G\}) &= \begin{cases} 1 & p = p' \text{ and } q = q' \\ 1 + \text{mDepth}_{p,q}(G) & \text{otherwise} \end{cases} \end{aligned}$$

Remark 3. Similarly to Remark 1, $\text{mDepth}_{p,q}(G) \leq n$ iff every path of length n down G contains an en-route transmission from p to q .

It is useful to understand how depths interact with global type semantics. Transitions excluding p either remove branches or remove type constructors excluding p , thus decreasing its depth.

Lemma 4 (Depth is decreasing). *Suppose that $G \xrightarrow{\alpha} G'$, then:*

- If $p \neq \text{subject}(\alpha)$ and $\text{depth}_p(G)$ exists, then $\text{depth}_p(G') \leq \text{depth}_p(G)$.
- If α is not of the form $p:q \& m(B)$ and $\text{mDepth}_{p,q}(G)$ exists, then $\text{mDepth}_{p,q}(G') \leq \text{mDepth}_{p,q}(G)$.

Proof. Proceed by induction on the definition of $G \xrightarrow{\alpha} G'$.

Suppose that $\alpha \neq p:q \oplus m$ or $p:q \& m$ for any q , and that $\text{depth}_p(G)$ exists. Suppose that $G \xrightarrow{\alpha} G'$ and consider the cases for the last transition rule:

[GR- μ] $G = \mu t.G''$ and $G''[\mu t.G''/t] \xrightarrow{\alpha} G'$. $\text{depth}_p(G) = \text{depth}_p(G'') = \text{depth}_p(G''[\mu t.G''/t])$ so by the I.H., $\text{depth}_p(G') \leq \text{depth}_p(G''[\mu t.G''/t]) = \text{depth}_p(G)$.

[GR- $\&$] $G = p' \xrightarrow{m} q : \{m(B).G'\}$, and $\alpha = q:p' \& m(B)$. Now, $\text{depth}_p(G) = \text{depth}_p(G') + 1 > \text{depth}_p(G')$.

[GR- $\oplus$] $G = p' \rightarrow q : \{m_i(B_i).G'_i\}_{i \in I}$, $\alpha = p':q \oplus m_j(B_j)$, and $G' = p' \xrightarrow{m_j} q : \{m_j(B_j).G'_j\}$. Now, $\text{depth}_p(G) = \max_{i \in I} \text{depth}_p(G'_i) + 1 \geq \text{depth}_p(G')$ if $p \neq q$ and $\text{depth}_p(G) = \text{depth}_p(G') = 1$ if $p = q$.

[GR-CTX-I] $G = p' \rightarrow q' : \{m_i(B_i).G'_i\}_{i \in I}$, $G' = p' \rightarrow q' : \{m_i(B_i).G''_i\}_{i \in I}$ and $G'_i \xrightarrow{\alpha} G''_i$ for $i \in I$. If $p \in \{p', q'\}$, $\text{depth}_p(G) = \text{depth}_p(G') = 1$. Otherwise, $\text{depth}_p(G'_i) < \text{depth}_p(G)$ for $i \in I$ so by the I.H., $\text{depth}_p(G'_i) \leq \text{depth}_p(G'_i) < \text{depth}_p(G)$ for $i \in I$, so $\text{depth}_p(G') \leq \text{depth}_p(G)$.

[GR-CTX-I'] $G = p' \rightarrow q' : \{m_i(B_i).G'_i\}_{i \in I}$, $G' = p' \rightarrow q' : \{m_i(B_i).G''_i\}_{i \in J}$ and $G'_i \xrightarrow{\alpha} G''_i$ for $i \in J \subseteq I$. If $p \in \{p', q'\}$, $\text{depth}_p(G) = \text{depth}_p(G') = 1$. Otherwise, $\text{depth}_p(G'_i) < \text{depth}_p(G)$ for $i \in J$ so by the I.H., $\text{depth}_p(G'_i) \leq \text{depth}_p(G'_i) < \text{depth}_p(G)$ for $i \in J$, so $\text{depth}_p(G') \leq \text{depth}_p(G)$.

[GR-CTX-II] $G = p' \xrightarrow{m} q' : \{m(B).G''\}$, $G' = p' \xrightarrow{m'} q' : \{m'(B').G'''\}$ and $G'' \xrightarrow{\alpha} G'''$. If $p = q'$, $\text{depth}_p(G) = \text{depth}_p(G') = 1$. Otherwise, $\text{depth}_p(G'') < \text{depth}_p(G)$ so by the I.H., $\text{depth}_p(G''') \leq \text{depth}_p(G'') < \text{depth}_p(G)$, so $\text{depth}_p(G') \leq \text{depth}_p(G)$.

Suppose that α is not of the form $p : q \& m(B)$ and $\text{mDepth}_{p,q}(G)$ exists. Suppose that $G \xrightarrow{\alpha} G'$ and consider the last case of the transition rule:

[GR- μ] $G = \mu t.G''$ and $G''[\mu t.G''/t] \xrightarrow{\alpha} G'$. $\text{mDepth}_{p,q}(G) = \text{mDepth}_{p,q}(G'') = \text{mDepth}_{p,q}(G''[\mu t.G''/t])$ so by the I.H., $\text{mDepth}_{p,q}(G') \leq \text{mDepth}_{p,q}(G''[\mu t.G''/t]) = \text{mDepth}_{p,q}(G)$.

[GR- $\&$] $G = p' \xrightarrow{m} q' : \{m(B).G'\}$, and $\alpha = q' : p' \& m(B)$. So, $p \neq q'$ or $q \neq p'$. Now, $\text{mDepth}_{p,q}(G) = \text{mDepth}_{p,q}(G') + 1 > \text{mDepth}_{p,q}(G')$.

[GR- $\oplus$] $G = p' \rightarrow q' : \{m_i(B_i).G'_i\}_{i \in I}$, $\alpha = p' : q' \oplus m_j(B_j)$, and $G' = p' \xrightarrow{m_j} q' : \{m_j(B_j).G'_j\}$. Now, $\text{mDepth}_{p,q}(G) = \text{mDepth}_{p,q}(G')$ if $(p', q') \neq (p, q)$ and $\text{mDepth}_{p,q}(G') = 1 \leq \text{mDepth}_{p,q}(G)$ otherwise.

[GR-CTX-I] $G = p' \rightarrow q' : \{m_i(B_i).G'_i\}_{i \in I}$, $G' = p' \rightarrow q' : \{m_i(B_i).G''_i\}_{i \in I}$ and $G'_i \xrightarrow{\alpha} G''_i$ for $i \in I$. $\text{mDepth}_{p,q}(G'_i) < \text{mDepth}_{p,q}(G)$ for $i \in I$ so by the I.H., $\text{mDepth}_{p,q}(G''_i) \leq \text{mDepth}_{p,q}(G'_i) < \text{mDepth}_{p,q}(G)$ for $i \in I$, so $\text{mDepth}_{p,q}(G') \leq \text{mDepth}_{p,q}(G)$.

[GR-CTX-I'] $G = p' \rightarrow q' : \{m_i(B_i).G'_i\}_{i \in I}$, $G' = p' \rightarrow q' : \{m_i(B_i).G''_i\}_{i \in J}$ and $G'_i \xrightarrow{\alpha} G''_i$ for $i \in J \subseteq I$. $\text{mDepth}_{p,q}(G'_i) < \text{mDepth}_{p,q}(G)$ for $i \in J$ so by the I.H., $\text{mDepth}_{p,q}(G''_i) \leq \text{mDepth}_{p,q}(G'_i) < \text{mDepth}_{p,q}(G)$ for $i \in J$, so $\text{mDepth}_{p,q}(G') \leq \text{mDepth}_{p,q}(G)$.

[GR-CTX-II] $G = p' \xrightarrow{m'} q' : \{m'(B').G''\}$, $G' = p' \xrightarrow{m'} q' : \{m'(B').G'''\}$ and $G'' \xrightarrow{\alpha} G'''$. If $p = p'$ and $q = q'$, $\text{mDepth}_{p,q}(G) = \text{mDepth}_{p,q}(G') = 1$. Otherwise, $\text{mDepth}_{p,q}(G'') < \text{mDepth}_{p,q}(G)$ so by the I.H., $\text{mDepth}_{p,q}(G''') \leq \text{mDepth}_{p,q}(G'') < \text{mDepth}_{p,q}(G)$, so $\text{mDepth}_{p,q}(G') \leq \text{mDepth}_{p,q}(G)$.

□

We need to proceed by induction on $\text{mDepth}_{p,q}(G)$ when trying to invert a projection onto q with a receive from p at the head and a message from p to q on the queue. We know that the $\text{depth}_p(G)$ exists for all $p \in \text{roles}(G)$ by balancedness; we would like a similar statement for mDepth and balanced^+ . Balanced^+ ensures the existence of $\text{mDepth}_{p,q}(G)$ for all $(p, q) \in \text{mRoles}(G)$.

Lemma 5 (En Route Transmissions are Bounded). *If G is balanced^+ , then for $(p, q) \in \text{mRoles}(G)$, $\text{mDepth}_{p,q}(G)$ exists.*

Proof. We show that if $\text{count}_{p,q}(G) > 0$, then $\text{mDepth}_{p,q}(G)$ exists by induction on the structure of G .

- If $G = \text{end}$, then $\text{count}_{p,q}(G) = 0$.
- If $G = t$, then $\text{count}_{p,q}(G) = 0$. If $G = \mu t.G'$ and $\text{count}_{p,q}(G) > 0$, then $\text{count}_{p,q}(G') > 0$ so $\text{mDepth}_{p,q}(G')$ exists by I.H., so $\text{mDepth}_{p,q}(G) = \text{mDepth}_{p,q}(G')$.
- If $G = p' \rightarrow q' : \{m_i(B_i).G'_i\}_{i \in I}$, then $\text{count}_{p,q}(G_i) > 0$ for $i \in I$, so $\text{mDepth}_{p,q}(G_i)$ exists for $i \in I$ by the I.H., so $\text{mDepth}_{p,q}(G) = 1 + \max_{i \in I} \text{mDepth}_{p,q}(G_i)$.
- If $G = p' \xrightarrow{m} q' : \{m(B).G'\}$ and $(p, q) \neq (p', q')$, then $\text{count}_{p,q}(G') > 0$, so $\text{mDepth}_{p,q}(G')$ exists by the I.H., so $\text{mDepth}_{p,q}(G) = 1 + \text{mDepth}_{p,q}(G')$.
- If $G = p \xrightarrow{m} q : \{m(B).G'\}$, then $\text{mDepth}_{p,q}(G) = 1$.

Next we show that if $\text{count}_{p,q}(G)$ exists and $(p, q) \in \text{mRoles}(G)$, then $\text{count}_{p,q}(G) > 0$. We proceed by induction on the structure of G .

- If $G = \mathbf{end}$, then $(p, q) \notin \mathbf{mRoles}(G)$.
- If $G = \mathbf{t}$, then $(p, q) \notin \mathbf{mRoles}(G)$.
- If $G = \mu \mathbf{t}.G'$ then $(p, q) \in \mathbf{mRoles}(G')$ and $\mathbf{count}_{p,q}(G')$ exists, so by I.H. $\mathbf{count}_{p,q}(G') > 0$, so $\mathbf{count}_{p,q}(G) = \mathbf{count}_{p,q}(G') > 0$.
- If $G = p' \rightarrow q' : \{\mathbf{m}_i(B_i).G_i\}_{i \in I}$, then $(p, q) \in \mathbf{mRoles}(G_i)$ and $\mathbf{count}_{p,q}(G_i)$ exists for $i \in I$, so by I.H. $\mathbf{count}_{p,q}(G_i) > 0$ for $i \in I$, so $\mathbf{count}_{p,q}(G) > 0$.
- If $G = p' \xrightarrow{\mathbf{m}} q' : \{\mathbf{m}(B).G'\}$ and $(p, q) \neq (p', q')$, then $(p, q) \in \mathbf{mRoles}(G')$ and $\mathbf{count}_{p,q}(G')$ exists, so by I.H. $\mathbf{count}_{p,q}(G') > 0$, so $\mathbf{count}_{p,q}(G) > 0$.
- If $G = p \xrightarrow{\mathbf{m}} q : \{\mathbf{m}(B).G'\}$, then $\mathbf{count}_{p,q}(G) \geq 1$.

Now, if G is balanced⁺ then, for $(p, q) \in \mathbf{mRoles}(G)$, $\mathbf{count}_{p,q}(G)$ exists, so $\mathbf{count}_{p,q}(G) > 1$, so $\mathbf{mDepth}_{p,q}(G)$ exists. $\square$

4.2. Properties of Full Coinductive Merging

In order to use the full coinductive projection, we first study basic properties of the full coinductive merge. Following from previous work on projections, we prove an operational correspondence between a merged type and its components (Lemma 7) and we prove that mergeability respects set inclusions (Lemma 10). Additionally, previous work on projections makes use of the standard subtyping (Def. 20) in their operational correspondence results. The standard subtyping provides an operational correspondence, which the precise asynchronous subtyping lacks; this makes the standard subtyping more useful in determining typing context properties. Once we define the standard subtyping, we shall show that merging produces a subtype (Lemma 8) and preserves subtyping (Lemma 9).

We will encounter merges being done iteratively. Under the view that merges produce witnesses to compatibility, surely we can collapse these repeated applications to a single one. This property is associativity.

Lemma 6 (Associativity of Merge). *Full coinductive merge is associative i.e. if $\bigcap_{i \in I_j} T_i \ni T'_j$ for all $j \in J$, then $\bigcap_{j \in J, i \in I_j} T_i = \bigcap_{j \in J} T'_j$.*

Proof. Consider

$$\mathfrak{R} = \bigcap \cup \left\{ (\mathcal{T}, T) : \begin{array}{c} \bigcup_{i \in I} \mathcal{T}_i = \mathcal{T} \\ \forall i \in I, \text{merge} \langle \mathcal{T}_i, T'_i \rangle \\ \text{merge} \langle \{T'_i : i \in I\}, T \rangle \end{array} \right\} \cup \left\{ (\{T'_i : i \in I\}, T) : \begin{array}{c} \bigcup_{i \in I} \mathcal{T}_i = \mathcal{T} \\ \forall i \in I, \text{merge} \langle \mathcal{T}_i, T'_i \rangle \\ \text{merge} \langle \mathcal{T}, T \rangle \end{array} \right\} \quad (52)$$

This satisfies the coinductive rules defining full merge, and so $\mathfrak{R} = \mathbf{merge}$. Suppose that $\{T_i\}_{i \in I} \mathfrak{R} T$. If $\mathbf{merge} \langle \{T_i\}_{i \in I}, T \rangle$, then:

$[\sqcap\text{-end}]$ If $\mathbf{unf}(T) = \mathbf{end}$, then $\mathbf{unf}(T_i) = \mathbf{end}$ for all $i \in I$

$[\sqcap\text{-}\oplus]$ If $\mathbf{unf}(T) = \mathbf{q} \oplus \{\mathbf{m}_j(B_j).T'_j\}_{j \in J}$, then $\mathbf{unf}(T_i) = \mathbf{q} \oplus \{\mathbf{m}_j(B_j).T_{i,j}\}_{j \in J}$ for all $i \in I$ where for all $j \in J$ $\mathbf{merge} \langle \{T_{i,j} : i \in I\}, T'_j \rangle$ so $\{T_{i,j} : i \in I\} \mathfrak{R} T'_j$

$[\sqcap\text{-}\&]$ If $\mathbf{unf}(T) = \mathbf{p} \& \{\mathbf{m}_j(B_j).T'_j\}_{j \in J}$, then $\mathbf{unf}(T_i) = \mathbf{p} \& \{\mathbf{m}_j(B_j).T_{i,j}\}_{j \in J_i}$ for all $i \in I$ where $J = \bigcup_{i \in I} J_i$ and for all $j \in J$ $\mathbf{merge} \langle \{T_{i,j} : j \in J_i\}, T'_j \rangle$ so $\{T_{i,j} : j \in J_i\} \mathfrak{R} T'_j$

If $\{I_j\}_{j \in J}$ is a cover of I of non-empty sets and for $j \in J$ there is T'_j with $\mathbf{merge} \langle \{T_i\}_{i \in I_j}, T'_j \rangle$ and $\mathbf{merge} \langle \{T'_j\}_{j \in J}, T \rangle$, then:

[$\sqcap$ -end] If $\text{unf}(T) = \mathbf{end}$, then $\text{unf}(T'_j) = \mathbf{end}$ for all $j \in J$, then $\text{unf}(T_i) = \mathbf{end}$ for all $i \in I$.

[$\sqcap$ - $\oplus$] If $\text{unf}(T) = \mathbf{q} \oplus \{\mathbb{m}_k(B_k).T''_k\}_{k \in K}$, then $\text{unf}(T'_j) = \mathbf{q} \oplus \{\mathbb{m}_k(B_k).T''_{j,k}\}_{k \in K}$ for all $j \in J$ where for all $k \in K$ merge $\left\langle \{T''_{j,k} : j \in J\}, T''_k \right\rangle$. So for all $j \in J$, $\text{unf}(T_i) = \mathbf{q} \oplus \{\mathbb{m}_k(B_k).T''_{i,k}\}_{k \in K}$ for all $i \in I_j$ where for all $k \in K$ merge $\left\langle \{T''_{i,k} : i \in I_j\}, T''_{j,k} \right\rangle$. Therefore, $\{T''_{i,k}\}_{i \in I} \mathfrak{R} T''_k$.

