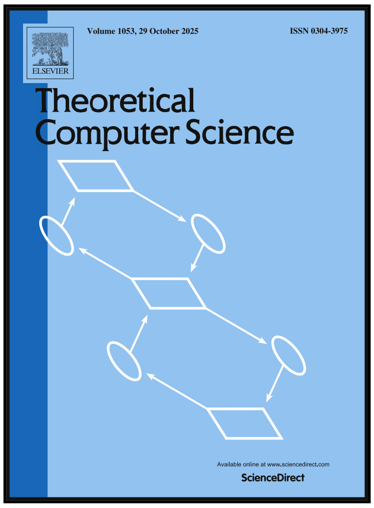
Editor: Dr. Vladimiro Sassone

Journal Pre-proof



Less is More Revisited Association with Global Protocols and Multiparty Sessions

Ping Hou, Nobuko Yoshida, Iona Kuhn

PII:	S0304-3975(26)00132-5
DOI:	https://doi.org/10.1016/j.tcs.2026.115873
Reference:	TCS 115873
To appear in:	*Theoretical Computer Science*
Received date:	18 September 2025
Revised date:	23 February 2026
Accepted date:	2 March 2026

Please cite this article as: Ping Hou, Nobuko Yoshida, Iona Kuhn, Less is More Revisited Association with Global Protocols and Multiparty Sessions, *Theoretical Computer Science* (2026), doi: https://doi.org/10.1016/j.tcs.2026.115873

Less is More Revisited
Association with Global Protocols and Multiparty Sessions

Ping Hou[a,*], Nobuko Yoshida[a], Iona Kuhn[b]

[a]Department of Computer Science, University of Oxford, United Kingdom
[b]Department of Computer Science, Saarland University, Germany

Abstract

Ensuring correctness of communication in distributed systems remains challenging. To address this, Multiparty session types (MPST), initially introduced by Honda et al. [52, 53], offer a type discipline in which a programmer or architect specifies an overall view of communication as a *global protocol* (*global type*), and each distributed program is locally type-checked against its *end-point projection*. In practice, the MPST framework has been integrated into over 25 programming languages or tools. Ten years after the emergence of MPST, Scalas and Yoshida [84] discovered that existing *proofs* of type safety using end-point projection with *mergeability* are flawed, where the mergeability operator enlarges the typability of MPST end-point programs, admits easy implementation, and is more efficient than alternative approaches, including model checking. Nevertheless, following the result in [84], the soundness of end-point projection (with mergeability) has been interpreted in the literature as *problematic*. We clarify this concern by proposing a new general proof technique for *type soundness* (*subject reduction*) of multiparty session π-calculus, which relies on an *association* relation between the behavioural semantics of a global type and its end-point projection. With this approach, *behavioural properties*, namely *session fidelity*, *deadlock freedom*, and *liveness*, are also guaranteed based on global types. Additionally, we provide detailed comparisons with existing MPST typing systems and discuss their respective proof methods for type soundness.

Keywords: Mobile processes, Session types, Global types, End-point projection, Subject reduction, Type soundness, Behavioural properties

1. Introduction

Distributed systems are built upon interactions between concurrent processes, implemented using *message-passing* communication abstractions. In this model, an interaction between processes can be interpreted as the exchange of messages, forming a *protocol* that consists of sending and receiving values, making choices between multiple possible paths, and repeating or terminating the interaction. Such protocols are carried out over communication transports used in web applications, ranging from request-response client-server interactions via HTTP to full-duplex communication channels via the WebSocket protocol [37].

Multiparty session types (MPST) [52, 53] constitute a type formalism inspired by the Web Service Choreography Description Language (WS-CDL) [47], originating as *abstract choreographies* or *choreography APIs* that extract communication flows and message types while abstracting away from program-level constructs such as conditionals, assignments, and concrete values. This formalism facilitates the description, specification, and verification of communication in concurrent and distributed systems based on *multiparty protocols*. Intuitively, MPST describe structured multiparty interactions closely related to interaction-based formalisms such as message sequence charts (MSCs) and choreographies; for instance, multiparty protocols

*Corresponding author
Email addresses: ping.hou@cs.ox.ac.uk (Ping Hou), nobuko.yoshida@cs.ox.ac.uk (Nobuko Yoshida), ioku00001@stud.uni-saarland.de (Iona Kuhn)

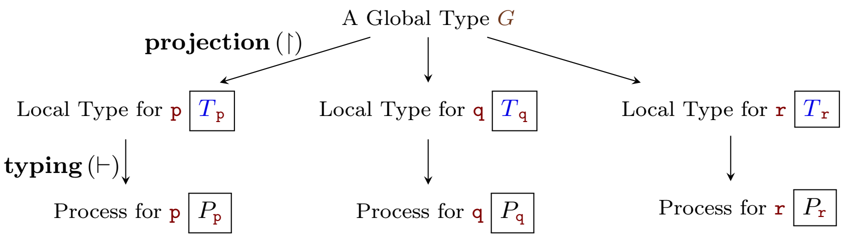


Figure 1: Top-down methodology of multiparty session types

are commonly presented in an MSC-like form (e.g. [53, Figs. 1 and 2]), while additionally providing a typing discipline and projection mechanism that support implementability and protocol compliance.

The methodology of MPST begins with specifying a protocol, called a *global type*, which describes a sequence of communication actions, choices, and recursions between two or more participants. The global type is then *projected* into a set of *end-point local* (*session*) *types*, each representing a participant's viewpoint. Well-typed *end-point implementations* (*processes*) that conform to a global type are guaranteed to be *correct by construction*, ensuring safety, deadlock-freedom, and liveness of interactions. On the practical side, MPST are *language-agnostic*, i.e. specifications and end-point projection algorithms do not depend on specific programming languages. The top-down approach has been implemented in a variety of mainstream programming languages, including Java [55, 56, 64, 6], Go [16], Rust [27, 65, 54, 95], TypeScript [68, 42], PureScript [62], Scala [82, 96, 24, 1, 36, 59], OCaml [103, 57], MPI-C [74], F⋆ [104], F♯ [70], Python [73, 72, 29], Erlang [38, 71, 34], C [75, 76], C♯ [61], and domain-specific actor languages [17, 18, 48]. These implementations adopt diverse styles, such as code generation, API generation, static type checking, protocol compliance, and runtime monitoring, facilitating their application in real-world programs [98].

Figure 1 illustrates the MPST workflow. The design of multiparty protocols begins with a global type G (top of Figure 1), and each participant's implementation (process) $P_{\mathtt{p}}$ (bottom) relies on its local (session) type $T_{\mathtt{p}}$ (middle), obtained via end-point projection of the global type $G \upharpoonright \mathtt{p}$. The global and local types capture the global and local communication behaviours, respectively. Since each process $P_{\mathtt{p}}$ conforms to its type $T_{\mathtt{p}}$, the resulting set of processes collectively makes progress according to the global type.

Top-Down Multiparty Session Types. In this paper, we demonstrate the rigour of the MPST *top-down* approach by revisiting the work of Scalas and Yoshida [84]. That work proposed a general MPST framework, known as the *bottom-up* approach, which does *not* require global types. Within the bottom-up framework, process safety is enforced by directly verifying a relevant set of local types. Udomsrirungruang and Yoshida [94] reveal that, compared with the top-down methodology, the bottom-up offers greater typability but incurs higher computational cost, particularly for liveness checking and type inference. Moreover, as shown in [84], in asynchronous MPST – where processes communicate via unbounded FIFO queues – type checking becomes undecidable under the bottom-up approach, whereas decidability is preserved by the top-down procedure, as type checking is performed with respect to end-point types obtained via a decidable projection of a well-formed global type [52, 53][1]. Intuitively, the global type fully characterises the communication structure in advance, thereby reducing type checking to verifying local conformance rather than inferring global compatibility among independently specified local behaviours.

The work in [84] also identified flaws in certain published type-safety proofs for top-down typing systems based on projection with *mergeability*. In subsequent literature, this issue has been referred to as "*broken proofs*" or "*unsound results*", as well as the claim that "*several versions of classical projection with the full merge are flawed*", framed in terms of the "*brittleness*" of merging-based projection mechanisms and the architectural limitations of existing MPST frameworks (e.g. [15, 58, 60, 85, 86]). In particular, this has led to a characterisation of these proof flaws as "*unsoundness under a specific formal definition*" (e.g. [67]), and

[1] Type checking discussed here does not involve subtyping checks; in particular, asynchronous subtyping checking is generally undecidable (e.g. [44]).

to accounts emphasising the error-proneness of the theory (e.g. [58, 50]). Without careful qualification, such assessments may be over-generalised into conclusions about the soundness of the top-down approach (with mergeability), or even about global types being *problematic* in general.

In this paper, we show that a sound typing system can indeed be built using end-point projection with mergeability. More importantly, we demonstrate that global types enable a clear and structured proof method for type soundness. To clarify the statement in [84] precisely, we summarise global types, end-point projections, and merging operators, following their chronological development in the literature.

End-point Projection with Plain Merging. Theories and implementations based on MPST require "*correct by construction*" protocols that prevent deadlocks and type errors during the interaction of endpoint programs. This is achieved by imposing a *well-formedness condition* on global types, known as *projectability*. A global type G is *projectable* if a set of end-point types can be generated from it, following formal rules or algorithms.

The MPST framework with global types and end-point projection was first introduced by Honda et al. [52], with *linearity conditions* ensuring both the well-formedness of global types annotated with type-level channel declarations and the *projectability* of local types. Subsequently, Bettini et al. [5] proposed a simplified MPST system without channel declarations, which has since been widely adopted in both theory and practice; the global types used in this paper also follow this channel-less style. The end-point projection defined in [52, 5] employs *plain merging*, which requires that, in a branching, the projections for uninvolved roles yield identical local types across all branches.

To illustrate end-point projection (with plain merging), consider a simple global type involving three participants: Alice, Bob, and Carol. In the initial choice, A sends to B *either* a request for addition or subtraction (each carrying an integer). In both cases, B continues by forwarding the answer int to C, and the session then **end**s. This protocol is specified by the following global type G_{p}:

$$G_{\mathrm{p}} = \mathtt{A}{\to}\mathtt{B}{:} \left\{ \begin{array}{l} \mathtt{add}(\mathrm{int}).\mathtt{B}{\to}\mathtt{C}{:}\mathtt{forward}(\mathrm{int}).\mathbf{end} \\ \mathtt{sub}(\mathrm{int}).\mathtt{B}{\to}\mathtt{C}{:}\mathtt{forward}(\mathrm{int}).\mathbf{end} \end{array} \right\} \tag{1}$$

Following [52, 5], the global type G_{p} is projected onto three local types (one for each role A, B, C):

$$T_{\mathtt{A}} = \mathtt{B}{\oplus}\left\{ \begin{array}{l} \mathtt{add}(\mathrm{int}).\mathbf{end} \\ \mathtt{sub}(\mathrm{int}).\mathbf{end} \end{array} \right\} \quad T_{\mathtt{B}} = \mathtt{A}\&\left\{ \begin{array}{l} \mathtt{add}(\mathrm{int}).\mathtt{C}{\oplus}\mathtt{forward}(\mathrm{int}).\mathbf{end} \\ \mathtt{sub}(\mathrm{int}).\mathtt{C}{\oplus}\mathtt{forward}(\mathrm{int}).\mathbf{end} \end{array} \right\} \quad T_{\mathtt{C}} = \mathtt{B}\&\left\{ \mathtt{forward}(\mathrm{int}).\mathbf{end} \right\} \tag{2}$$

Here, $T_{\mathtt{A}}$ represents the interface of A in G_{p}: it must send ($\oplus$) to B either `add` or `sub`. In the first case, B receives the `add` message with $\&$, performs an addition with some specific integer, `forward`s the result to C, and the session ends. Otherwise, the message `sub` with $\&$ is received at B; after the subtraction at B, the result is `forward`ed to C, and the session likewise ends.

We highlight the projection onto C: its local behaviour is determined by merging the two choices at B. In this example, both branches deliver the same message `forward` to C, so the merge succeeds and the projection is well-defined. If instead the messages or payload types differed, the merge would fail, illustrating the constraint imposed by plain merging on the projectability of global types.

End-point Projection with Full Merging. We now modify G_{p} to obtain the following global type G_{f}, where the message sent from B to C depends on the choice made by A:

$$G_{\mathrm{f}} = \mathtt{A}{\to}\mathtt{B}{:} \left\{ \begin{array}{l} \mathtt{add}(\mathrm{int}).\mathtt{B}{\to}\mathtt{C}{:} \left\{ \begin{array}{l} \mathtt{add}(\mathrm{int}).\mathbf{end} \\ \mathtt{forward}(\mathrm{int}).\mathbf{end} \end{array} \right\} \\ \mathtt{sub}(\mathrm{int}).\mathtt{B}{\to}\mathtt{C}{:} \left\{ \begin{array}{l} \mathtt{sub}(\mathrm{int}).\mathbf{end} \\ \mathtt{forward}(\mathrm{int}).\mathbf{end} \end{array} \right\} \end{array} \right\} \tag{3}$$

Under *plain merging*, G_{f} is no longer projectable onto C, since the projections for C differ across branches.

However, if C prepares for all three message labels, `add`, `sub` and `forward`, its implementation will not get stuck. That is, we should be able to merge not only a common label like `forward`, but also input choice (branching) types from the same role with disjoint labels `add` and `sub`, into a single input type that offers both options. This motivates *full merging*, which generalises plain merging. Under the full merging strategy, projection of the global type G_f yields the following local types T'_A, T'_B, and T'_C:

$$T'_A = B\oplus\begin{Bmatrix}\texttt{add}(\text{int}).\textbf{end}\\ \texttt{sub}(\text{int}).\textbf{end}\end{Bmatrix} \quad T'_B = A\&\begin{Bmatrix}\texttt{add}(\text{int}).C\oplus\begin{Bmatrix}\texttt{add}(\text{int}).\textbf{end}\\ \texttt{forward}(\text{int}).\textbf{end}\end{Bmatrix}\\ \texttt{sub}(\text{int}).C\oplus\begin{Bmatrix}\texttt{sub}(\text{int}).\textbf{end}\\ \texttt{forward}(\text{int}).\textbf{end}\end{Bmatrix}\end{Bmatrix} \quad T'_C = B\&\begin{Bmatrix}\texttt{add}(\text{int}).\textbf{end}\\ \texttt{sub}(\text{int}).\textbf{end}\\ \texttt{forward}(\text{int}).\textbf{end}\end{Bmatrix}$$

To enlarge the set of projectable global types, end-point projection with full merging was first introduced by Yoshida et al. [99] and further developed in [31, 30, 19, 92]. Its aim is to broaden the typability of processes by allowing more global types to be considered projectable (well-formed) than under the original, more limited projection in [52]. Mergeability is implemented in the Scribble protocol description language [51, 102, 103] and other related MPST tools. As illustrated by G_f (3), without full merging, a broad range of global protocols, including those with branch-dependent continuations, are not projectable. We define the projection and mergeability formally in Definition 3.1.

Subject Reduction Theorem. We now revisit the issue highlighted in [84]. While the top-down approach to MPST guarantees *type soundness*, also known as *subject reduction*, a subtle problem arises in the *proofs* of the Subject Reduction Theorem given in [99, 30, 19, 92]. These proofs rely on an *invariance property* that is valid under projection with plain merging but *invalid* under full merging. Consequently, although both the theorem and the projection algorithm under full merging remain correct, the existing proofs are *unsound* in the general setting with full merging.

Intuitively, the *Subject Reduction Theorem* states that typed processes reduce only to typed processes and, therefore, no (untypable) error state can be reached, i.e. *"typed processes never go wrong"*.

Formally, subject reduction is stated in terms of the typing judgement:

$$\Gamma \vdash P \triangleright \Delta$$

which asserts that the process P conforms to the standard *typing context* Γ and the *session typing context* Δ. Specifically, Γ assigns base types to variables, while Δ is a collection of session types, recording for each communication channel its current protocol state.

Typically, one expects a formulation of subject reduction similar to that of the simply typed λ-calculus:

(SR1) *Assume* $\Gamma \vdash P \triangleright \Delta$. *If* $P \to P'$, *then we have* $\Gamma \vdash P' \triangleright \Delta$.

However, this statement is *too strong*, since the session typing context Δ may change when P reduces to P' due to communication. Session types are *behavioural*: channel types can evolve during interactions.

A weaker property is therefore:

(SR2) *Assume* $\Gamma \vdash P \triangleright \Delta$. *If* $P \to P'$, *then there exists* Δ' *such that* $\Gamma \vdash P' \triangleright \Delta'$.

Unfortunately, this condition is *too general (i.e., too weak)* to ensure type safety, since Δ' may be *arbitrary*.

To recover type safety, subject reduction must be strengthened by imposing an appropriate invariance condition:

(SR3) *Assume* $\Gamma \vdash P \triangleright \Delta$ *and* $\underline{\Delta \text{ satisfies property } \varphi}$. *If* $P \to P'$, *then there exists* Δ' *such that* $\Gamma \vdash P' \triangleright \Delta'$ *and* $\underline{\Delta' \text{ satisfies property } \varphi}$.

The invariance condition required by **(SR3)** is introduced at the typing of session restriction. In top-down MPST, a channel-restricted process $(\nu s)\,P$ denotes a complete session s, where each participant playing role $\mathtt{p}$ is typed by the projection $G\restriction\mathtt{p}$. Restricted processes are typed using the following rule:

$$\frac{\Delta' = \{s[\mathtt{p}] : G\restriction\mathtt{p}\}_{\mathtt{p}\in\text{roles}(G)} \qquad s \notin \Delta \qquad \Gamma \vdash P \triangleright \Delta, \Delta'}{\Gamma \vdash (\nu s : \Delta')\, P \triangleright \Delta}\ \text{[T-}\nu\text{ClassicG]} \tag{4}$$

This rule is crucial in subject reduction proofs, as it determines the initial session typing context Δ' for a newly created session s, and thereby fixes the invariance property required by **(SR3)** to hold initially.

Problem: An Incorrect Invariant φ for the Type System with Projection Using Full Merging. In binary session types, such an invariance property φ is known as *balancedness* [40], which requires the two endpoints of a channel to carry *dual types*: one offering selection ($\oplus$) while the other branching ($\&$). For multiparty session types, the challenge is therefore to identify an appropriate invariance property φ that supports a sound subject reduction argument.

Following the intuition of balancedness, a natural attempt in the multiparty setting is to define the invariance property φ as a form of duality between local types. The system in [5] introduced *consistency* (also called *coherence*, cf. [31]) as a generalisation of binary duality: a global type G projects to *consistent* local types if projecting G onto any two participants yields local types whose *partial projections* – capturing their respective behaviours towards each other – are dual.

As an illustration, consider the local types $T_{\mathtt{A}}$, $T_{\mathtt{B}}$, and $T_{\mathtt{C}}$ in (2), projected from G_{p} in (1).

For the pair $(\mathtt{A}, \mathtt{B})$, the corresponding partial projections

$$T_{\mathtt{A}} \upharpoonright \mathtt{B} \;=\; \oplus\begin{Bmatrix}\mathtt{add}(\mathrm{int}).\mathbf{end}\\ \mathtt{sub}(\mathrm{int}).\mathbf{end}\end{Bmatrix} \quad \text{and} \quad T_{\mathtt{B}} \upharpoonright \mathtt{A} \;=\; \&\begin{Bmatrix}\mathtt{add}(\mathrm{int}).\mathbf{end}\\ \mathtt{sub}(\mathrm{int}).\mathbf{end}\end{Bmatrix}$$

are dual. The same holds for the remaining pairs; hence, the projections of G_{p} are consistent.

Under plain merging, *consistency* is sufficient to establish subject reduction. Plain merging enforces structural alignment of local types – sharing the same labels, payloads, and mergeable continuations – so that projection yields consistent local types, which are preserved under reduction. However, this guarantee does not extend to full merging: **full merging admits inconsistent local types**, thereby violating **(SR3)**.

As a counterexample, consider the global type G_{w} updated from G_{p} in (1) as follows:

$$G_{\mathrm{w}} = \mathtt{A}{\to}\mathtt{B}\colon \begin{Bmatrix} \mathtt{add}(\mathrm{int}).\mathtt{B}{\to}\mathtt{C}{:}\mathtt{add}(\mathrm{int}).\mathtt{C}{\to}\mathtt{A}{:}\mathtt{forward}(\mathrm{int}).\mathbf{end} \\ \mathtt{sub}(\mathrm{int}).\mathtt{B}{\to}\mathtt{C}{:}\mathtt{forward}(\mathrm{int}).\mathbf{end} \end{Bmatrix} \tag{5}$$

The local types projected from G_{w} using full merging are:

$$T''_{\mathtt{A}} = \mathtt{B}{\oplus}\begin{Bmatrix}\mathtt{add}(\mathrm{int}).\mathtt{C}\&\mathtt{forward}(\mathrm{int})\\ \mathtt{sub}(\mathrm{int})\end{Bmatrix} \quad T''_{\mathtt{B}} = \mathtt{A}\&\begin{Bmatrix}\mathtt{add}(\mathrm{int}).\mathtt{C}{\oplus}\mathtt{add}(\mathrm{int})\\ \mathtt{sub}(\mathrm{int}).\mathtt{C}{\oplus}\mathtt{forward}(\mathrm{int})\end{Bmatrix}$$
$$T''_{\mathtt{C}} = \mathtt{B}\&\begin{Bmatrix}\mathtt{add}(\mathrm{int}).\mathtt{A}{\oplus}\mathtt{forward}(\mathrm{int})\\ \mathtt{forward}(\mathrm{int})\end{Bmatrix} \tag{6}$$

The resulting local types fail to be consistent, since the partial projections $T''_{\mathtt{A}} \upharpoonright \mathtt{C}$ and $T''_{\mathtt{C}} \upharpoonright \mathtt{A}$ are *undefined*. Intuitively, the interaction between $\mathtt{C}$ and $\mathtt{A}$ is branch-dependent on $\mathtt{B}$. Such inter-role dependencies are not captured by the syntactic nature of projection and duality checks, and may therefore lead to inconsistency.

More specifically, the issue in the subject reduction proofs of [99, 31, 30, 19, 92] results from their reliance on consistency as the underlying invariance assumption. Moreover, [31, p. 28] and [19, Prop. 2] explicitly claim that projecting a global type with full merging yields a consistent typing context. Therefore, a broader invariance condition is required for fully mergeable global types.

Solution: Association. In this paper, we prove the Subject Reduction Theorem under full merging by applying an invariance notion, namely *association*, which captures compatibility between a global type and its local types. Intuitively, association allows local types to safely conform to the global protocol, rather than requiring them to be exact projections – a relaxation that is essential under full merging.

Formally, association relates a global type and a typing context for a given multiparty session via *subtyping*. The subtyping relation $\leqslant$ on local types is typically used to enhance expressiveness by enabling more flexible processes to be typed while preserving behavioural soundness.

For instance, consider the global type G_{w} (5) and its projection on $\mathtt{B}$, $T''_{\mathtt{B}}$ (6), where $\mathtt{B}$ branches on inputs from $\mathtt{A}$. Under our subtyping discipline, branching is contravariant: a subtype of $T''_{\mathtt{B}}$ may accept additional labels, e.g. `multiplus`, beyond `add` and `sub`. A process implementing $\mathtt{B}$ can therefore support extra operations, e.g. by offering a further branch labelled `multiplus`, and remains typable by $T''_{\mathtt{B}}$ via subsumption. In this way, subtyping permits such local refinements without altering the communication pattern fixed by G_{w}; association formalises precisely this flexibility.

More precisely, a typing context Δ is *associated* with a global type G for a session s, written $\Delta \sqsubseteq_s G$, if for every role $\mathtt{p}$ of G, the endpoint $s[\mathtt{p}]$ in Δ has a type satisfying $\Delta(s[\mathtt{p}]) \leqslant G\upharpoonright\mathtt{p}$, and all other endpoints of that session are terminated (i.e. assigned **end**). Moreover, we say that Δ *is associated* if, for every session s occurring in Δ, there exists a global type G such that $\Delta_s \sqsubseteq_s G$, where Δ_s denotes the restriction of Δ to s.