[$\sqcap$ - $\&$] If $\text{unf}(T) = \mathbf{p} \& \{\mathbb{m}_k(B_k).T''_k\}_{k \in K}$, then $\text{unf}(T'_j) = \mathbf{p} \& \{\mathbb{m}_k(B_k).T''_{j,k}\}_{k \in K_j}$ for all $j \in J$ where $K = \bigcup_{j \in J} K_j$ and for all $k \in K$ merge $\left\langle \{T''_{j,k} : k \in K_j\}, T''_k \right\rangle$. So for all $j \in J$, $\text{unf}(T_i) = \mathbf{p} \& \{\mathbb{m}_k(B_k).T''_{i,k}\}_{k \in K_i}$ for all $i \in I_j$ where $K_j = \bigcup_{i \in I_j} K_i$ and for all $k \in K_j$ merge $\left\langle \{T''_{i,k} : k \in K_i, i \in I_j\}, T''_{j,k} \right\rangle$. Therefore, $\{T''_{i,k} : k \in K_i\} \mathfrak{R} T''_k$.

If merge $\left\langle \{T'_j\}_{j \in J_i}, T_i \right\rangle$ for $i \in I$ and merge $\left\langle \{T'_j\}_{j \in J_i, i \in I}, T \right\rangle$, then:

[$\sqcap$ -end] If $\text{unf}(T) = \mathbf{end}$, then $\text{unf}(T'_j) = \mathbf{end}$ for all $j \in J_i$ and $i \in I$. So, $\text{unf}(T_i) = \mathbf{end}$ for all $i \in I$.

[$\sqcap$ - $\oplus$] If $\text{unf}(T) = \mathbf{q} \oplus \{\mathbb{m}_k(B_k).T''_k\}_{k \in K}$, then $\text{unf}(T'_j) = \mathbf{q} \oplus \{\mathbb{m}_k(B_k).T''_{j,k}\}_{k \in K}$ for all $j \in J_i$ and $i \in I$ where for all $k \in K$ merge $\left\langle \{T''_{j,k} : j \in J_i, i \in I\}, T''_k \right\rangle$. So for $i \in I$, $\text{unf}(T_i) = \mathbf{q} \oplus \{\mathbb{m}_k(B_k).T''_{i,k}\}_{k \in K}$ where for all $k \in K$ merge $\left\langle \{T''_{j,k} : j \in J_i\}, T''_{i,k} \right\rangle$. Therefore, $\{T''_{j,k} : j \in J_i, i \in I\} \mathfrak{R} T''_k$.

[$\sqcap$ - $\&$] If $\text{unf}(T) = \mathbf{p} \& \{\mathbb{m}_k(B_k).T''_k\}_{k \in K}$, then $\text{unf}(T'_j) = \mathbf{p} \& \{\mathbb{m}_k(B_k).T''_{j,k}\}_{k \in K_j}$ for all $j \in J_i$ and $i \in I$ where $K = \bigcup_{j \in J} K_j$ and for all $k \in K$ merge $\left\langle \{T''_{j,k} : k \in K_j\}, T''_k \right\rangle$. So for $i \in I$, $\text{unf}(T_i) = \mathbf{p} \& \{\mathbb{m}_k(B_k).T''_{i,k}\}_{k \in K_i}$ where $K_i = \bigcup_{j \in J_i} K_j$ and for all $k \in K_j$ merge $\left\langle \{T''_{j,k} : k \in K_j, j \in J_i\}, T''_{i,k} \right\rangle$. Therefore, $\{T''_{i,k} : k \in K_i\} \mathfrak{R} T''_k$.

□

In the literature, the typing context transition relation is sometimes built up from a local type transition relation, using additional rules for queue manipulation and inter-participant communication. We instead define context transitions directly (Def. 9), which streamlines our proofs by avoiding this layered construction. A separate transition relation on local types alone remains useful when reasoning about syntactic properties of local types such as standard subtyping of local types alone without queues. We therefore introduce local type LTSs in order to state an operational correspondence for merge (Lem. 7), which we use in turn to relate merging to standard subtyping (Lems. 8 and 9).

Definition 19 (Local type LTSs). We define the local type LTSs, denoted by $T \xrightarrow{\alpha} T'$ by the following rules:

$$\frac{T[\mu\mathbf{t}.T/\mathbf{t}] \xrightarrow{\alpha} T'}{\mu\mathbf{t}.T \xrightarrow{\alpha} T'} \quad [\text{LR-}\mu] \quad \frac{j \in I}{\mathbf{p} \oplus \{\mathbb{m}_i(B_i).T_i\}_{i \in I} \xrightarrow{\mathbf{p} \oplus \mathbb{m}_j(B_j)} T_j} \quad [\text{LR-}\oplus] \quad \frac{j \in I}{\mathbf{p} \& \{\mathbb{m}_i(B_i).T_i\}_{i \in I} \xrightarrow{\mathbf{p} \& \mathbb{m}_j(B_j)} T_j} \quad [\text{LR-}\&]$$

Lemma 7 (Merge has an Operational Correspondence). • If $T_i \xrightarrow{\alpha} T'_i$ for all $i \in I$ and $\sqcap \{T_i : i \in I\} \ni T$, then there exists T' such that $T \xrightarrow{\alpha} T'$ and $\sqcap \{T'_i : i \in I\} \ni T'$.

• If $\sqcap \{T_i : i \in I\} \ni T$ and $T \xrightarrow{\alpha} T'$, then there is some non-empty $J \subseteq I$ such that $T_i \xrightarrow{\alpha} T'_i$ for $i \in J$ and $\sqcap \{T'_i : i \in J\} \ni T'$.

Proof. Suppose that $\text{merge} \langle \{T_i : i \in I\}, T \rangle$.

- Suppose that $T_i \xrightarrow{p \oplus m(B)} T'_i$ for all $i \in I$. So, for $i \in I$, $\text{unf}(T_i) = p \oplus \left\{ m_{i,j}(B_{i,j}).T'_{i,j} \right\}_{j \in J_i}$ and there is $j \in J$ with $m_{i,j} = m$, $B_{i,j} = B$, and $T'_{i,j} = T'_i$. $\text{merge} \langle \{T_i : i \in I\}, T \rangle$, so we may assume that $J_i = J$, $m_{i,j} = m_j$, and $B_{i,j} = B_j$ for all $j \in J$. Say $k \in J$ has $m_k = m$, $B_k = B$, and $T'_{i,k} = T'_i$. We also have that $\text{unf}(T) = p \oplus \left\{ m_j(B_j).T''_j \right\}_{j \in J}$, where $\text{merge} \langle \{T'_{i,j} : i \in I\}, T''_j \rangle$ for $j \in J$. So, $T \xrightarrow{p \oplus m(B)} T''_k$ and $\text{merge} \langle \{T'_i : i \in I\}, T''_k \rangle$. The proof in the case that $T_i \xrightarrow{p \& m(B)} T'_i$ for all $i \in I$ is similar.
- Suppose that $T \xrightarrow{p \oplus m(B)} T'$. So, $\text{unf}(T) = p \oplus \left\{ m_j(B_j).T'_j \right\}_{j \in J}$ and there is $k \in J$ such that $m_k = m$, $B_k = B$, and $T'_k = T'$. $\text{merge} \langle \{T_i : i \in I\}, T \rangle$ so for $i \in I$, $\text{unf}(T_i) = p \oplus \left\{ m_j(B_j).T'_{i,j} \right\}_{j \in J}$, where $\text{merge} \langle \{T'_{i,j} : i \in I\}, T'_j \rangle$ for $j \in J$. So, $T_i \xrightarrow{p \oplus m(B)} T'_{i,k}$ for $i \in I$, $\text{merge} \langle \{T'_{i,k} : i \in I\}, T' \rangle$, and $\emptyset \neq I \subseteq I$.
- Suppose that $T \xrightarrow{p \& m(B)} T'$. So, $\text{unf}(T) = p \& \left\{ m_j(B_j).T'_j \right\}_{j \in J}$ and there is $k \in J$ such that $m_k = m$, $B_k = B$, and $T'_k = T'$. $\text{merge} \langle \{T_i : i \in I\}, T \rangle$ so there are $J_j \subseteq I$, non-empty, for $j \in J$ such that for $i \in I$, $\text{unf}(T_i) = p \oplus \left\{ m_j(B_j).T'_{i,j} \right\}_{j : i \in J_j}$, where $\text{merge} \langle \{T'_{i,j} : i \in J_j\}, T'_j \rangle$ for $j \in J$. So, $T_i \xrightarrow{p \oplus m(B)} T'_{i,k}$ for $i \in J_k$, $\text{merge} \langle \{T'_{i,k} : i \in J_k\}, T' \rangle$, and $\emptyset \neq J_k \subseteq I$.

□

We shall now define the standard subtyping relation $\leq$ [10, 13, 19, 45], on local types, which is precise in the synchronous setting. This standard subtyping is contained within the precise asynchronous subtyping. We will use the standard subtyping during our proof of safety, deadlock-freedom, and liveness, as there is an operational correspondence between instances of the standard subtyping. In fact, the operational correspondences from Lemma 7 will be sufficient to show that merge behaves well with regard to the standard subtyping.

Definition 20 (Subtyping). The *session subtyping relation* $\leq$ is coinductively defined:

$$\begin{array}{c}
 \frac{\forall i \in I \quad B_i <: B'_i \quad T_i \leq T'_i}{p \& \{m_i(B'_i).T_i\}_{i \in I \cup J} \leq p \& \{m_i(B_i).T'_i\}_{i \in I}} \text{[SUB-}\&\text{]} \quad \frac{T[\mu t.T/t] \leq T'}{\mu t.T \leq T'} \text{[SUB-}\mu\text{L]} \quad \frac{T \leq T'[\mu t.T'/t]}{T \leq \mu t.T'} \text{[SUB-}\mu\text{R]} \\
 \\
 \frac{}{\text{end} \leq \text{end}} \text{[SUB-end]} \quad \frac{\forall i \in I \quad B_i <: B'_i \quad T_i \leq T'_i}{p \oplus \{m_i(B_i).T_i\}_{i \in I} \leq p \oplus \{m_i(B'_i).T'_i\}_{i \in I \cup J}} \text{[SUB-}\oplus\text{]}
 \end{array}$$

Note that the definition of $\leq$ is far simpler than $\leq_a$, and this reflects in the fact that $\leq$ is computable while $\leq_a$ is not [8, 37]. The intuition behind $\leq$ is that if you expect a type to send a message from a set of messages, then a type sending a message from a smaller set will be permissible, and similarly if you expect a type to be able to receive a message from a set of messages, then a type receiving a message from a larger set will be permissible. In fact, $\leq$ is sound and complete with respect to local type transitions, if we enforce that subtypes can always send fewer messages, subtypes can always receive more messages, and **end** types (up-to unfolding) are only related to other **end** types. It is now important to note that the precise asynchronous subtyping does not satisfy this operation correspondence.

Example 15 (Precise Asynchronous Subtyping does not provide an Operational Correspondence). Recall the projection of the ring protocol onto q and its optimisation.

$$T_q = \mu t. p \& \{ \text{add}(\text{int}).r \oplus \{ \text{add}(\text{int}).t, \text{sub}(\text{int}).t \} \} \quad (53)$$

$$T_q^{\text{opt}} = \mu t. r \oplus \{ \text{add}(\text{int}).p \& \{ \text{add}(\text{int}).t \}, \text{sub}(\text{int}).p \& \{ \text{add}(\text{int}).t \} \} \quad (54)$$

We have that $T_q^{\text{opt}} \leq_a T_q$, $T_q \xrightarrow{p \& \text{add}(\text{int})}$, and $T_q^{\text{opt}} \xrightarrow{r \oplus \text{add}(\text{int})}$, but $T_q \not\xrightarrow{r \oplus \text{add}(\text{int})}$ and $T_q^{\text{opt}} \not\xrightarrow{p \& \text{add}(\text{int})}$. Thus, although $T_q^{\text{opt}} \leq_a T_q$, neither type can match the other's immediate transition, so $\leq_a$ does not provide the operational correspondence enjoyed by standard subtyping.

Lemma 8 (Merge is a Subtype). *If $\prod \{T_i : i \in I\} \ni T$, then for all $j \in I$ $T \leq T_j$.*

Proof. Consider $\mathfrak{R} = \{(T, T') : \exists \mathcal{T} \ni T', \text{merge} \langle \mathcal{T}, T \rangle\}$. This clearly satisfies the laws for subtyping by Lem. 7, so $\mathfrak{R} \subseteq \leq$. $\square$

Lemma 9 (Merge preserves Subtypes). *If $\prod \{T_i : i \in I\} \ni T$ and $T' \leq T_i$ for all $i \in I$, then $T' \leq T$.*

Proof. Consider $\mathfrak{R} = \{(T, T') : \exists \mathcal{T}, \text{merge} \langle \mathcal{T}, T' \rangle \text{ and } T \leq T'' \text{ for all } T'' \in \mathcal{T}\}$. By Lem. 7, this clearly satisfies the laws for subtyping, so $\mathfrak{R} \subseteq \leq$. $\square$

A set of local types is mergeable if they are all ‘compatible’. We should expect this ‘compatibility’ property to be decreasing (with respect to subsets).

Lemma 10 (Mergeability respects Set Inclusion). *If $\{T_i : i \in I\}$ is mergeable, then $\{T_i : i \in J\}$ is mergeable for all non-empty $J \subseteq I$.*

Proof. Suppose that $\prod \{T_i : i \in I\} \ni T$. We construct an LTS as follows:

1. Start at node $(T, T_i : i \in I)$.
2. At node $(T', T'_i : i \in I')$, if $T' \xrightarrow{\alpha} T''$, then let $I'' = \{i \in I : \exists T''_i, T'_i \xrightarrow{\alpha} T''_i\}$. Add an edge from $(T', T'_i : i \in I')$ to $(T'', T''_i : i \in I'')$ labelled by α .

Now, let $J \subseteq I$ be non-empty. Remove all nodes, $(T', T'_i : i \in I')$, with $I' \cap J = \emptyset$. This graph is a correct type graph, so let $\mathbb{U}$ be the corresponding type. We relabel the nodes.

1. The root node is relabelled $(\mathbb{U}, T_i : i \in J)$.
2. If there is an edge from $(\mathbb{U}', T'_i : i \in J')$ to $(T'', T''_i : i \in I'')$ with α , then there is $\mathbb{U}''$ such that $\mathbb{U}' \xrightarrow{\alpha} \mathbb{U}''$. Relabel $(T'', T''_i : i \in I'')$ to $(\mathbb{U}'', T''_i : i \in I'' \cap J)$.

The nodes of this graph satisfy the coinductive rules of merge , so $\text{merge} \langle \{T_i : i \in J\}, \mathbb{U} \rangle$. $\square$

5. Global and Local Type Asynchronous Association

Following the introduction of LTS semantics for global types (Def. 7) and typing contexts (Def. 9), we establish a relationship between these two semantics using the projection relation $\downarrow_p$ (Def. 4) and the subtyping relation $\leq_a$ (Def. 5). By analysing the structural consequences (§5.1) of asynchronous subtyping and projection, we derive an operational correspondence from this relation (Thms. 11 and 12). Additionally, we use this relation and the structural consequences of projection to guarantee safety, deadlock-freedom, and liveness (Thm. 13).

Definition 21 (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 contains projections of G : $\text{dom}(\Delta) \supseteq \text{roles}(G)$; and $\forall p \in \text{roles}(G) \exists T_p, \sigma_p$ such that $\Delta(p) = (\sigma'_p, T'_p)$ and $T'_p \leq_a T_p$ and $\sigma'_p \leq_a \sigma_p$ and $G \downarrow_p(\sigma_p, T_p)$.

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 p , its corresponding entry in the typing context $(\Delta(p))$ must be a subtype (Def. 5) of the projection of the global type onto this role. Looking again at the ring protocol example (§1.2), we can observe how the transitions 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.

$$\begin{array}{ccccccccccc}
 G_{\text{ring}} & \xrightarrow{p:q \oplus \text{add}} & G_{\text{ring}}^{(1)} & \xrightarrow{q:r \oplus \text{sub}} & G_{\text{ring}}^{(2)} & \xrightarrow{q:p \& \text{add}} & G_{\text{ring}}^{(3)} & \xrightarrow{r:q \& \text{sub}} & G_{\text{ring}}^{(4)} & \xrightarrow{r:p \oplus \text{sub}} & G_{\text{ring}}^{(5)} \\
 \downarrow \tilde{\cdot} & & \downarrow \tilde{\cdot} & & \downarrow \tilde{\cdot} & & \downarrow \tilde{\cdot} & & \downarrow \tilde{\cdot} & & \downarrow \tilde{\cdot} \\
 \Delta_0 & \xrightarrow{p:q \oplus \text{add}} & \Delta_1 & \xrightarrow{q:r \oplus \text{sub}} & \Delta_2 & \xrightarrow{q:p \& \text{add}} & \Delta_3 & \xrightarrow{r:q \& \text{sub}} & \Delta_4 & \xrightarrow{r:p \oplus \text{sub}} & \Delta_5
 \end{array} \tag{55}$$

This idea is illustrated through two main theorems: Thm. 12 shows that the reducibility of a global type aligns with that of its associated typing context; while Thm. 11 illustrates that each possible transition of a typing context is simulated by an action in the transitions of the associated global type. The following two subsections (§5.1 and 5.2) are dedicated to the proofs of these results.

Theorem 11 (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' and α' such that $\alpha \leq \alpha'$, $\Delta' \sqsubseteq_a G'$, and $G \xrightarrow{\alpha'} G'$.*

Theorem 12 (Soundness of Association). *Let $\Delta \sqsubseteq_a G$ and assume $G \xrightarrow{\alpha} G'$. Then there exist actions $\alpha' \leq \alpha''$, a context Δ' , and a global type G'' such that $G \xrightarrow{\alpha''} G''$, $\Delta \xrightarrow{\alpha'} \Delta'$, and $\Delta' \sqsubseteq_a G''$.*

Remark 4 (Thms. 11 and 12 from the Perspective of Processes). In Thms. 29 and 31, we shall see that Thms. 11 and 12 are analogues for subject reduction and session fidelity, respectively, with typing contexts and global types taking the place of sessions and typing contexts. In both cases, Thms. 11 and 12 bridge the gap between typing sessions with contexts and typing them with global types.

Remark 5 (Sufficiency of Soundness). As in [36], our soundness condition necessitates that we change our transition label. However, unlike in the synchronous case, we can no longer guarantee that the participants remain the same. In the synchronous setting, subtyping will allow for fewer selections in associated contexts than in global types, meaning that the global transition label may use a selection label that is absent locally, forcing us to change to a label that is present. In the asynchronous setting this can still occur, but now the asynchronous subtyping will also prevent us from transitioning on the same participants locally. For example, if the global type has a branching label from p to q (corresponding to a head transition and so reflected in the projected context), then the asynchronous subtyping may have moved a branching from r to q to the head of the context's type for q , preventing q from performing any action. This behaviour can be witnessed below in Eq. (56).

$$p : ((q, m), \text{end}), q : (\emptyset, r \& \{m'. p \& \{m\}\}), r : (\emptyset, q \oplus \{m'\}) \sqsubseteq_a p \xrightarrow{m} q : \{m. r \rightarrow q : \{m'\}\} \quad (56)$$

Before we can use the association to define the top-down typing system as a bottom-up typing system, we must ensure that associated contexts are safe, deadlock-free, and live.

Theorem 13 (Associated Context Properties). *If $\Delta \sqsubseteq_a G$, then Δ is safe, deadlock-free, and live.*

Section 5.3 is dedicated to the proof of this result.

5.1. Structure of Asynchronous Subtyping

The issue that we introduce by using asynchronous subtyping, is that local asynchronous transitions, with participant p , need not correspond to the shallowest occurrences of p in the corresponding global type. We follow Ghilezan et al. [28] to resolve this issue, by introducing local and global type contexts, where the holes correspond to redexes that are reduced during asynchronous communication. However, our choice of global type transition has introduced non-determinism, which we must resolve when identifying local transitions to global transitions. Our solution is to introduce ‘forgetful holes’, $\langle \cdot \rangle_i$, both locally and globally; these holes will denote where the asynchronous subtyping forgets selections, so that we can identify which branches to forget when making global transitions.

Remark 6 (Inductive Contexts over Coinductive Types). The contexts defined below (Defs. 22 and 24) are *inductively* defined and do not include a case for recursive types. This is because they operate on the *tree representations* $\mathfrak{T}(G)$ and $\mathfrak{T}(T)$, which are coinductive (potentially infinite) trees obtained by fully unfolding all μ -binders. Since the tree representations contain no μ -binders, the context grammars need no corresponding μ case. The contexts themselves are always finite, capturing only the structure between the root of the tree and the finitely many redex positions, while the (potentially infinite) subtrees are used to fill in the holes.

Definition 22 (Global Type Contexts). Analogously to Ghilezan et al. [28, Def A.19], we define global type contexts, with indexed holes, inductively as follows, where we assume $(a, b) \neq (p, q)$, $c \neq p$, and $I_R \neq \emptyset$:

$$\mathbb{G}_{\oplus}^{(p,q)} = a \rightarrow b : \left\{ m_i(B_i). \mathbb{G}_{\oplus}^{(p,q)} \right\}_{i \in I} \mid r \xrightarrow{m} s : \left\{ m(B). \mathbb{G}_{\oplus}^{(p,q)} \right\} \mid [\cdot]_i$$

$$\begin{aligned}
& | \text{p} \rightarrow \text{b} : \left\{ \mathfrak{m}_i(B_i) \cdot \langle \cdot \rangle_{l_i} \right\}_{i \in I_L} + \text{p} \rightarrow \text{b} : \left\{ \mathfrak{m}_i(B_i) \cdot \mathbb{G}_{\oplus}^{(p,q)} \right\}_{i \in I_R} \\
\mathbb{G}_{\&}^{(q,p)} &= \text{c} \rightarrow \text{r} : \left\{ \mathfrak{m}_i(B_i) \cdot \mathbb{G}_{\&}^{(q,p)} \right\}_{i \in I} \mid \text{a} \xrightarrow{\text{m}} \text{c} : \left\{ \mathfrak{m}(B) \cdot \mathbb{G}_{\&}^{(q,p)} \right\} \mid [\cdot]_i
\end{aligned}$$

We define separate contexts for situations in which it is safe to reduce transmissions ($\mathbb{G}_{\oplus}^{(p,q)}$) and en-route transmissions ($\mathbb{G}_{\&}^{(q,p)}$) respectively. In either case, given $\mathbb{G}_{\star}^{(p,q)}$ where $\star \in \{\oplus, \&\}$, the context has some hole indexing set I and some forgetful hole indexing set I' . We require the indexing sets of continuations to be disjoint and for I and I' to be disjoint. We write $\mathbb{G}_{\star}^{(p,q)}[G_i]_{i \in I} \langle G'_i \rangle_{i \in I'}$ for the tree given by populating the holes labelled with i by the global type tree G_i for $i \in I$ and the forgetful holes labelled with i by the global type tree G'_i for $i \in I'$.

Of course, we will need to specify the shape of a global type after it performs an asynchronous transition. Via [GR-CTX-I'], we can do this by explicitly removing the choices that result in a forgetful hole.

Definition 23 (Forgetting Forgetful Holes). Given a context $\mathbb{G}$, we define $\overline{\mathbb{G}}$ to be the context where we have removed all of the forgetful holes from $\mathbb{G}$ recursively:

- $\overline{\text{p} \rightarrow \text{q} : \left\{ \mathfrak{m}_i(B_i) \cdot \langle \cdot \rangle_{l_i} \right\}_{i \in I_L} + \text{p} \rightarrow \text{q} : \left\{ \mathfrak{m}_i(B_i) \cdot \mathbb{G}_i \right\}_{i \in I_R}} = \text{p} \rightarrow \text{q} : \left\{ \mathfrak{m}_i(B_i) \cdot \overline{\mathbb{G}}_i \right\}_{i \in I_R}$
- $\overline{[\cdot]_i} = [\cdot]_i$
- $\overline{\text{p} \xrightarrow{\text{m}} \text{q} : \left\{ \mathfrak{m}(B) \cdot \mathbb{G} \right\}} = \text{p} \xrightarrow{\text{m}} \text{q} : \left\{ \mathfrak{m}(B) \cdot \overline{\mathbb{G}} \right\}$

We now provide similar definitions for local type contexts, so that we can observe asynchronous semantics in projected types.

Definition 24 (Local Type Tree Context). Define a local type tree context inductively as follows ($q \neq p$):

$$\begin{aligned}
\mathbb{L}_{\oplus}^{(p)} &= [\cdot]_i \mid \text{r} \& \left\{ \mathfrak{m}_i(B_i) \cdot \mathbb{L}_{\oplus}^{(p)} \right\}_{i \in I} \mid \text{q} \oplus \left\{ \mathfrak{m}_i(B_i) \cdot \langle \cdot \rangle_{l_i} \right\}_{i \in I_L} + \text{q} \oplus \left\{ \mathfrak{m}_i(B_i) \cdot \mathbb{L}_{\oplus}^{(p)} \right\}_{i \in I_R} \\
\mathbb{L}_{\&}^{(p)} &= [\cdot]_i \mid \text{q} \& \left\{ \mathfrak{m}_i(B_i) \cdot \mathbb{L}_{\&}^{(p)} \right\}_{i \in I}
\end{aligned}$$

We define separate contexts for situations in which it is safe to reduce transmissions ($\mathbb{L}_{\oplus}^{(p)}$) and en-route transmissions ($\mathbb{L}_{\&}^{(p)}$) respectively. In either case, given $\mathbb{L}_{\star}^{(p)}$ where $\star \in \{\oplus, \&\}$, the context has some hole indexing set I and some forgetful hole indexing set I' . We require the indexing sets of continuations to be disjoint and for I and I' to be disjoint. We write $\mathbb{L}_{\star}^{(p,q)}[T_i]_{i \in I} \langle T'_i \rangle_{i \in I'}$ for the tree given by populating the holes labelled with i by the local type tree T_i for $i \in I$ and the forgetful holes labelled with i by the local type tree T'_i for $i \in I'$.

Definition 25 (Forgetting Forgetful Holes Locally). Given a context $\mathbb{L}$, we define $\overline{\mathbb{L}}$ to be the context where we have removed all of the forgetful holes from $\mathbb{L}$ recursively:

- $\overline{\text{q} \oplus \left\{ \mathfrak{m}_i(B_i) \cdot \langle \cdot \rangle_{l_i} \right\}_{i \in I_L} + \text{q} \oplus \left\{ \mathfrak{m}_i(B_i) \cdot \mathbb{L}_i \right\}_{i \in I_R}} = \text{q} \oplus \left\{ \mathfrak{m}_i(B_i) \cdot \overline{\mathbb{L}}_i \right\}_{i \in I_R}$
- $\overline{\text{q} \& \left\{ \mathfrak{m}_i(B_i) \cdot \mathbb{L}_i \right\}_{i \in I}} = \text{q} \& \left\{ \mathfrak{m}_i(B_i) \cdot \overline{\mathbb{L}}_i \right\}_{i \in I}$
- $\overline{[\cdot]_i} = [\cdot]_i$

Our aim is to use local contexts to determine redexes for the asynchronous subtyping, to identify them in global contexts. Therefore, it will be necessary to link global contexts with local contexts in some way. Taking inspiration from our types, we provide a partial projection function. Note that as en-route transmissions induced by a global transition may occur both in front of some global redexes and following some global redexes, we must keep track of the queue up to each hole, instead of requiring uniformity of the queues.

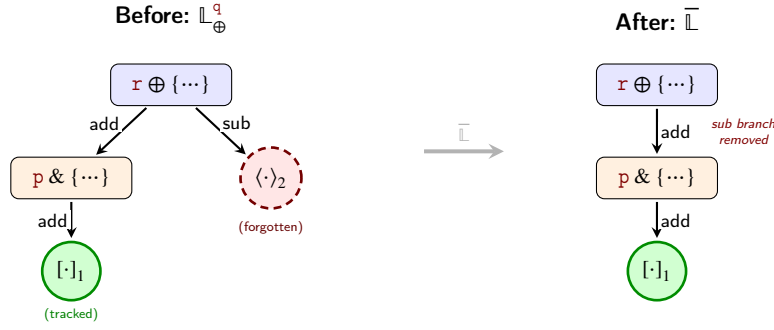


Figure 4: Local type context with normal and forgetful holes for T_q^{opt} from the ring protocol. **Left:** The context where q selects a branch to send to r . The add branch leads to a normal hole $[·]_1$ (tracked continuation), while sub leads to a forgetful hole $\langle·\rangle_2$ (to be forgotten). **Right:** After applying $\bar{L}$, only the add branch remains.

Definition 26 (Context Projection). We define a partial function from global type contexts with indexed holes to queues indexed by the global holes and local type contexts, written $\mathbb{G} \upharpoonright r = (\{\sigma_i\}_{i \in I}, \mathbb{L})$ with indexes being sets of indexes from the global context, inductively as follows:

- $[·]_i \upharpoonright r = (\{i \mapsto \emptyset\}, [·]_{\{i\}})$
- $\langle·\rangle_i \upharpoonright r = (\{i \mapsto \emptyset\}, \langle·\rangle_{\{i\}})$
- $p \rightarrow r: \{m_1(B_i).G_i\}_{i \in I} \upharpoonright r = (\bigcup_{i \in I} (G_i \upharpoonright r)_1, p \& \{m_1(B_i).(G_i \upharpoonright r)_2\}_{i \in I})$
- $p \xrightarrow{m} r: \{m(B).G'\} \upharpoonright r = ((G' \upharpoonright r)_1, p \& \{m(B).(G' \upharpoonright r)_2\})$
- $r \rightarrow q: \{m_1(B_i).G_i\}_{i \in I} \upharpoonright r = (\bigcup_{i \in I} (G_i \upharpoonright r)_1, q \oplus \{m_1(B_i).(G_i \upharpoonright r)_2\}_{i \in I})$
- $r \xrightarrow{m} q: \{m(B).G'\} \upharpoonright r = (\{(q, m(B)) \cdot (G' \upharpoonright r)_1(j)\}_j, q \oplus \{m(B).(G' \upharpoonright r)_2\})$
- $p \rightarrow q: \{m_1(B_i).G_i\}_{i \in I} \upharpoonright r = (\bigcup_{i \in I} (G_i \upharpoonright r)_1, \prod_{i \in I} (G_i \upharpoonright r)_2)$
- $p \xrightarrow{m} q: \{m(B).G'\} \upharpoonright r = ((G' \upharpoonright r)_1, (G' \upharpoonright r)_2)$

where we define the merge of contexts, where the hole indexes are sets of indexes, by

- $\prod_{i \in I} [·]_I = [·]_{\{i: i \in I\}}$
- $\prod_{i \in I} \langle·\rangle_I = \langle·\rangle_{\{i: i \in I\}}$
- $\prod_{i \in I} p \& \{m_j(B_j).L_{I,j}\}_{j \in J_I} = p \& \{m_j(B_j). \prod \{L_{I,j} : j \in J_I\}\}_{j \in J}$ where $J = \bigcup_{i \in I} J_I$ and $\{m_j\}_{j \in J}$ are distinct
- $\prod_{i \in I} p \oplus \{m_j(B_j).L_{I,j}\}_{j \in J} = p \oplus \{m_j(B_j). \prod_{i \in I} L_{I,j}\}_{j \in J}$

The definition of context projections and context merges, follow the structure of type projections and type merges. The exception being that, in the context case, we will not determine the entire queue before reaching a hole so we must remember the queue upto each hole.

Now to make use of the context projection, we observe that it indeed corresponds to type projections.

Lemma 14 (Context Merges correspond to Merges). Suppose that for $i \in I$, $\mathbb{L}_i$ are local contexts where the hole indexes are disjoint sets. Say $\mathbb{L}_i$ has hole index set J_i and forgetful hole index set J'_i . Suppose that $\mathbb{L} = \prod_{i \in I} \mathbb{L}_i$, with hole indexes J and forgetful hole indexes J' . Suppose that $\mathfrak{T}(T) = \mathbb{L}[\mathfrak{T}(T_j)]_{j \in J} \langle \mathfrak{T}(T_j) \rangle_{j \in J'}$, for all $i \in I$ $\mathfrak{T}(T'_i) = \mathbb{L}_i[\mathfrak{T}(T'_{i,j})]_{j \in J'_i} \langle \mathfrak{T}(T'_{i,j}) \rangle_{j \in J'_i}$, and for all $j \in J$ $\prod \{T'_{i,j'} : i \in I, j' \in J_i \cup J'_i, j' \subseteq j\} \ni T_j$. Then $\prod_{i \in I} T'_i \ni T$.

Proof. By induction on local contexts. $\square$

Lemma 15 (Context Projections correspond to Projections). *Suppose that $\mathbb{G} \uparrow \mathbf{r} = (\{\sigma_i\}_{i \in I \cup J}, \mathbb{L})$ with index sets $I \cup J$ and $I' \cup J'$, respectively. Suppose that $\mathfrak{Z}(G) = \mathbb{G}[G'_i]_{i \in I} \langle G'_i \rangle_{i \in J}$, $\mathfrak{Z}(T) = \mathbb{L}[T'_i]_{i \in I'} \langle T'_i \rangle_{i \in J'}$, $G'_i = \mathfrak{Z}(G_i)$ for $i \in I \cup J$, $T'_i = \mathfrak{Z}(T_i)$ for $i \in I' \cup J'$, and for $i \in I \cup J$ there is T''_i such that $G_i \uparrow \mathbf{r}(\sigma'_i, T''_i)$, and $\prod_{i \in J} T''_i \ni T_J$ for $J \in I'$, and $\sigma = \sigma_i \cdot \sigma'_i$ for $i \in I$. Then, $G \uparrow \mathbf{r}(\sigma, T)$.*

Proof. We proceed by induction on global contexts, using Lem. 14 $\square$

With all of our tools in place, we can now see how local contexts and global contexts can be used to capture redexes up to asynchronous subtyping. Asynchronous subtyping allows redexes to move, thus the top-level redex of an asynchronous subtype appears somewhere in its supertype.

Lemma 16 (Inversion of subtyping).

- (1) If $\mathbf{p} \oplus \{\mathbf{m}_i(B_i).T_i\}_{i \in I} \leq_a T$, then $\mathfrak{Z}(T) = \mathbb{L}_{\oplus}^{(\mathbf{p})}[\mathbf{p} \oplus \{\mathbf{m}_i(B_{i,j}).\mathbb{T}'_{i,j}\}_{i \in I_j}]_{j \in J} \langle \mathbb{T}''_j \rangle_{j \in J'}$ with $I \subseteq I_j$ for all $j \in J$ and for all $i \in I$ and $j \in J$, we have $B_i <: B_{i,j}$, and if $\mathfrak{Z}(T'_i) = \mathbb{L}_{\oplus}^{(\mathbf{p})}[\mathbb{T}'_{i,j}]_{j \in J} \langle \mathbb{T}''_j \rangle_{j \in J'}$ then $T_i \leq_a T'_i$.
- (2) If $\mathbf{p} \& \{\mathbf{m}_i(B_i).T_i\}_{i \in I} \leq_a T$, then $\mathfrak{Z}(T) = \mathbb{L}_{\&}^{(\mathbf{p})}[\mathbf{p} \& \{\mathbf{m}_i(B_{i,j}).\mathbb{T}'_{i,j}\}_{i \in I_j}]_{j \in J}$ with $I_j \subseteq I$ for all $j \in J$, $I = \bigcup_{j \in J} I_j$, for all $j \in J$ and $i \in I$ $B_{i,j} <: B_i$, and if $\mathfrak{Z}(T'_i) = \mathbb{L}_{\&}^{(\mathbf{p})}[\mathbb{T}'_{i,j}]_{j \in J}$ for some $i \in \bigcap_{j \in J} I_j$ then $T_i \leq_a T'_i$.