With association in place, we prove the following result:

(SR) *Assume* $\Gamma \vdash P \triangleright \Delta$ *and* Δ *is associated. If* $P \to P'$*, then there exists* Δ' *such that* $\Gamma \vdash P' \triangleright \Delta'$ *with* Δ' *associated.*

This is achieved by demonstrating a *sound* and *complete* operational correspondence between global and local types with respect to association. Finally, by applying the standard subsumption rule, which allows typing contexts to be widened, we conclude that the typing system equipped with rule (4) satisfies the Subject Reduction Theorem.

Outline. **§2** introduces a multiparty session π-calculus, including its syntax and operational semantics. The calculus builds on that of [84], extended with values, expressions, and conditional statements.
§3 presents a multiparty session type theory. **§3.1** defines the syntax of global and local types, as well as projection and subtyping. **§3.2** and **§3.3** provide the semantics of global types and typing contexts (sets of local types), respectively. **§3.4** introduces the *association relation* $\Delta \sqsubseteq_s G$, which relates a global type G with a typing context Δ on a session s via projection and subtyping. The relation establishes an operational correspondence between global types and typing contexts (Thms. 3.1 and 3.2). **§3.5** motivates the need for association through an illustrative example. **§3.6** demonstrates that association ensures key typing context properties – communication safety, deadlock-freedom, and liveness (Thm. 3.3). **§3.7** clarifies the relationships among these properties (Thm. 3.4).
§4 develops a typing system for our multiparty session π-calculus. **§4.1** formalises the typing rules. **§4.2** establishes subject reduction: first with respect to association (Thm. 4.1), and subsequently using projection (Thm. 4.2), with type safety following as a corollary (Cor. 4.3). **§4.3** demonstrates session fidelity, also known as protocol conformance, i.e. that well-typed processes behave according to their session types, in an analogous way: initially via association (Thm. 4.4), and thereafter by projection (Thm. 4.5). **§4.4** shows that process properties – deadlock-freedom and liveness – are guaranteed by construction (Thm. 4.6).
We discuss related work in **§5** and conclude in **§6**. Detailed proofs are provided in the appendix.

Extensions and Refinements of Yoshida and Hou [101]. This work extends Yoshida and Hou [101], which introduced association as a proof technique for establishing type soundness in MPST. We advance this line by clarifying certain misunderstandings and challenges surrounding type soundness proofs in the MPST community, making these explicit in the Introduction. A dedicated section (§ 3.5) further illustrates the necessity of association within the framework. On the technical side, we present a full multiparty session π-calculus that extends [101] with constructs such as expressions and conditionals. Our system adopts a distinct subtyping discipline and typing judgements, yielding a substantially different yet more expressive type system. The definition of association is refined to align with the subtyping order, while the session restriction rule ($\lfloor$T-G-$\nu\rfloor$ in Fig. 8,§ 4) is reformulated to use projection directly rather than association. In this setting, we formalise subject reduction and session fidelity through projection – results not developed in [101] – in addition to the association-based theorems. Moreover, the definition of process liveness (Def. 4.2 in § 4.4) is strengthened by incorporating an additional side condition. Finally, we include a comprehensive discussion of related work (§ 5) and provide complete proofs.

Comparison with Scalas and Yoshida [84]. Since [84] serves as a key benchmark for this work, we summarise below the differences between the two frameworks, providing a clear and quick reference.

- Top-down vs. bottom-up methodology: the main difference lies in the underlying paradigm. Scalas and Yoshida [84] develop a bottom-up MPST theory that is independent of global types and instead founded on a parametric safety invariant. In contrast, this work follows a top-down MPST approach, in which correctness is ensured by construction through global types. This leads to distinct technical structures and guarantees, as follows.

– Global types and projection: the proposed framework provides explicit global types together with their syntax and semantics, and supports end-point projection with full merging (§ 3.1 and 3.2); such mechanisms are absent from the theory in [84].
– Core invariant: type soundness in this work (established via the subject reduction theorem, Thm. 4.1) is based on a global-to-local association invariant induced by projection and subtyping, whereas [84] relies on preserving a parametric typing-context safety invariant that is independent of global types ([84, Theorem 4.6]).
– Behavioural property guarantees: in this framework, once a process is typed under a typing context associated with a global type, key behavioural properties, such as session fidelity, communication safety, deadlock-freedom, and liveness, follow by construction (§4.3 and 4.4). In contrast, the guarantees in [84], established in [84, §5.1, 5.2 and 5.5], depend on the particular instantiations of the safety invariant ([84, §5.3 and 5.4]). To support this approach, [84] provides a model-checking–based mechanism ([84, §6]) to verify specific invariant instantiations at the type level, from which the corresponding process-level guarantees are derived.

- Technical correction: the definition of process liveness in [84, Def. 5.1] contains a subtle issue: it allows the liveness condition to hold without requiring executable communication. We address this by refining the definition (Def. 4.2) to include an explicit side condition and presenting a counterexample in § 4.4.
- Minor differences (without impact on the theoretical results).

– Multiparty session π-calculus: the calculus presented in § 2 is an extension of that in [84, §2.1], incorporating values, expressions, and conditional statements.
– Basic types: to support values and expressions, basic types are introduced in § 3.1, which are absent in [84].
– Subtyping: the subtyping discipline formalised in Def. 3.2 follows the *process-oriented* approach [12, 28, 69, 21, 20], in which branching is contravariant and selection covariant. This contrasts with the *channel-oriented* subtyping order of [84, Def. 2.5], where the variance is reversed. A detailed comparison of these two approaches is provided in [39]. Our association method is parametric in the chosen subtyping discipline and applies to *both*.
– Typing contexts, judgements, and rules: the typing contexts (Def. 3.5) are extended to account for expression variables and basic types; consequently, separate judgements (§4.1) are applied to the typing of expressions and processes, with process typing formulated differently from that in [84]. The typing rules (Fig. 8) are reformulated accordingly, resulting in a more expressive typing discipline.
– Typing context liveness: the notion of typing context liveness adopted here (Def. 3.11) is stronger than the liveness condition in [84, Fig. 5] and is closer to liveness$^+$.
– Synchronous vs. asynchronous MPST: Scalas and Yoshida [84] extend their theory to asynchronous MPST and establish a corresponding subject reduction result ([84, §7]), whereas this work focuses on the synchronous setting.

2. Multiparty Session π-Calculus

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

Syntax of Processes in Multiparty Session π-Calculus. A *session* is a sequence of interactions, typically including send and receive operations, performed by a set of *roles* (*participants*) in a communication protocol. The multiparty session π-calculus models the behaviour of processes that interact using multiparty channels.

We use the following basic notations: *basic values*, denoted by $v, v', v_i, \ldots$, *expressions*, denoted by $e, e', e_i, \ldots$, *expression variables*, denoted by $x, x', x_i, \ldots$, *channels*, denoted by $c, c', c_i, \ldots$, *channel variables*, denoted by $y, y', y_i, \ldots$, *roles*, denoted by $\mathtt{p}, \mathtt{p}', \mathtt{q}, \mathtt{q}', \mathtt{p}_i, \ldots$, *sessions*, denoted by $s, s', s_i, \ldots$, *message labels*, denoted by $\mathtt{m}, \mathtt{m}', \mathtt{m}_i, \ldots$, *processes*, denoted by $P, P', Q, Q', P_i, \ldots$, and *process variables*, denoted by $X, X', Y, Y', X_i, \ldots$.

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

$$\frac{e_1 \downarrow v}{e_1 \otimes e_2 \downarrow v} \quad \frac{e_2 \downarrow v}{e_1 \otimes e_2 \downarrow v} \quad \texttt{i}_1 < \texttt{i}_2 \downarrow \begin{cases} \texttt{true} & \text{if } \texttt{i}_1 < \texttt{i}_2 \\ \texttt{false} & \text{otherwise} \end{cases}$$

Figure 2: Expression evaluation.

Definition 2.1 (Syntax of Multiparty Session π-Calculus). The *multiparty session π-calculus* syntax is defined as follows:

$$\begin{array}{rcll}
v & ::= & \texttt{n} \mid \texttt{i} \mid \texttt{true} \mid \texttt{false} \mid \texttt{""} \mid \texttt{()} \mid \cdots & \text{(natural number, integer, boolean, string, or unit, \ldots)} \\
e & ::= & x \mid v \mid \texttt{succ}(e) \mid \texttt{neg}(e) \mid & \\
 & & \neg e \mid e \otimes e' \mid e < e' \mid \cdots & \text{(expression variable, basic value, or expression term)} \\
c & ::= & y \mid s[\texttt{p}] & \text{(channel variable or channel for session } s \text{ with role } \texttt{p}) \\
z & ::= & x \mid y & \text{(expression variable or channel variable)} \\
d & ::= & e \mid c & \text{(expression, channel variable, or channel with role)} \\
w & ::= & e \mid s[\texttt{p}] & \text{(expression or channel with role)} \\
u & ::= & v \mid s[\texttt{p}] & \text{(value or channel with role)} \\
P, Q & ::= & \mathbf{0} \mid (\nu s)\, P & \text{(inaction, restriction)} \\
 & & c[\texttt{q}] \oplus \texttt{m}\langle d \rangle . P & \text{(selection towards role } \texttt{q}) \\
 & & c[\texttt{q}] \& \{\texttt{m}_i(z_i).P_i\}_{i \in I} & \text{(branching from role } \texttt{q} \text{ with an index set } I \neq \emptyset) \\
 & & \textbf{def } X(x_1, \ldots, x_n, y_1, \ldots, y_m) = P \textbf{ in } Q & \text{(process definition)} \\
 & & X\langle e_1, \ldots, e_n, c_1, \ldots, c_m \rangle & \text{(process call)} \\
 & & \mathsf{if}\; e \;\mathsf{then}\; P \;\mathsf{else}\; Q & \text{(conditional)} \\
 & & P \mid Q \mid \texttt{err} & \text{(parallel composition, error)}
\end{array}$$

Restriction, branching, and process definitions and declarations act as binders, as expected. $\text{fc}(P)$ denotes the set of *free channels with roles* in P, $\text{fev}(P)$ denotes the set of *free expression variables* in P, and $\text{fcv}(P)$ denotes the set of *free channel variables* in P. We adopt a form of Barendregt convention: bound sessions and process variables are assumed pairwise distinct, and different from free ones. We write $\Pi_{i \in I} P_i$ for the parallel composition of processes P_i.

The syntax of our session π-calculus (Def. 2.1) is mostly standard [84], with extensions for expressions and "if... then... else" statements. A basic value v can be a natural number $\texttt{n}$, an integer $\texttt{i}$, a boolean $\texttt{true}$ or $\texttt{false}$, a string $\texttt{""}$, a unit $\texttt{()}$ (often omitted for brevity), or any other specific tailored value. An expression e can be an expression variable x, a basic value v, or a term built from expressions by applying operators, e.g. $\texttt{succ}, \texttt{neg}, \neg, \otimes, <$. A channel c can be either a channel variable y or a *channel with role* (a.k.a. *session endpoint*) $s[\texttt{p}]$, a multiparty communication endpoint whose user plays role $\texttt{p}$ in the session s.

A process acts as a communication agent within a session, representing the behaviour and actions of a role in the session. *Inaction* $\mathbf{0}$ represents a terminated process (often omitted for brevity). *Session restriction* $(\nu s)\, P$ declares a new session s, with its scope restricted to the process P. *Selection* (a.k.a. *internal choice*) $c[\texttt{q}] \oplus \texttt{m}\langle d \rangle . P$ sends a message $\texttt{m}$ with payload d to role $\texttt{q}$ via endpoint c, where d may be either an expression or a channel. *Branching* (a.k.a. *external choice*) $c[\texttt{q}] \& \{\texttt{m}_i(z_i).P_i\}_{i \in I}$ expects to receive a message $\texttt{m}_i$ (for some $i \in I$) from role $\texttt{q}$ via endpoint c, and then continues as P_i.

The *conditional* process $\mathsf{if}\; e \;\mathsf{then}\; P \;\mathsf{else}\; Q$ represents the internal choice between processes P and Q, with the selection of the branch determined by the *evaluation* of expression e (as explained later in the semantics). *Process definition* $\textbf{def } X(x_1, \ldots, x_n, y_1, \ldots, y_m) = P \textbf{ in } Q$ and *process call* $X\langle e_1, \ldots, e_n, c_1, \ldots, c_m \rangle$ capture recursion: the call invokes X by expanding it into Q and replacing its formal parameters with the actual arguments. *Parallel composition* $P \mid Q$ denotes two processes capable of concurrent execution and potential communication. Finally, $\texttt{err}$ represents the *error* process.

Operational Semantics of Multiparty Session π-Calculus. The value of an expression is computed as shown in Fig. 2, where $e \downarrow v$ denotes that an expression e evaluates to a value v. Note that Fig. 2 can be extended

[R-⊕&-V]	$s[\mathtt{p}][\mathtt{q}]\&\{\mathtt{m}_i(z_i).P_i\}_{i\in I} \mid s[\mathtt{q}][\mathtt{p}]\oplus\mathtt{m}_k\langle e\rangle.Q \;\to\; P_k\{v/z_k\} \mid Q$	$k\in I$ and $e\downarrow v$
[R-⊕&-D]	$s[\mathtt{p}][\mathtt{q}]\&\{\mathtt{m}_i(z_i).P_i\}_{i\in I} \mid s[\mathtt{q}][\mathtt{p}]\oplus\mathtt{m}_k\langle s'[\mathtt{r}]\rangle.Q \;\to\; P_k\{s'[\mathtt{r}]/z_k\} \mid Q$	$k\in I$
[R-X]	$\mathbf{def}\ X(x_1,\ldots,x_n,y_1,\ldots,y_m)=P\ \mathbf{in}\ (X\langle e_1,\ldots,e_n,s_1[\mathtt{p}_1],\ldots,s_m[\mathtt{p}_m]\rangle \mid Q)$ $\to\ \mathbf{def}\ X(x_1,\ldots,x_n,y_1,\ldots,y_m)=P\ \mathbf{in}$ $(P\{v_1/x_1\}\cdots\{v_n/x_n\}\{s_1[\mathtt{p}_1]/y_1\}\cdots\{s_m[\mathtt{p}_m]/y_m\} \mid Q)$	$\forall i\in\{1,\ldots,n\}: e_i\downarrow v_i$
[R-Cond-T]	$\mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \;\to\; P$	$e\downarrow\mathtt{true}$
[R-Cond-F]	$\mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \;\to\; Q$	$e\downarrow\mathtt{false}$
[R-Ctx]	$P\to P'$ implies $\mathbb{C}[P]\to\mathbb{C}[P']$	
[R-Err]	$s[\mathtt{p}][\mathtt{q}]\&\{\mathtt{m}_i(z_i).P_i\}_{i\in I} \mid s[\mathtt{q}][\mathtt{p}]\oplus\mathtt{m}\langle w\rangle.Q \;\to\; \mathtt{err}$	$\forall i\in I: \mathtt{m}_i\neq\mathtt{m}$
[R-≡]	$P'\equiv P\to Q\equiv Q'$ implies $P'\to Q'$	

Figure 3: Operational semantics of multiparty session π-calculus.

$$P\mid Q\equiv Q\mid P \quad (P\mid Q)\mid R\equiv P\mid(Q\mid R) \quad P\mid\mathbf{0}\equiv P \quad (\nu s)\,\mathbf{0}\equiv\mathbf{0}$$
$$(\nu s)\,(\nu s')\,P\equiv(\nu s')\,(\nu s)\,P \qquad (\nu s)\,(P\mid Q)\equiv P\mid(\nu s)\,Q \ \text{ if } s\notin\mathrm{fc}(P)$$
$$\mathbf{def}\ D\ \mathbf{in}\ \mathbf{0}\equiv\mathbf{0} \qquad \mathbf{def}\ D\ \mathbf{in}\ (\nu s)\,P\equiv(\nu s)\,(\mathbf{def}\ D\ \mathbf{in}\ P) \ \text{ if } s\notin\mathrm{fc}(D)$$
$$\mathbf{def}\ D\ \mathbf{in}\ (P\mid Q)\equiv(\mathbf{def}\ D\ \mathbf{in}\ P)\mid Q \ \text{ if } \mathrm{dpv}(D)\cap\mathrm{fpv}(Q)=\emptyset$$
$$\mathbf{def}\ D\ \mathbf{in}\ (\mathbf{def}\ D'\ \mathbf{in}\ P)\equiv\mathbf{def}\ D'\ \mathbf{in}\ (\mathbf{def}\ D\ \mathbf{in}\ P)$$
$$\text{if } (\mathrm{dpv}(D)\cup\mathrm{fpv}(D))\cap\mathrm{dpv}(D')=(\mathrm{dpv}(D')\cup\mathrm{fpv}(D'))\cap\mathrm{dpv}(D)=\emptyset$$

Figure 4: Standard structural congruence rules, where $\mathrm{fpv}(D)$ is the set of *free process variables* in D, and $\mathrm{dpv}(D)$ is the set of *declared process variables* in D.

to include additional expressions. The successor operation `succ` is defined for natural numbers, negation `neg` and comparison $<$ for integers, and logical negation $\neg$ for boolean values. The nondeterministic choice $e_1\otimes e_2$ evaluates to either the value of e_1 or the value of e_2.

We provide the operational semantics of our multiparty session π-calculus in Def. 2.2, using a standard *structural congruence* $\equiv$ defined in Fig. 4.

Definition 2.2 (Semantics of Multiparty Session π-Calculus). A *reduction context* $\mathbb{C}$ is defined as:
$$\mathbb{C} ::= \mathbb{C}\mid P \;\big|\; (\nu s)\,\mathbb{C} \;\big|\; \mathbf{def}\ D\ \mathbf{in}\ \mathbb{C} \;\big|\; [\,]$$
The *reduction* relation $\to$ on processes is inductively defined by the rules in Fig. 3. A process P is said to *have an error* if $P=\mathbb{C}[\mathtt{err}]$ for some reduction context $\mathbb{C}$. A relation $\leadsto$ on reduction contexts is defined by $\mathbb{C}\leadsto\mathbb{C}'$ iff $\forall P.\,\mathbb{C}[P]\to\mathbb{C}'[P]$.

Our operational semantics for the multiparty session π-calculus retains the standard rules from [84], while incorporating additional rules for conditionals and value exchange. A reduction context $\mathbb{C}$ denotes a process with a single hole $[\,]$, occurring in place of a subterm P. In addition, a relation $\leadsto$ on reduction contexts is introduced to capture context-level reductions that hold uniformly for all process instantiations; it will be used in the definition of process liveness in § 4.4. The reflexive-transitive closures of $\to$ and $\leadsto$ are denoted by $\to^*$ and $\leadsto^*$, respectively.

Rules [R-⊕&-V] and [R-⊕&-D] describe communication on session s involving receiver $\mathtt{p}$ and sender $\mathtt{q}$, where a value or an endpoint is exchanged, respectively, provided that the transmitted message $\mathtt{m}_k$ is handled by the receiver ($k\in I$). In case of a message label mismatch, rule [R-Err] is invoked to trigger an `err`or.

Rules [R-Cond-T] and [R-Cond-F] pertain to conditionals, which are self-explanatory. Rule [R-X] initiates the expansion of process definitions when called. Additionally, rules [R-Ctx] and [R-≡] facilitate the reduction of processes under reduction contexts and modulo structural congruence ($\equiv$), respectively.

Example 2.1 (Syntax and Semantics of Session π-Calculus, adapted from [3]). Processes P and Q below communicate over session s: P uses endpoint $s[\mathtt{p}]$ to send endpoint $s[\mathtt{r}]$ to role $\mathtt{q}$, while Q uses endpoint $s[\mathtt{q}]$ to receive it, and then sends a message to role $\mathtt{p}$ via $s[\mathtt{r}]$.

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

By Def. 2.2, successful reductions yield:

$$\begin{array}{rl} & (\nu s)\,(P\,|\,Q) \\ = & (\nu s)\,(s[\mathtt{p}][\mathtt{q}]\oplus\mathtt{m}'\langle s[\mathtt{r}]\rangle.s[\mathtt{p}][\mathtt{r}]\&\mathtt{m}(z).\mathbf{0}\,|\,s[\mathtt{q}][\mathtt{p}]\&\mathtt{m}'(z).z[\mathtt{p}]\oplus\mathtt{m}\langle 42\rangle.\mathbf{0}) \\ \to & (\nu s)\,(s[\mathtt{p}][\mathtt{r}]\&\mathtt{m}(z).\mathbf{0}\,|\,s[\mathtt{r}][\mathtt{p}]\oplus\mathtt{m}\langle 42\rangle.\mathbf{0}) \;\to\; (\nu s)\,(\mathbf{0}\,|\,\mathbf{0}) \equiv \mathbf{0} \end{array}$$

Example 2.2 (OAuth Process, extended from [84]). The following process interacts on session s using channels with role $s[\mathtt{s}]$, $s[\mathtt{c}]$, $s[\mathtt{a}]$, corresponding respectively to roles $\mathtt{s}$, $\mathtt{c}$, and $\mathtt{a}$. For brevity, irrelevant message payloads are omitted.

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

Here, $(\nu s)\,(P_{\mathtt{s}}\,|\,P_{\mathtt{c}}\,|\,P_{\mathtt{a}})$ is the parallel composition of processes $P_{\mathtt{s}}$, $P_{\mathtt{c}}$, $P_{\mathtt{a}}$ in the scope of session s. In $P_{\mathtt{s}}$, "$s[\mathtt{s}][\mathtt{c}]\oplus\mathtt{cancel}$" means using $s[\mathtt{s}]$ to send $\mathtt{cancel}$ to $\mathtt{c}$. Process $P_{\mathtt{c}}$ uses $s[\mathtt{c}]$ to receive $\mathtt{login}$, $\mathtt{cancel}$, or $\mathtt{fatal}$ from $\mathtt{s}$; then, in the first case, it uses $s[\mathtt{c}]$ to send $\mathtt{passwd}$ to $\mathtt{a}$; in the second case, it uses $s[\mathtt{c}]$ to send $\mathtt{quit}$ to $\mathtt{a}$; in the third case, it uses $s[\mathtt{c}]$ to send $\mathtt{fatal}$ to $\mathtt{a}$. By Def. 2.2, we have the reductions:

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

3. Multiparty Session Types

This section introduces multiparty session types. In § 3.1, we provide an extensive exploration of global and local types, including their syntax, projection, and subtyping. In §3.2, we establish a Labelled Transition System (LTS) semantics for global types, and in § 3.3, for typing contexts. The operational relationship between these semantics is explained in §3.4 via association. In §3.5, we show the necessity of association: an example illustrates that projection alone does not preserve this operational correspondence. Finally, in § 3.6, we demonstrate that a typing context associated with a global type ensures safety, deadlock-freedom, and liveness.

3.1. Global and Local Types

The Multiparty Session Type (MPST) theory utilises *global types* to provide a comprehensive overview of communication between *roles*, such as $\mathtt{p}, \mathtt{q}, \mathtt{s}, \mathtt{t}, \ldots$, belonging to a set $\mathcal{R}$. In contrast, it employs *local types*, derived via *projection* from a global type, to describe the communication of an *individual* role from a local perspective. The syntax for global and local types is presented in Fig. 5, where most constructs are standard [84].

Basic types, taken from a set $\mathcal{B}$, describe the types of values, including natural numbers, integers, booleans, strings, units, and others.

Global types, represented as $G, G', G_i, \ldots$, describe the high-level behaviour of all roles. The set of roles in a global type G is denoted by $\mathrm{roles}(G)$. Each syntactic construct of global types is explained as follows:

- $\mathtt{p}\to\mathtt{q}{:}\,\{\mathtt{m_i}(S_i).G_i\}_{i\in I}$: a *transmission* of a message from role $\mathtt{p}$ to role $\mathtt{q}$, consisting of a label $\mathtt{m}_i$, a payload of type S_i (either a basic or local type), and a continuation G_i, where i is taken from a non-empty index set I. The labels $\mathtt{m}_i$ must be pairwise distinct, and self receptions are excluded (i.e. $\mathtt{p} \neq \mathtt{q}$).