Proof. We only show item (1) (as item (2) is dual). Write $S = \mathbf{p} \oplus \{\mathbf{m}_i(B_i).T_i\}_{i \in I}$ and assume $S \leq_a T$.

By Def. 5, for every $\mathbb{U} \in \llbracket \mathfrak{Z}(S) \rrbracket_{\text{SO}}$ and every $\mathbb{V}' \in \llbracket \mathfrak{Z}(T) \rrbracket_{\text{SI}}$ there exist $\mathbb{W} \in \llbracket \mathbb{U} \rrbracket_{\text{SI}}$ and $\mathbb{W}' \in \llbracket \mathbb{V}' \rrbracket_{\text{SO}}$ with $\mathbb{W} \lesssim \mathbb{W}'$. The clause for $\llbracket \cdot \rrbracket_{\text{SO}}$ on S yields, for each $i \in I$, an element

$$\mathbb{U}_i = \mathbf{p}! \mathbf{m}_i(B_i); \mathbb{U}'_i \quad \text{with} \quad \mathbb{U}'_i \in \llbracket \mathfrak{Z}(T_i) \rrbracket_{\text{SO}}.$$

Fix $i \in I$ and an arbitrary $\mathbb{V}' \in \llbracket \mathfrak{Z}(T) \rrbracket_{\text{SI}}$. The subtyping premise gives SISO trees $\mathbb{W}_i \in \llbracket \mathbb{U}_i \rrbracket_{\text{SI}}$ and $\mathbb{W}'_i \in \llbracket \mathbb{V}' \rrbracket_{\text{SO}}$ with $\mathbb{W}_i \lesssim \mathbb{W}'_i$. Unfolding $\llbracket \cdot \rrbracket_{\text{SI}}$ on $\mathbb{U}_i$ forces $\mathbb{W}_i = \mathbf{p}! \mathbf{m}_i(B_i); \mathbb{W}'_i$ for some $\mathbb{W}'_i \in \llbracket \mathbb{U}'_i \rrbracket_{\text{SI}}$. Invert the last rule of the refinement derivation of $\mathbb{W}_i \lesssim \mathbb{W}'_i$. Because $\mathbb{W}_i$ is headed by an output, the final rule is either [REF-OUT] or [REF-B].

- [REF-OUT]: $\mathbb{W}'_i = \mathbf{p}! \mathbf{m}_i(B_{i,j}); \mathbb{W}''_i$ with $B_i <: B_{i,j}$ and premise $\mathbb{W}'_i \lesssim \mathbb{W}''_i$. Then we must have that $T = \mathbf{p} \oplus \{\mathbf{m}_i(B_i).T_i\}_{i \in I'}$ with $I \subseteq I'$.
- [REF-B]: $\mathbb{W}'_i = \mathcal{B}^{(\mathbf{p})}; \mathbf{p}! \mathbf{m}_i(B_{i,j}); \mathbb{W}''_i$ for some non-empty finite $\mathcal{B}^{(\mathbf{p})}$, with $B_i <: B_{i,j}$, $\text{act}(\mathbb{W}'_i) = \text{act}(\mathcal{B}^{(\mathbf{p})}; \mathbb{W}''_i)$, and premise $\mathbb{W}'_i \lesssim \mathcal{B}^{(\mathbf{p})}; \mathbb{W}''_i$.

For [REF-B], write $\mathcal{B}^{(\mathbf{p})} = a; \mathcal{B}_{\text{tail}}^{(\mathbf{p})}$ with a the first action. We reason by case analysis on a , using that $\mathbb{W}'_i \in \llbracket \mathbb{V}' \rrbracket_{\text{SO}}$ and the definitions of $\llbracket \cdot \rrbracket_{\text{SO}}$ (where $\mathbf{r}! \mathbf{m}(B)$ and $\mathbf{r}? \mathbf{m}(B)$ are the single-branch forms of internal and external choice, respectively):

- If $a = \mathbf{r}? \mathbf{m}'(B')$, then $\llbracket \mathbb{V}' \rrbracket_{\text{SO}}$ can contain such a prefix only if $T = \mathbf{r} \oplus \{\mathbf{m}_k(B_k). \mathbb{V}'_k\}_{k \in K}$ with some k satisfying $\mathbf{m}_k = \mathbf{m}'$ and $B' <: B_k$; moreover $\mathbb{W}'_i \in \llbracket \mathbb{V}'_k \rrbracket_{\text{SO}}$.
- If $a = \mathbf{q}! \mathbf{m}'(B')$, with $\mathbf{p} \neq \mathbf{q}$, and then $\llbracket \mathbb{V}' \rrbracket_{\text{SO}}$ can contain such a prefix only if $T = \mathbf{q} \& \{\mathbf{m}_k(B_k). \mathbb{V}'_k\}_{k \in K}$ with some k satisfying $\mathbf{m}_k = \mathbf{m}'$ and $B' <: B'$, and again $\mathbb{W}'_i \in \llbracket \mathbb{V}'_k \rrbracket_{\text{SO}}$.

In either subcase we strip the head constructor of $\mathfrak{Z}(T)$, using forgetful holes to cover any branches not in I , obtaining a continuation, yielding a shorter prefix $\mathcal{B}_{\text{tail}}^{(\mathbf{p})}$. The induction hypothesis on $\mathcal{B}_{\text{tail}}^{(\mathbf{p})}$ yields a decomposition in which every branch reached after $\mathcal{B}_{\text{tail}}^{(\mathbf{p})}$ still contains an internal choice by $\mathbf{p}$ whose branch-set includes all labels in I with payloads extending the corresponding B_i . Re-wrapping the stripped constructor a preserves this property for the continuation, showing that along every branch of $\mathfrak{Z}(T)$ compatible with $\mathcal{B}^{(\mathbf{p})}$ there is an occurrence of $\mathbf{p} \oplus \{\mathbf{m}_i(B_{i,j}).\mathbb{T}'_{i,j}\}_{i \in I_j}$ with

$I \subseteq I_j$ and $B_i <: B_{i,j}$. So the overall shape of $\mathfrak{Z}(T)$ is $\mathbb{L}_{\oplus}^{(\mathbf{p})}[\mathbf{p} \oplus \{\mathbf{m}_i(B_{i,j}).\mathbb{T}'_{i,j}\}_{i \in I_j}]_{j \in J} \langle \mathbb{T}''_j \rangle_{j \in J'}$ as claimed. $\square$

Recall the context rules [GR-CTX-I], [GR-CTX-I'] and [GR-CTX-II] from Def. 7; these exactly say that sends occur under $\mathbb{G}_{\oplus}^{(p,q)}$ contexts and receives occur under $\mathbb{G}_{\&}^{(q,p)}$ contexts.

Lemma 17 (Global Type Tree Context Transitions).

1. Let $\mathbb{G}_{\oplus}^{(p,q)}$ be a context with index set J and forgetful index set J' . Let $k \in \bigcap_{j \in J} I_j$. We have that $\mathbb{G}_{\oplus}^{(p,q)}[p \rightarrow q : \{m_i(B_i).G_{i,j}\}_{i \in I_j}]_{j \in J} \langle G'_j \rangle_{j \in J'} \xrightarrow{p:q \oplus m_k(B_k)} \mathbb{G}_{\oplus}^{(p,q)}[G_{k,j}]_{j \in J}$.
2. Let $\mathbb{G}_{\&}^{(q,p)}$ be a context with index set J . We have that $\mathbb{G}_{\&}^{(q,p)}[q \rightsquigarrow p : \{m(B).G_j\}]_{j \in J} \xrightarrow{p:q \& m(B)} \mathbb{G}_{\oplus}^{(p,q)}[G_j]_{j \in J}$.

Proof. The proof in each case is by induction on the structure of the context.

1. By induction on the structure of $\mathbb{G}_{\oplus}^{(p,q)}$.
 - **Case** $\mathbb{G}_{\oplus}^{(p,q)} = [\cdot]$: Immediate from [GR- $\oplus$].
 - **Case** $\mathbb{G}_{\oplus}^{(p,q)} = a \rightarrow b : \{m'_i(B'_i).G_{\oplus}^{(p,q)}\}_{i \in I}$: Applying [GR-CTX-I] to the induction hypothesis.
 - **Case** $\mathbb{G}_{\oplus}^{(p,q)} = p \rightarrow b : \{m_i(B_i).\langle \cdot \rangle_{I_i}\}_{i \in I_L} + p \rightarrow b : \{m_i(B_i).G_{\oplus}^{(p,q)}\}_{i \in I_R}$: Applying [GR-CTX-I'] to the induction hypothesis.
 - **Case** $\mathbb{G}_{\oplus}^{(p,q)} = r \rightsquigarrow s : \{m'_i(B'_i).G_{\oplus}^{(p,q)}\}_{i \in I}$: Applying [GR-CTX-II] to the induction hypothesis.
2. By induction on the structure of $\mathbb{G}_{\&}^{(q,p)}$.
 - **Case** $\mathbb{G}_{\&}^{(q,p)} = [\cdot]$: Immediate from [GR- $\&$].
 - **Case** $\mathbb{G}_{\&}^{(q,p)} = c \rightarrow r : \{m'_i(B'_i).G_{\oplus}^{(p,q)}\}_{i \in I}$: Applying [GR-CTX-I] to the induction hypothesis.
 - **Case** $\mathbb{G}_{\&}^{(q,p)} = a \rightsquigarrow b : \{m'_i(B'_i).G_{\&}^{(q,p)}\}_{i \in I}$: Applying [GR-CTX-II] to the induction hypothesis.

□

5.2. Completeness and Soundness

In § 5.1, we related local asynchronous semantics to global semantics, in a manner that can be reflected by our projections, via type contexts, and in § 4.1, we determined properties that being balanced⁺ ensures. We can now combine these results; we may now proceed by induction on the depth functions provided by balanced⁺, to invert projections i.e. produce global contexts from local contexts.

We begin by observing that the semantics of a typing context informs us of the structure of its local types and queues.

Lemma 18 (Inversion of Context Semantics). *Suppose that $\Delta \xrightarrow{\alpha} \Delta'$.*

1. If $\alpha = p:q \oplus m(B)$ then $\Delta(p) = (\sigma_p, T_p)$, with $T_p = q \oplus \{m_i(B_i).T_i\}_{i \in I}$ where $m_j = m$, $B_j = B$ for some $j \in I$. And $\Delta'(p) = (\sigma_p \cdot (q, m(B)), T_j)$ with $\Delta'(r) = \Delta(r)$ for all $r \in \text{dom}(\Delta)$ with $r \neq p$.
2. If $\alpha = p:q \& m(B)$ then $\Delta(p) = (\sigma_p, T_p)$, with $T_p = q \& \{m_i(B_i).T_i\}_{i \in I}$ where $m_j = m$, $B_j = B$ for some $j \in I$. And $\Delta(q) = (\sigma_q \cdot (p, m(B')), T_q)$ with $B' <: B$. Finally, $\Delta'(p) = (\sigma_p, T_j)$ and $\Delta'(q) = (\sigma_q, T_q)$ with $\Delta'(r) = \Delta(r)$ for all $r \in \text{dom}(\Delta)$ with $r \notin \{p, q\}$.

Proof. By rule-induction on $\Delta \xrightarrow{\alpha} \Delta'$ (see Def. 9).

□

As alluded to above, we can proceed by induction on our depth functions to match local sends and receives in projected types to communications in global types.

Lemma 19 (Head Inversion of Projection). Assume $G \vdash_p(\sigma, T)$.

1. If $T = q \oplus \{m_i : T_i\}_{i \in I}$, then $\mathfrak{Z}(G) = \mathbb{G}[p \rightarrow q : \{m_i(B_i).G_{i,j}\}_{i \in I}]_{j \in J}$ where $G'_{i,j} \vdash_p(\sigma_j, T_{i,j})$, $\prod \{T_{i,j} : j \in J\} \ni T_i$, $\mathfrak{Z}(G'_{i,j}) = G_{i,j}$, $\mathbb{G} \vdash_p = (\{\sigma'_j\}_{j \in J}, [\cdot]_J)$, for $j \in J$, $\sigma = \sigma'_j \cdot \sigma_j$, and $\mathbb{G}$ does not contain p .
2. If $T = q \& \{m_i : T_i\}_{i \in I}$ and $G \vdash_q((p, m(B)) \cdot \sigma'', T'')$, then $\mathfrak{Z}(G) = \mathbb{G}[q \xrightarrow{m} p : \{m(B).G_j\}_{j \in J}]$ and $T = q \& \{m : T'\}$ where $G'_j \vdash_p(\sigma_j, T_j)$ for all $j \in J$, $\prod \{T_j : j \in J\} \ni T'$, $\mathfrak{Z}(G'_j) = G_j$, $\mathbb{G} \vdash_p = (\{\sigma'_j\}_{j \in J}, [\cdot]_J)$ for $j \in J$, $\sigma = \sigma'_j \cdot \sigma_j$, and $\mathbb{G}$ does not contain p except as the sender of an en-route transmission.
3. If $T = q \& \{m_i : T_i\}_{i \in I}$ and $G \vdash_q(\sigma_q, T'')$ with no element with destination p , then $\mathfrak{Z}(G) = \mathbb{G}[q \rightarrow p : \{m_i(B_i).G_{i,j}\}_{i \in I}]_{j \in J}$ where $I_j \subseteq I$, $I = \bigcup_{j \in J} I_j$, $G'_{i,j} \vdash_p(\sigma_j, T_{i,j})$ for all $j \in J$ and $i \in I_j$, $\prod \{T_{i,j} : i \in I_j\} \ni T_i$ for all $i \in I$, $\mathfrak{Z}(G'_{i,j}) = G_{i,j}$, $\mathbb{G} \vdash_p = (\{\sigma'_j\}_{j \in J}, [\cdot]_J)$ for $j \in J$, $\sigma = \sigma'_j \cdot \sigma_j$, and $\mathbb{G}$ does not contain p .

Proof. As $\text{unf}(T) \neq \text{end}$, $p \in \text{roles}(G)$. As G is balanced, $\text{depth}_p(G)$ exists. We shall proceed by induction on $\text{depth}_p(G)$. Suppose that $\text{depth}_p(G) = 1$, so p appears at the head of G .

1. If $T = q \oplus \{m_i.T_i\}_{i \in I}$, then by the definition of the projection, $\text{unf}(G) = p \rightarrow q : \{m_i(B_i).G_i\}_{i \in I}$ where $G_i \vdash_p(\sigma, T_i)$. $[\cdot]_I \vdash_p = (\{I \mapsto \emptyset\}, [\cdot]_I)$.
2. If $T = q \& \{m_i.T_i\}_{i \in I}$ and $G \vdash_q(\sigma_q, T'')$ with σ_q having no element with destination p , then by definition of the projection, $\text{unf}(G) = q \rightarrow p : \{m_i(B_i).G_i\}_{i \in I}$ where $G_i \vdash_p(\sigma, T_i)$.
3. If $T = q \& \{m_i.T_i\}_{i \in I}$ and $G \vdash_q((p, m(B)) \cdot \sigma'', T'')$, then by definition of the projection and as $\text{count}_{q,p}(G)$ exists, we have that $\text{unf}(G) = q \xrightarrow{m} p : \{m(B).G'\}$, and $T = q \& \{m : T'\}$, where $G' \vdash_p(\sigma, T')$.

Suppose that $\text{depth}_p(G) > 1$, so p is not at the head of G . We consider the cases for the head of G .

1. $\text{unf}(G) = p' \rightarrow q' : \{m_i(B_i).G_i\}_{i \in I}$ where $p \notin \{p', q'\}$. For $i \in I$, we have T_i such that $G_i \vdash_p(\sigma, T_i)$ and $\prod_{i \in I} T_i \ni T$, and if $G \vdash_q(\sigma', T')$ then $G_i \vdash_q(\sigma', \star)$. We then apply the I.H., reindex the contexts, and use associativity of the full coinductive merge to conclude.
2. $\text{unf}(G) = p' \xrightarrow{m} q' : \{m(B).G'\}$ where $p \notin \{p', q'\}$. We have $G' \vdash_p(\sigma, T)$, and if $G \vdash_q(\sigma', T')$ then $G' \vdash_q(\sigma'', \star)$ where σ'' and σ' , restricted to p , are the same. We then apply the I.H. to conclude.
3. $\text{unf}(G) = p \xrightarrow{m} q' : \{m(B).G'\}$. We have σ''' such that $G' \vdash_p(\sigma''', T)$ and $G' \vdash_q((p, m(B)) \cdot \sigma'', T'')$. We then apply the I.H. to conclude.

□

Since we are working with asynchronous subtyping, the head of a local type is insufficient. However by Lem. 16, it will suffice to consider local asynchronous redexes in their respective local contexts. Further, it suffices to apply Lem. 19 at each step to extend it to local contexts.