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

Figure 5: Syntax of types.

- $\mu\mathbf{t}.G$: a *recursive* global type, subject to contractive requirements [80, §21.8], meaning that each recursion variable $\mathbf{t}$ is bound within a $\mu\mathbf{t}.\ldots$ and is guarded.
- $\mathbf{end}$: a *terminated* global type, omitted when unambiguous.

The set of free variables in G, denoted by $\text{fv}(G)$, is defined as usual.

Local types (or **session types**), represented as $T, U, T', T_i, U', U_i, \ldots$, describe the behaviour of a single role. Each syntactic construct of local types is elucidated below:

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

Similar to global types, we require pairwise-distinct, non-empty labels in local types.

Projection In the top-down approach of MPST, local types are obtained by projecting a global type onto roles. *Projection* is a *partial* function that yields the local type associated with a participating role in a global type. Specifically, it takes a global type G and a role $\mathtt{p}$, returning the corresponding local type. Our definition of projection, provided in Def. 3.1 below, follows the standard approach [84].

Definition 3.1 (Global Type Projection). The *projection of a global type G onto a role $\mathtt{p}$*, written $G\upharpoonright\mathtt{p}$, is:

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

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

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

$$
\mathtt{p}\oplus\{\mathtt{m_i}(S_i).T_i\}_{i\in I} \sqcap \mathtt{p}\oplus\{\mathtt{m_i}(S_i).T'_i\}_{i\in I} = \mathtt{p}\oplus\{\mathtt{m_i}(S_i).(T_i\sqcap T'_i)\}_{i\in I}
$$

$$
\mu\mathbf{t}.T \sqcap \mu\mathbf{t}.T' = \mu\mathbf{t}.(T\sqcap T') \qquad \mathbf{t}\sqcap\mathbf{t}=\mathbf{t} \qquad \mathbf{end}\sqcap\mathbf{end}=\mathbf{end}
$$

$$
\mathtt{p}\&\{\mathtt{m_i}(S_i).T_i\}_{i\in I} \sqcap \mathtt{p}\&\{\mathtt{m_j}(S_j).T'_j\}_{j\in J} = \mathtt{p}\&\{\mathtt{m_k}(S_k).T''_k\}_{k\in I\cup J} \text{ where } T''_k = \begin{cases} T_k\sqcap T'_k & \text{if } k\in I\cap J \\ T_k & \text{if } k\in I\setminus J \\ T'_k & \text{if } k\in J\setminus I \end{cases}
$$

Note that some global types cannot be projected onto all of their participants. This occurs when certain global types describe inherently meaningless protocols, leading to undefined merging operations, as illustrated in Ex. 3.2.

If a global type G begins with a transmission from role $\mathtt{p}$ to role $\mathtt{q}$, projecting it onto role $\mathtt{p}$ (resp. $\mathtt{q}$) results in an internal (resp. external) choice, provided that the continuation of each branching of G is also projectable. When projecting G onto other participants $\mathtt{r}$ ($\mathtt{r} \neq \mathtt{p}$ and $\mathtt{r} \neq \mathtt{q}$), a merge operator, as defined in Def. 3.1 and exemplified in Ex. 3.2, is used to ensure that the projections of all continuations are "compatible". If a global type G is a termination or a type variable, projecting it onto any role results in a termination or type variable, respectively. Finally, the projection of a recursive global type $\mu \mathbf{t}.G$ preserves its recursive structure when projected onto a role in G, or if $\mu \mathbf{t}.G$ contains no free type variables; otherwise, the projection yields termination.

Example 3.1 (Types of OAuth 2.0, from [84]). Consider a protocol from OAuth 2.0 [77]: the service sends to the client *either* a request to login, *or* cancel; in the first case, c continues by sending passwd (carrying a string) to the authorisation server, who in turn sends auth to s (with a boolean, telling whether the client is authorised), and the session **end**s; in the second case, c sends quit to a, and the session **end**s. This protocol can be represented by the global type G_{auth}:

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

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

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

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

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

Example 3.2 (Merge and Projection, originated from [68]). Two external choice (branching) types from the same role with disjoint labels can be merged into a type carrying both labels, e.g.

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

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

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

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

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

Consider a global type:

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

G cannot be projected onto role C since:

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

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

Subtyping We introduce a *subtyping* relation $\leqslant$ on local types, as defined in Def. 3.2. Intuitively, subtyping captures safe substitutability: when a local type T is a subtype of T', the behaviour specified by T and T' is compatible; consequently, a process typed by T can be safely used wherever one typed by T' is expected, without introducing communication errors.

Definition 3.2 (Subtyping). The *session subtyping relation* $\leqslant$ is coinductively defined:

$$\dfrac{}{B \leqslant B}\ [\text{SUB-}B] \qquad \dfrac{\forall i \in I \quad S_i \leqslant S'_i \quad T_i \leqslant T'_i}{\mathtt{p}\&\{\mathtt{m_i}(S_i).T_i\}_{i\in I\cup J} \leqslant \mathtt{p}\&\{\mathtt{m_i}(S'_i).T'_i\}_{i\in I}}\ [\text{SUB-}\&]$$

$$\dfrac{}{\mathbf{end} \leqslant \mathbf{end}}\ [\text{SUB-}\mathbf{end}] \qquad \dfrac{\forall i \in I \quad S'_i \leqslant S_i \quad T_i \leqslant T'_i}{\mathtt{p}\oplus\{\mathtt{m_i}(S_i).T_i\}_{i\in I} \leqslant \mathtt{p}\oplus\{\mathtt{m_i}(S'_i).T'_i\}_{i\in I\cup J}}\ [\text{SUB-}\oplus]$$

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

Rule [SUB-B] states that any basic type is its own subtype. Rule [SUB-$\oplus$] allows the subtype of an internal choice to encompass a narrower range of message labels while permitting the sending of more generic payloads. Conversely, rule [SUB-&] dictates that the subtype of an external choice must support a broader set of input message labels and less generic payloads. Rule [SUB-**end**] specifies that the type **end** is its own subtype. Finally, rules [SUB-μL] and [SUB-μR] define the relationship between recursive types up to their unfolding.

Example 3.3 (Subtyping). Recall the local types $T_{\mathtt{s}}$, $T_{\mathtt{c}}$, and $T_{\mathtt{a}}$ from Ex. 3.1:

$$T_{\mathtt{s}} = \mathtt{c}\oplus\begin{Bmatrix}\mathtt{login}.\mathtt{a}\&\mathtt{auth}(\text{bool})\\ \mathtt{cancel}\end{Bmatrix} \quad T_{\mathtt{c}} = \mathtt{s}\&\begin{Bmatrix}\mathtt{login}.\mathtt{a}\oplus\mathtt{passwd}(\text{str})\\ \mathtt{cancel}.\mathtt{a}\oplus\mathtt{quit}\end{Bmatrix}$$

$$T_{\mathtt{a}} = \mathtt{c}\&\begin{Bmatrix}\mathtt{passwd}(\text{str}).\mathtt{s}\oplus\mathtt{auth}(\text{bool})\\ \mathtt{quit}\end{Bmatrix}$$

Additionally, consider the following local types:

$$T'_{\mathtt{s}} = s[\mathtt{s}] : \mathtt{c}\oplus\mathtt{cancel} \qquad T'_{\mathtt{c}} = s[\mathtt{c}] : \mathtt{s}\&\begin{Bmatrix}\mathtt{login}.\mathtt{a}\oplus\mathtt{passwd}(\text{str})\\ \mathtt{cancel}.\mathtt{a}\oplus\mathtt{quit}\\ \mathtt{fail}.\mathtt{a}\oplus\mathtt{fatal}\end{Bmatrix}$$

$$T'_{\mathtt{a}} = s[\mathtt{a}] : \mathtt{c}\&\begin{Bmatrix}\mathtt{passwd}(\text{str}).\mathtt{s}\oplus\mathtt{auth}(\text{bool})\\ \mathtt{quit}\\ \mathtt{fatal}\end{Bmatrix}$$

It holds that $T'_{\mathtt{s}} \leqslant T_{\mathtt{s}}$, because a subtype is allowed to offer fewer options in an internal choice, while $T'_{\mathtt{c}} \leqslant T_{\mathtt{c}}$ and $T'_{\mathtt{a}} \leqslant T_{\mathtt{a}}$, as a subtype in an external choice may accept more options.

3.2. Semantics of Global Types

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

Definition 3.3 (Transition Labels). Let α be a transition label of the form:

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

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

$$\begin{array}{rcl}\text{subject}(s[\mathtt{p}]{:}\mathtt{q}\&\mathtt{m}(S)) & = & \text{subject}(s[\mathtt{p}]{:}\mathtt{q}\oplus\mathtt{m}(S)) = \{\mathtt{p}\}\\ \text{subject}(s[\mathtt{p}][\mathtt{q}]\mathtt{m}) & = & \{\mathtt{p},\mathtt{q}\}\end{array}$$

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

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

Figure 6: Global type transition rules.

The label $s[\mathtt{p}][\mathtt{q}]\mathtt{m}$ denotes a synchronising communication between $\mathtt{p}$ and $\mathtt{q}$ via a message label $\mathtt{m}$; the subject of this label includes *both* roles. The labels $s[\mathtt{p}]{:}\mathtt{q}{\oplus}\mathtt{m}(S)$ and $s[\mathtt{p}]{:}\mathtt{q}\&\mathtt{m}(S)$ describe sending and receiving actions, respectively, and are specifically used in the typing context semantics in § 3.3.

We proceed to give a Labelled Transition System (LTS) semantics to a global type G in Def. 3.4.

Definition 3.4 (Global Type Semantics). The global type transition $\xrightarrow{\alpha}$ is inductively defined by the rules in Fig. 6. We use $G \to G'$ if there exists α such that $G \xrightarrow{\alpha} G'$; we write $G \to$ if there exists G' such that $G \to G'$, and $G \not\to$ for its negation (i.e. there is no G' such that $G \to G'$). Finally, $\to^*$ denotes the transitive and reflexive closure of $\to$.

Fig. 6 depicts the standard global type transition rules [100]. Rule [GR-$\oplus\&$] models communication between two roles, while rule [GR-μ] addresses recursion. Finally, rule [GR-Ctx] permits the transition of a global type that is causally independent of its prefix, provided that all continuations can perform the transition with that label. This causal independence is indicated by the subjects of the label being disjoint from the prefix of the global type. For example, consider the global type $G = \mathtt{p}{\to}\mathtt{q}{:}\mathtt{m_1}.\mathtt{r}{\to}\mathtt{u}{:}\mathtt{m_2}.\mathbf{end}$. Given that the subjects of the label $s[\mathtt{r}][\mathtt{u}]\mathtt{m_2}$ are disjoint from $\mathtt{p}$ and $\mathtt{q}$, i.e. $\mathrm{subject}(s[\mathtt{r}][\mathtt{u}]\mathtt{m_2}) \cap \{\mathtt{p},\mathtt{q}\} = \emptyset$, and $\mathtt{r}{\to}\mathtt{u}{:}\mathtt{m_2}.\mathbf{end} \xrightarrow{s[\mathtt{r}][\mathtt{u}]\mathtt{m_2}} \mathbf{end}$, rule [GR-Ctx] allows G to perform the transition $G \xrightarrow{s[\mathtt{r}][\mathtt{u}]\mathtt{m_2}} \mathtt{p}{\to}\mathtt{q}{:}\mathtt{m_1}.\mathbf{end}$.

3.3. Semantics of Typing Context

Following the introduction of global type semantics, we present an LTS semantics for *typing contexts*, which are collections of local types. The formal definition of a typing context is provided in Def. 3.5, with its transition rules in Def. 3.6.

Definition 3.5 (Typing Contexts). Γ denotes a partial mapping from expression variables to basic types, and from process variables to sequences of basic types and session types, while Δ denotes a partial mapping from channels to session types. Their syntax is defined as:

$$\Gamma \ ::= \ \emptyset \ \big|\ \Gamma, x : B \ \big|\ \Gamma, X : B_1, \ldots, B_n, T_1, \ldots, T_m \qquad \Delta \ ::= \ \emptyset \ \big|\ \Delta, c : T$$

The *context composition* Δ_1, Δ_2 is defined iff $\mathrm{dom}(\Delta_1) \cap \mathrm{dom}(\Delta_2) = \emptyset$. We write $\bigcup_{i\in I} \Delta_i$ for the composition of contexts Δ_i. We write $s \in \Delta$ iff $\exists \mathtt{p} : s[\mathtt{p}] \in \mathrm{dom}(\Delta)$ (i.e. session s occurs in Δ). Conversely, we write $s \notin \Delta$ iff $\forall \mathtt{p} : s[\mathtt{p}] \notin \mathrm{dom}(\Delta)$ (i.e. session s does not occur in Δ). We write $\mathrm{dom}(\Delta) = \{s\}$ iff $\forall c \in \mathrm{dom}(\Delta)$ there exists $\mathtt{p}$ such that $c = s[\mathtt{p}]$ (i.e. Δ only contains session s). We write $\Delta \leqslant \Delta'$ iff $\mathrm{dom}(\Delta) = \mathrm{dom}(\Delta')$ and $\forall c \in \mathrm{dom}(\Delta){:}\ \Delta(c) \leqslant \Delta'(c)$. We write Δ_s iff $\mathrm{dom}(\Delta_s) = \{s\}$, $\mathrm{dom}(\Delta_s) \subseteq \mathrm{dom}(\Delta)$, and $\forall s[\mathtt{p}] \in \mathrm{dom}(\Delta)$: $\Delta(s[\mathtt{p}]) = \Delta_s(s[\mathtt{p}])$ (i.e. restriction of Δ to session s).

Definition 3.6 (Typing Context Semantics). The *typing context transition* $\xrightarrow{\alpha}$ is inductively defined by the rules in Fig. 7. We write $\Delta \xrightarrow{\alpha}$ if there exists Δ' such that $\Delta \xrightarrow{\alpha} \Delta'$. We define two *reductions* $\Delta \to_s \Delta'$ and $\Delta \to \Delta'$, as follows:

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

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

$$\frac{\Delta_1 \xrightarrow{s[\mathtt{p}]:\mathtt{q}\oplus\mathtt{m}(S)} \Delta_1' \quad \Delta_2 \xrightarrow{s[\mathtt{q}]:\mathtt{p}\&\mathtt{m}(S')} \Delta_2' \quad S' \leqslant S}{\Delta_1, \Delta_2 \xrightarrow{s[\mathtt{p}][\mathtt{q}]\mathtt{m}} \Delta_1', \Delta_2'}\ [\Delta\text{-}\oplus\&] \qquad \frac{s[\mathtt{p}] : T\{\mu\mathbf{t}.T/\mathbf{t}\} \xrightarrow{\alpha} \Delta'}{s[\mathtt{p}] : \mu\mathbf{t}.T \xrightarrow{\alpha} \Delta'}\ [\Delta\text{-}\mu] \qquad \frac{\Delta \xrightarrow{\alpha} \Delta'}{\Delta, c:T \xrightarrow{\alpha} \Delta', c:T}\ [\Delta\text{-},]$$

Figure 7: Typing context transition rules.

- $\Delta \to \Delta'$ holds iff $\Delta \to_s \Delta'$ for some s: this means that Δ can progress via message transmission on any session. We write $\Delta\!\to$ iff $\Delta \to \Delta'$ for some Δ', and $\Delta\!\not\to$ for its negation, and we denote $\to^*$ as the reflexive and transitive closure of $\to$.

Fig. 7 presents rules for typing context transitions, which are similar to those in [84]. Rule [Δ-⊕] (resp. [Δ-&]) states that a typing context entry can execute an output (resp. input) transition. Rule [Δ-⊕&] ensures synchronised matching of input and output transitions, with payload compatibility through subtyping; consequently, the context progresses via a message transmission label $s[\mathtt{p}][\mathtt{q}]\mathtt{m}$. Rule [Δ-μ] handles recursion, while rule [Δ-,] lifts transitions to larger contexts.

3.4. Relating Semantics between Global Types and Typing Contexts

With the LTS semantics for global types (Def. 3.4) and typing contexts (Def. 3.6) defined, we establish a relationship between these semantics using the projection operator $\upharpoonright$ (Def. 3.1) and the subtyping relation $\leqslant$ (Def. 3.2).

Definition 3.7 (Association of Global Types and Typing Contexts). A typing context Δ is associated with a global type G for a multiparty session s, written $\Delta \sqsubseteq_s G$, iff Δ can be split into two disjoint (possibly empty) sub-contexts $\Delta = \Delta_{\upharpoonright\leqslant}, \Delta_{\mathbf{end}}$ where:

1. $\Delta_{\upharpoonright\leqslant}$ contains subtypes of projections of G: $\mathrm{dom}(\Delta_{\upharpoonright\leqslant}) = \{s[\mathtt{p}] \mid \mathtt{p} \in \mathrm{roles}(G)\}$, and $\forall \mathtt{p} \in \mathrm{roles}(G)$: $\Delta_{\upharpoonright\leqslant}(s[\mathtt{p}]) \leqslant G\!\upharpoonright\!\mathtt{p}$;
2. $\Delta_{\mathbf{end}}$ contains only terminated endpoints: either $\Delta_{\mathbf{end}} = \emptyset$, or $\mathrm{dom}(\Delta_{\mathbf{end}}) = s$ and $\forall s[\mathtt{q}] \in \mathrm{dom}(\Delta_{\mathbf{end}})$: $\Delta_{\mathbf{end}}(s[\mathtt{q}]) = \mathbf{end}$.

The association $\cdot \sqsubseteq_{\cdot} \cdot$ is a binary relation over typing contexts Δ and global types G, parameterised by multiparty sessions s. The association relation requires that, for each role $\mathtt{p}$ in the global type, its corresponding entry in the typing context ($\Delta(s[\mathtt{p}])$) must be a subtype (Def. 3.2) of the projection of the global type onto that role ($G\!\upharpoonright\!\mathtt{p}$).

Remark 3.1. It is evident that a typing context $\Delta = \{s[\mathtt{p}] : G\!\upharpoonright\!\mathtt{p}\}_{\mathtt{p}\in\mathrm{roles}(G)}$, which contains the projections of all roles in G, is associated with the global type G for s. ◀

Example 3.4 (Association). Recall the global type G_{auth} and its projected local types $T_{\mathtt{s}}, T_{\mathtt{c}}, T_{\mathtt{a}}$ from Ex. 3.1, as well as the local types $T'_{\mathtt{s}}, T'_{\mathtt{c}}, T'_{\mathtt{a}}$ from Ex. 3.3. Consider the typing context:

$$\Delta_{\mathrm{auth}} = \Delta_{\mathrm{auth}_{\mathtt{s}}}, \Delta_{\mathrm{auth}_{\mathtt{c}}}, \Delta_{\mathrm{auth}_{\mathtt{a}}}$$

where $\Delta_{\mathrm{auth}_{\mathtt{s}}} = s[\mathtt{s}] : T'_{\mathtt{s}}$, $\Delta_{\mathrm{auth}_{\mathtt{c}}} = s[\mathtt{s}] : T'_{\mathtt{c}}$, and $\Delta_{\mathrm{auth}_{\mathtt{a}}} = s[\mathtt{s}] : T'_{\mathtt{a}}$.

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

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

- $\mathrm{roles}(G_{\mathrm{auth}}) = \{\mathtt{s}, \mathtt{c}, \mathtt{a}\}$ and $\mathrm{dom}(\Delta_{\mathrm{auth}}) = \{s[\mathtt{s}], s[\mathtt{c}], s[\mathtt{a}]\}$
- $\Delta_{\mathrm{auth}}(s[\mathtt{s}]) = T'_{\mathtt{s}} \leqslant G_{\mathrm{auth}}\!\upharpoonright\!\mathtt{s} = T_{\mathtt{s}}$

- $\Delta_{\mathrm{auth}}(s[\mathtt{c}]) = T'_{\mathtt{c}} \leqslant G_{\mathrm{auth}} \upharpoonright \mathtt{c} = T_{\mathtt{c}}$
- $\Delta_{\mathrm{auth}}(s[\mathtt{a}]) = T'_{\mathtt{a}} \leqslant G_{\mathrm{auth}} \upharpoonright \mathtt{a} = T_{\mathtt{a}}$

We demonstrate the *operational correspondence* between a global type and any *associated* typing context through two main theorems: Thm. 3.1 shows that the transition behaviour of a global type corresponds to that of its associated typing context, while Thm. 3.2 illustrates that each possible transition of a typing context is reflected by an action in the transitions of the associated global type.

Theorem 3.1 (Soundness of Association)**.** *Given associated global type G and typing context Δ for session s: $\Delta \sqsubseteq_s G$. If $G \xrightarrow{\alpha} G'$ where $\alpha = s[\mathtt{p}][\mathtt{q}]\mathtt{m}$, then there exist $\mathtt{m}'$, α', Δ', and G'', such that $\alpha' = s[\mathtt{p}][\mathtt{q}]\mathtt{m}'$, $G \xrightarrow{\alpha'} G''$, $\Delta' \sqsubseteq_s G''$, and $\Delta \xrightarrow{\alpha'} \Delta'$.*

Proof. By induction on global type transitions (Def. 3.4). □

Theorem 3.2 (Completeness of Association)**.** *Given associated global type G and typing context Δ for session s: $\Delta \sqsubseteq_s G$. If $\Delta \xrightarrow{\alpha} \Delta'$ where $\alpha = s[\mathtt{p}][\mathtt{q}]\mathtt{m}$, then there exists G' such that $\Delta' \sqsubseteq_s G'$ and $G \xrightarrow{\alpha} G'$.*

Proof. By induction on typing context transitions (Def. 3.6). □

Example 3.5 (Soundness and Completeness of Association). Consider the associated global type G_{auth} and typing context Δ_{auth} in Ex. 3.4. We have the global type transition $G_{\mathrm{auth}} \xrightarrow{s[\mathtt{s}][\mathtt{c}]\mathtt{login}}$, and, by the soundness of association (Thm. 3.1), there exist $\alpha = s[\mathtt{s}][\mathtt{c}]\mathtt{cancel}$, G'_{auth}, and Δ'_{auth} such that

$$G_{\mathrm{auth}} \xrightarrow{s[\mathtt{s}][\mathtt{c}]\mathtt{cancel}} G'_{\mathrm{auth}} = \mathtt{c}\to\mathtt{a}\mathtt{:quit} \quad\text{and}\quad \Delta_{\mathrm{auth}} \xrightarrow{s[\mathtt{s}][\mathtt{c}]\mathtt{cancel}} \Delta'_{\mathrm{auth}} = \Delta'_{\mathrm{auth}_{\mathtt{s}}}, \Delta'_{\mathrm{auth}_{\mathtt{c}}}, \Delta'_{\mathrm{auth}_{\mathtt{a}}}$$

where:

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

By Def. 3.7, it is straightforward to check that $\Delta'_{\mathrm{auth}} \sqsubseteq_s G'_{\mathrm{auth}}$. Further, Δ'_{auth} admits the transition:

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

which, by the completeness of association (Thm. 3.2), relates to the transition: $G'_{\mathrm{auth}} \xrightarrow{s[\mathtt{c}][\mathtt{a}]\mathtt{quit}} \mathbf{end}$, with $\Delta''_{\mathrm{auth}} \sqsubseteq_s \mathbf{end}$.

Remark 3.2 (Sufficiency of Soundness Theorem). Inquisitive readers may question why we opted to formulate a soundness theorem that does not directly mirror the completeness theorem, as in common existing literature [32]. This decision stems from our use of the subtyping (particularly rule [Sub-⊕]). A local type in the typing context may offer fewer branches for selection compared to the projected local type, leading to transmission actions in the global type that remain uninhabited.

For example, consider the global type G_{auth} and its associated typing context Δ_{auth} from Ex. 3.4. While the global type G_{auth} can transition via $s[\mathtt{s}][\mathtt{c}]\mathtt{login}$, no corresponding transition is available for the associated typing context Δ_{auth}.

Importantly, the message $\mathtt{m}'$ appearing in the soundness theorem is not arbitrary: it must correspond to a branch enabled by the global type and realisable by the associated local types, although it need not coincide with the particular label $\mathtt{m}$. Consequently, soundness of association requires the existence of a *compatible* typing context transition, rather than preservation of the identical message transmission.

Despite this formulation, the soundness theorem is *sufficient* to ensure the key properties guaranteed by association, including safety, deadlock-freedom, and liveness, as demonstrated in § 3.6. ◀

3.5. Association vs Projection: Necessity of Association

As introduced in § 1, the association defined in § 3.4 is applied as the invariant in the proof of subject reduction, where typing contexts used to type processes are incrementally constructed along transitions while maintaining the association. A natural question, therefore, concerns the necessity of association: why cannot projection be used directly in its place?

The difficulty lies in the fact that projection is not preserved under transitions. More specifically, if the conditions $\Delta \sqsubseteq_s G$ and $\Delta' \sqsubseteq_s G'$ in Thms. 3.1 and 3.2 are replaced with $\Delta = \{s[\mathtt{r}] : G \upharpoonright \mathtt{r}\}_{\mathtt{r} \in \mathrm{roles}(G)}$ and $\Delta' = \{s[\mathtt{r}] : G' \upharpoonright \mathtt{r}\}_{\mathtt{r} \in \mathrm{roles}(G)}$, so that typing contexts are derived directly from projections, the operational correspondence no longer holds. The following example illustrates this issue.

Consider the global type $G_\times$:

$$\mathtt{A} \to \mathtt{B}: \left\{ \begin{array}{l} \mathtt{add}(\mathrm{int}).\mathtt{B} \to \mathtt{C}: \left\{ \begin{array}{l} \mathtt{add}(\mathrm{int}).\mathbf{end} \\ \mathtt{forward}(\mathrm{int}).\mathbf{end} \end{array} \right\} \\ \mathtt{sub}(\mathrm{int}).\mathtt{B} \to \mathtt{C}: \left\{ \begin{array}{l} \mathtt{sub}(\mathrm{int}).\mathbf{end} \\ \mathtt{forward}(\mathrm{int}).\mathbf{end} \end{array} \right\} \end{array} \right\}$$

and the typing context $\Delta_\times$:

$$s[\mathtt{A}] : \mathtt{B} \oplus \left\{ \begin{array}{l} \mathtt{add}(\mathrm{int}).\mathbf{end} \\ \mathtt{sub}(\mathrm{int}).\mathbf{end} \end{array} \right\} \qquad s[\mathtt{B}] : \mathtt{A} \& \left\{ \begin{array}{l} \mathtt{add}(\mathrm{int}).\mathtt{C} \oplus \left\{ \begin{array}{l} \mathtt{add}(\mathrm{int}).\mathbf{end} \\ \mathtt{forward}(\mathrm{int}).\mathbf{end} \end{array} \right\} \\ \mathtt{sub}(\mathbf{end}).\mathtt{C} \oplus \left\{ \begin{array}{l} \mathtt{sub}(\mathrm{int}).\mathbf{end} \\ \mathtt{forward}(\mathrm{int}).\mathbf{end} \end{array} \right\} \end{array} \right\}$$

$$s[\mathtt{C}] : \mathtt{B} \& \left\{ \begin{array}{l} \mathtt{add}(\mathrm{int}).\mathbf{end} \\ \mathtt{sub}(\mathrm{int}).\mathbf{end} \\ \mathtt{forward}(\mathrm{int}).\mathbf{end} \end{array} \right\}$$

It is straightforward to verify that $\Delta_\times$ is a projection of $G_\times$ onto session s. There are two possible transitions between $\mathtt{A}$ and $\mathtt{B}$ in $G_\times$: either $\mathtt{add}$ or $\mathtt{sub}$. Since both cases are symmetrical, we focus on the transition labeled $\mathtt{add}$, i.e. $\alpha = s[\mathtt{A}][\mathtt{B}]\mathtt{add}$. Consequently, $G_\times$ reduces to

$$G'_\times = \mathtt{B} \to \mathtt{C}: \left\{ \begin{array}{l} \mathtt{add}(\mathrm{int}).\mathbf{end} \\ \mathtt{forward}(\mathrm{int}).\mathbf{end} \end{array} \right\}$$

The typing context obtained through projection from $G'_\times$ is:

$$\Delta'_\times = s[\mathtt{A}] : \mathbf{end},\ s[\mathtt{B}] : \mathtt{C} \oplus \left\{ \begin{array}{l} \mathtt{add}(\mathrm{int}).\mathbf{end} \\ \mathtt{forward}(\mathrm{int}).\mathbf{end} \end{array} \right\},\ s[\mathtt{C}] : \mathtt{B} \& \left\{ \begin{array}{l} \mathtt{add}(\mathrm{int}).\mathbf{end} \\ \mathtt{forward}(\mathrm{int}).\mathbf{end} \end{array} \right\}$$

However, $\Delta_\times \xrightarrow{\alpha} \Delta'_\times$ does not hold. The only possible transition of $\Delta_\times$ with $\alpha = s[\mathtt{A}][\mathtt{B}]\mathtt{add}$ leads to:

$$\Delta''_\times = s[\mathtt{A}] : \mathbf{end},\ s[\mathtt{B}] : \mathtt{C} \oplus \left\{ \begin{array}{l} \mathtt{add}(\mathrm{int}).\mathbf{end} \\ \mathtt{forward}(\mathrm{int}).\mathbf{end} \end{array} \right\},\ s[\mathtt{C}] : \mathtt{B} \& \left\{ \begin{array}{l} \mathtt{add}(\mathrm{int}).\mathbf{end} \\ \mathtt{sub}(\mathrm{int}).\mathbf{end} \\ \mathtt{forward}(\mathrm{int}).\mathbf{end} \end{array} \right\}$$

Thus, direct projection cannot, in general, be substituted for association in Thm. 3.1, and a similar limitation arises in Thm. 3.2. Association therefore provides the required transition-preserving relation between global types and typing contexts, acting as the intermediate invariant for subject reduction.

3.6. Typing Context Properties Guaranteed via Association

The design of multiparty session type theory provides substantial advantages in guaranteeing desirable properties. Processes that adhere to the local types obtained from projections are inherently *correct by construction*. This subsection highlights three key properties: *communication safety*, *deadlock-freedom*, and *liveness*, which are ensured by typing contexts associated with global types.

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

Definition 3.8 (Typing Context Safety). Given a session s, we say that φ is an *s-safety property* of typing contexts iff, whenever $\varphi(\Delta)$, we have:

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

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

Our safety property is derived from a fundamental feature of generalised MPST systems [84, Def. 4.1]. According to Def. 3.8, safety is a *coinductive* property [81]. This implies that, for a given a session s, s-safe is the largest s-safety property, including the union of all s-safety properties. To demonstrate the s-safety of a typing context Δ, we need to identify a property φ such that $\Delta \in \varphi$, and then prove that φ qualifies as an s-safety property. Specifically, if such a φ exists, it can be formulated as a set containing Δ and all its reductums (via reduction $\to_s^*$). We then verify whether all elements of φ satisfy each clause of Def. 3.8.

As specified by clause $[\text{S-}\oplus\&]$, whenever two roles $\mathtt{p}$ and $\mathtt{q}$ attempt communication, the receiving role $\mathtt{q}$ must accommodate all output messages of the sending role $\mathtt{p}$ with compatible payload types, ensuring the feasibility of the communication. Clause $[\text{S-}\mu]$ unfolds any recursive entry, while clause $[\text{S-}\to_s]$ states that any typing context Δ' reachable from Δ through reduction on session s, must also belong to φ, indicating that Δ' is s-safe as well. Note that any entry $s[\mathtt{p}] : \mathbf{end}$ in Δ satisfies all clauses.

Example 3.6 (Typing Context Safety). Consider the typing context Δ_{auth} from Ex. 3.4. We show that Δ_{auth} is s-safe by inspecting its transitions. Specifically, we have

$$\Delta_{\text{auth}} \xrightarrow{s[\mathtt{s}][\mathtt{c}]\mathtt{cancel}} \cdot \xrightarrow{s[\mathtt{c}][\mathtt{a}]\mathtt{quit}} \Delta''_{\text{auth}} = s[\mathtt{s}] : \mathbf{end},\, s[\mathtt{c}] : \mathbf{end},\, s[\mathtt{a}] : \mathbf{end}$$

where each transition complies with all clauses of Def. 3.8.

The typing context $\Delta_A = s[\mathtt{p}] : \mathtt{q}\oplus\mathtt{m}_1.\mathtt{r}\oplus\mathtt{m}_3,\, s[\mathtt{q}] : \mathtt{p}\&\mathtt{m}_2,\, s[\mathtt{r}] : \mathtt{p}\&\mathtt{m}_4$ is *not* s-safe. Any property φ containing such a typing context is *not* a s-safety property, as it violates $[\text{S-}\oplus\&]$ of Def. 3.8: $\Delta_A \xrightarrow{s[\mathtt{p}]:\mathtt{q}\oplus\mathtt{m}_1}$ and $\Delta_A \xrightarrow{s[\mathtt{q}]:\mathtt{p}\&\mathtt{m}_2}$, but $\Delta_A \xrightarrow{s[\mathtt{p}][\mathtt{q}]\mathtt{m}_1}$ does not hold.

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

Deadlock-Freedom. The property of deadlock-freedom ensures that a typing context does not get "stuck" during reduction – that is, it either always has a transition available or has reached a terminal state.

Definition 3.9 (Deadlock-Free Typing Contexts). Given a session s, a typing context Δ is *s-deadlock-free*, written $\text{df}(s, \Delta)$, iff $\Delta \to_s^* \Delta' \not\to_s$ implies $\forall s[\mathtt{p}] \in \text{dom}(\Delta') : \Delta'(s[\mathtt{p}]) = \mathbf{end}$.

Notably, a typing context that reduces infinitely adheres to deadlock-freedom, as it consistently undergoes further reductions. Alternatively, when a terminal typing context is reached, all entries must successfully terminate with **end**. Consequently, a deadlock-free typing context either continues to reduce perpetually or terminates successfully.

Example 3.7 (Typing Context Deadlock-Freedom). The typing context Δ_{auth} from Ex. 3.4 is s-deadlock-free, as any terminal typing context Δ'_{auth}, reached from Δ_{auth} via $\Delta_{\text{auth}} \to_s^* \Delta'_{\text{auth}} \not\to_s$, contains only **end** entries, which can be easily verified.

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

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

Liveness. The liveness property ensures that every pending internal or external choice eventually gets triggered through a message transmission. It relies on fairness – specifically, the *strong fairness of components* [46, Fact 2] – to guarantee that every enabled message transmission is eventually executed. Definitions of fair and live paths for typing contexts are provided in Def. 3.10, and these paths are used to formalise the liveness for typing contexts in Def. 3.11.

Definition 3.10 (Fair, Live Paths). A *path* is a possibly infinite sequence of typing contexts $(\Delta_n)_{n\in N}$, where $N = \{0, 1, 2, \ldots\}$ is a (finite or infinite) set of consecutive natural numbers, and, $\forall n, n+1 \in N$, $\Delta_n \to \Delta_{n+1}$.

We say that a path $(\Delta_n)_{n\in N}$ is *fair for session* s iff, $\forall n \in N$: $\Delta_n \xrightarrow{s[\mathtt{p}][\mathtt{q}]\mathtt{m}}$ implies $\exists k, \mathtt{m}'$ such that $N \ni k \geq n$, and $\Delta_k \xrightarrow{s[\mathtt{p}][\mathtt{q}]\mathtt{m}'} \Delta_{k+1}$.

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

(L1) $\Delta_n \xrightarrow{s[\mathtt{p}]:\mathtt{q}\oplus\mathtt{m}(S)}$ implies $\exists k, \mathtt{m}'$ such that $N \ni k \geq n$ and $\Delta_k \xrightarrow{s[\mathtt{p}][\mathtt{q}]\mathtt{m}'} \Delta_{k+1}$;

(L2) $\Delta_n \xrightarrow{s[\mathtt{q}]:\mathtt{p}\&\mathtt{m}(S)}$ implies $\exists k, \mathtt{m}'$ such that $N \ni k \geq n$ and $\Delta_k \xrightarrow{s[\mathtt{p}][\mathtt{q}]\mathtt{m}'} \Delta_{k+1}$.

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

Definition 3.11 (Live Typing Contexts). Given a session s, a typing context Δ is s-*live*, written $\text{live}(s, \Delta)$, iff all paths starting with Δ that are fair for session s are also live for s.

A typing context Δ is s-live if it consistently generates a live path for s under fairness.

Example 3.8 (Fairness and Typing Context Liveness, originated from [44, 84]). Consider the typing context:

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

which is s-safe and s-deadlock-free. There is an infinite path starting with Δ_E, where $\mathtt{q}$ and $\mathtt{q}'$ continue communicating in session s, while $\mathtt{p}$ and $\mathtt{p}'$ never trigger a transition to interact, i.e.

$$\Delta_E \xrightarrow{s[\mathtt{q}][\mathtt{q}']\mathtt{m}_2} \Delta_E \xrightarrow{s[\mathtt{q}][\mathtt{q}']\mathtt{m}_2} \cdots$$

Such a path is *not* fair for s because although message transmission between $\mathtt{p}$ and $\mathtt{p}'$ is enabled, it will never occur. Alternatively, along any fair path of Δ_E, all inputs and outputs are eventually fired on s, indicating that Δ_E is s-live.

Now consider another typing context:

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

which is s-safe and s-deadlock-free. Δ'_E has fair and live paths, where in s, $\mathtt{m}_2$ is transmitted from $\mathtt{q}$ to $\mathtt{p}$, and then to $\mathtt{r}$. However, there is also a fair path, where in s, $\mathtt{m}_1$ is consistently transmitted from $\mathtt{q}$ to $\mathtt{p}$:

$$\Delta'_E \xrightarrow{s[\mathtt{q}][\mathtt{p}]\mathtt{m}_1} \Delta'_E \xrightarrow{s[\mathtt{q}][\mathtt{p}]\mathtt{m}_1} \cdots$$

In this case, $\mathtt{r}$ indefinitely awaits an input ($\mathtt{m}_2$ from $\mathtt{q}$) that will never arrive. The path is fair, as the message to $\mathtt{r}$ is not enabled – that is, the action $s[\mathtt{q}][\mathtt{r}]\mathtt{m}_2$ is never selected, so its absence does not violate fairness. Therefore, this path is fair for s but not live, meaning that Δ'_E is not s-live.

Finally, consider the typing context $\Delta''_E = s[\mathtt{p}] : \mathtt{q}\oplus\{\mathtt{m}_1, \mathtt{m}_2\}, s[\mathtt{q}] : \mathtt{p}\&\{\mathtt{m}_1, \mathtt{m}_3\}$, which is s-live. There is only one path starting from Δ''_E, i.e.

$$\Delta''_E \xrightarrow{s[\mathtt{p}][\mathtt{q}]\mathtt{m}_1} s[\mathtt{p}] : \mathbf{end}, s[\mathtt{q}] : \mathbf{end}$$

which is fair and live for s; however it is not s-safe.

Properties by Association. To conclude, Thm. 3.3 asserts that a typing context associated with a global type is constructed to guarantee the properties of safety, deadlock-freedom, and liveness.

Theorem 3.3 (Safety, Deadlock-Freedom, and Liveness by Association)**.** *Let G be a global type, Δ a typing context, and s a session. If Δ is associated with G for s: $\Delta \sqsubseteq_s G$, then Δ is s-safe, s-deadlock-free, and s-live.*

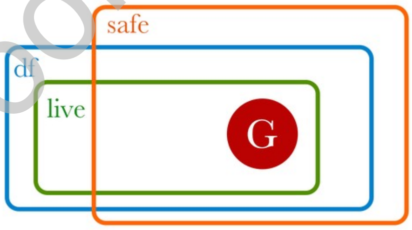
Example 3.9 (Typing Context Properties Guaranteed by Association). The typing context Δ_{auth}, described in Ex. 3.4, is associated with G_{auth} for session s, i.e. $\Delta_{\text{auth}} \sqsubseteq_s G_{\text{auth}}$. Consequently, Δ_{auth} possesses the desirable properties of being s-safe, s-deadlock-free (as demonstrated in Ex. 3.6 and Ex. 3.7, respectively), and s-live.

3.7. Relationships Between Typing Context Properties

We now examine both the relationships between typing context properties, and how they relate to association, as formalised in Thm. 3.4.

Theorem 3.4. *For any typing context Δ and session s, the following statements are valid:*

(1) $\text{df}(s, \Delta) \not\Longleftarrow\ \not\Longrightarrow \text{safe}(s, \Delta)$;
(2) $\text{live}(s, \Delta) \not\Longleftarrow\ \not\Longrightarrow \text{safe}(s, \Delta)$;
(3) $\text{live}(s, \Delta) \not\Longleftarrow\!\!\Longrightarrow \text{df}(s, \Delta)$;
(4) $\exists G : \Delta \sqsubseteq_s G \not\Longleftarrow\!\!\Longrightarrow \text{safe}(s, \Delta)$;
(5) $\exists G : \Delta \sqsubseteq_s G \not\Longleftarrow\!\!\Longrightarrow \text{df}(s, \Delta)$;
(6) $\exists G : \Delta \sqsubseteq_s G \not\Longleftarrow\!\!\Longrightarrow \text{live}(s, \Delta)$.



In the diagram, the "safe" set (resp. "df" set, "live" set) contains all typing contexts that are s-safe (resp. s-deadlock-free, s-live). The red set $\mathbb{G}$ encompasses all typing contexts associated with some global type for s. The negated implications stated in Thm. 3.4 are illustrated in Ex. 3.7, Ex. 3.8, and Ex. 3.10, respectively.

Example 3.10 (Non-Associated Typing Context, adapted from [4, 84]). The typing context $\Delta_F = s[\mathtt{p}] : T, s[\mathtt{q}] : \mathtt{p}\&\mathtt{m}(T).\mathbf{end}$, with $T = \mu\mathbf{t}.\mathtt{q}\oplus\mathtt{m}(\mathbf{t}).\mathbf{end}$ (from [4, Ex. 1.2]), is s-safe, s-deadlock-free, and s-live, but *not* associated with any global type for session s. This is due to the occurrence of the recursion variable $\mathbf{t}$ as a payload in T, which is not allowed by Def. 3.2 since the subtyping relation provides no rules for recursion variables; consequently, the subtyping condition in Def. 3.7 is not satisfied.

4. Multiparty Session Typing System

This section presents a type system for the multiparty session π-calculus (defined in § 2). In § 4.1, we introduce the typing rules. § 4.2 and 4.3 demonstrate the main properties of typed processes: *subject reduction* and *session fidelity*. Finally, § 4.4 shows how process properties such as deadlock-freedom and liveness can be guaranteed by construction.

4.1. Typing Rules

Two kinds of typing contexts, as introduced in Def. 3.5, are used in our type system: Γ, which assigns a sequence of basic types and session types to each process variable X, as well as a basic type to each expression variable x; and Δ, which maps channels to session types. Γ is utilised in judgements for expressions, while both Γ and Δ are jointly applied in judgements for processes. The typing judgements are formulated as:

$$\Gamma \vdash e : B \quad \text{and} \quad \Gamma \vdash P \triangleright \Delta$$

The judgement for expressions is standard: given the expression variables and basic types in Γ, e is of basic type B. For processes, based on the types assigned to expressions and process variables in Γ, P uses its channels *linearly* as specified by Δ.

$$\Gamma \vdash \mathtt{n} : \mathrm{nat} \quad \Gamma \vdash \mathtt{i} : \mathrm{int} \quad \Gamma \vdash \mathtt{true} : \mathrm{bool} \quad \Gamma \vdash \mathtt{false} : \mathrm{bool} \quad \Gamma \vdash \mathtt{""} : \mathrm{str} \quad \Gamma \vdash \mathtt{()} : \mathrm{unit} \quad \Gamma, x{:}B \vdash x : B$$

$$\frac{\Gamma \vdash e : \mathrm{nat}}{\Gamma \vdash \mathtt{succ}(e) : \mathrm{nat}} \quad \frac{\Gamma \vdash e : \mathrm{int}}{\Gamma \vdash \mathtt{neg}(e) : \mathrm{nat}} \quad \frac{\Gamma \vdash e : \mathrm{bool}}{\Gamma \vdash \neg e : \mathrm{bool}} \quad \frac{\Gamma \vdash e_1 : \mathrm{int} \quad \Gamma \vdash e_2 : \mathrm{int}}{\Gamma \vdash e_1 < e_2 : \mathrm{bool}} \quad \frac{\Gamma \vdash e_1 : B \quad \Gamma \vdash e_2 : B}{\Gamma \vdash e_1 \otimes e_2 : B}$$

$$\frac{\forall c \in \mathrm{dom}(\Delta) \quad \Delta(c) = \mathbf{end}}{\mathrm{end}(\Delta)}\ [\text{T-end}] \qquad \frac{\mathrm{end}(\Delta)}{\Gamma \vdash \mathbf{0} \triangleright \Delta}\ [\text{T-}\mathbf{0}] \qquad \frac{\Gamma \vdash P \triangleright \Delta \quad \Delta \leqslant \Delta'}{\Gamma \vdash P \triangleright \Delta'}\ [\text{T-Sub}]$$

$$\frac{\forall i \in 1\ldots n \quad \Gamma \vdash e_i : B_i \quad \mathrm{end}(\Delta)}{\Gamma, X : B_1, \ldots, B_n, T_1, \ldots, T_m \vdash X\langle e_1, \ldots, e_n, c_1, \ldots, c_m\rangle \triangleright \Delta, c_1 : T_1, \ldots, c_m : T_m}\ [\text{T-Call}]$$

$$\frac{\begin{array}{c}\Gamma, X : B_1, \ldots, B_n, T_1, \ldots, T_m, x_1 : B_1, \ldots, x_n : B_n \vdash P \triangleright y_1 : T_1, \ldots, y_m : T_m \\ \Gamma, X : B_1, \ldots, B_n, T_1, \ldots, T_m \vdash Q \triangleright \Delta\end{array}}{\Gamma \vdash \mathbf{def}\ X(x_1 : B_1, \ldots, x_n : B_n, y_1 : T_1, \ldots, y_m : T_m) = P\ \mathbf{in}\ Q \triangleright \Delta}\ [\text{T-}\mathbf{def}]$$

$$\frac{\Gamma \vdash P_1 \triangleright \Delta_1 \quad \Gamma \vdash P_2 \triangleright \Delta_2}{\Gamma \vdash P_1 \mid P_2 \triangleright \Delta_1, \Delta_2}\ [\text{T-}|] \qquad \frac{\Gamma \vdash e : \mathrm{bool} \quad \Gamma \vdash P \triangleright \Delta \quad \Gamma \vdash Q \triangleright \Delta}{\Gamma \vdash \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \triangleright \Delta}\ [\text{T-If}]$$

$$\frac{\forall i \in I \quad \left\{\begin{array}{ll}\Gamma, z_i : B \vdash P_i \triangleright \Delta, c : T_i & \text{if } S_i = B \\ \Gamma \vdash P_i \triangleright \Delta, z_i : T, c : T_i & \text{if } S_i = T\end{array}\right\}}{\Gamma \vdash c[\mathtt{q}]\&\{\mathtt{m}_i(z_i).P_i\}_{i\in I} \triangleright \Delta, c : \mathtt{q}\&\{\mathtt{m}_i(S_i).T_i\}_{i\in I}}\ [\text{T-}\&]$$

$$\frac{\Gamma \vdash e : B \quad \Gamma \vdash P \triangleright \Delta, c : T}{\Gamma \vdash c[\mathtt{q}]\oplus\mathtt{m}\langle e\rangle.P \triangleright \Delta, c : \mathtt{q}\oplus\{\mathtt{m}(B).T\}}\ [\text{T-}\oplus\text{-V}] \qquad \frac{\Gamma \vdash P \triangleright \Delta, c : T}{\Gamma \vdash c[\mathtt{q}]\oplus\mathtt{m}\langle c'\rangle.P \triangleright \Delta, c : \mathtt{q}\oplus\{\mathtt{m}(T').T\}, c' : T'}\ [\text{T-}\oplus\text{-D}]$$

$$\frac{\Delta' = \{s[\mathtt{p}] : G\!\restriction\!\mathtt{p}\}_{\mathtt{p}\in \mathrm{roles}(G)} \quad s \notin \Delta \quad \Gamma \vdash P \triangleright \Delta, \Delta'}{\Gamma \vdash (\nu s : \Delta')\, P \triangleright \Delta}\ [\text{T-}G\text{-}\nu]$$

Figure 8: Typing rules for expressions (top) and processes (bottom).