Lemma 20 (Inversion of Projection). Assume $G \vdash_p(\sigma, T)$

1. If $\mathfrak{Z}(T) = \mathbb{L}_{\oplus}^{(q)}[q \oplus \{m_{i,j}.T_{i,j}\}_{i \in I_j}]_{j \in J} \langle T_j \rangle_{j \in K}$ then $\mathfrak{Z}(G) = \mathbb{G}_{\oplus}^{(p,q)}[p \rightarrow q : \{m_{i,f(j)}(B_{i,f(j)}).G_{i,j}\}_{i \in I_{f(j)}}]_{j \in J'} \langle G'_j \rangle_{j \in K'}$ where σ is a reindexing of $\mathbb{L}_{\oplus}^{(q)}$, $\mathbb{G}_{\oplus}^{(p,q)} \vdash_p = (\{\sigma'_j\}_{j \in J' \cup K'}, \sigma \mathbb{L}_{\oplus}^{(q)})$, and $f(j) = k \in J \cup K : j \in \sigma k$ for $j \in J \cup K$. We additionally have that $G'_{i,j} \vdash_{\mathbf{r}}(\sigma''_j, T'_{i,j})$ for $j \in J'$, $i \in I_{f(j)}$, $G'_j \vdash_{\mathbf{r}}(\sigma''_j, T'_j)$ for $j \in K'$, $T'_{i,k} \in \prod_{j \in \sigma k} T'_{i,j}$ for $k \in J$, $i \in I_k$, and $T'_k \in \prod_{j \in \sigma k} T'_j$ for $k \in K$, where $\mathfrak{Z}(T'_{i,k}) = T_{i,k}$ for $k \in J$ and $i \in I_k$, $\mathfrak{Z}(T'_k) = T_k$ for $k \in K$, $\mathfrak{Z}(G'_{i,j}) = G_{i,j}$ for $j \in J'$ and $i \in I_{f(j)}$, $\mathfrak{Z}(G'_j) = G_j$ for $j \in K'$, and $\sigma = \sigma'_j \cdot \sigma''_j$ for $j \in J' \cup K'$.

2. If $\mathfrak{Z}(T) = \mathbb{L}_{\&}^{(q)}[q \& \{m_{i,j}.T_{i,j}\}_{i \in I_j}]_{j \in J}$ and $G \downarrow_q((p, m(B)) \cdot \sigma_q, T'_q)$, then for all $j \in J$, $|I_j| = 1$ so relabel $T_{i,j}$ to T_j , also $m_{i,j} = m$ and $B_{i,j} = B$ for the unique $i \in I$, $\mathfrak{Z}(G) = \mathbb{G}_{\&}^{(q,p)}[q \rightsquigarrow p : \{m(B).G_j\}]_{j \in J'}$ where σ is a reindexing of $\mathbb{L}_{\&}^{(q)}$, $\mathbb{G}_{\&}^{(q,p)} \downarrow p = (\{\sigma'_j\}_{j \in J}, \sigma \mathbb{L}_{\&}^{(q)})$. We additionally have that $G'_j \downarrow_r(\sigma''_j, T'_j)$ for $j \in J'$ and $T''_k \in \bigcap_{j \in \sigma k} T'_j$ for $k \in J$, where $\mathfrak{Z}(T''_k) = T_k$, $\mathfrak{Z}(G'_j) = G_j$, and $\sigma = \sigma'_j \cdot \sigma''_j$ for $j \in J'$.

Proof. Induction on local contexts. The base case and inductive cases are instances of Lem. 19. $\square$

We will now be able to perform global transitions upon observing local asynchronous transitions, such that the subject of the transition can still be projected onto a supertype of the resultant local type. To ensure that association is preserved, we must also consider the other projections. It can be easily observed that performing transitions with subject p does not drastically impact the projection onto $r (\neq p)$.

Lemma 21 (Global Type Transitions). *Suppose that $G \downarrow_r(\sigma, T)$ and $G \xrightarrow{\alpha} G'$.*

1. If $\alpha = p : q \& m(B)$ and $r \neq q$, then $G' \downarrow_r(\sigma, T)$.
2. If $\alpha = p : r \& m(B)$, then $\sigma = (p, m(B)) \cdot \sigma'$ and $G' \downarrow_r(\sigma', T)$.
3. Otherwise, there is T' such that $\bigcap \{T, T'\} \ni T$ and $G' \downarrow_r(\sigma, T')$.

Note that $\bigcap \{T, T'\} \ni T$ implies that $T \leq T'$ by Lemma 8.

Proof. Induction on the relation $G \xrightarrow{\alpha} G'$, making use of Lem. 6 and Lem. 10 to maintain the invariant specified in Item 3. $\square$

Given a transition from Δ associated to a balanced⁺ global type, G , to Δ' , we can now: identify the redexes inside the projection of G ; identify corresponding redexes inside G' ; perform a corresponding transition of G to G' ; and project G' to observe that Δ' is associated to G' . We state this formally, focusing on a single participant, in Lem. 22, before applying this to contexts, proving Thm. 11.

Lemma 22 (Global Type Mirrors Local Actions). *Assume*

$$G \downarrow_p(\sigma_p, T'_p) \quad \text{with} \quad T_p \leq_a T'_p.$$

1. If $\text{unf}(T_p) = q \oplus \{m_i(B_i).T_i\}_{i \in I}$ then for any $k \in I$, $\exists G'$ such that $G' \downarrow_p(\sigma_p \cdot (q, m_k(B')), T'_k)$ and $B_k <: B'$ and $T_k \leq_a T'_k$ and $G \xrightarrow{p : q \oplus m_k(B')} G'$.
2. If $\text{unf}(T_p) = q \& \{m_i(B_i).T_i\}_{i \in I}$ and $G \downarrow_q((p, m(B)) \cdot \sigma_q, T'_q)$ with $T_q \leq_a T'_q$ where $m = m_k$ and $B <: B_k$ for some $k \in I$, then $G' \downarrow_p(\sigma_p, T'_k)$ and $T_k \leq_a T'_k$ and $G' \downarrow_q T'_q$ and $G \xrightarrow{p : q \& m_k(B)} G'$.
3. If $\text{unf}(T_p) = \text{end}$ then $\sigma_p = \emptyset$ and $\text{unf}(T'_p) = \text{end}$.

Proof.

1. $\text{unf}(T_p) = q \oplus \{m_i(B_i).T_i\}_{i \in I}$. Applying Lem. 16, we have that $\mathfrak{Z}(T) = \mathbb{L}_{\oplus}^{(p)}[p \oplus \{m_i(B_{i,j}).T'_{i,j}\}]_{i \in I_j} \langle T''_j \rangle_{j \in J'}$ with $I \subseteq I_j$ for all $j \in J$ and for all $i \in I$ and $j \in J$, we have $B_i <: B_{i,j}$, and if $\mathfrak{Z}(T'_i) = \mathbb{L}_{\oplus}^{(p)}[T'_{i,j}]_{j \in J} \langle T''_j \rangle_{j \in J'}$ then $T_i \leq_a T'_i$. Now we apply Lem. 20 to get that $\mathfrak{Z}(G) = \mathbb{G}_{\oplus}^{(p,q)}[p \rightarrow q : \{m_{i,f(j)}(B_{i,f(j)}).G_{i,j}\}_{i \in I_{f(j)}}]_{j \in K} \langle G''_j \rangle_{j \in K'}$. We apply Lem. 22 to get that $\mathfrak{Z}(G) = \mathbb{G}_{\oplus}^{(p,q)}[p \rightarrow q : \{m_{i,f(j)}(B_{i,f(j)}).G_{i,j}\}_{i \in I_{f(j)}}]_{j \in K} \langle G''_j \rangle_{j \in K'} \rightarrow \mathbb{G}_{\oplus}^{(p,q)'}[G_{k,j}]_{j \in K}$ for $k \in I$. Then we use the additional conclusions to reconstruct the subtyping derivation from before, with updated premises taken from the results of applying the lemma, to deduce that $G' \downarrow_p T'_k$ where $\mathfrak{Z}(T'_k) = \mathbb{L}_{\oplus}^{(p,q)'}[T_{k,j}]_{j \in J}$ and $\mathfrak{Z}(G') = \mathbb{G}_{\oplus}^{(p,q)'}[G_{k,j}]_{j \in K}$.

2. (Similar to previous case).
3. From the projection rules and applying Lem. 1.

□

Theorem 11 (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' and α' such that $\alpha \leq \alpha'$, $\Delta' \sqsubseteq_a G'$, and $G \xrightarrow{\alpha'} G'$.*

Proof. By case analysis on α . **Case $\alpha = p : q \oplus m(B)$ (a send by a participant).**

Applying Lem. 18, for some label m , payload B , sender p and receiver q , we have

$$\Delta(p) = (\sigma_p, T_p) \\ \text{with } T_p = q \oplus \{m_i(B_i).T_i\}_{i \in I} \text{ where } m_j = m, B_j = B.$$

We also know the endpoint at p is updated in the new context, $\Delta'(p) = (\sigma_p \cdot (q, m(B)), T_j)$, while all other endpoints stay unchanged. As we know $T_p \neq \text{end}$, then $p \in \text{dom}(\Delta)$, and so by association there exists T'_p such that

$$G \upharpoonright_p (\sigma'_p, T'_p) \quad \text{with} \quad T_p \leq_a T'_p \text{ and } \sigma_p \leq_a \sigma'_p \quad (57)$$

Hence we can apply Lem. 22 to obtain the desired result.

$$G' \upharpoonright_p (\sigma'_p \cdot (q, m(B')), T'_j) \quad \text{with} \quad T_j \leq_a T'_j \quad \text{and} \quad B <: B' \quad (58)$$

And so, by also applying Lem. 21, we have $\Delta' \sqsubseteq_a G'$ and $G \xrightarrow{p : q \oplus m(B')} G'$ in this case.

Case $\alpha = q : p \& m(B)$ (receive by a participant).

Dually to the above case, we can again apply Lem. 18, for some label m , payload B , sender q and receiver p . Then we have

$$\Delta(p) = (\sigma_p, T_p) \quad \text{and} \quad \Delta(q) = ((p, m(B')) \cdot \sigma_q, T_q) \\ \text{with } T_p = q \& \{m_i(B_i).T_i\}_{i \in I} \text{ where } m_j = m, B' <: B_j = B. \quad (59)$$

Now the endpoints at p and q are updated, so $\Delta'(p) = (\sigma_p, T_j)$, and $\Delta'(q) = (\sigma_q, T_q)$, while other endpoints in Δ' stay the same. Again, given $T_p \neq \text{end}$, there exists T'_p such that

$$G \upharpoonright_p (\sigma'_p, T'_p) \quad \text{with} \quad T_p \leq_a T'_p \text{ and } \sigma_p \leq_a \sigma'_p. \quad (60)$$

and

$$G \upharpoonright_q ((p, m(B'')) \cdot \sigma'_q, T'_q) \quad \text{with} \quad T_q \leq_a T'_q \quad \text{and} \quad B' <: B'' \text{ and } \sigma_q \leq_a \sigma'_q. \quad (61)$$

Hence we can apply Lem. 22 to obtain the desired result.

$$G' \upharpoonright_p (\sigma'_p, T'_j) \quad \text{with} \quad T_j \leq_a T'_j. \quad (62)$$

and

$$G' \upharpoonright_q (\sigma'_q, T'_q). \quad (63)$$

And so we have $\Delta' \sqsubseteq_a G'$ and $G \xrightarrow{q : p \& m(B'')} G'$. □

Now that we have Thm. 11, Thm. 12 is truly quite weak. Indeed, we only need to know that contexts, associated to a non-**end** global type, can move.

Lemma 23 (Projection preserves operational semantics). *If $G \xrightarrow{\cdot}$, $G \upharpoonright_p \Delta(p)$ for all $p \in \text{dom}(\Delta) \supseteq \text{roles}(G)$, then $\Delta \xrightarrow{\cdot}$.*

Proof. If $G \rightarrow$, then $\text{unf}(G) \neq \text{end}$ so write $\text{unf}(G) = p \rightarrow q : \{m_1(B_i).G_i\}_{i \in I}$ or $p \xrightarrow{m_j} q : \{m_1(B_i).G_i\}_{i \in I}$. If $\text{unf}(G) = p \rightarrow q : \{m_1(B_i).G_i\}_{i \in I}$, then let $\Delta_G(p) = (\sigma, T)$ so $\text{unf}(T) = q \oplus \{m_1(B_i).T_i\}_{i \in I}$ so $\Delta \xrightarrow{p:q \oplus m_j}$ for $j \in I$. If $\text{unf}(G) = p \xrightarrow{m_j} q : \{m_1(B_i).G_i\}_{i \in I}$, then let $\Delta_G(q) = (\sigma, T)$ and $\Delta_G(p) = (\sigma', T')$ so $\sigma' = (q, m_j(B_j)) \cdot \sigma'$ and $\text{unf}(T) = p \& \{m_1(B_i).T_i\}_{i \in I}$ so $\Delta \xrightarrow{q:p \& m_j}$. $\square$

Theorem 12 (Soundness of Association). *Let $\Delta \sqsubseteq_a G$ and assume $G \xrightarrow{\alpha} G'$. Then there exist actions $\alpha' \leq \alpha''$, a context Δ' , and a global type G'' such that $G \xrightarrow{\alpha''} G''$, $\Delta \xrightarrow{\alpha'} \Delta'$, and $\Delta' \sqsubseteq_a G''$.*

Proof. By association, we know there exists a context Δ'' such that $G \upharpoonright_p \Delta''(p)$ for all $p \in \text{dom}(\Delta'')$ with $\Delta \leq_a \Delta''$.

Then we can apply Lem. 23 to get that $\Delta'' \xrightarrow{\alpha}$. So, $\Delta \xrightarrow{\alpha'}$ as the precise asynchronous subtyping relates non-deadlocked contexts to non-deadlocked contexts. Then the desired result is obtained by applying Thm. 11. $\square$

5.3. Properties of Associated Contexts

Now only Thm. 13 remains to be shown. Similarly to the synchronous setting, liveness implies deadlock-freedom. However unlike in the synchronous setting, liveness also implies safety. Therefore, it suffices to prove that associated contexts are live. With the observation that liveness is downward closed under the precise asynchronous subtyping, we need only focus on projected contexts.

Lemma 24 (Liveness is Downwards closed under subtyping [29, Lemma 4.10]). *If Δ is live and $\Delta' \leq_a \Delta$ then Δ' is live.*

Now, the precise asynchronous subtyping cannot cause any further issues. The proof that a projected context is live is very similar to the proofs in the inductive case and in the synchronous setting. We have already observed sufficient operational correspondence between global types and projected contexts to derive liveness.

Lemma 25 (Standard Association is Complete). *If $G \upharpoonright_p \Delta'(p)$ for all $p \in \text{dom}(\Delta)$, $\text{roles}(G) \subseteq \text{dom}(\Delta)$, $\Delta \leq \Delta'$, and $\Delta \xrightarrow{\alpha} \Delta''$, then there exist G' , $\alpha' \geq \alpha$, and Δ''' such that $G' \upharpoonright_p \Delta'''(p)$ for all $p \in \text{dom}(\Delta)$, $\Delta' \leq \Delta'''$, and $G \xrightarrow{\alpha'} G'$.*

Proof. Follows the same structure as the proof of Thm. 11. $\square$

Lemma 26 (Standard Association is Head-Sound). *If $G \upharpoonright_p \Delta(p)$ for all $p \in \text{dom}(\Delta)$, $\text{roles}(G) \subseteq \text{dom}(\Delta)$, then:*

- If $\text{unf}(G) = p \rightarrow q : \{m_1(B_i).G_i\}_{i \in I}$, then $\Delta \xrightarrow{p:q \oplus m_k(B_k)}$ for some $k \in I$.
- If $\text{unf}(G) = p \xrightarrow{m} q : \{m(B).G'\}$, then $\Delta \xrightarrow{p:q \& m(B)}$.

Proof. Clear from the definition of projection. $\square$

With this operational correspondence, we can proceed normally. We take fair sequences from a projected context and use Lem. 25 to determine a corresponding sequence of global types. By Lem. 26 and fairness, every global type head must be reduced. Finally by balanced⁺, we deduce that every redex will either be reduced or, eventually, become the head of the global type, and then be reduced.

Lemma 27 (Projection ensures Liveness). *If $G \upharpoonright_p \Delta(p)$ for all $p \in \text{dom}(\Delta)$ and $\text{roles}(G) \subseteq \text{dom}(\Delta)$, then Δ is live.*

Proof. Let $(\Delta_n)_{n \in \mathbb{N}}$ with $(\alpha_n)_{n, n+1 \in \mathbb{N}}$ be a fair sequence starting from Δ . By Lem. 25, find $(G_n)_{n \in \mathbb{N}}$ ($G_0 = G$) and $(\Delta'_n)_{n \in \mathbb{N}}$ ($\Delta'_0 = \Delta$) such that $G_n \upharpoonright_p \Delta'_n(p)$ for all $p \in \text{dom}(\Delta)$, $\Delta_n \leq \Delta'_n$, and $G_n \xrightarrow{\alpha_n} G_{n+1}$. If $\Delta_i(p)$ has a message for q in its queue, then $(p, q) \in \text{mRoles}(G_i)$ so $\text{mDepth}_{p,q}(G_i)$ exists. Proceed by induction on $\text{mDepth}_{p,q}(G_i)$. Suppose that this message is never dequeued. Now apply Lem. 26 via fairness to get to $j > i$ and Lem. 4 by our assumption. The message is still on the queue by our assumption, and $\text{mDepth}_{p,q}(G_j) < \text{mDepth}_{p,q}(G_i)$. By the I.H, the message will be dequeued. The case for $\Delta_i(p)$ having a receive at the head is similar, but we proceed by induction on $\text{depth}_p(G_i)$ instead of $\text{mDepth}_{p,q}(G_i)$. $\square$

Theorem 13 (Associated Context Properties). *If $\Delta \sqsubseteq_a G$, then Δ is safe, deadlock-free, and live.*

Proof. Let $G \vdash_p \Delta'(\mathbf{p})$ for all $\mathbf{p} \in \text{dom}(\Delta)$ and $\Delta \leq_a \Delta'$, by association. By Lem. 27 and Lem. 24, Δ' is live so Δ is live. $\square$

6. 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 from Ghilezan et al. [29, Theorems 4.11, 4.12 and 4.13].

6.1. Asynchronous Multiparty Session Processes

We summarise the calculus from Ghilezan et al. [29] and main properties.

Process Syntax. We define *processes* $(P, Q, P_i, \dots)$ by the following grammar:

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

where $\mathbf{q}!\ell\langle e \rangle.P$ is a *send* (selection) to role $\mathbf{q}$ with label ℓ and payload expression e , followed by continuation P ; $\sum_{i \in I} \mathbf{q}?\ell_i(x_i).P_i$ is a *receive* (branching) from role $\mathbf{q}$, where a message with label ℓ_k binds variable x_k and continues as P_k ; $\text{if } e \text{ then } P \text{ else } Q$ is a conditional; $\mathbf{0}$ is the inactive process; $\mu X.P$ and X are for recursion.

Queue Syntax. We define *message queues* $(h, h_i, \dots)$ by the following grammar:

$$h ::= \epsilon \mid (\mathbf{q}, \ell(v)) \mid h_1 \cdot h_2$$

where ϵ is the empty queue; $(\mathbf{q}, \ell(v))$ is the queue denoting that the message with label ℓ and value v has been sent to $\mathbf{q}$; and $h_1 \cdot h_2$ is the concatenation of the queues h_1 and h_2 . We consider message queues up to an equivalence relation $\equiv$, similar to Def. 1 but with ϵ for $\emptyset$, values for basic types, and message queues for syntactic queue types.

Remark 7 (Session Interleaving and Delegation). This calculus excludes session interleaving and delegation elements. This choice was made by Ghilezan et al. [29], to simplify the calculus by Coppo et al. [15] and focus on subtyping. For more detail and examples of how to tackle this, see the work by Coppo et al. [16] and van den Heuvel and Pérez [56].

Typing Judgements. We write $\Theta \vdash P \triangleright T$ to denote that process P has local type T under process variable environment Θ , and $\vdash h \triangleright \sigma$ to denote that queue h has queue type σ . The typing rules for processes are standard and are shared by both the top-down and bottom-up systems. We write Θ for the process-variable and expression-variable context, mapping process variables to their local types and expression variables to their sorts. [T-0] types the inactive process; [T-VAR] and [T-REC] allow recursion at the process level; [T- $\oplus$] and [T- $\&$] allow typing of selections and branchings; [T-COND] types conditionals; and the subsumption rule [T-SUB] allows for subtyping. Queue rules type the empty queue, single messages, and concatenation. We assume a separate sorting system for expressions, at least providing judgements of the form $\Theta \vdash e \triangleright \text{bool}$ and $\Theta \vdash e \triangleright \text{int}$ for boolean and integer expressions.

Definition 27 (Typing Rules for Processes [29]).

$$\begin{array}{c} \frac{}{\Theta \vdash \mathbf{0} \triangleright \text{end}} \text{[T-0]} \quad \frac{X : \mathbf{t} \in \Theta}{\Theta \vdash X \triangleright \mathbf{t}} \text{[T-VAR]} \quad \frac{\Theta \vdash P \triangleright T \quad T \leq_a T'}{\Theta \vdash P \triangleright T'} \text{[T-SUB]} \\[10pt] \frac{\Theta \vdash e_j \triangleright B_j \quad \Theta \vdash P_j \triangleright T_j \quad j \in I}{\Theta \vdash \mathbf{q}!\ell_j\langle e_j \rangle.P_j \triangleright \mathbf{q}\oplus\{\ell_i(B_i).T_i\}_{i \in I}} \text{[T-}\oplus\text{]} \quad \frac{\forall i \in I \quad \Theta, x_i : B_i \vdash P_i \triangleright T_i}{\Theta \vdash \sum_{i \in I} \mathbf{q}?\ell_i(x_i).P_i \triangleright \mathbf{q}\&\{\ell_i(B_i).T_i\}_{i \in I}} \text{[T-}\&\text{]} \\[10pt] \frac{\Theta \vdash e \triangleright \text{bool} \quad \Theta \vdash P_1 \triangleright T \quad \Theta \vdash P_2 \triangleright T}{\Theta \vdash \text{if } e \text{ then } P_1 \text{ else } P_2 \triangleright T} \text{[T-COND]} \quad \frac{\Theta, X : \mathbf{t} \vdash P \triangleright T}{\Theta \vdash \mu X.P \triangleright \mu \mathbf{t}.T} \text{[T-REC]} \end{array}$$

The typing rules for queues are:

$$\frac{}{\vdash \epsilon \triangleright \emptyset} \quad \frac{\vdash v \triangleright B}{\vdash (\mathbf{q}, \ell(v)) \triangleright (\mathbf{q}, \ell(B))} \quad \frac{\vdash h_1 \triangleright \sigma_1 \quad \vdash h_2 \triangleright \sigma_2}{\vdash h_1 \cdot h_2 \triangleright \sigma_1 \cdot \sigma_2}$$

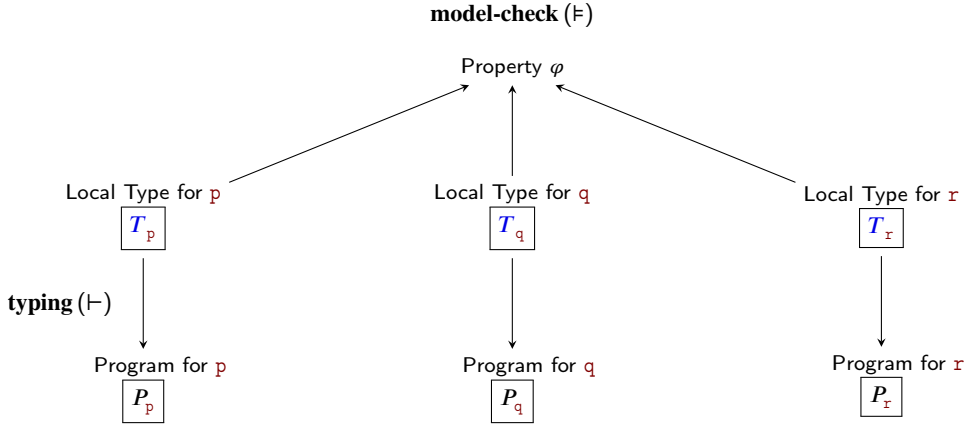


Figure 5: Bottom-up methodology for multiparty session types. Processes P_p , P_q and P_r are typed by local types T_p , T_q and T_r , which are jointly model-checked against the property ϕ .

The subsumption rule [T-SUB] can be explained by Liskov and Wing’s substitution principle [43]: if T is a subtype of T' , then an object of type T can always replace an object of type T' .

Example 16 (Ring Protocol Processes). Returning to the ring-choice protocol G_{ring} from § 1.2, a set of processes implementing the protocol can be given as follows:

$$P_p = \mu X. q!add\langle n \rangle. (r?add(x).X + r?sub(x).X) \quad (64)$$

$$P_q = \mu X. \text{if } e \text{ then } r!add\langle m \rangle. p?add(y).X \text{ else } r!sub\langle m \rangle. p?add(y).X \quad (65)$$

$$P_r = \mu X. q?add(z). p!add\langle z+k \rangle. X + q!sub(z). p!sub\langle z-k \rangle. X \quad (66)$$

Processes P_p and P_r are typed by the corresponding projected local types from § 1.2: $\vdash P_p \triangleright T_p$ and $\vdash P_r \triangleright T_r$.

Observe that, as far as T_q is concerned, q ’s outgoing message to r need not depend on the value received from p . A programmer can therefore write P_q to send to r *before* receiving from p , allowing the $q \rightarrow r$ and $p \rightarrow q$ communications to proceed concurrently, improving throughput as illustrated in Fig. 3(b). The projected type T_q requires the receive first, but P_q is typed by T_q^{opt} , and since $T_q^{\text{opt}} \leq_a T_q$, the subsumption rule [T-SUB] accepts P_q in place of any process typed by T_q . In this way, the global protocol G_{ring} guarantees safety, deadlock-freedom, and liveness for the system, while $\leq_a$ gives the programmer the freedom to choose a more efficient implementation for q .

Figure 5 highlights the bottom-up workflow that we use in this section: local types are jointly model-checked against the property ϕ and guide the typing of the programs.

Asynchronous Multiparty Sessions. We define *asynchronous multiparty session* $(M, M_i, \dots)$ as:

$$M ::= p \triangleleft P_p \mid p \triangleleft h_p \mid M \mid M'$$

where $p \triangleleft P_p$ denotes a process P_p plays as a role p .

The structural precongruence relation $\Rightarrow$ used in Def. 28 below allows us to disregard inactive processes, unfolds recursion, and is closed under reflexivity and transitivity:

$$\begin{array}{c} \overline{p \triangleleft \mathbf{0} \mid p \triangleleft \epsilon \mid \mathcal{M} \Rightarrow \mathcal{M}} \quad [\text{SC-IDLE}] \qquad \overline{\mathcal{M} \Rightarrow \mathcal{M}} \quad [\text{SC-REFL}] \\[10pt] \overline{\mu X. P \Rightarrow P\{\mu X. P/X\}} \quad [\text{SC-UNFOLD}] \qquad \frac{\mathcal{M} \Rightarrow \mathcal{M}' \quad \mathcal{M}' \Rightarrow \mathcal{M}''}{\mathcal{M} \Rightarrow \mathcal{M}''} \quad [\text{SC-TRANS}] \end{array}$$

Definition 28 (Reduction relation on sessions (asynchronous)).

[R-SEND]	$p \triangleleft q! \ell \langle e \rangle . P \mid p \triangleleft h_p \mid \mathcal{M}$	$\rightarrow p \triangleleft P \mid p \triangleleft h_p \cdot (q, \ell(v)) \mid \mathcal{M} \quad (e \downarrow v)$
[R-RCV]	$p \triangleleft \sum_{i \in I} q? \ell_i(x_i) . P_i \mid p \triangleleft h_p$ $\mid q \triangleleft Q \mid q \triangleleft (p, \ell_k(v)) \cdot h \mid \mathcal{M}$	$\rightarrow p \triangleleft P_k \{v/x_k\} \mid p \triangleleft h_p \quad (k \in I)$ $\mid q \triangleleft Q \mid q \triangleleft h \mid \mathcal{M}$
[R-COND-T]	$p \triangleleft \text{if } e \text{ then } P \text{ else } Q \mid p \triangleleft h \mid \mathcal{M}$	$\rightarrow p \triangleleft P \mid p \triangleleft h \mid \mathcal{M} \quad (e \downarrow \text{true})$
[R-COND-F]	$p \triangleleft \text{if } e \text{ then } P \text{ else } Q \mid p \triangleleft h \mid \mathcal{M}$	$\rightarrow p \triangleleft Q \mid p \triangleleft h \mid \mathcal{M} \quad (e \downarrow \text{false})$
[R-STRUCT]	$\mathcal{M}_1 \Rightarrow \mathcal{M}'_1 \quad \mathcal{M}'_1 \rightarrow \mathcal{M}'_2 \quad \mathcal{M}'_2 \Rightarrow \mathcal{M}_2$	$\Rightarrow \mathcal{M}_1 \rightarrow \mathcal{M}_2$
[ERR-MISM]	$p \triangleleft \sum_{i \in I} q? \ell_i(x_i) . P_i \mid p \triangleleft h_p \mid q \triangleleft Q \mid q \triangleleft (p, \ell(v)) \cdot h \mid \mathcal{M}$	$\rightarrow \text{err} \quad (\forall i \in I. \ell_i \neq \ell)$
[ERR-EVAL]	$p \triangleleft \text{if } e \text{ then } P \text{ else } Q \mid p \triangleleft h \mid \mathcal{M}$	$\rightarrow \text{err} \quad (e \not\downarrow \text{true and } e \not\downarrow \text{false})$

Safety and Deadlock-Freedom. We first define safety and deadlock-freedom for multiparty sessions.

Definition 29 (Safety and Deadlock-Freedom). A session M is:

1. *communication safe* iff for all M' such that $M \rightarrow^* M'$, we have $M' \not\rightarrow \text{err}$;
2. *deadlock-free* iff for all M' such that $M \rightarrow^* M'$ and $M' \not\rightarrow$, we have $M' \equiv \prod_{i \in I} (p_i \triangleleft 0 \mid p_i \triangleleft e)$.

Intuitively, safety ensures that no reachable session can reduce to an error state. Deadlock-freedom ensures that if a session cannot reduce further, then all processes have terminated and all queues are empty.

Session Liveness. Liveness is a stronger property that subsumes both safety and deadlock-freedom (recall Defs. 10 to 12 for the corresponding context-level properties). We define liveness for multiparty sessions, ensuring that under fair scheduling, all pending actions are eventually performed. A *session path* is a (potentially infinite) sequence of sessions $(M_i)_{i \in I}$, where $I = \{0, 1, 2, \dots\}$ is a set of consecutive natural numbers, and for all $i \in I$, $M_i \rightarrow M_{i+1}$.

Definition 30 (Fair and Live Session Paths [29]). A session path $(M_i)_{i \in I}$ is *fair* iff, for all $i \in I$:

- SF1 if $M_i \equiv p \triangleleft q! \ell \langle e \rangle . P \mid p \triangleleft h \mid M'$, then $\exists k$ such that $I \ni k \geq i$ and $M_k \rightarrow M_{k+1}$ via [R-SEND] at role p
- SF2 if $M_i \equiv p \triangleleft \sum_{j \in J} q? \ell_j(x_j) . P_j \mid p \triangleleft h \mid M'$ and $q \triangleleft (p, \ell_k(v)) \cdot h' \mid M''$ for some $k \in J$, then $\exists k'$ such that $I \ni k' \geq i$ and $M_{k'} \rightarrow M_{k'+1}$ via [R-RCV] at role p
- SF3 if $M_i \equiv p \triangleleft \text{if } e \text{ then } P \text{ else } Q \mid p \triangleleft h \mid M'$, then $\exists k$ such that $I \ni k \geq i$ and $M_k \rightarrow M_{k+1}$ via [R-COND-T] or [R-COND-F] at role p

A session path $(M_i)_{i \in I}$ is *live* iff, for all $i \in I$:

- SL1 if $M_i \equiv p \triangleleft \text{if } e \text{ then } P \text{ else } Q \mid p \triangleleft h \mid M'$, then $\exists k$ such that $I \ni k \geq i$ and, for some M'' , $M_k \equiv p \triangleleft P \mid p \triangleleft h \mid M''$ or $M_k \equiv p \triangleleft Q \mid p \triangleleft h \mid M''$
- SL2 if $M_i \equiv p \triangleleft q! \ell \langle e \rangle . P \mid p \triangleleft h \mid M'$, then $\exists k$ such that $I \ni k \geq i$ and $M_k \rightarrow M_{k+1}$ via [R-SEND] at role p
- SL3 if $M_i \equiv p \triangleleft P \mid p \triangleleft (q, \ell(v)) \cdot h \mid M'$, then $\exists k$ such that $I \ni k \geq i$ and $M_k \rightarrow M_{k+1}$ via [R-RCV] at role q
- SL4 if $M_i \equiv p \triangleleft \sum_{j \in J} q? \ell_j(x_j) . P_j \mid p \triangleleft h \mid M'$, then $\exists k, \ell'$ such that $I \ni k \geq i$, $\ell' \in \{\ell_j\}_{j \in J}$, and $M_k \rightarrow M_{k+1}$ via [R-RCV] at role p receiving label ℓ'

A session M is *live* iff all fair paths starting from M are live.

Intuitively, fairness (SF1–SF3) requires that if a component (process or queue) is ready to fire an action, then that action will eventually be performed. Liveness (SL1–SL4) then ensures that under fair execution: conditionals are eventually resolved (SL1), pending outputs are eventually performed (SL2), queued messages are eventually received (SL3), and pending inputs eventually receive a message (SL4).

6.2. Deriving the Main Theorems

We recall the bottom-up typing system for a multiparty session:

$$\frac{\forall p \in \text{dom}(\Delta) \quad \vdash P_p \triangleright T_p \quad \vdash h_p \triangleright \sigma_p \quad \Delta(p) = (\sigma_p, T_p) \quad \varphi(\Delta)}{\vdash^{\text{bot}} \Pi_{p \in \text{dom}(\Delta)} (p \triangleleft P_p \mid p \triangleleft h_p) \triangleright \Delta} \text{[SESSBOT]}$$

where $\vdash P_p \triangleright T_p$ states that process P_p has type T_p , $\vdash h_p \triangleright \sigma_p$ types message queue h_p has a queue type σ_p , and φ is some desired property, which is usually a *safety* property—a selected label is always available at the branching process [36,51]. In the work of Ghilezan et al. [29], a *liveness* property [29, Definition 4.17] is used instead for proving the preciseness of $\leq_a$. To derive the main theorems, we do not require the details of typing rules, hence we omit. See Ghilezan et al. [29, § 7.1] for the full typing system and its explanations.

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.

Theorem 28 (Subject Reduction, [29, Theorem 4.11]). *Assume $\vdash^{\text{bot}} M \triangleright \Delta$ with Δ live and $M \rightarrow^* M'$. Then there exist live Δ', Δ'' such that $\vdash^{\text{bot}} M' \triangleright \Delta'$ with $\Delta'' \leq_a \Delta$ and $\Delta'' \rightarrow^* \Delta'$.*

Theorem 29 (Subject Reduction of the Top-Down System). *Assume $\vdash^{\text{top}} M \triangleright \Delta$, $\Delta \sqsubseteq_a G$, and $M \rightarrow^* M'$. Then there exists G' such that $\vdash^{\text{top}} M' \triangleright \Delta'$, $G \rightarrow^* G'$ and $\Delta' \sqsubseteq_a G'$.*

Proof. We prove the following stronger statement, from which we can derive the above theorem

Assume $\vdash^{\text{top}} M \triangleright \Delta$, $\Delta \sqsubseteq_a G$, and $M \rightarrow^* M'$. Then there exist Δ', Δ'' , and G' such that $\vdash^{\text{top}} M' \triangleright \Delta'$ with $\Delta'' \leq_a \Delta$, $\Delta'' \rightarrow^* \Delta'$, $G \rightarrow^* G'$, and $\Delta' \sqsubseteq_a G'$.