The typing system is defined inductively by the rules depicted in Fig. 8. The typing rules for expressions are straightforward, while we focus on elaborating the rules for processes.

Rule [T-end] introduces a predicate $\mathrm{end}(\cdot)$ on typing contexts, which denotes the termination of all endpoints (note that $\mathrm{end}(\emptyset)$ holds). This predicate is used in [T-$\mathbf{0}$] to type an inactive process $\mathbf{0}$. Rule [T-Sub] incorporates subtyping within typing contexts. Rules [T-Call] and [T-**def**] handle recursive process calls and declarations, respectively. Rule [T-|] divides the typing context *linearly* into two parts, each used to type one of the sub-processes. Rule [T-If] is used for typing conditionals. In rule [T-&], we distinguish whether the payload in each branch is a basic type or a session type. Similarly, for typing selections, we apply two rules, [T-⊕-V] and [T-⊕-D], to differentiate the payload types. Finally, rule [T-G-ν] uses a typing context derived from a global type via projection to enforce session restriction. Note that association can be applied in place of projection in this rule, as a more general approach.

Example 4.1 (Typed Process). Consider the processes P and Q from Ex. 2.1, along with a type context $\Delta_H = s[\mathtt{p}] : T_{H_\mathtt{p}}, s[\mathtt{q}] : T_{H_\mathtt{q}}, s[\mathtt{r}] : T_{H_\mathtt{r}}$, where $T_{H_\mathtt{p}} = \mathtt{q}\oplus\mathtt{m}'(T_{H_\mathtt{r}}).\mathtt{r}\&\mathtt{m}(\mathrm{int}).\mathbf{end}$, $T_{H_\mathtt{q}} = \mathtt{p}\&\mathtt{m}'(T_{H_\mathtt{r}}).\mathbf{end}$, $T_{H_\mathtt{r}} = \mathtt{p}\oplus\mathtt{m}(\mathrm{int}).\mathbf{end}$. The context Δ_H can type the process $P \mid Q$ through the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathrm{end}(s[\mathtt{p}] : \mathbf{end})}{z : \mathrm{int} \vdash \mathbf{0} \triangleright s[\mathtt{p}] : \mathbf{end}}\,[\text{T-}\mathbf{0}]}{\emptyset \vdash s[\mathtt{p}][\mathtt{r}]\&\mathtt{m}(z).\mathbf{0} \triangleright s[\mathtt{p}] : \mathtt{r}\&\mathtt{m}(\mathrm{int}).\mathbf{end}}\,[\text{T-}\&]}{\emptyset \vdash P \triangleright s[\mathtt{p}] : T_{H_\mathtt{p}}, s[\mathtt{r}] : T_{H_\mathtt{r}}}\,[\text{T-}\oplus\text{-D}] \quad \dfrac{\dfrac{\emptyset \vdash 42 : \mathrm{int} \quad \dfrac{\mathrm{end}(s[\mathtt{q}] : \mathbf{end}, z : \mathbf{end})}{\emptyset \vdash \mathbf{0} \triangleright s[\mathtt{q}] : \mathbf{end}, z : \mathbf{end}}\,[\text{T-}\mathbf{0}]}{\emptyset \vdash z[\mathtt{p}]\oplus\mathtt{m}\langle 42\rangle.\mathbf{0} \triangleright s[\mathtt{q}] : \mathbf{end}, z : T_{H_\mathtt{r}}}\,[\text{T-}\oplus\text{-V}]}{\emptyset \vdash Q \triangleright s[\mathtt{q}] : T_{H_\mathtt{q}}}\,[\text{T-}\&]}{\emptyset \vdash P \mid Q \triangleright \Delta_H}\,[\text{T-}|]$$

Moreover, since Δ_H is obtained by projecting a global type $G_H = \mathtt{p}{\to}\mathtt{q}{:}\mathtt{m}'(\mathtt{p}\oplus\mathtt{m}(\mathrm{int})).\mathtt{r}{\to}\mathtt{p}{:}\mathtt{m}(\mathrm{int}).\mathbf{end}$ onto roles in session s, i.e. $\Delta_H = \{s[\mathtt{p}] : G_H \upharpoonright \mathtt{p}\}_{\mathtt{p} \in \mathrm{roles}(G_H)}$, it follows from [T-G-$\nu$] that the process $P \mid Q$ is closed under Δ_H, i.e. $\emptyset \vdash (\nu s : \Delta_H)\, P \mid Q \triangleright \emptyset$.

Example 4.2 (Typed Process of OAuth). Recall the typing contexts $\Delta_{\mathrm{auth}_{\mathtt{s}}}$, $\Delta_{\mathrm{auth}_{\mathtt{c}}}$, and $\Delta_{\mathrm{auth}_{\mathtt{a}}}$ from Ex. 3.4, and the processes $P_{\mathtt{s}}$, $P_{\mathtt{c}}$, and $P_{\mathtt{a}}$ from Ex. 2.2. These contexts enable the typing of the respective processes. Therefore, the context Δ_{auth} (from Ex. 3.4) can type the process $P_{\mathtt{s}} \mid P_{\mathtt{c}} \mid P_{\mathtt{a}}$.

Furthermore, the typing context $s[\mathtt{s}] : T_{\mathtt{s}}$, using the local type $T_{\mathtt{s}}$ from Ex. 3.1, can type the process $P_{\mathtt{s}}$, since $\Delta_{\mathrm{auth}_{\mathtt{s}}} \leqslant s[\mathtt{s}] : T_{\mathtt{s}}$ (as demonstrated in Ex. 3.4), and $\emptyset \vdash P_{\mathtt{s}} \triangleright \Delta_{\mathrm{auth}_{\mathtt{s}}}$. Similarly, $s[\mathtt{c}] : T_{\mathtt{c}}$ and $s[\mathtt{a}] : T_{\mathtt{a}}$ type $P_{\mathtt{c}}$ and $P_{\mathtt{a}}$, respectively. Thus, by [T-G-ν], $P_{\mathtt{s}} \mid P_{\mathtt{c}} \mid P_{\mathtt{a}}$ is closed under $\{s[\mathtt{p}] : G_{\mathrm{auth}} \upharpoonright \mathtt{p}\}_{\mathtt{p} \in \mathrm{roles}(G_{\mathrm{auth}})}$ (where G_{auth} is from Ex. 3.1).

4.2. Subject Reduction

Subject reduction (type soundness) ensures the preservation of well-typedness under reduction: if a process P is typable and $P \to P'$, then P' remains typable. Consequently, no reduction sequence starting from a well-typed process can reach an untypable state.

We present a subject reduction result in Thm. 4.1, where P is constructed from a global type via association (Def. 3.7), i.e. it is typed by a context associated with the global type. Thm. 4.1 serves as the fundamental technical result, establishing such a property as an invariant preserved throughout reduction.

Theorem 4.1 (Subject Reduction via Association)**.** *Assume $\Gamma \vdash P \triangleright \Delta$ where $\forall s \in \Delta : \exists G_s : \Delta_s \sqsubseteq_s G_s$. If $P \to P'$, then $\exists \Delta'$ such that $\Delta \to^* \Delta'$, $\Gamma \vdash P' \triangleright \Delta'$, and $\forall s \in \Delta' : \exists G'_s : G_s \to^* G'_s$ and $\Delta'_s \sqsubseteq_s G'_s$.*

Proof. By induction on the derivation of $P \to P'$ (Def. 2.2). □

Furthermore, since a typing context obtained from a global type via projection is a supertype of the context associated with it, and a process can be typed using a broader context, Thm. 4.2 – our key subject reduction result – demonstrates that well-typedness is maintained for processes constructed from global types via projection. This result recovers the standard projection-based form of subject reduction and reinforces compatibility with global types as a behavioural invariant through reductions, ensuring correctness by construction in alignment with the MPST top-down methodology.

Theorem 4.2 (Subject Reduction)**.** *Assume $\Gamma \vdash P \triangleright \Delta$, where $\forall s \in \Delta : \exists G_s : \Delta_s = \{s[\mathtt{p}] : G_s \upharpoonright \mathtt{p}\}_{\mathtt{p} \in \mathrm{roles}(G_s)}$. If $P \to P'$, then $\exists \Delta'$ such that $\Gamma \vdash P' \triangleright \Delta'$, and $\forall s \in \Delta' : \exists G'_s : G_s \to^* G'_s$ and $\Delta'_s = \{s[\mathtt{p}] : G'_s \upharpoonright \mathtt{p}\}_{\mathtt{p} \in \mathrm{roles}(G'_s)}$.*

Proof. By the definition of association (Def. 3.7), subject reduction via association (Thm. 4.1), and the application of [T-Sub] as necessary. □

As a corollary of Thm. 4.2, well-typed processes communicate without `errors`.

Corollary 4.3 (Type Safety)**.** *Assume $\emptyset \vdash P \triangleright \emptyset$. If $P \to^* P'$, then P' has no error.*

Example 4.3 (Subject Reduction). Recall the typed process $P \mid Q$, the typing context Δ_H, and the global type G_H from Ex. 4.1. Using rule [R-⊕&-D], the process $P \mid Q$ reduces to

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

The typing context Δ_H, with $\Delta_H \sqsubseteq_s G_H$, transitions via rule [Δ-⊕&] to

$$\Delta'_H = s[\mathtt{p}] : \mathtt{r}\&\mathtt{m}(\mathrm{int}).\mathbf{end}, s[\mathtt{q}] : \mathbf{end}, s[\mathtt{r}] : \mathtt{p}\oplus\mathtt{m}(\mathrm{int}).\mathbf{end}$$

which types $P' \mid Q'$ and is associated with the global type $G'_H = \mathtt{r} \to \mathtt{p}{:}\mathtt{m}(\mathrm{int}).\mathbf{end}$, obtained by the transition $G_H \xrightarrow{s[\mathtt{p}][\mathtt{q}]\mathtt{m}'} G'_H$. Moreover, the typing context $\Delta''_H = \{s[\mathtt{p}] : G'_H \upharpoonright \mathtt{p}\}_{\mathtt{p} \in \mathrm{roles}(G'_H)}$ also types $P' \mid Q'$.

Note that the typing context $\{s[\mathtt{p}] : G_{\mathrm{auth}} \upharpoonright \mathtt{p}\}_{\mathtt{p} \in \mathrm{roles}(G_{\mathrm{auth}})}$ and the process $P_{\mathtt{s}} \mid P_{\mathtt{c}} \mid P_{\mathtt{a}}$ from Ex. 4.2 also adhere to subject reduction, thereby ensuring type safety.

4.3. Session Fidelity

Session fidelity asserts the converse implication concerning subject reduction: if a process P is typed by Δ, and Δ can reduce along session s, then P can replicate at least one of the reductions performed by Δ (though not necessarily all, as Δ may over-approximate the behaviour of P).

However, this property does *not* hold universally for all well-typed processes. For example, a well-typed process may diverge due to unguarded recursion (e.g. $\mathbf{def}\ X(\ldots) = X\ \mathbf{in}\ X$), or may deadlock due to intricate interleavings of communications across multiple sessions [26].

To address this, and in line with [84] and most existing work on session types, we establish session fidelity specifically for processes featuring guarded recursion and implementing a single multiparty session, realised as a parallel composition of one sub-process per role. The guarantees for process properties given in § 4.4 are demonstrated under the same guarded-recursion, single-session assumption.

Note that alternative approaches exist for handling interleaved multiparty sessions with delegation [79, 26, 49, 22, 23]. For example, a decentralised analysis is proposed in [49] to ensure session fidelity and deadlock freedom for implementations with interleaved and delegated multiparty sessions. In addition, the type system introduced in [22, 23] guarantees termination of well-typed processes under fairness assumptions in the presence of session chaining, nesting, interleaving, delegation, and dynamic session creation.

The formalisation of our session fidelity, as provided in Thms. 4.4 and 4.5 below, builds upon the concepts introduced in Def. 4.1.

Definition 4.1 (from [84]). Assume $\emptyset \vdash P \triangleright \Delta$. We say that P:

(1) *has guarded definitions* if and only if in each process definition in P of the form $\mathbf{def}\ X(x_1 : B_1, \ldots, x_n : B_n, \ldots, y_1 : T_1, \ldots, y_m : T_m) = Q\ \mathbf{in}\ P'$, for all $i \in 1\ldots m$, a call $Y\langle \ldots, y_i, \ldots \rangle$ can only occur in Q as a subterm of $y_i[\mathtt{q}]\&\{\mathtt{m}_j(z_j).P_j\}_{j\in J}$ or $y_i[\mathtt{q}]\oplus\mathtt{m}\langle d\rangle.P''$ (i.e. after using y_i for input or output);

(2) *only plays role* $\mathtt{p}$ *in* s*, by* Δ if and only if: (i) P has guarded definitions; (ii) $\mathrm{fev}(P) = \emptyset$; (iii) $\mathrm{fcv}(P) = \emptyset$; (iv) $\Delta = \Delta_0, s[\mathtt{p}] : T$ with $T \not\leqslant \mathbf{end}$ and $\mathrm{end}(\Delta_0)$; (v) for all subterms $(\nu s' : \Delta')\, P'$ in P, $\mathrm{end}(\Delta')$.

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

In Def. 4.1, item (1) formalises guarded recursion for processes, while item (2) defines a process that plays exactly *one* role on *one* session. It is evident that an ensemble of such processes cannot deadlock by waiting for each other across multiple sessions.

Example 4.4 (Playing Only Role). Consider the processes P, Q, and the typing context Δ_H from Ex. 4.1. Observe that P does not only play either $\mathtt{p}$ or $\mathtt{r}$ in s. This is because P can only be typed by a context of the form $s[\mathtt{p}] : T_{\mathtt{p}}, s[\mathtt{r}] : T_{\mathtt{r}}$, where neither $\mathrm{end}(s[\mathtt{p}] : T_{\mathtt{p}})$ nor $\mathrm{end}(s[\mathtt{r}] : T_{\mathtt{r}})$ holds, thus violating item (2) of Def. 4.1. Conversely, Q only plays role $\mathtt{q}$ in s, by $s[\mathtt{q}] : T_{H_{\mathtt{q}}}$, with all required conditions satisfied.

Additionally, the processes $P_{\mathtt{s}}$, $P_{\mathtt{c}}$, and $P_{\mathtt{a}}$ from Ex. 4.2 only play roles $\mathtt{s}$, $\mathtt{c}$, and $\mathtt{a}$, respectively, in s, which can be easily verified.

We now formalise our session fidelity properties (Thms. 4.4 and 4.5). Similar to subject reduction in §4.2, Thm. 4.4 utilises a typing context associated with a global type for a specific session s to type the process, while Thm. 4.5, as a corollary, applies a typing context projected from a global type, with both asserting that a process constructed from a global type preserves its structure after reductions.

Theorem 4.4 (Session Fidelity via Association). *Assume* $\emptyset \vdash P \triangleright \Delta$*, with* $\Delta \sqsubseteq_s G$*,* $P \equiv \Pi_{\mathtt{p}\in I} P_{\mathtt{p}}$*, and* $\Delta = \bigcup_{\mathtt{p}\in I} \Delta_{\mathtt{p}}$ *such that for each* $P_{\mathtt{p}}$*: (1)* $\emptyset \vdash P_{\mathtt{p}} \triangleright \Delta_{\mathtt{p}}$*, and (2) either* $P_{\mathtt{p}} \equiv \mathbf{0}$*, or* $P_{\mathtt{p}}$ *only plays* $\mathtt{p}$ *in* s*, by* $\Delta_{\mathtt{p}}$*. Then,* $\Delta \rightarrow_s$ *implies* $\exists \Delta', G', P'$ *such that* $\Delta \rightarrow_s \Delta'$*,* $G \rightarrow G'$*,* $P \rightarrow^* P'$*, and* $\emptyset \vdash P' \triangleright \Delta'$*, with* $\Delta' \sqsubseteq_s G'$*,* $P' \equiv \Pi_{\mathtt{p}\in I} P'_{\mathtt{p}}$*, and* $\Delta' = \bigcup_{\mathtt{p}\in I} \Delta'_{\mathtt{p}}$ *such that for each* $P'_{\mathtt{p}}$*: (1)* $\emptyset \vdash P'_{\mathtt{p}} \triangleright \Delta'_{\mathtt{p}}$*, and (2) either* $P'_{\mathtt{p}} \equiv \mathbf{0}$*, or* $P'_{\mathtt{p}}$ *only plays* $\mathtt{p}$ *in* s*, by* $\Delta'_{\mathtt{p}}$*.*

Proof. By induction on the derivation of $\Delta \rightarrow_s$ (Def. 3.6). □

Theorem 4.5 (Session Fidelity)**.** *Assume* $\emptyset \vdash P \triangleright \{s[\mathtt{p}] : G \upharpoonright \mathtt{p}\}_{\mathtt{p} \in I}$*, with* $\mathrm{roles}(G) \subseteq I$ *and* $P \equiv \Pi_{\mathtt{p} \in I} P_{\mathtt{p}}$ *such that for each* $P_{\mathtt{p}}$*: (1)* $\emptyset \vdash P_{\mathtt{p}} \triangleright s[\mathtt{p}] : G \upharpoonright \mathtt{p}$*, and (2) either* $P_{\mathtt{p}} \equiv \mathbf{0}$*, or* $P_{\mathtt{p}}$ *only plays* $\mathtt{p}$ *in* s*, by* $s[\mathtt{p}] : G \upharpoonright \mathtt{p}$*. Then,* $G \rightarrow$ *implies* $\exists G', P'$ *such that* $G \rightarrow G'$*,* $P \rightarrow^* P'$*, and* $\emptyset \vdash P' \triangleright \{s[\mathtt{p}] : G' \upharpoonright \mathtt{p}\}_{\mathtt{p} \in I}$*, with* $\mathrm{roles}(G') \subseteq I$ *and* $P' \equiv \Pi_{\mathtt{p} \in I} P'_{\mathtt{p}}$ *such that for each* $P'_{\mathtt{p}}$*: (1)* $\emptyset \vdash P'_{\mathtt{p}} \triangleright s[\mathtt{p}] : G' \upharpoonright \mathtt{p}$*, and (2) either* $P'_{\mathtt{p}} \equiv \mathbf{0}$*, or* $P'_{\mathtt{p}}$ *only plays* $\mathtt{p}$ *in* s*, by* $s[\mathtt{p}] : G' \upharpoonright \mathtt{p}$*.*

Proof. By the definition of association (Def. 3.7), the soundness of association (Thm. 3.1), and session fidelity via association (Thm. 4.4). □

Example 4.5 (Guarded Definitions in Session Fidelity, from [84]). According to rule [T-def] in Fig. 8, an unguarded definition $X(y : T) = X\langle y \rangle$ can be typed with *any* T. Therefore, we have:

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

The unguarded process above reduces vacuously by infinitely invoking X, without aligning with any typing context reduction. This highlights the necessity of guarded definitions for session fidelity (Thms. 4.4 and 4.5).

Observe that the process P from Ex. 4.1 does not satisfy (2) of Def. 4.1, while Q does, as shown in Ex. 4.4. However, as demonstrated in Ex. 4.1, $P \mid Q$ adheres to session fidelity. Readers might question the necessity of enforcing the strict condition in session fidelity that requires a process to play exactly one role in a single session. The following example highlights its importance.

Example 4.6 (Playing Only Role in Session Fidelity). Take the processes $P_z = s[\mathtt{A}][\mathtt{B}] \oplus \mathtt{m}.s[\mathtt{C}][\mathtt{D}] \oplus \mathtt{m}$, $Q_z = s[\mathtt{D}][\mathtt{C}] \& \mathtt{m}.s[\mathtt{B}][\mathtt{A}] \& \mathtt{m}$, and the typing context $\Delta_z = s[\mathtt{A}] : \mathtt{B} \oplus \mathtt{m}, s[\mathtt{B}] : \mathtt{A} \& \mathtt{m}, s[\mathtt{C}] : \mathtt{D} \oplus \mathtt{m}, s[\mathtt{D}] : \mathtt{C} \& \mathtt{m}$, which can be obtained by projecting the global type $G_z = \mathtt{A} \to \mathtt{B}{:}\mathtt{m}.\mathtt{C} \to \mathtt{D}{:}\mathtt{m}$ onto s. It is trivial to verify that neither P_z nor Q_z satisfies (2) of Def. 4.1. Moreover, it is evident that $P_z \mid Q_z$ does not satisfy session fidelity, as the process cannot reduce further, whereas Δ_z remains reducible.

4.4. Properties of Typed Processes

We finalise this section by showcasing that processes constructed from global types guarantee desirable run-time properties, including *deadlock-freedom* and *liveness*, as formalised in Def. 4.2. Deadlock-freedom indicates that if a process cannot reduce, it consists only of inactive sub-processes ($\mathbf{0}$), while liveness (a.k.a. "lock-freedom" [63, 78]) ensures that every pending input or output action of a process can eventually engage in communication, i.e. there exists a future execution in which it synchronises with the corresponding dual.

Definition 4.2 (Runtime Process Properties). We say P is:
(1) *deadlock-free* iff $P \rightarrow^* P' \not\rightarrow$ implies $P' \equiv \mathbf{0}$;
(2) *live* iff $P \rightarrow^* P' \equiv \mathbb{C}[Q]$ implies:
  (a) if $Q = c[\mathtt{q}] \oplus \mathtt{m}\langle w \rangle.Q'$, then $\exists \mathbb{C}' : \mathbb{C} \rightsquigarrow^* \mathbb{C}'$ and $P' \rightarrow^* \mathbb{C}'[Q']$;
  (b) if $Q = c[\mathtt{q}] \& \{\mathtt{m}_i(x_i).Q'_i\}_{i \in I}$, then $\exists \mathbb{C}', k \in I, u : \mathbb{C} \rightsquigarrow^* \mathbb{C}'$ and $P' \rightarrow^* \mathbb{C}'[Q'_k\{u/x_k\}]$.

Remark 4.1 (Correction of Process Liveness in Scalas and Yoshida [84]). In [84, Def. 5.1], a process P is defined to be live iff for every P' such that $P \rightarrow^* P' \equiv \mathbb{C}[Q]$,

- if $Q = c[\mathtt{q}] \oplus \mathtt{m}\langle w \rangle.Q'$, then $\exists \mathbb{C}' : P' \rightarrow^* \mathbb{C}'[Q']$;
- if $Q = c[\mathtt{q}] \& \{\mathtt{m}_i(x_i).Q'_i\}_{i \in I}$, then $\exists \mathbb{C}', k \in I, u : P' \rightarrow^* \mathbb{C}'[Q'_k\{u/x_k\}]$.

In this formulation, the witness context $\mathbb{C}'$ is unrestricted and need not be related to the original context $\mathbb{C}$. Consequently, the liveness condition may be satisfied by selecting an arbitrary context that enables the desired reduction.

For example, consider the process $P = s[\mathtt{p}][\mathtt{q}] \oplus \mathtt{m}\langle 5 \rangle.\mathbf{0}$, which is *not* live since no matching input action exists. Under the above definition, however, one may take $\mathbb{C}' = [\,] \mid P$ to obtain $P \rightarrow^* \mathbf{0} \mid P = \mathbb{C}'[\mathbf{0}]$, making the liveness notion hold vacuously, despite the absence of any communication.