Assume $M \equiv \Pi_{p \in \text{dom}(\Delta)} (p \triangleleft P_p \mid p \triangleleft h_p)$ and $\vdash^{\text{top}} M \triangleright \Delta$ is derived with

$$\forall p \in \text{dom}(\Delta) \quad \vdash P_p \triangleright T_p \quad \vdash h_p \triangleright \sigma_p \quad \Delta(p) = (\sigma_p, T_p) \quad \Delta \sqsubseteq_a G \quad (67)$$

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

Note that Δ is live by Thm. 13. Hence by Thm. 28, there exist live Δ', Δ'' such that $\vdash^{\text{bot}} \Pi_{p \in \text{role}(G)} P'_p \triangleright \Delta''$ with $\Delta'' \leq_a \Delta$ and $\Delta'' \rightarrow^* \Delta'$. By Def. 21, $\Delta' \sqsubseteq_a G$. Then by Thm. 11, $\Delta'' \rightarrow^* \Delta'$ implies $G \rightarrow^* G'$ and $\Delta' \sqsubseteq_a G'$. Hence $\vdash^{\text{top}} \Pi_{p \in \text{dom}(\Delta')} P'_p \triangleright \Delta'$ as desired. $\square$

Remark 8 (Subject Reduction Labels). By examining the proof of Thm. 28, we can deduce the labels of the context transition sequence from the reduction rules used in the session reduction sequence. For simplicity, we consider a single session reduction as this implies the full subject reduction. Suppose that $M \rightarrow M' \not\equiv \text{err}$ and consider the non-[R-STRUCT] rule deriving it.

[R-SEND] If the reduction occurred via [R-SEND] at role p with label ℓ and destination q , then $\Delta' \xrightarrow{p:q \oplus m(B)} \Delta''$ for some ground type B .

[R-RCV] If the reduction occurred via [R-RCV] at role p receiving label ℓ and with source q , then $\Delta' \xrightarrow{p:q \& m(B)} \Delta''$ for some ground type B .

[R-COND-T] $\Delta' = \Delta''$.

[R-COND-F] $\Delta' = \Delta''$.

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 30 (Session Fidelity, [29, Theorem 4.13]). *Assume $\vdash^{\text{bot}} M \triangleright \Delta$ with Δ live. Assume $\Delta \rightarrow$. Then there exist M', Δ' , and Δ'' such that $M \rightarrow^+ M'$, $\Delta'' \rightarrow \Delta'$, $\Delta'' \leq_a \Delta$, and $\vdash^{\text{bot}} M' \triangleright \Delta'$.*

Theorem 31 (Session Fidelity of the Top-Down System). *Assume $\vdash^{\text{top}} M \triangleright \Delta$ is derived by $\Delta \sqsubseteq_a G$ and $G \rightarrow$. Then there exist M' , Δ' , and G' such that $M \rightarrow^+ M'$, $G \rightarrow G'$, and $\vdash^{\text{top}} M' \triangleright \Delta'$ with $\Delta' \sqsubseteq_a G'$.*

Proof. We prove the following stronger statement, by which the above theorem is derived.

Assume $\vdash^{\text{top}} M \triangleright \Delta$ is derived by $\Delta \sqsubseteq_a G$ and $G \rightarrow$. Then there exist M' , Δ' , Δ'' , and G' such that $M \rightarrow^+ M'$, $\Delta'' \rightarrow \Delta'$, $\Delta'' \leq_a \Delta$, $G \rightarrow G'$, and $\vdash^{\text{top}} M' \triangleright \Delta'$ with $\Delta' \sqsubseteq_a G'$.

Assume $\Delta \sqsubseteq_a G$. By the soundness of the association, $G \rightarrow$ implies $\Delta \rightarrow$. Suppose $M \equiv \prod_{p \in \text{dom}(\Delta)} (p \triangleleft P_p \mid p \triangleleft h_p)$ and $\vdash^{\text{top}} M' \triangleright \Delta$ is derived with (67) above. By Thm. 30, there exist M' , Δ' , and Δ'' such that $M \rightarrow^+ M'$, $\Delta'' \rightarrow \Delta'$, and $\Delta'' \leq_a \Delta$. Hence by the completeness of the association, and Thm. 29, $G \rightarrow G'$ and $\Delta' \sqsubseteq_a G'$ with $\vdash^{\text{top}} M' \triangleright \Delta'$, as desired. $\square$

Remark 9 (No Session Fidelity Labels). Unlike Remark 8, we cannot identify the rules used in the session reduction from the initial global transition. We cannot even identify the participant undertaking the reduction! The reason is the same as in Remark 5, the asynchronous subtyping allows us to reorganise participants' actions. Therefore, the global type transition may reflect an action that can be brought forward by supertyping, but that the session cannot do right now. Regardless, the concluding transitions are induced by Thm. 29, hence the labels can be deduced from Remark 8.

Next we show that typed multiparty sessions are live.

Theorem 32 (Session Liveness, [29, Theorem 4.12]). *Assume $\vdash^{\text{bot}} M \triangleright \Delta$ with Δ live. Then M is live.*

Theorem 33 (Liveness of the Top-Down System). *Assume $\vdash^{\text{top}} M \triangleright \Delta$. Then, 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 by Thm. 13 and we have that $\vdash^{\text{bot}} M \triangleright \Delta$ as $\vdash^{\text{top}} M \triangleright \Delta$. Then by Thm. 32, M is live. $\square$

Example 17 (Program Requiring Message Reordering). Consider a program that may exhibit different behaviours depending on non-deterministic control flow; this may require the precise asynchronous subtyping to type, since the different behaviours may need to be reconciled via message reordering.

Recall $G_{\text{non-det}}$ from Ex. 7, its projections, and their subtypes:

$$G_{\text{non-det}} = \mu t. p \rightarrow q : \left\{ \begin{array}{l} m_1 . q \rightarrow r : m_1 . t \\ m_2 . q \rightarrow r : m_2 . p \rightarrow r : m \end{array} \right\} \quad (68)$$

$$T_p = \mu t. q \oplus \left\{ \begin{array}{l} m_1 . t \\ m_2 . r \oplus m \end{array} \right\} \quad T_q = \mu t. p \& \left\{ \begin{array}{l} m_1 . r \oplus m_1 . t \\ m_2 . r \oplus m_2 \end{array} \right\} \quad T_r = \mu t. q \& \left\{ \begin{array}{l} m_1 . t \\ m_2 . p \& m \end{array} \right\} \quad (69)$$

$$T_n = r \oplus m . \underbrace{q \oplus m_1 \cdots q \oplus m_1 . q \oplus m_2}_n \quad T' = \mu t. q \oplus m_1 . t \quad (70)$$

We can use these to type a protocol that cannot be typed without the precise asynchronous subtyping. Suppose that e_1 and e_2 are non-deterministic boolean typed expressions.

$$P_p = \begin{array}{l} \text{if } e_1 \text{ then } r!m.q!m_2 \\ \text{else if } e_2 \text{ then } r!m.q!m_1.q!m_2 \\ \text{else } \mu X.q!m_1.X \end{array} \quad P_q = \mu X. \sum \left\{ \begin{array}{l} p?m_1.r!m_1.X \\ p?m_2.r!m_2 \end{array} \right\} \quad P_r = \mu X. \sum \left\{ \begin{array}{l} q?m_1.X \\ q?m_2.p?m \end{array} \right\} \quad (71)$$

We can derive the following type judgements without using subtyping:

$$\vdash P_q \triangleright T_q \quad \vdash P_r \triangleright T_r \quad \vdash r!m.q!m_2 \triangleright T_0 \quad \vdash r!m.q!m_1.q!m_2 \triangleright T_1 \quad \vdash \mu X.q!m_1.X \triangleright T' \quad (72)$$

We have that, for all $n \geq 0$, $T_n \leq_a T_p$ and $T' \leq_a T_p$, so, using [T-SUB], we get the following type judgements:

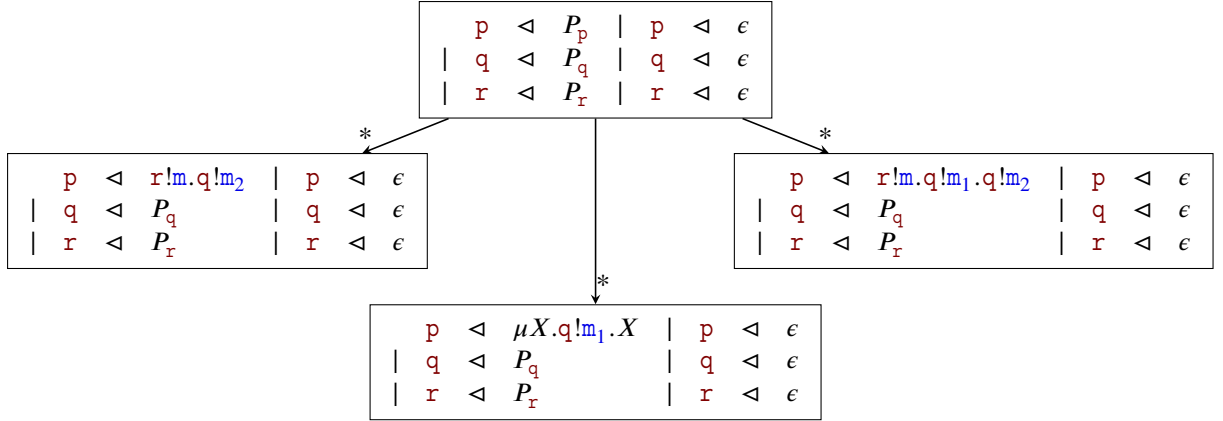
$$\vdash r!m.q!m_2 \triangleright T_p \quad \vdash r!m.q!m_1.q!m_2 \triangleright T_p \quad \vdash \mu X.q!m_1.X \triangleright T_p \quad (73)$$

Hence, we can type $\vdash P_p \triangleright T_p$, allowing us to type the session using $G_{\text{non-det}}$:

$$\vdash (p \triangleleft P_p \mid p \triangleleft \epsilon \mid q \triangleleft P_q \mid q \triangleleft \epsilon \mid r \triangleleft P_r \mid r \triangleleft \epsilon) \triangleright (p : T_p, q : T_q, r : T_r) \sqsubseteq_a G'_{\text{non-det}} \quad (74)$$

Therefore, the session is safe, deadlock-free and live by Thm. 33. If our subtyping did not include asynchronous message reordering, then we would be unable to reconcile the control flows of P_p , making it untypable.

We use this session to witness the utility of the non-deterministic global transition relation, as in Ex. 7, by considering the three possible control flows:



All three sessions are typed by $G_{\text{non-det}}$, but, after p enqueues the next message to r in the left and right sessions, the session on the left can be typed by G' while the session on the right cannot be. Where:

$$G' = p \rightarrow q : m_2 . q \rightarrow r : m_2 . p \xrightarrow{m} r : m$$

This explains why the transitions of $G_{\text{non-det}}$ allow for non-determinism: it needs to handle all processes of the above form (with any number of communications from p to q), which no single transition can handle simultaneously.

7. Related Work

Preciseness of Subtyping. For synchronous programs, two subtyping approaches have been proposed: one allowing the safe substitution of channels (called *channel subtyping*) [27] and the other allowing the safe substitution of processes (called *process subtyping*) [19]. In this work, we use process subtyping, which represents refinement of process behaviours.

Preciseness of subtyping was first proposed for the call-by-value λ -calculus with iso-recursive types by Ligatti et al. [42]. Chen et al. [13] introduced and proved *preciseness* of synchronous [19, 27] and asynchronous subtyping for the binary (2-party) session types. Later, the asynchronous subtyping was found to be *undecidable*, independently by Bravetti et al. [8], and by Lange and Yoshida [37]. This provoked active studies on identifying a set of binary session types where asynchronous subtyping is decidable; and proposing *sound* algorithms [4, 7] extending the formalism to *binary communicating automata* [6]. A variant of asynchronous binary session types based on service contract theory, called the *fair* asynchronous subtyping relation, was also proposed and proved sound and complete in [9].

In the multiparty setting, Ghilezan et al. [28, 29] proposed precise synchronous and asynchronous session subtyping, and our theory is centred on precise multiparty asynchronous subtyping. Recently, Bocchi et al. [5] have shown a sound algorithm for asynchronous multiparty session types that can decide subtyping for non-trivial multiparty communication patterns, extending an abstract interpretation framework from Bocchi et al [4]. Li et al. [38] proposed a sound and complete algorithm for asynchronous behavioural contract refinement subtyping on communicating state machines (CSMs) [6], which is distinct from the precise asynchronous subtyping: as it is defined to be complete with respect to subprotocol fidelity. As noted in [4, § 7], classically, the subtyping question is answered pairwise: by comparing a type against a candidate subtype, without reference to a global type. Our classical formulation is favourable when types are mined from source code, allowing *local* optimisations of specifications and code per participant. See the paragraph on **Implementations** below.

Top-Down Asynchronous Systems. The first multiparty session types system [34, 35] (MPST) introduced *asynchronous multiparty session processes* which interact through FIFO queues. The typing system uses the most restricted end-point projection (called an inductive projection with plain merging) and does not consider subtyping. Later this projection was extended to an inductive projection with full merging with synchronous subtyping relations, which is used in several asynchronous MPST systems (e.g., [22]). Scalas and Yoshida have discovered that proofs of an inductive projection with full merging in the literature are flawed [51]. Recently this proof was fixed by Hou and Yoshida [36] for a *synchronous calculus*, using an association relation between a global type and typing context up to the synchronous subtyping relation. We apply their proof method to an asynchronous calculus up to the precise asynchronous subtyping, with a coinductive projection with full merging; the coinductive full merge subsumes the inductive full merge of earlier MPST systems, since any inductively-derivable merge also satisfies our coinductive rules. Hence our type system offers strictly larger typable processes than those typed by the association relation in [36]. Barbanera and Dezani-Ciancaglini [2] extend global types with syntax for delegation and separate the sending and receiving syntax, allowing some of the expressivity given by the precise asynchronous subtyping. They give a decidable type system directly between sessions and global types, without using a projection. Notably, their global types do not include en-route transmissions and are not given semantics; instead they define *type configurations*, which are pairs of global types and queues, and give these semantics. Unlike Thms. 29 and 31, their subject reduction and session fidelity theorems refer to type configurations *not* global types. This subsumes points (1) and (2) from Def. 16, and avoids point (3) since messages can be globally dequeued at only point, independent of the structure of the global type.

Asynchronous local types are often viewed as Communicating Finite State Machines (CFSMs) [6]. The work [20] which first studied a connection between multiparty local types and CFSMs proposed a graphical asynchronous calculus with mixed choice which is typed by a graphical global type with mixed choice and parallel composition. After this work, researchers started studying properties of global types or global protocols using the theory of CFSMs and applied their theories to implementations of MPSTs, see [60]. Extending methodologies of message sequence charts (MSC), recent work focus on *implementability* or *realisability*—whether there is a CFSM implementation of the specified global protocol (or MSC) [23, 39, 44, 52]. Their approach aims to produce decidable, sound and complete (with respect to traces) projections from sender-driven global types (or global state machines) to CFSMs.

While CFSMs related session type works only focus on CFSMs (and global protocols), recent work in [53] has proposed Protocol State Machines (PSMs) and Automata-based Multiparty Protocols (AMP) as an alternative framework for asynchronous message passing processes. Their PSMs project onto CFSMs, which are then used to type processes, similar to how global types project onto local types. While a typing context projected from a global type (modulo the synchronous subtyping) can only produce 1-synchronous and half-duplex behaviours (as proven *multiparty compatibility* in [21]), their work [20, 53] and ours present more expressive global type behaviours. By extending en-route transmissions notation and context transmission rules for global types, our global and local type semantics do not impose half-duplex limitations.

The most important distinction between all of the above CFSM systems and ours is the property of *liveness*; using well-formed global types enforces liveness in the presence of single selection and conditional processes (features of the original session calculi [32, 54]), while the calculi from [20, 53] exclude these constructs. In our system, liveness is downward closed with respect to $\leq_a$ as proven in [29, Lemma 4.10]. See [51, Example 5.14] for a counterexample which demonstrates liveness without the fair path assumption (weaker liveness) is *not* downward closed with respect to subtyping relations, thus the typing system is unsound under their weaker liveness (which corresponds to orphan message freedom in [20, 21, 53]). In contrast, we have proven that the coinductive projection with full merging is sound and complete with respect to asynchronous trace reordering, demonstrating the correctness of the top-down MPST with asynchronous precise multiparty session subtyping relation.

Implementations of Asynchronous Multiparty Session Subtyping. Asynchronous session subtyping was first introduced for messaging optimisation in session-based high-performance computing platforms, i.e., multicore C programming [33, 59] and MPI-C [46, 47]. Castro-Perez and Yoshida [12] proposed CAMP, which is a static performance analysis framework for message-passing concurrent and distributed systems based on MPST. CAMP augments MPST with annotations of communication latency and local computation cost, defined as estimated execution times, which is used to extract cost equations from protocol descriptions and to statically predict the communication cost. CAMP is also extended to analyse asynchronous communication optimised programs. The tool, based on cost theory, is applicable to different existing benchmarks and usecases in the literature with a wide range of communication protocols.

The Rust programming framework, Rumpsteak [18], incorporates multiparty asynchronous subtyping to optimise asynchronous message-passing in the Rust programming language. It proposes an algorithm for asynchronous subtyping based on the session decomposition technique that is bounded by a number of iterations and proved to be sound and decidable. They evaluate the performance and expressiveness of Rumpsteak against previous Rust implementations and algorithms, and show that Rumpsteak is more efficient and can safely implement case studies by offering arbitrary ordering of messages.