Def. 4.2 addresses this issue by requiring $\mathbb{C} \rightsquigarrow^* \mathbb{C}'$, thereby enforcing a structural relationship between $\mathbb{C}'$ and $\mathbb{C}$. This constraint ensures that, under this liveness formalisation, the execution of an output or input action arises from reductions of the process itself, rather than from an arbitrarily extended environment.

Note that the process liveness property adopted in existing work (e.g. [65, 3, 7]) relies on the definition in [84], and may therefore require corresponding revision. ◂

Finally, we illustrate how a process, typed with a typing context obtained from a global type via projection, ensures both deadlock-freedom and liveness.

Theorem 4.6 (Process Deadlock-Freedom, Liveness)**.** *Assume* $\emptyset \vdash P \triangleright \{s[\mathtt{p}] : G \upharpoonright \mathtt{p}\}_{\mathtt{p} \in \mathrm{roles}(G)}$*, where* $P \equiv \Pi_{\mathtt{p} \in \mathrm{roles}(G)} P_{\mathtt{p}}$ *and for each* $P_{\mathtt{p}}$*,* $\emptyset \vdash P_{\mathtt{p}} \triangleright s[\mathtt{p}] : G \upharpoonright \mathtt{p}$*. Further, assume that each* $P_{\mathtt{p}}$ *is either* $\mathbf{0}$ *(up to* $\equiv$*), or only plays* $\mathtt{p}$ *in* s*, by* $s[\mathtt{p}] : G \upharpoonright \mathtt{p}$*. Then,* P *is deadlock-free and live.*

Example 4.7 (Typed Process Properties). The process $P_{\mathtt{s}} \mid P_{\mathtt{c}} \mid P_{\mathtt{a}}$ from Ex. 4.2 is both deadlock-free and live, as can be easily verified by applying either Def. 4.2 or Thm. 4.6.

5. Related Work

In this section, we explore related work on multiparty session types (MPST). We begin by discussing top-down frameworks, with emphasis on their proof methodologies for type soundness (subject reduction), while also including work on the projectability and implementability of global types. We then cover developments based on the bottom-up approach. Additionally, we discuss mechanisations of MPST, focusing on projection-based settings. This overview is intended to trace the development of these approaches rather than to provide an exhaustive survey, and therefore not all related work is included.

5.1. Top-Down Multiparty Session Types

The classic top-down MPST framework, with its notions of *global types* and *projections*, was first introduced in Honda et al. [52] and fully developed in Honda et al. [53], where *linearity conditions* were incorporated to ensure the well-formedness of global types and the *projectability* of local types. Subsequently, Bettini et al. [5] proposed a simplified MPST system without type-level channel declarations, which has since been widely adopted in most works, both theoretical and practical, including ours. Later, Carbone et al. [14, 13] and Caires and Pérez [10] investigated the logical foundations of MPST under restricted classes of global types.

We now classify some related works according to their treatment of *projection*, *consistency*, and *association*, as well as the correctness of their proofs for the subject reduction theorem.

Papers	Projection	Consistency	Association	Subject Reduction
(a) [52, 53, 5, 14, 13, 25]	$\leq$ plain	yes	no	correct
(b) [99, 31, 30, 19, 92]	$\geq$ full	no	no	flawed
(c) [82, 93]	full	yes (required)	no	correct
(d) [1, 2, 101, 54]	full	no	yes	correct
(e) [33, 43, 10, 65]	full	no	no	correct

Row (a) lists works that use *plain* (or stricter) global type projection, which guarantees *consistency*. Consequently, their proofs of type soundness, i.e. subject reduction, are correct, as they rely on consistency as the invariant φ to be satisfied in **(SR3)** in § 1. However, this plain projection is overly restrictive, excluding many valid protocols – even a simple one such as $G_{\mathtt{f}}$ in § 1 – at the cost of guaranteeing consistency.

Row (b) lists works that adopt *full* (or more flexible) global type projection, originally introduced in Yoshida et al. [99] to support a wider range of protocols. These works overlook the consistency requirement, and, as shown in § 1, subject reduction proofs that rely on full projection (without consistency) are flawed.

To "fix" these issues within MPST theory, the works in **row (c)** enforce consistency while keeping full projection, which restricts typability and thus falls back into over-restriction. In contrast, our proposed proof technique, based on the *association* relation under full projection, establishes subject reduction within the top-down MPST framework without losing expressivity.

Row (d) specifies works that employ *association* relations similar to the one used in our work, thereby guaranteeing subject reduction. The notion of association was first proposed in Barwell et al. [1] and further developed in Barwell et al. [2] within an MPST framework with crash-stop failures, and later extended in Hou et al. [54] to incorporate time and failure handling. In these settings, the association relation is used to establish a sound and complete operational correspondence between global and local semantics, ensuring that key global type properties are preserved in local types through projection. Our work, however, develops this idea into the foundation of a new invariant-based proof approach to subject reduction. Moreover, the framework of [1, 2] is limited to a *single-session* type system with first-order session types (i.e. without channel passing), and both [1, 2] and [54] assume asynchronous communication and adopt a channel-oriented subtyping discipline. As a result, their association relations differ substantially from ours, which is formulated for a synchronous model with a distinct subtyping direction and different applicability.

Additionally, there are several MPST works that fall within the top-down framework but are *neither* based on the classic projection+consistency approach *nor* employ the association relation as in our work, as shown in **row (e)**. Dezani-Ciancaglini et al. [33], with a journal version in Ghilezan et al. [43], introduce a single-session type system without channel passing, similar to that of Barwell et al. [1, 2]. Rooted in global types and their projections, it does *not* require consistency. Such a system is strictly subsumed by our framework, which, in contrast, supports higher-order types and multiple interleaved sessions. Moreover, their subject reduction proof strategy proceeds by reasoning directly on global types and their semantics, whereas ours relies on the operational correspondence between global types and typing contexts via association, making explicit use of typing contexts in the proof.

Caires and Pérez [10] develop a theory of multiparty session types encoded into binary sessions, with a type system based on linear logic from Caires and Pfenning [11] and Wadler [97]. A related multiparty-to-binary session decomposition was later studied in Scalas et al. [82], with a crucial difference: in [82], consistency is a *necessary* requirement (formalised in their Theorem 6.3), whereas in [10] it is not, despite supporting full projection and merging. This distinction arises because the decomposition in [10] introduces a centralised *medium process* (similar to the *arbiter* in Carbone et al. [13]) that receives and forwards all messages between processes playing different roles, whereas the decomposition in [82] preserves the peer-to-peer nature of MPST interactions. This suggests that, when decomposing multiparty choreographies into linear binary interactions, consistency is necessary if and only if no centralised medium process is introduced. Our present work supports binary sessions as a special case, without requiring either consistency or medium processes.

Finally, Lagaillardie et al. [65] present an MPST framework with affine communication channels and implicit/explicit cancellation mechanisms, where subject reduction is formalised in the style of Scalas and Yoshida [84], with typing-context safety as the key invariant. Since this property is guaranteed by projection, it suffices to establish correctness. Our approach, by contrast, adopts a different invariant – association – and explicitly incorporates both global types and projection into the theorem, aligning more closely with the top-down formulation.

Projectability and Implementability of Global Types. A complementary line of top-down work focuses on the projectability and implementability of global types, addressing the existence and behavioural correctness of projections. In this setting, the implementability problem asks whether there exist local specifications for all roles such that their composition is deadlock-free and generates exactly the executions specified by the global type.

Stutz [85] establishes decidability of implementability for a class of MPSTs via a reduction to safe realisability of globally cooperative high-level message sequence charts. Li et al. [66] introduce an automata-theoretic projection framework that separates synthesis from implementability checking, yielding a practical sound and complete projection operator for general MPSTs with a PSPACE decision procedure. Stutz and D'Osualdo [86] further generalise this approach by extending the projection algorithm of [66] to Protocol State Machines, a highly expressive formalism for global protocol specifications, and deriving Communicating State Machines as local specifications.

In contrast, the works discussed earlier in this section that employ projection – including our own – do not focus on completeness or decidability of projection. Instead, these approaches typically rely on syntactic projection operators and study how type soundness can be ensured in the presence of a fixed projection.

5.2. Bottom-Up Multiparty Session Types

The first attempt to develop a theory of MPST based on a bottom-up method, rather than the top-down approach with global types and projections, was presented in Scalas and Yoshida [83]. Subsequently, Scalas and Yoshida [84] introduced a general MPST typing system that does not rely on global types: it ensures desired process properties by checking the corresponding properties of typing contexts. These properties are specified in modal μ-calculus formulae and verified using the mCRL2 checker from Bunte et al. [9]. Unlike the top-down approach, it is not constrained by the projectability or implementability of global types, thereby capturing the full class of well-behaved processes while preserving type soundness. However, Udomsrirungruang and Yoshida [94] observe that this gain in typability comes at a higher computational cost. In asynchronous MPST with unbounded FIFO queues, the choice of methodology becomes even more crucial: type checking is *undecidable* under the bottom-up approach, as shown by Scalas and Yoshida [84], but remains decidable in the top-down approach we adopt, thanks to the decidability of end-point projection, as presented in Honda et al. [52, 53] and Scalas and Yoshida [84].

Over time, this line of work has also inspired a number of extensions. For example, Harvey et al. [48] adapt the bottom-up approach to model actor systems with explicit connection types from Hu and Yoshida [56]; Barwell et al. [3] apply it to account for crash-stop failures; Brun and Dardha [7] broaden it to address a wider range of fault models; Brun et al. [8] extend it into a session-typed multiparty process calculus with replication and first-class roles; and Giunti and Yoshida [45] develop it further with iso-recursion.

5.3. Mechanisations in Multiparty Session Types

Due to the complexity of MPST theories, numerous mechanisations have been developed across different frameworks to ensure correctness rigorously. To focus the discussion, we restrict attention to works that employ mechanised proofs in MPST frameworks involving projection.

Castro-Perez et al. [15] introduce Zooid, a domain-specific language embedded in Rocq for certified multiparty communication. Zooid formalises the syntaxes of global and local types inductively, together with a coinductive tree-based representation supporting a semantic interpretation. By defining an unravelling relation from types to trees and a projection operation from global to local specifications, they mechanise the result that projection is preserved under unravelling. Moreover, Zooid provides mechanisation proofs of sound and complete correspondence between the labelled transition systems of global and local types, in terms of execution-trace equivalence, thereby ensuring properties such as deadlock-freedom and protocol compliance. Additionally, the syntax and semantics of processes, together with a typing system, are formalised, and type preservation is mechanised.

Tirore et al. [89], with a journal version in Tirore et al. [91], define a computable projection function from global types to local types and provide a Rocq mechanisation of its soundness and completeness with respect to a coinductive tree semantics. Their work addresses limitations of existing computable projections by introducing a function that is equally expressive as its coinductive, non-computable counterpart, while remaining decidable. However, the mechanisation focuses on projection, independently of any process or typing calculus; consequently, properties such as subject reduction, progress, and type safety for processes are not addressed.

Jacobs et al. [58] extend earlier work on a compiler for a functional language with binary session types by Tassarotti et al. [87], based on a simplified variant of the GV ("Good Variation") system by Gay and Vasconcelos [41], to MPGV, which enriches a linear lambda calculus with multiparty sessions and supports participant redirection and dynamic thread spawning. Their type system incorporates both global and local types, with projection operations similar to those of Castro-Perez et al. [15]. All guarantees for well-typed processes in MPGV are formally mechanised in Rocq, including type safety, session fidelity, global progress, deadlock-freedom, and leak-freedom.

Additionally, several closely related mechanisation works address subject reduction. Tirore [88] provides a Rocq formalisation of subject reduction for the multiparty session π-calculus of Honda et al. [52, 53], including session initialisation and delegation. The type system is based on channel-explicit global and local types, with projections derived from Tirore et al. [89, 91]. Channel-explicit types require additional linearity checks to guarantee the projectability of global types. In follow-up work, Tirore et al. [90] extend these results by formalising proofs of communication safety and safety preservation in Rocq.

Ekici et al. [35] develop a Rocq mechanisation of subject reduction and progress for MPST using coinductive reasoning over type trees derived from global and local types. Their approach exploits structural properties of these trees to refine projection accuracy in the presence of the precise subtyping discipline introduced by Ghilezan et al. [43], integrating subtyping into type checking and thereby extending the expressiveness of the type system.

All the mechanisations discussed above rely on plain (or stricter) merging.

Finally, regarding the mechanisation of implementability of global protocols, Li and Wies [67] present a Rocq formalisation of the precise implementability characterisation by Li et al. [66], discussed earlier in § 5.1. Their work unifies distinct frameworks, simplifies existing proof arguments, and makes explicit the construction of canonical implementations. The mechanisation further reveals a subtle issue in the semantics of infinite behaviours and shows that the implementability characterisation extends to protocols with infinitely many participants.

6. Conclusion

This paper addresses a recent concern in the multiparty session type community: namely, that the top-down approach with mergeability is unsound, or more strongly, that global types themselves are inherently problematic, and thus correctness-by-construction cannot be achieved from them. To clarify this, we introduce a new notion, *association*, which relates global types to sets of local types, and apply it to develop a general proof technique for establishing type soundness in top-down MPST frameworks. With this method, we show that a sound typing system can indeed be obtained via endpoint projection with mergeability, thereby demonstrating that global types provide a clear and principled foundation for proving type soundness. Moreover, we establish that the top-down typing discipline, supported by endpoint projection, guarantees type safety, deadlock-freedom, and liveness of session processes by construction.

Future work includes applying the association-based proof technique to MPST with alternative subtyping disciplines and projection methods, as well as to extended variants such as those incorporating probabilistic behaviour or more advanced fault-tolerant models. These directions will further evaluate the robustness and generality of the association method, and provide a unified foundation for establishing type soundness across a broader spectrum of MPST frameworks. In addition, we plan to verify the correctness of our approach through mechanisation, particularly with respect to full merging.

Acknowledgments

We thank the reviewers for their detailed and constructive comments and suggestions. We are particularly grateful to the reviewers, as well as to Jake Masters and Joe Paulus, for prompting a correction to the definition of process liveness (Definition 4.2). This work was partially supported by EPSRC grants EP/T006544/2, EP/T014709/2, EP/Y005244/1, EP/V000462/1, EP/X015955/1, EP/Z0005801/1; Horizon EU TaRDIS 101093006 (UKRI No. 10066667); Advanced Research and Invention Agency (ARIA) Safeguarded AI; and a grant from the Simons Foundation.

Declaration of interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Barwell, A.D., Hou, P., Yoshida, N., Zhou, F., 2023. Designing Asynchronous Multiparty Protocols With Crash Stop Failures, in: European Conference on Object Oriented Programming, pp. 1:1–1:30. doi:10.4230/LIPIcs.ECOOP.2023.1.

[2] Barwell, A.D., Hou, P., Yoshida, N., Zhou, F., 2025. Crash-Stop Failures in Asynchronous Multiparty Session Types. Logical Methods in Computer Science 21(2). doi:10.46298/lmcs-21(2:5)2025.

[3] Barwell, A.D., Scalas, A., Yoshida, N., Zhou, F., 2022. Generalised Multiparty Session Types with Crash-Stop Failures, in: 33rd International Conference on Concurrency Theory, Dagstuhl. pp. 35:1–35:25. doi:10.4230/LIPIcs.CONCUR.2022.35.

[4] Bernardi, G., Hennessy, M., 2016. Using higher-order contracts to model session types. LMCS 12(2). doi:10.2168/LMCS-12(2:10)2016.

[5] Bettini, L., Coppo, M., D'Antoni, L., De Luca, M., Dezani-Ciancaglini, M., Yoshida, N., 2008. Global Progress in Dynamically Interleaved Multiparty Sessions, in: CONCUR. doi:10.1007/978-3-540-85361-9_33.

[6] Bouma, J., de Gouw, S., Jongmans, S., 2023. Multiparty session typing in java, deductively, in: Sankaranarayanan, S., Sharygina, N. (Eds.), Tools and Algorithms for the Construction and Analysis of Systems - 29th International Conference, TACAS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Paris, France, April 22-27, 2023, Proceedings, Part II, Springer. pp. 19–27. URL: https://doi.org/10.1007/978-3-031-30820-8_3, doi:10.1007/978-3-031-30820-8_3.

[7] Brun, M.A.L., Dardha, O., 2023. Magπ: Types for failure-prone communication, in: Wies, T. (Ed.), Programming Languages and Systems - 32nd European Symposium on Programming, ESOP 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2023, Paris, France, April 22-27, 2023, Proceedings, Springer. pp. 363–391. URL: https://doi.org/10.1007/978-3-031-30044-8_14, doi:10.1007/978-3-031-30044-8_14.

[8] Brun, M.A.L., Fowler, S., Dardha, O., 2025. Multiparty session types with a bang!, in: Vafeiadis, V. (Ed.), 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, Springer. pp. 125–153. URL: https://doi.org/10.1007/978-3-031-91121-7_6, doi:10.1007/978-3-031-91121-7_6.

[9] Bunte, O., Groote, J.F., Keiren, J.J.A., Laveaux, M., Neele, T., de Vink, E.P., Wesselink, W., Wijs, A., Willemse, T.A.C., 2019. The mcrl2 toolset for analysing concurrent systems - improvements in expressivity and usability, in: Vojnar, T., Zhang, L. (Eds.), Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part II, Springer. pp. 21–39. URL: https://doi.org/10.1007/978-3-030-17465-1_2, doi:10.1007/978-3-030-17465-1_2.

[10] Caires, L., Pérez, J.A., 2016. Multiparty session types within a canonical binary theory, and beyond, in: FORTE. doi:10.1007/978-3-319-39570-8_6.

[11] Caires, L., Pfenning, F., 2010. Session types as intuitionistic linear propositions, in: CONCUR. doi:10.1007/978-3-642-15375-4_16.

[12] Carbone, M., Honda, K., Yoshida, N., 2012. Structured communication-centered programming for web services. ACM Trans. Program. Lang. Syst. 34, 8:1–8:78. URL: https://doi.org/10.1145/2220365.2220367, doi:10.1145/2220365.2220367.

[13] Carbone, M., Lindley, S., Montesi, F., Schürmann, C., Wadler, P., 2016. Coherence generalises duality: A logical explanation of multiparty session types, in: CONCUR. doi:10.4230/LIPIcs.CONCUR.2016.33.

[14] Carbone, M., Montesi, F., Schürmann, C., Yoshida, N., 2015. Multiparty session types as coherence proofs, in: CONCUR. doi:10.4230/LIPIcs.CONCUR.2015.412.

[15] Castro-Perez, D., Ferreira, F., Gheri, L., Yoshida, N., 2021. Zooid: a DSL for certified multiparty computation: from mechanised metatheory to certified multiparty processes, in: Freund, S.N., Yahav, E. (Eds.), PLDI '21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 2021, ACM. pp. 237–251. URL: https://doi.org/10.1145/3453483.3454041, doi:10.1145/3453483.3454041.

[16] Castro-Perez, D., Hu, R., Jongmans, S., Ng, N., Yoshida, N., 2019. Distributed programming using role-parametric session types in go: statically-typed endpoint APIs for dynamically-instantiated communication structures. Proc. ACM Program. Lang. 3, 29:1–29:30. doi:10.1145/3290342.

[17] Charalambides, M., Dinges, P., Agha, G., 2012. Parameterized concurrent multi-party session types, in: Kokash, N., Ravara, A. (Eds.), Proceedings 11th International Workshop on Foundations of Coordination Languages and Self Adaptation, FOCLASA 2012, Newcastle, U.K., September 8, 2012, pp. 16–30. URL: https://doi.org/10.4204/EPTCS.91.2, doi:10.4204/EPTCS.91.2.

[18] Charalambides, M., Dinges, P., Agha, G.A., 2016. Parameterized, concurrent session types for asynchronous multi-actor interactions. Sci. Comput. Program. 115-116, 100–126. URL: https://doi.org/10.1016/j.scico.2015.10.006, doi:10.1016/J.SCICO.2015.10.006.

[19] Chen, T.C., 2015. Lightening global types. JLAMP 84. doi:https://doi.org/10.1016/j.jlamp.2015.06.003.

[20] Chen, T.C., Dezani-Ciancaglini, M., Scalas, A., Yoshida, N., 2017. On the preciseness of subtyping in session types. Logical Methods in Computer Science 13. doi:10.23638/LMCS-13(2:12)2017.

[21] Chen, T.C., Dezani-Ciancaglini, M., Yoshida, N., 2014. On the preciseness of subtyping in session types, in: PPDP. doi:10.1145/2643135.2643138. Journal version: [20].

[22] Ciccone, L., Dagnino, F., Padovani, L., 2022. Fair termination of multiparty sessions, in: Ali, K., Vitek, J. (Eds.), 36th European Conference on Object-Oriented Programming, ECOOP 2022, Berlin, Germany, June 6-10, 2022, Schloss Dagstuhl - Leibniz-Zentrum für Informatik. pp. 26:1–26:26. URL: https://doi.org/10.4230/LIPIcs.ECOOP.2022.26, doi:10.4230/LIPICS.ECOOP.2022.26.

[23] Ciccone, L., Dagnino, F., Padovani, L., 2024. Fair termination of multiparty sessions. J. Log. Algebraic Methods Program. 139, 100964. URL: https://doi.org/10.1016/j.jlamp.2024.100964, doi:10.1016/J.JLAMP.2024.100964.

[24] Cledou, G., Edixhoven, L., Jongmans, S.S., Proença, J., 2022. API Generation for Multiparty Session Types, Revisited and Revised Using Scala 3, in: Ali, K., Vitek, J. (Eds.), 36th European Conference on Object-Oriented Programming (ECOOP 2022), Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany. pp. 27:1–27:28. URL: https://drops.dagstuhl.de/opus/volltexte/2022/16255.

[25] Coppo, M., Dezani-Ciancaglini, M., Padovani, L., Yoshida, N., 2015a. A Gentle Introduction to Multiparty Asynchronous Session Types, in: Formal Methods for Multicore Programming. doi:10.1007/978-3-319-18941-3_4.

[26] Coppo, M., Dezani-Ciancaglini, M., Yoshida, N., Padovani, L., 2015b. Global progress for dynamically interleaved multiparty sessions. MSCS 760. doi:10.1017/S0960129514000188.

[27] Cutner, Z., Yoshida, N., Vassor, M., 2022. Deadlock-Free Asynchronous Message Reordering in Rust with Multiparty Session Types, in: 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, ACM. pp. 261–246. doi:10.1145/3503221.3508404.

[28] Demangeon, R., Honda, K., 2011. Full abstraction in a subtyped pi-calculus with linear types, in: Katoen, J., König, B. (Eds.), CONCUR 2011 - Concurrency Theory - 22nd International Conference, CONCUR 2011, Aachen, Germany, September 6-9, 2011. Proceedings, Springer. pp. 280–296. URL: https://doi.org/10.1007/978-3-642-23217-6_19, doi:10.1007/978-3-642-23217-6_19.

[29] Demangeon, R., Honda, K., Hu, R., Neykova, R., Yoshida, N., 2015. Practical Interruptible Conversations: Distributed Dynamic Verification with Multiparty Session Types and Python. Formal Methods Syst. Des. 46, 197–225. doi:10.1007/s10703-014-0218-8.

[30] Deniélou, P., Yoshida, N., 2012. Multiparty Session Types Meet Communicating Automata, in: ESOP. doi:10.1007/978-3-642-28869-2_10.

[31] Deniélou, P., Yoshida, N., Bejleri, A., Hu, R., 2012. Parameterised Multiparty Session Types. LMCS 8. doi:10.2168/LMCS-8(4:6)2012.

[32] Deniélou, P.M., Yoshida, N., 2013. Multiparty Compatibility in Communicating Automata: Characterisation and Synthesis of Global Session Types, in: 40th International Colloquium on Automata, Languages and Programming, Springer, Berlin, Heidelberg. pp. 174–186. doi:10.1007/978-3-642-39212-2_18.

[33] Dezani-Ciancaglini, M., Ghilezan, S., Jaksic, S., Pantovic, J., Yoshida, N., 2015. Precise subtyping for synchronous multiparty sessions, in: PLACES. doi:10.4204/EPTCS.203.3. Journal version: [43].

[34] Egidi, L., Giannini, P., Ventura, L., 2022. Multiparty-session-types coordination for core erlang, in: Fill, H., van Sinderen, M., Maciaszek, L.A. (Eds.), Proceedings of the 17th International Conference on Software Technologies, ICSOFT 2022, Lisbon, Portugal, July 11-13, 2022, SCITEPRESS. pp. 532–541. URL: https://doi.org/10.5220/0011316600003266, doi:10.5220/0011316600003266.

[35] Ekici, B., Kamegai, T., Yoshida, N., 2025. Formalising subject reduction and progress for multiparty session processes, in: Forster, Y., Keller, C. (Eds.), 16th International Conference on Interactive Theorem Proving, ITP 2025, Reykjavik, Iceland, September 28 - October 1, 2025, Schloss Dagstuhl - Leibniz-Zentrum für Informatik. pp. 19:1–19:23. URL: https://doi.org/10.4230/LIPIcs.ITP.2025.19, doi:10.4230/LIPICS.ITP.2025.19.

[36] Ferreira, F., Jongmans, S., 2023. Oven: Safe and live communication protocols in scala, using synthetic behavioural type analysis, in: Just, R., Fraser, G. (Eds.), Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2023, Seattle, WA, USA, July 17-21, 2023, ACM. pp. 1511–1514. URL: https://doi.org/10.1145/3597926.3604926, doi:10.1145/3597926.3604926.

[37] Fette, I., Melnikov, A., 2011. The WebSocket Protocol. RFC 6455. RFC Editor. URL: https://www.rfc-editor.org/rfc/rfc1654.txt.

[38] Fowler, S., 2016. An Erlang Implementation of Multiparty Session Actors, in: Bartoletti, M., Henrio, L., Knight, S., Vieira, H.T. (Eds.), Proceedings 9th Interaction and Concurrency Experience, ICE 2016, Heraklion, Greece, 8-9 June 2016, pp. 36–50. doi:10.4204/EPTCS.223.3.

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

[40] Gay, S.J., Hole, M., 2005. Subtyping for session types in the π-calculus. Acta Inf. 42. doi:10.1007/s00236-005-0177-z.

[41] Gay, S.J., Vasconcelos, V.T., 2010. Linear type theory for asynchronous session types. J. Funct. Program. 20, 19–50. URL: https://doi.org/10.1017/S0956796809990268, doi:10.1017/S0956796809990268.

[42] Gheri, L., Lanese, I., Sayers, N., Tuosto, E., Yoshida, N., 2022. Design-by-contract for flexible multiparty session protocols, in: Ali, K., Vitek, J. (Eds.), 36th European Conference on Object-Oriented Programming, ECOOP 2022, June 6-10, 2022, Berlin, Germany, Schloss Dagstuhl - Leibniz-Zentrum für Informatik. pp. 8:1–8:28. URL: https://doi.org/10.4230/LIPIcs.ECOOP.2022.8, doi:10.4230/LIPICS.ECOOP.2022.8.

[43] Ghilezan, S., Jaksic, S., Pantovic, J., Scalas, A., Yoshida, N., 2019. Precise subtyping for synchronous multiparty sessions. J. Log. Algebraic Methods Program. 104, 127–173. URL: https://doi.org/10.1016/j.jlamp.2018.12.002, doi:10.1016/J.JLAMP.2018.12.002.

[44] Ghilezan, S., Pantović, J., Prokić, I., Scalas, A., Yoshida, N., 2023. Precise Subtyping for Asynchronous Multiparty Sessions. ACM Transactions on Computational Logic Volume 24, Issue 2, 1–73. doi:10.1145/3568422.

[45] Giunti, M., Yoshida, N., 2025. Iso-recursive multiparty sessions and their automated verification, in: Vafeiadis, V. (Ed.), 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 I, Springer. pp. 349–378. URL: https://doi.org/10.1007/978-3-031-91118-7_14, doi:10.1007/978-3-031-91118-7_14.

[46] van Glabbeek, R., Höfner, P., Horne, R., 2021. Assuming just enough fairness to make session types complete for lock-freedom, in: 2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), pp. 1–13. doi:10.1109/LICS52264.2021.9470531.

[47] Web Services Choreography Working Group, 2003. http://w3.org/2002/ws/chor/.

[48] Harvey, P., Fowler, S., Dardha, O., Gay, S.J., 2021. Multiparty Session Types for Safe Runtime Adaptation in an Actor Language, in: Møller, A., Sridharan, M. (Eds.), 35th European Conference on Object-Oriented Programming, ECOOP

2021, July 11-17, 2021, Aarhus, Denmark (Virtual Conference), Schloss Dagstuhl - Leibniz-Zentrum für Informatik. pp. 10:1–10:30. URL: https://doi.org/10.4230/LIPIcs.ECOOP.2021.10, doi:10.4230/LIPICS.ECOOP.2021.10.

[49] van den Heuvel, B., Pérez, J.A., 2022. A decentralized analysis of multiparty protocols. Sci. Comput. Program. 222, 102840. URL: https://doi.org/10.1016/j.scico.2022.102840, doi:10.1016/J.SCICO.2022.102840.

[50] Hinrichsen, J.K., Jacobs, J., Krebbers, R., 2024. Multris: Functional verification of multiparty message passing in separation logic. Proc. ACM Program. Lang. 8, 1446–1474. URL: https://doi.org/10.1145/3689762, doi:10.1145/3689762.

[51] Honda, K., Mukhamedov, A., Brown, G., Chen, T.C., Yoshida, N., 2011. Scribbling interactions with a formal foundation, in: ICDCIT, Springer. pp. 55–75.

[52] Honda, K., Yoshida, N., Carbone, M., 2008. Multiparty asynchronous session types, in: POPL. doi:10.1145/1328438.1328472. Journal version: [53].

[53] Honda, K., Yoshida, N., Carbone, M., 2016. Multiparty Asynchronous Session Types. J. ACM 63. doi:10.1145/2827695.

[54] Hou, P., Lagaillardie, N., Yoshida, N., 2024. Fearless asynchronous communications with timed multiparty session protocols, in: Aldrich, J., Salvaneschi, G. (Eds.), 38th European Conference on Object-Oriented Programming, ECOOP 2024, September 16-20, 2024, Vienna, Austria, Schloss Dagstuhl - Leibniz-Zentrum für Informatik. pp. 19:1–19:30. URL: https://doi.org/10.4230/LIPIcs.ECOOP.2024.19, doi:10.4230/LIPICS.ECOOP.2024.19.

[55] Hu, R., Yoshida, N., 2016. Hybrid Session Verification through Endpoint API Generation, in: 19th International Conference on Fundamental Approaches to Software Engineering, Springer, Berlin, Heidelberg. pp. 401–418. doi:10.1007/978-3-662-49665-7_24.

[56] Hu, R., Yoshida, N., 2017. Explicit Connection Actions in Multiparty Session Types, in: FASE. doi:10.1007/978-3-662-54494-5_7.

[57] Imai, K., Neykova, R., Yoshida, N., Yuen, S., 2020. Multiparty Session Programming with Global Protocol Combinators, in: 34th European Conference on Object-Oriented Programming, Schloss Dagstuhl–Leibniz-Zentrum für Informatik. pp. 9:1–9:30. doi:10.4230/LIPIcs.ECOOP.2020.9.

[58] Jacobs, J., Balzer, S., Krebbers, R., 2022. Multiparty GV: functional multiparty session types with certified deadlock freedom. Proc. ACM Program. Lang. 6, 466–495. URL: https://doi.org/10.1145/3547638, doi:10.1145/3547638.

[59] Jongmans, S., 2025. Multiparty session typing, embedded, in: Gurfinkel, A., Heule, M. (Eds.), Tools and Algorithms for the Construction and Analysis of Systems - 31st International Conference, TACAS 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 I, Springer. pp. 145–164. URL: https://doi.org/10.1007/978-3-031-90643-5_8, doi:10.1007/978-3-031-90643-5_8.

[60] Jongmans, S., Ferreira, F., 2023. Synthetic behavioural typing: Sound, regular multiparty sessions via implicit local types, in: Ali, K., Salvaneschi, G. (Eds.), 37th European Conference on Object-Oriented Programming, ECOOP 2023, Seattle, Washington, United States, July 17-21, 2023, Schloss Dagstuhl - Leibniz-Zentrum für Informatik. pp. 42:1–42:30. URL: https://doi.org/10.4230/LIPIcs.ECOOP.2023.42, doi:10.4230/LIPICS.ECOOP.2023.42.

[61] Kimura, S., Imai, K., 2020. Fluent Session Programming in C#, in: Balzer, S., Padovani, L. (Eds.), Proceedings of the 12th International Workshop on Programming Language Approaches to Concurrency- and Communication-cEntric Software, PLACES@ETAPS 2020, Dublin, Ireland, 26th April 2020, pp. 61–75. URL: https://doi.org/10.4204/EPTCS.314.6, doi:10.4204/EPTCS.314.6.

[62] King, J., Ng, N., Yoshida, N., 2019. Multiparty Session Type-safe Web Development with Static Linearity, in: Martins, F., Orchard, D. (Eds.), Proceedings Programming Language Approaches to Concurrency- and Communication-cEntric Software, PLACES@ETAPS 2019, Prague, Czech Republic, 7th April 2019, pp. 35–46. URL: https://doi.org/10.4204/EPTCS.291.4, doi:10.4204/EPTCS.291.4.

[63] Kobayashi, N., Sangiorgi, D., 2010. A hybrid type system for lock-freedom of mobile processes. TOPLAS 32. doi:10.1145/1745312.1745313.

[64] Kouzapas, D., Dardha, O., Perera, R., Gay, S.J., 2018. Typechecking protocols with Mungo and StMungo: A session type toolchain for Java. Sci. Comput. Program. 155, 52–75. URL: https://doi.org/10.1016/j.scico.2017.10.006, doi:10.1016/J.SCICO.2017.10.006.

[65] Lagaillardie, N., Neykova, R., Yoshida, N., 2022. Stay safe under panic: Affine rust programming with multiparty session types, in: Ali, K., Vitek, J. (Eds.), 36th European Conference on Object-Oriented Programming, ECOOP 2022, June 6-10, 2022, Berlin, Germany, Schloss Dagstuhl - Leibniz-Zentrum für Informatik. pp. 4:1–4:29. URL: https://doi.org/10.4230/LIPIcs.ECOOP.2022.4, doi:10.4230/LIPICS.ECOOP.2022.4.

[66] Li, E., Stutz, F., Wies, T., Zufferey, D., 2023. Complete multiparty session type projection with automata, in: Enea, C., Lal, A. (Eds.), Computer Aided Verification - 35th International Conference, CAV 2023, Paris, France, July 17-22, 2023, Proceedings, Part III, Springer. pp. 350–373. URL: https://doi.org/10.1007/978-3-031-37709-9_17, doi:10.1007/978-3-031-37709-9_17.

[67] Li, E., Wies, T., 2025. Certified implementability of global multiparty protocols, in: Forster, Y., Keller, C. (Eds.), 16th International Conference on Interactive Theorem Proving, ITP 2025, Reykjavik, Iceland, September 28 - October 1, 2025, Schloss Dagstuhl - Leibniz-Zentrum für Informatik. pp. 15:1–15:20. URL: https://doi.org/10.4230/LIPIcs.ITP.2025.15, doi:10.4230/LIPICS.ITP.2025.15.

[68] Miu, A., Ferreira, F., Yoshida, N., Zhou, F., 2021. Communication-Safe Web Programming in TypeScript with Routed Multiparty Session Types, in: International Conference on Compiler Construction, pp. 94–106. doi:10.1145/3446804.3446854.

[69] Mostrous, D., Yoshida, N., 2009. Session-based communication optimisation for higher-order mobile processes, in: Curien, P. (Ed.), Typed Lambda Calculi and Applications, 9th International Conference, TLCA 2009, Brasilia, Brazil, July 1-

3, 2009. Proceedings, Springer. pp. 203–218. URL: https://doi.org/10.1007/978-3-642-02273-9_16, doi:10.1007/978-3-642-02273-9_16.
[70] Neykova, R., Hu, R., Yoshida, N., Abdeljallal, F., 2018. A Session Type Provider: Compile-time API Generation for Distributed Protocols with Interaction Refinements in F#, in: 27th International Conference on Compiler Construction, ACM. pp. 128–138. doi:10.1145/3178372.3179495.
[71] Neykova, R., Yoshida, N., 2017a. Let It Recover: Multiparty Protocol-Induced Recovery, in: CC. doi:10.1145/3033019.3033031.
[72] Neykova, R., Yoshida, N., 2017b. Multiparty Session Actors. Logical Methods in Computer Science 13, 1–30. doi:10.23638/LMCS-13(1:17)2017.
[73] Neykova, R., Yoshida, N., Hu, R., 2013. SPY: local verification of global protocols, in: Legay, A., Bensalem, S. (Eds.), Runtime Verification - 4th International Conference, RV 2013, Rennes, France, September 24-27, 2013. Proceedings, Springer. pp. 358–363. URL: https://doi.org/10.1007/978-3-642-40787-1_25, doi:10.1007/978-3-642-40787-1_25.
[74] Ng, N., Coutinho, J.G., Yoshida, N., 2015. Protocols by Default: Safe MPI Code Generation based on Session Types, in: 24th International Conference on Compiler Construction, Springer. pp. 212–232. doi:10.1007/978-3-662-46663-6_11.
[75] Ng, N., Yoshida, N., Honda, K., 2012a. Multiparty Session C: Safe Parallel Programming with Message Optimisation, in: 50th International Conference on Objects, Models, Components, Patterns, Springer. pp. 202–218. doi:10.1007/978-3-642-30561-0_15.
[76] Ng, N., Yoshida, N., Niu, X., Tsoi, K.H., Luk, W., 2012b. Session Types: Towards safe and fast reconfigurable programming. ACM SIGARCH Computer Architecture News 40, 22–27. doi:10.1145/2460216.2460221.
[77] OAuth Working Group, 2012. RFC 6749: OAuth 2.0 framework. http://tools.ietf.org/html/rfc6749.
[78] Padovani, L., 2014. Deadlock and lock freedom in the linear π-calculus, in: CSL-LICS. doi:10.1145/2603088.2603116.
[79] Padovani, L., Vasconcelos, V.T., Vieira, H.T., 2014. Typing liveness in multiparty communicating systems, in: Kühn, E., Pugliese, R. (Eds.), Coordination Models and Languages - 16th IFIP WG 6.1 International Conference, COORDINATION 2014, Held as Part of the 9th International Federated Conferences on Distributed Computing Techniques, DisCoTec 2014, Berlin, Germany, June 3-5, 2014, Proceedings, Springer. pp. 147–162. URL: https://doi.org/10.1007/978-3-662-43376-8_10, doi:10.1007/978-3-662-43376-8_10.
[80] Pierce, B.C., 2002. Types and Programming Languages. 1st ed., The MIT Press.
[81] Sangiorgi, D., 2011. Introduction to Bisimulation and Coinduction. Cambridge University Press. doi:10.1017/CBO9780511777110.
[82] Scalas, A., Dardha, O., Hu, R., Yoshida, N., 2017. A Linear Decomposition of Multiparty Sessions for Safe Distributed Programming, in: ECOOP. doi:10.4230/LIPIcs.ECOOP.2017.24.
[83] Scalas, A., Yoshida, N., 2018. Multiparty session types, beyond duality. doi:10.1016/j.jlamp.2018.01.001.
[84] Scalas, A., Yoshida, N., 2019. Less is More: Multiparty Session Types Revisited. Proc. ACM Program. Lang. 3, 30:1–30:29. doi:10.1145/3290343.
[85] Stutz, F., 2023. Asynchronous multiparty session type implementability is decidable - lessons learned from message sequence charts, in: Ali, K., Salvaneschi, G. (Eds.), 37th European Conference on Object-Oriented Programming, ECOOP 2023, Seattle, Washington, United States, July 17-21, 2023, Schloss Dagstuhl - Leibniz-Zentrum für Informatik. pp. 32:1–32:31. URL: https://doi.org/10.4230/LIPIcs.ECOOP.2023.32, doi:10.4230/LIPICS.ECOOP.2023.32.
[86] Stutz, F., D'Osualdo, E., 2025. An automata-theoretic basis for specification and type checking of multiparty protocols, in: Vafeiadis, V. (Ed.), 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, Springer. pp. 314–346. URL: https://doi.org/10.1007/978-3-031-91121-7_13, doi:10.1007/978-3-031-91121-7_13.
[87] Tassarotti, J., Jung, R., Harper, R., 2017. A higher-order logic for concurrent termination-preserving refinement, in: Yang, H. (Ed.), Programming Languages and Systems - 26th European Symposium on Programming, ESOP 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Springer. pp. 909–936. URL: https://doi.org/10.1007/978-3-662-54434-1_34, doi:10.1007/978-3-662-54434-1_34.
[88] Tirore, D.L., 2024. A Mechanisation of Multiparty Session Types. Ph.D. thesis. ITU, Copenhagen.
[89] Tirore, D.L., Bengtson, J., Carbone, M., 2023. A sound and complete projection for global types, in: Naumowicz, A., Thiemann, R. (Eds.), 14th International Conference on Interactive Theorem Proving, ITP 2023, July 31 to August 4, 2023, Białystok, Poland, Schloss Dagstuhl - Leibniz-Zentrum für Informatik. pp. 28:1–28:19. URL: https://doi.org/10.4230/LIPIcs.ITP.2023.28, doi:10.4230/LIPICS.ITP.2023.28.
[90] Tirore, D.L., Bengtson, J., Carbone, M., 2025a. Multiparty asynchronous session types: A mechanised proof of subject reduction, in: Aldrich, J., Silva, A. (Eds.), 39th European Conference on Object-Oriented Programming, ECOOP 2025, June 30 to July 2, 2025, Bergen, Norway, Schloss Dagstuhl - Leibniz-Zentrum für Informatik. pp. 31:1–31:30. URL: https://doi.org/10.4230/LIPIcs.ECOOP.2025.31, doi:10.4230/LIPICS.ECOOP.2025.31.
[91] Tirore, D.L., Bengtson, J., Carbone, M., 2025b. A sound and complete projection for global types. J. Autom. Reason. 69, 14. URL: https://doi.org/10.1007/s10817-025-09726-9, doi:10.1007/S10817-025-09726-9.
[92] Toninho, B., Yoshida, N., 2016. Certifying Data in Multiparty Session Types, in: A List of Successes That Can Change the World: Essays Dedicated to Philip Wadler on the Occasion of His 60th Birthday. doi:10.1007/978-3-319-30936-1_23.
[93] Toninho, B., Yoshida, N., 2017. Certifying data in multiparty session types. JLAMP 90. doi:10.1016/j.jlamp.2016.11.005.
[94] Udomsrirungruang, T., Yoshida, N., 2025. Top-Down or Bottom-Up? Complexity Analyses of Synchronous Multiparty Session Types. Proc. ACM Program. Lang. 9, 1040–1071. URL: https://doi.org/10.1145/3704872, doi:10.1145/

3704872.

[95] Vassor, M., Yoshida, N., 2024. Refinements for multiparty message-passing protocols: Specification-agnostic theory and implementation, in: Aldrich, J., Salvaneschi, G. (Eds.), 38th European Conference on Object-Oriented Programming, ECOOP 2024, September 16-20, 2024, Vienna, Austria, Schloss Dagstuhl - Leibniz-Zentrum für Informatik. pp. 41:1–41:29. URL: https://doi.org/10.4230/LIPIcs.ECOOP.2024.41, doi:10.4230/LIPICS.ECOOP.2024.41.

[96] Viering, M., Hu, R., Eugster, P., Ziarek, L., 2021. A Multiparty Session Typing Discipline for Fault-Tolerant Event-Driven Distributed Programming. Proc. ACM Program. Lang. 5. URL: https://doi.org/10.1145/3485501, doi:10.1145/3485501.

[97] Wadler, P., 2012. Propositions as sessions, in: ICFP. doi:10.1145/2364527.2364568.

[98] Yoshida, N., 2024. Programming language implementations with multiparty session types, in: de Boer, F.S., Damiani, F., Hähnle, R., Johnsen, E.B., Kamburjan, E. (Eds.), Active Object Languages: Current Research Trends. Springer. volume 14360 of *Lecture Notes in Computer Science*, pp. 147–165. URL: https://doi.org/10.1007/978-3-031-51060-1_6, doi:10.1007/978-3-031-51060-1_6.

[99] Yoshida, N., Deniélou, P., Bejleri, A., Hu, R., 2010. Parameterised Multiparty Session Types, in: FOSSACS. doi:10.1007/978-3-642-12032-9_10. Journal version: [31].

[100] Yoshida, N., Gheri, L., 2020. A very gentle introduction to multiparty session types, in: Hung, D.V., D'Souza, M. (Eds.), Distributed Computing and Internet Technology - 16th International Conference, ICDCIT 2020, Bhubaneswar, India, January 9-12, 2020, Proceedings, Springer. pp. 73–93. doi:10.1007/978-3-030-36987-3_5.

[101] Yoshida, N., Hou, P., 2024. Less is more revisited: Association with global multiparty session types, in: Cavalcanti, A., Baxter, J. (Eds.), The Practice of Formal Methods: Essays in Honour of Cliff Jones, Part II. Springer. volume 14781 of *Lecture Notes in Computer Science*, pp. 268–291. URL: https://doi.org/10.1007/978-3-031-66673-5_14, doi:10.1007/978-3-031-66673-5_14.

[102] Yoshida, N., Hu, R., Neykova, R., Ng, N., 2013. The Scribble Protocol Language, in: Trustworthy Global Computing, Springer, Buenos Aires, Argentina. pp. 22–41.

[103] Yoshida, N., Zhou, F., Ferreira, F., 2021. Communicating Finite State Machines and an Extensible Toolchain for Multiparty Session Types, in: 23rd International Symposium on Fundamentals of Computation Theory, Springer. pp. 18–35. doi:10.1007/978-3-030-86593-1_2.

[104] Zhou, F., Ferreira, F., Hu, R., Neykova, R., Yoshida, N., 2020. Statically Verified Refinements for Multiparty Protocols, in: OOPSLA 2020: Conference on Object-Oriented Programming Systems, Languages and Applications, ACM. pp. 148:1–148:30. doi:10.1145/3428216.

A. Proofs for § 3

A.1. Roles

Definition A.1 (Roles in Global Types). The set of roles in a global type G, denoted by $\text{roles}(G)$, is defined inductively as:

$$\text{roles}(\mathtt{p}\to\mathtt{q}\colon\{\mathtt{m_i}(B_i).G_i\}_{i\in I}) = \{\mathtt{p},\mathtt{q}\} \cup \bigcup_{i\in I} \text{roles}(G_i)$$
$$\text{roles}(\mathbf{end}) = \text{roles}(\mathbf{t}) = \emptyset \qquad \text{roles}(\mu\mathbf{t}.G) = \text{roles}(G\{^{\mu\mathbf{t}.G}/_{\mathbf{t}}\})$$

A.2. Unfolding

Definition A.2 (Type Unfolding). The *unfolding* of a global type G, written $\text{unf}(G)$, is defined as:

$$\text{unf}(\mu\mathbf{t}.G) = G\{^{\mu\mathbf{t}.G}/_{\mathbf{t}}\} \qquad \text{unf}(G) = G \quad \text{if } G \neq \mu\mathbf{t}.G'$$

The *unfolding* of a local type T, written $\text{unf}(T)$, is defined similarly:

$$\text{unf}(\mu\mathbf{t}.T) = T\{^{\mu\mathbf{t}.T}/_{\mathbf{t}}\} \qquad \text{unf}(T) = T \quad \text{if } T \neq \mu\mathbf{t}.T'$$

Proposition A.1. *For a closed, well-guarded global type G, $\text{unf}(G)$ can only be of form $\mathbf{end}$, or $\mathtt{p}\to\mathtt{q}\colon\{\cdots\}$. For a closed, well-guarded local type T, $\text{unf}(T)$ can only be of form $\mathbf{end}$, $\mathtt{p}\oplus\{\cdots\}$, or $\mathtt{p}\&\{\cdots\}$.*

Proof. $\mathbf{t}$ will not appear since we require closed types. $\mu\mathbf{t}.G'\{^{\mu\mathbf{t}.G'}/_{\mathbf{t}}\} \neq \mu\mathbf{t}.G'$ since we require well-guarded types (recursive types are contractive). Similar argument for local types. □

Proposition A.2. $\text{roles}(G) = \text{roles}(\text{unf}(G))$.