Mechanisations of Asynchronous Subtyping Relations. Hinrichsen et al. [31] introduce Actris, a Iris Rocq that integrates separation logics and asynchronous binary session types with the asynchronous subtyping by Chen et al. [13]. The first mechanisation of multiparty asynchronous subtyping by the Rocq proof assistant is proposed by Ekici and Yoshida [25,26]. Their Rocq formalisation follows a session tree decomposition approach, using the greatest fixed point of the least fixed point technique, facilitated by the paco library, to define coinductive predicates. Li et al. [41] have formalised correctness of their implementation problem of refinement global specifications [40] in Rocq. It is an interesting future work to combine it with the Rocq mechanisation of the top-down approach by Ekici et al. [24], to formalise the subject reduction, deadlock-freedom and liveness theorems based on this work.

8. 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 coinductive 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 provided by Barwell et al. [3] and Honda et al. [35]. 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 [29, Theorem 4.11 and 4.13]. By proving that the projection Δ of G is safe, deadlock-free and live, we can derive that asynchronous multiparty session processes typed by the top-down typing system ([SESSTOP]) are also safe, deadlock-free and live (Theorem 33).

While Ghilezan et al. [29] have proved the subject reduction theorem and session fidelity theorem under the subsumption rule of $\leq_a$, it does not use the top-down typing system. On the other hand, Ghilezan et al. [28] have shown that multiparty synchronous subtyping is precise in the synchronous multiparty session calculus using the top-down system. None of the previous works has studied: (1) properties of coinductive full merge projection; (2) liveness of asynchronously associated typing contexts; nor (3) association with respect to asynchronous subtyping. An interesting open question is whether the association theorems hold for the sound decidable asynchronous subtyping relations [5, 18] so that we can derive the subject reduction theorems under those relations.

Pischke and Yoshida [50] have shown that in synchronous semantics, the top-down system has the same typability as the bottom-up system, which ensures liveness, if the top-down system meets three conditions: (1) global types are balanced; (2) the projection is coinductive with full merging; and (3) the subsumption rule uses $\leq$. We have proven the correctness of the bottom-up system with $\leq_a$ implies the correctness of the top-down system where (1) a global type is balanced⁺; (2) a projection is coinductive with full merging (soundness of typability). It is an open question whether our top-down system can *completely* type sessions which are typable by the bottom-up system, which ensures asynchronous liveness (completeness of typability).

We have demonstrated the usefulness of association in deriving the main theorems of the top-down system, by *reusing* the theorems in [29]. We have not yet reached a stage to claim that MPST is a theoretical framework for building asynchronous software systems. There still needs to be more effort applied to developing practical applications of MPST with asynchronous subtyping relations, for testing and maintaining compositionality and reusability of protocols. The most challenging topic is to type individual asynchronous *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 distributed 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 will continue our avenue to promote asynchronous optimisations with MPST for specifying and implementing real-world distributed components.

Acknowledgements. We thank the reviewers for their detailed comments and helpful suggestions.

References

- [1] Message sequence chart (msc). Recommendation Z.120, International Telecommunication Union (ITU-T), Geneva, October 1996. (10/96). Revised and approved by WTSC (Geneva, 9–18 Oct 1996).
- [2] Franco Barbanera and Mariangiola Dezani-Ciancaglini. Asynchronous multiparty sessions with internal delegation - dedicated to rocco de nicola on the occasion of his 70th birthday. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation. REoCAS Colloquium in Honor of Rocco De Nicola - 12th International Symposium, ISO LA 2024, Crete, Greece, October 27-31, 2024, Proceedings, Part I*, volume 15219 of *Lecture Notes in Computer Science*, pages 322–339. Springer, 2024.
- [3] Adam D. Barwell, Ping Hou, Nobuko Yoshida, and Fangyi Zhou. Crash-Stop Failures in Asynchronous Multiparty Session Types. *Logical Methods in Computer Science*, 2025.
- [4] Laura Bocchi, Andy King, and Maurizio Murgia. Asynchronous subtyping by trace relaxation. In Bernd Finkbeiner and Laura Kovács, editors, *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, April 6-11, 2024, Proceedings, Part I*, volume 14570 of *Lecture Notes in Computer Science*, pages 207–226. Springer, 2024.
- [5] Laura Bocchi, Andy King, Maurizio Murgia, and Simon Thompson. Abstract Subtyping for Asynchronous Multiparty Sessions. In Patricia Bouyer and Jaco van de Pol, editors, *36th International Conference on Concurrency Theory (CONCUR 2025)*, volume 348 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:19, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.
- [6] Daniel Brand and Pitro Zafiropulo. On Communicating Finite-State Machines. *Journal of the ACM*, 30(2):323–342, April 1983.
- [7] Mario Bravetti, Marco Carbone, Julien Lange, Nobuko Yoshida, and Gianluigi Zavattaro. A Sound Algorithm for Asynchronous Session Subtyping and its Implementation. *Logical Methods in Computer Science*, Volume 17, Issue 1, March 2021.
- [8] Mario Bravetti, Marco Carbone, and Gianluigi Zavattaro. Undecidability of asynchronous session subtyping. *Inf. Comput.*, 256:300–320, 2017.
- [9] Mario Bravetti, Luca Padovani, and Gianluigi Zavattaro. A Sound and Complete Characterization of Fair Asynchronous Session Subtyping. In Patricia Bouyer and Jaco van de Pol, editors, *36th International Conference on Concurrency Theory (CONCUR 2025)*, volume 348 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:17, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.
- [10] Marco Carbone, Kohei Honda, and Nobuko Yoshida. Structured communication-centered programming for web services. *ACM Trans. Program. Lang. Syst.*, 34(2):8:1–8:78, 2012.
- [11] David Castro-Perez, Francisco Ferreira, Lorenzo Gheri, and Nobuko Yoshida. Zood: a DSL for certified multiparty computation: from mechanised metatheory to certified multiparty processes. In Stephen N. Freund and Eran Yahav, editors, *PLDI '21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 2021*, pages 237–251. ACM, 2021.
- [12] David Castro-Perez and Nobuko Yoshida. CAMP: cost-aware multiparty session protocols. *Proc. ACM Program. Lang.*, 4(OOPSLA):155:1–155:30, 2020.
- [13] Tzu Chun Chen, Mariangiola Dezani-Ciancaglini, Alceste Scalas, and Nobuko Yoshida. On the preciseness of subtyping in session types. *Logical Methods in Computer Science*, 13(2), June 2017.

- [14] Tzu-Chun Chen, Mariangiola Dezani-Ciancaglini, and Nobuko Yoshida. 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, New York, NY, USA, 2024. Association for Computing Machinery.
- [15] Mario Coppo, Mariangiola Dezani-Ciancaglini, Luca Padovani, and Nobuko Yoshida. A Gentle Introduction to Multiparty Asynchronous Session Types. In *15th International School on Formal Methods for the Design of Computer, Communication and Software Systems: Multicore Programming*, volume 9104 of *LNCS*, pages 146–178. Springer, 2015.
- [16] Mario Coppo, Mariangiola Dezani-Ciancaglini, Nobuko Yoshida, and Luca Padovani. Global Progress for Dynamically Interleaved Multiparty Sessions. *MSCS*, 26:238–302, 2015.
- [17] Zak Cutner and Nobuko Yoshida. Safe session-based asynchronous coordination in rust. In *Coordination Models and Languages: 23rd IFIP WG 6.1 International Conference, COORDINATION 2021, Held as Part of the 16th International Federated Conference on Distributed Computing Techniques, DisCoTec 2021, Valletta, Malta, June 14–18, 2021, Proceedings*, page 80–89, Berlin, Heidelberg, 2021. Springer-Verlag.
- [18] Zak Cutner, Nobuko Yoshida, and Martin Vassor. Deadlock-Free Asynchronous Message Reordering in Rust with Multiparty Session Types. In *27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, volume abs/2112.12693 of *PPoPP '22*, pages 246–261. ACM, 2022.
- [19] Romain Demangeon and Kohei Honda. Full abstraction in a subtyped pi-calculus with linear types. In *Proceedings of CONCUR 2011*, volume 6901 of *LNCS*, pages 280–296. Springer, 2011.
- [20] Pierre-Malo Deniélou and Nobuko Yoshida. Multiparty session types meet communicating automata. In *ESOP*, volume 7211 of *LNCS*, pages 194–213. Springer, 2012.
- [21] Pierre-Malo Deniélou and Nobuko Yoshida. Multiparty compatibility in communicating automata: Characterisation and synthesis of global session types. In *ICALP 2013*, pages 174–186, 2013.
- [22] Pierre-Malo Deniélou, Nobuko Yoshida, Andi Bejleri, and Raymond Hu. Parameterised multiparty session types. *Logical Methods in Computer Science*, 8(4), 2012.
- [23] Cinzia Di Giusto, Etienne Lozes, and Pascal Urso. Realisability and complementability of multiparty session types. In *Proceedings of the 27th International Symposium on Principles and Practice of Declarative Programming*, PPDP '25, New York, NY, USA, 2025. Association for Computing Machinery.
- [24] Burak Ekici, Tadayoshi Kamegai, and Nobuko Yoshida. Formalising Subject Reduction and Progress for Multiparty Session Processes. In *16th International Conference on Interactive Theorem Proving, 2025, Reykjavik, Iceland*, volume 352 of *LIPICs*, pages 19:1–19:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025.
- [25] Burak Ekici and Nobuko Yoshida. Completeness of Asynchronous Session Tree Subtyping in Coq. In *15th International Conference on Interactive Theorem Proving, 2024, Tbilisi, Georgia*, volume 309, pages 6:1–6:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.
- [26] Burak Ekici and Nobuko Yoshida. Formalising Asynchronous Session Subtyping. *ACM Transactions on Computational Logic*, 27:1–45, 2026.
- [27] Simon Gay and Malcolm Hole. Subtyping for session types in the pi calculus. *Acta Informatica*, 42(2-3):191–225, November 2005.
- [28] Silvia Ghilezan, Svetlana Jakšić, Jovanka Pantović, Alceste Scalas, and Nobuko Yoshida. Precise subtyping for synchronous multiparty sessions. *Journal of Logical and Algebraic Methods in Programming*, 104:127–173, April 2019.
- [29] Silvia Ghilezan, Jovanka Pantović, Ivan Prokić, Alceste Scalas, and Nobuko Yoshida. Precise subtyping for asynchronous multiparty sessions. *ACM Trans. Comput. Logic*, 24(2), nov 2023.

- [30] Peter Graubmann, Ekkart Rudolph, and Jens Grabowski. The standardization of message sequence charts. In *Proceedings of the IEEE Software Engineering Standards Symposium (SESS '93)*, pages 48–63, Brighton, United Kingdom, September 1993. IEEE Computer Society.
- [31] Jonas Kastberg Hinrichsen, Jesper Bengtson, and Robbert Krebbers. Actris 2.0: Asynchronous session-type based reasoning in separation logic. *Logical Methods in Computer Science*, Volume 18, Issue 2, Jun 2022.
- [32] Kohei Honda, Vasco T. Vasconcelos, and Makoto Kubo. Language primitives and type disciplines for structured communication-based programming. In *Proceedings of ESOP 1998*, volume 1381 of *LNCS*, pages 22–138. Springer, 1998.
- [33] Kohei Honda, Vasco Thudichum Vasconcelos, and Nobuko Yoshida. Type-Directed Compilation for Multicore Programming. *ENTCS*, 241:101–111, 2009.
- [34] Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty Asynchronous Session Types. In *Proceedings of POPL 2008*, pages 273–284. ACM, 2008.
- [35] Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. *Journal of the ACM*, 63(1):9:1–9:67, 2016.
- [36] Ping Hou, Nobuko Yoshida, and Iona Kuhn. Less is More Revisited: Association with Global Protocols and Multiparty Sessions. *Theoretical Computer Science*, 1076:–, 2026.
- [37] Julien Lange and Nobuko Yoshida. On the Undecidability of Asynchronous Session Subtyping. In *Proceedings of FOSSACS 2017*, volume 10203 of *LNCS*, pages 441–457. Springer, 2017.
- [38] Elaine Li, Felix Stutz, and Thomas Wies. Deciding subtyping for asynchronous multiparty sessions. In *Lecture Notes in Computer Science*, Lecture notes in computer science, pages 176–205. Springer Nature Switzerland, Cham, 2024.
- [39] Elaine Li, Felix Stutz, Thomas Wies, and Damien Zufferey. Complete multiparty session type projection with automata. In *Lecture Notes in Computer Science*, Lecture notes in computer science, pages 350–373. Springer Nature Switzerland, Cham, 2023.
- [40] Elaine Li, Felix Stutz, Thomas Wies, and Damien Zufferey. Characterizing implementability of global protocols with infinite states and data. *Proc. ACM Program. Lang.*, 9(OOPSLA1), April 2025.
- [41] Elaine Li and Thomas Wies. Certified Implementability of Global Multiparty Protocols. In Yannick Forster and Chantal Keller, editors, *16th International Conference on Interactive Theorem Proving (ITP 2025)*, volume 352 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 15:1–15:20, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.
- [42] Jay Ligatti, Jeremy Blackburn, and Michael Nachtigal. On subtyping-relation completeness, with an application to iso-recursive types. *ACM Trans. Program. Lang. Syst.*, 39(1), March 2017.
- [43] Barbara H Liskov and Jeannette M Wing. A behavioral notion of subtyping. *ACM Trans. Program. Lang. Syst.*, 16(6):1811–1841, November 1994.
- [44] Rupak Majumdar, Madhavan Mukund, Felix Stutz, and Damien Zufferey. Generalising projection in asynchronous multiparty session types. In Serge Haddad and Daniele Varacca, editors, *32nd International Conference on Concurrency Theory, CONCUR 2021, August 24-27, 2021, Virtual Conference*, volume 203 of *LIPIcs*, pages 35:1–35:24. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.
- [45] Dimitris Mostrous and Nobuko Yoshida. Session-based communication optimisation for higher-order mobile processes. In Pierre-Louis Curien, editor, *Typed Lambda Calculi and Applications, 9th International Conference, TLCA 2009, Brasilia, Brazil, July 1-3, 2009. Proceedings*, volume 5608 of *Lecture Notes in Computer Science*, pages 203–218. Springer, 2009.

- [46] Nicholas Ng, Jose G.F. Coutinho, and Nobuko Yoshida. Protocols by Default: Safe MPI Code Generation based on Session Types. In *24th International Conference on Compiler Construction*, volume 9031 of *LNCS*, pages 212–232. Springer, 2015.
- [47] Nicholas Ng, Nobuko Yoshida, and Kohei Honda. Multiparty Session C: Safe Parallel Programming with Message Optimisation. In *50th International Conference on Objects, Models, Components, Patterns*, volume 7304 of *LNCS*, pages 202–218. Springer, 2012.
- [48] Benjamin Pierce. *Types and Programming Languages*. MIT Press, 2002.
- [49] Kai Pischke and Nobuko Yoshida. Asynchronous Global Protocols, Precisely. In *Components Operationally: Reversibility and System Engineering*, volume 16065 of *LNCS*, pages 116–133. Springer Nature, 2025.
- [50] Kai Pischke and Nobuko Yoshida. Top Down=Bottom Up: Sound and Complete Characterisations of Liveness by Multiparty Global Protocols. *Proc. ACM Program. Lang.*, 2026.
- [51] Alceste Scalas and Nobuko Yoshida. Less is more: Multiparty session types revisited. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–29, January 2019.
- [52] Felix Stutz. Asynchronous Multiparty Session Type Implementability is Decidable - Lessons Learned from Message Sequence Charts. In Karim Ali and Guido Salvaneschi, editors, *37th European Conference on Object-Oriented Programming (ECOOP 2023)*, volume 263 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 32:1–32:31, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.
- [53] Felix Stutz and Emanuele D’Osualdo. An automata-theoretic basis for specification and type checking of multiparty protocols. In *Programming Languages and Systems: 34th European Symposium on Programming, ESOP 2025, Held as Part of the International Joint Conferences on Theory and Practice of Software, ETAPS 2025, Hamilton, ON, Canada, May 3–8, 2025, Proceedings, Part II*, page 314–346, Berlin, Heidelberg, 2025. Springer-Verlag.
- [54] Kaku Takeuchi, Kohei Honda, and Makoto Kubo. An Interaction-based Language and its Typing System. In Constantine Halatsis, Dimitris G. Maritsas, George Philokyprou, and Sergios Theodoridis, editors, *PARLE ’94: Parallel Architectures and Languages Europe, 6th International PARLE Conference, Athens, Greece, July 4-8, 1994, Proceedings*, volume 817 of *Lecture Notes in Computer Science*, pages 398–413. Springer, 1994.
- [55] Thien Udomsrirungruang and Nobuko Yoshida. Top-down or bottom-up? complexity analyses of synchronous multiparty session types. *Proc. ACM Program. Lang.*, 9(POPL):1040–1071, 2025.
- [56] Bas van den Heuvel and Jorge A Pérez. A decentralized analysis of multiparty protocols. *Sci. Comput. Program.*, 222(102840):102840, October 2022.
- [57] Nobuko Yoshida. Programming language implementations with multiparty session types. In Frank S. de Boer, Ferruccio Damiani, Reiner Hähnle, Einar Broch Johnsen, and Eduard Kamburjan, editors, *Active Object Languages: Current Research Trends*, volume 14360 of *Lecture Notes in Computer Science*, pages 147–165. Springer, 2024.
- [58] Nobuko Yoshida, Raymond Hu, Romyana Neykova, and Nicholas Ng. The Scribble Protocol Language. In *8th International Symposium on Trustworthy Global Computing*, volume 8358 of *LNCS*, pages 22–41. Springer, 2013.
- [59] Nobuko Yoshida, Vasco Thudichum Vasconcelos, Hervé Paulino, and Kohei Honda. Session-based compilation framework for multicore programming. In *FMCO 2008*, pages 226–246, 2008.
- [60] Nobuko Yoshida, Fangyi Zhou, and Francisco Ferreira. Communicating Finite State Machines and an Extensible Toolchain for Multiparty Session Types. In *23rd International Symposium on Fundamentals of Computation Theory*, volume 12867 of *LNCS*, pages 18–35. Springer, 2021.