Proof. Follows directly from Defs. A.1 and A.2. □

A.3. Subtyping

Lemma A.3 (Subtyping is Reflexive)**.** *For any closed, well-guarded local type T, $T \leqslant T$ holds.*

Proof. We construct a relation $R = \{(T, T)\}$. It is trivial to show that R satisfies all clauses of Def. 3.2, and hence, $R \subseteq \leqslant$. □

Lemma A.4 (Subtyping is Transitive)**.** *For any closed, well-guarded local types T_1, T_2, T_3, if $T_1 \leqslant T_2$ and $T_2 \leqslant T_3$, then $T_1 \leqslant T_3$ holds.*

Proof. By constructing a relation $R = \{(T_1, T_3) \mid \exists T_2 \text{ such that } T_1 \leqslant T_2 \text{ and } T_2 \leqslant T_3\}$, and showing that $R \subseteq \leqslant$. □

Lemma A.5 (Subtyping with Unfolding)**.** *For any closed, well-guarded local type T, (1) $\text{unf}(T) \leqslant T$, and (2) $T \leqslant \text{unf}(T)$.*

Proof. (1) By [Sub-μR] if $T = \mu\mathbf{t}.T'$. Otherwise, by reflexivity (Lem. A.3).

(2) By [Sub-μL] if $T = \mu\mathbf{t}.T'$. Otherwise, by reflexivity (Lem. A.3). □

Lemma A.6 (Inversion of Subtyping)**.**

1. *If $\mathtt{p}\oplus\{\mathtt{m_i}(S_i).T_i\}_{i\in I} \leqslant U$, then $\text{unf}(U) = \mathtt{p}\oplus\{\mathtt{m'_j}(S'_j).T'_j\}_{j\in J}$, and $I \subseteq J$, and $\forall i \in I : \mathtt{m}_i = \mathtt{m'}_i, S'_i \leqslant S_i$ and $T_i \leqslant T'_i$.*
2. *If $\mathtt{p}\&\{\mathtt{m_i}(S_i).T_i\}_{i\in I} \leqslant U$, then $\text{unf}(U) = \mathtt{p}\&\{\mathtt{m'_j}(S'_j).T'_j\}_{j\in J}$, and $J \subseteq I$, and $\forall i \in J : \mathtt{m}_i = \mathtt{m'}_i, S_i \leqslant S'_i$ and $T_i \leqslant T'_i$.*
3. *If $U \leqslant \mathtt{p}\oplus\{\mathtt{m_i}(S_i).T_i\}_{i\in I}$, then $\text{unf}(U) = \mathtt{p}\oplus\{\mathtt{m'_j}(S'_j).T'_j\}_{j\in J}$, and $J \subseteq I$, and $\forall i \in I : \mathtt{m}_i = \mathtt{m'}_i, S_i \leqslant S'_i$ and $T'_i \leqslant T_i$.*

4. *If* $U \leqslant \mathtt{p}\&\{\mathtt{m_i}(S_i).T_i\}_{i\in I}$, *then* $\mathrm{unf}(U) = \mathtt{p}\&\{\mathtt{m'}_j(S'_j).T'_j\}_{j\in J}$, *and* $I \subseteq J$, *and* $\forall i \in J : \mathtt{m}_i = \mathtt{m'}_i, S'_i \leqslant S_i$ *and* $T'_i \leqslant T_i$.

Proof. By Lem. A.5, the transitivity of subtyping (Lem. A.4), Prop. A.1, and the definition of subtyping (Def. 3.2), in particular rules [Sub-&] and [Sub-⊕]. □

Lemma A.7. *Given a collection of mergeable local types* T_i $(i \in I)$. *For all* $j \in I$, $\bigsqcap_{i\in I} T_i \leqslant T_j$ *holds.*

Proof. By constructing a relation $R = \{(\bigsqcap_{i\in I} T_i, T_j) \mid j \in I\}$, and showing that $R \subseteq \leqslant$. □

Lemma A.8. *Given a collection of mergeable local types* T_i $(i \in I)$. *If for all* $i \in I$, $T_i \leqslant U$ *for some local type* U, *then* $\bigsqcap_{i\in I} T_i \leqslant U$.

Proof. By constructing a relation $R = \{(\bigsqcap_{i\in I} T_i, U)\}$, and showing that $R \subseteq \leqslant$. □

Lemma A.9. *Given a collection of mergeable local types* T_i $(i \in I)$. *If for all* $i \in I$, $U \leqslant T_i$ *for some local type* U, *then* $U \leqslant \bigsqcap_{i\in I} T_i$.

Proof. By constructing a relation $R = \{(U, \bigsqcap_{i\in I} T_i)\}$, and showing that $R \subseteq \leqslant$. (As $U \leqslant T_i$ holds for all $i \in I$, U contains every needed external choice. For the external choices holds that they need to be the same in all T_i and hence U has a subset of them.) □

Lemma A.10. *Given two collections of mergeable local types* S_i, T_i $(i \in I)$. *If for all* $i \in I$, $S_i \leqslant T_i$, *then* $\bigsqcap_{i\in I} S_i \leqslant \bigsqcap_{i\in I} T_i$.

Proof. By constructing a relation $R = \{(\bigsqcap_{i\in I} S_i, \bigsqcap_{i\in I} T_i)\}$, and showing that $R \subseteq \leqslant$. □

Lemma A.11 (Subtyping of Projection with Unfolding)**.** *For any closed, well-guarded global type* G *and role* $\mathtt{p}$, $\mathrm{unf}(G)\!\restriction\!\mathtt{p} \leqslant G\!\restriction\!\mathtt{p}$ *and* $G\!\restriction\!\mathtt{p} \leqslant \mathrm{unf}(G)\!\restriction\!\mathtt{p}$.

Proof. We proceed by cases on G.

- Case $G \neq \mu\mathbf{t}.G'$: by Def. A.2, $\mathrm{unf}(G) = G$. The thesis follows by reflexivity of subtyping (Lem. A.3).
- Case $G = \mu\mathbf{t}.G'$: by Def. A.2, $\mathrm{unf}(G) = G'\{^{\mu\mathbf{t}.G'}/_{\mathbf{t}}\}$. The proof follows by induction on the structure of G'.
  - Case $G' = \mathbf{end}$: by the definition of projection (Def. 3.1) and the subtyping rule [Sub-**end**].
  - Case $G' = \mathtt{q}\to\mathtt{r}: \{\mathtt{m_i}(S_i).G''_i\}_{i\in I}$: subcases are considered on $\mathtt{p}$.
    * Subcase $\mathtt{p} = \mathtt{q}$ or $\mathtt{p} = \mathtt{r}$: by the definition of projection (Def. 3.1), the induction hypothesis, and the definition of subtyping (Def. 3.2).
    * $\mathtt{p} \neq \mathtt{q}$ and $\mathtt{p} \neq \mathtt{r}$: by the definition of projection (Def. 3.1), the induction hypothesis, the definition of subtyping (Def. 3.2), and Lem. A.10. □

A.4. Properties of Global Type and Typing Context Transitions

Lemma A.12. $G \xrightarrow{\alpha} G'$ *iff* $\mathrm{unf}(G) \xrightarrow{\alpha} G'$.

Proof. By inverting or applying [GR-μ] when necessary. □

Lemma A.13 (Progress of Global Types)**.** *If* $G \neq \mathbf{end}$ *(where* G *is a projectable global type), then there exists* G' *such that* $G \to G'$.

Proof. By Lem. A.12, we only consider unfoldings.

- Case $\mathrm{unf}(G) = \mathbf{end}$: the premise does not hold;
- Case $\mathrm{unf}(G) = \mathtt{p}\to\mathtt{q}: \{\mathtt{m_i}(S_i).G_i\}_{i\in I}$: apply [GR-⊕&] to reduce the global type. □

Lemma A.14. *If* $\mathtt{p} \notin \mathrm{roles}(G)$*, then* $G \upharpoonright \mathtt{p} = \mathbf{end}$.

Proof. By induction on the structure of G:

- Case $G = \mathbf{end}$: trivial as $\mathbf{end} \upharpoonright \mathtt{p} = \mathbf{end}$.
- Case $G = \mathtt{q} \to \mathtt{r} \colon \{\mathtt{m_i}(S_i).G_i\}_{i \in I}$: we have $\mathtt{p} \neq \mathtt{q}$, $\mathtt{p} \neq \mathtt{r}$, and $\forall i \in I : \mathtt{p} \notin \mathrm{roles}(G_i)$. The thesis holds by $G \upharpoonright \mathtt{p} = \bigsqcap_{i \in I} G_i \upharpoonright \mathtt{p}$ and the induction hypothesis that $\forall i \in I : G_i \upharpoonright \mathtt{p} = \mathbf{end}$.
- Case $G = \mu \mathbf{t}.G'$: assume $\mathrm{fv}(\mu \mathbf{t}.G') \neq \emptyset$. By induction hypothesis, $G \upharpoonright \mathtt{p} = \mu \mathbf{t}.G' \upharpoonright \mathtt{p} = \mu \mathbf{t}.\mathbf{end}$, which is an unguarded recursion – a contradiction. Therefore, the thesis holds directly, as $\mu \mathbf{t}.G' \upharpoonright \mathtt{p} = \mathbf{end}$. □

Lemma A.15. *If* $\Delta \xrightarrow{\alpha} \Delta'$*, then* $\mathrm{dom}(\Delta) = \mathrm{dom}(\Delta')$.

Proof. By induction on typing context transitions. □

Lemma A.16. *If* $\Delta \xrightarrow{\alpha} \Delta'$ *and* $\mathrm{dom}(\Delta) = \{s\}$*, then for all* $s[\mathtt{p}] \in \mathrm{dom}(\Delta)$ *with* $\mathtt{p} \notin \mathrm{subject}(\alpha)$*,* $\Delta(s[\mathtt{p}]) = \Delta'(s[\mathtt{p}])$.

Proof. By induction on typing context transitions. □

Lemma A.17 (Inversion of Typing Context Transition)**.**

1. *If* $\Delta \xrightarrow{s[\mathtt{p}]:\mathtt{q}\oplus\mathtt{m_k}(S_k)} \Delta'$*, then* $\mathrm{unf}(\Delta(s[\mathtt{p}])) = \mathtt{q}\oplus\{\mathtt{m_i}(S_i).T_i\}_{i \in I}$*,* $k \in I$*, and* $\Delta'(s[\mathtt{p}]) = T_k$*;*
2. *If* $\Delta \xrightarrow{s[\mathtt{q}]:\mathtt{p}\&\mathtt{m_k}(S_k)} \Delta'$*, then* $\mathrm{unf}(\Delta(s[\mathtt{q}])) = \mathtt{p}\&\{\mathtt{m_i}(S_i).T_i\}_{i \in I}$*,* $k \in I$*, and* $\Delta'(s[\mathtt{q}]) = T_k$.
3. *If* $\Delta \xrightarrow{s[\mathtt{p}][\mathtt{q}]\mathtt{m}} \Delta'$*, then* $\mathrm{unf}(\Delta(s[\mathtt{p}])) = \mathtt{q}\oplus\{\mathtt{m_i}(S_i).T_i\}_{i \in I}$*,* $\mathrm{unf}(\Delta(s[\mathtt{q}])) = \mathtt{p}\&\{\mathtt{m'_j}(S'_j).T'_j\}_{j \in J}$*, and there exist* $k \in I$ *and* $l \in J$ *such that* $\mathtt{m}_k = \mathtt{m}'_l = \mathtt{m}$*,* $S'_l \leqslant S_k$*, with* $\Delta'(s[\mathtt{p}]) = T_k$ *and* $\Delta'(s[\mathtt{q}]) = T'_l$.

Proof. By applying and inverting [Δ-$\oplus$], [Δ-$\&$], and [Δ-μ] (when necessary). □

Lemma A.18 (Determinism of Typing Context Transition)**.** *If* $\Delta \xrightarrow{\alpha} \Delta'$ *and* $\Delta \xrightarrow{\alpha} \Delta''$*, then* $\Delta' = \Delta''$.

Proof. By induction on typing context transitions. □

Lemma A.19.

1. *If* $\Delta \xrightarrow{s[\mathtt{p}]:\mathtt{q}\oplus\mathtt{m_k}(S_k)} \Delta'$*, then for any channel with role* $c \in \mathrm{dom}(\Delta)$ *with* $c \neq s[\mathtt{p}]$*,* $\Delta(c) = \Delta'(c)$.
2. *If* $\Delta \xrightarrow{s[\mathtt{q}]:\mathtt{p}\&\mathtt{m_k}(S_k)} \Delta'$*, then for any channel with role* $c \in \mathrm{dom}(\Delta)$ *with* $c \neq s[\mathtt{q}]$*,* $\Delta(c) = \Delta'(c)$.
3. *If* $\Delta \xrightarrow{s[\mathtt{p}][\mathtt{q}]\mathtt{m}} \Delta'$*, then for any channel with role* $c \in \mathrm{dom}(\Delta)$ *with* $c \neq s[\mathtt{p}]$ *and* $c \neq s[\mathtt{q}]$*,* $\Delta(c) = \Delta'(c)$.

Proof. By induction on typing context transitions. □

A.5. Properties of Association

Lemma A.20. $T \leqslant G \upharpoonright \mathtt{p}$ *if and only if* $T \leqslant \mathrm{unf}(G) \upharpoonright \mathtt{p}$

Proof. Follows directly from Lem. A.11 and transitivity of subtyping (Lem. A.4). □

Proposition A.21. $\Delta \sqsubseteq_s G$ *if and only if* $\Delta \sqsubseteq_s \mathrm{unf}(G)$.

Proof. Direct from Def. 3.7, Prop. A.2, and Lem. A.20. □

Lemma A.22 (Relating Terminations)**.** *If* $G = \mathbf{end}$ *and* $\Delta \sqsubseteq_s G$*, then* $\forall s[\mathtt{p}] \in \mathrm{dom}(\Delta) : \Delta(s[\mathtt{p}]) = \mathbf{end}$.

Proof. By the definition of association (Def. 3.7), we know that $\Delta = \Delta_G, \Delta_{\mathbf{end}}$, where, by the hypothesis $G = \mathbf{end}$, $\mathrm{dom}(\Delta_G) = \emptyset$. Hence, $\Delta = \Delta_{\mathbf{end}}$, which is the thesis. □

Lemma A.23 (Inversion of Projection). *Given a local type T, which is a subtype of projection from a global type G on a role $\mathtt{p}$, i.e. $T \leqslant G\!\upharpoonright\!\mathtt{p}$, then:*

1. *If* $\mathrm{unf}(T) = \mathtt{q}\oplus\{\mathtt{m_i}(S_i).T_i\}_{i\in I}$, *then either*
   (a) $\mathrm{unf}(G) = \mathtt{p}\to\mathtt{q}\colon\{\mathtt{m}'_\mathtt{i}(S'_i).G_i\}_{i\in I'}$, *where $I \subseteq I'$, and for all $i \in I$: $\mathtt{m}_i = \mathtt{m}'_i$, $S'_i \leqslant S_i$, and $T_i \leqslant G_i\!\upharpoonright\!\mathtt{p}$; or,*
   (b) $\mathrm{unf}(G) = \mathtt{s}\to\mathtt{t}\colon\{\mathtt{m}'_\mathtt{j}(S'_j).G_j\}_{j\in J}$, *where for all $j \in J$: $T \leqslant G_j\!\upharpoonright\!\mathtt{p}$, with $\mathtt{p} \neq \mathtt{s}$ and $\mathtt{p} \neq \mathtt{t}$.*
2. *If* $\mathrm{unf}(T) = \mathtt{q}\&\{\mathtt{m_i}(S_i).T_i\}_{i\in I}$, *then either*
   (a) $\mathrm{unf}(G) = \mathtt{q}\to\mathtt{p}\colon\{\mathtt{m}'_\mathtt{i}(S'_i).G_i\}_{i\in I'}$, *where $I' \subseteq I$, and for all $i \in I'$: $\mathtt{m}_i = \mathtt{m}'_i$, $S_i \leqslant S'_i$, and $T_i \leqslant G_i\!\upharpoonright\!\mathtt{p}$; or,*
   (b) $\mathrm{unf}(G) = \mathtt{s}\to\mathtt{t}\colon\{\mathtt{m}'_\mathtt{j}(S'_j).G_j\}_{j\in J}$, *where for all $j \in J$: $T \leqslant G_j\!\upharpoonright\!\mathtt{p}$, with $\mathtt{p} \neq \mathtt{s}$ and $\mathtt{p} \neq \mathtt{t}$.*
3. *If* $\mathrm{unf}(T) = \mathbf{end}$, *then* $\mathtt{p} \notin \mathrm{roles}(G)$.

Proof. By Lems. A.5 and A.20, transitivity of subtyping (Lem. A.4), and the definition of global type projection (Def. 3.1). For the subcases (b) in items (1) and (2), apply Lem. A.7 additionally. □

Lemma A.24 (Matching Communication Under Projection). *If two local types T, U are subtypes of an internal choice and an external choice with matching roles, obtained via projection from a global type G, i.e.* $\mathrm{unf}(T) = \mathtt{q}\oplus\{\mathtt{m_i}(S_i).T_i\}_{i\in I_\mathtt{p}} \leqslant G\!\upharpoonright\!\mathtt{p}$ *and* $\mathrm{unf}(U) = \mathtt{p}\&\{\mathtt{m'}_j(S'_j).T'_j\}_{j\in I_\mathtt{q}} \leqslant G\!\upharpoonright\!\mathtt{q}$, *then $I_\mathtt{p} \subseteq I_\mathtt{q}$, and $\forall i \in I_\mathtt{p} : \mathtt{m}_i = \mathtt{m'}_i$ and $S'_i \leqslant S_i$.*

Proof. By induction on items (1) and (2) of Lem. A.23 simultaneously.

(a) We have $\mathrm{unf}(G) = \mathtt{p}\to\mathtt{q}\colon\{\mathtt{m}''_\mathtt{i}(S''_i).G_i\}_{i\in I}$, $I_\mathtt{p} \subseteq I \subseteq I_\mathtt{q}$, $\forall i \in I_\mathtt{p} : \mathtt{m}_i = \mathtt{m}''_i$ and $S''_i \leqslant S_i$, and $\forall i \in I$: $\mathtt{m}''_i = \mathtt{m'}_i$ and $S'_i \leqslant S''_i$. We have $I_\mathtt{p} \subseteq I_\mathtt{q}$ (by transitivity of $\subseteq$), and $\forall i \in I_\mathtt{p} : \mathtt{m}_i = \mathtt{m'}_i$ (by transitivity of $=$) and $S'_i \leqslant S_i$ (by transitivity of $\leqslant$).

(b) We have $\mathrm{unf}(G) = \mathtt{s}\to\mathtt{t}\colon\{\mathtt{m}''_\mathtt{j}(S''_j).G_j\}_{j\in J}$, where for all $j \in J$: $T \leqslant G_j\!\upharpoonright\!\mathtt{p}$, $U \leqslant G_j\!\upharpoonright\!\mathtt{q}$, $\{\mathtt{p},\mathtt{q}\}\cap\{\mathtt{s},\mathtt{t}\} = \emptyset$. Apply induction on $T \leqslant G_j\!\upharpoonright\!\mathtt{p}$ and $U \leqslant G_j\!\upharpoonright\!\mathtt{p}$ on any $j \in J$. □

Lemma A.25 (Inversion of Association). *Let $\Delta \sqsubseteq_s G$.*

1. *If* $\mathrm{unf}(\Delta(s[\mathtt{p}])) = \mathtt{q}\oplus\{\mathtt{m_i}(S_i).T_i\}_{i\in I}$, *then either*
   (a) $\mathrm{unf}(G) = \mathtt{p}\to\mathtt{q}\colon\{\mathtt{m_i}(S'_i).G_i\}_{i\in I'}$, *where $I \subseteq I'$, and for all $i \in I$: $\mathtt{m}_i = \mathtt{m}_i$, $S'_i \leqslant S_i$, and $T_i \leqslant G_i\!\upharpoonright\!\mathtt{p}$; or,*
   (b) $\mathrm{unf}(G) = \mathtt{s}\to\mathtt{t}\colon\{\mathtt{m_j}(S'_j).G_j\}_{j\in J}$, *where for all $j \in J$: $\Delta(s[\mathtt{p}]) \leqslant G_j\!\upharpoonright\!\mathtt{p}$, with $\mathtt{p} \neq \mathtt{s}$ and $\mathtt{p} \neq \mathtt{t}$.*
2. *If* $\mathrm{unf}(\Delta(s[\mathtt{p}])) = \mathtt{q}\&\{\mathtt{m_i}(S_i).T_i\}_{i\in I}$, *then either*
   (a) $\mathrm{unf}(G) = \mathtt{q}\to\mathtt{p}\colon\{\mathtt{m_i}(S'_i).G_i\}_{i\in I'}$, *where $I' \subseteq I$, and for all $i \in I'$: $\mathtt{m}_i = \mathtt{m}_i$, $S_i \leqslant S'_i$, and $T_i \leqslant G_i\!\upharpoonright\!\mathtt{p}$; or,*
   (b) $\mathrm{unf}(G) = \mathtt{s}\to\mathtt{t}\colon\{\mathtt{m_j}(S'_j).G_j\}_{j\in J}$, *where for all $j \in J$: $\Delta(s[\mathtt{p}]) \leqslant G_j\!\upharpoonright\!\mathtt{p}$, with $\mathtt{p} \neq \mathtt{s}$ and $\mathtt{p} \neq \mathtt{t}$.*
3. *If* $\mathrm{unf}(\Delta(s[\mathtt{p}])) = \mathbf{end}$, *then* $\mathtt{p} \notin \mathrm{roles}(G)$.

Proof. Follows directly from Def. 3.7 and Lem. A.23. □

Lemma A.26 (Simultaneous Inversions of Association). *Let* $\Delta \sqsubseteq_s G$. *If* $\mathrm{unf}(\Delta(s[\mathtt{p}])) = \mathtt{q}\oplus\{\mathtt{m_i}(S_i).T_i\}_{i\in I_{\mathtt{p}}}$ *and* $\mathrm{unf}(\Delta(s[\mathtt{q}])) = \mathtt{p}\&\{\mathtt{m_i}(S'_i).T'_i\}_{i\in I_{\mathtt{q}}}$*, then either*

1. $\mathrm{unf}(G) = \mathtt{p}\to\mathtt{q}\colon\{\mathtt{m_i}(S''_i).G_i\}_{i\in I}$*, where* $I_{\mathtt{p}} \subseteq I \subseteq I_{\mathtt{q}}$, $\forall i \in I_{\mathtt{p}} : S''_i \leqslant S_i$, $\forall i \in I : S'_i \leqslant S''_i$, $\forall i \in I_{\mathtt{p}} : T_i \leqslant G_i\!\restriction\!\mathtt{p}$*, and* $\forall i \in I : T'_i \leqslant G_i\!\restriction\!\mathtt{q}$*; or,*
2. $\mathrm{unf}(G) = \mathtt{s}\to\mathtt{t}\colon\{\mathtt{m_j}(S''_j).G_j\}_{j\in J}$*, where for all* $j \in J : \Delta(s[\mathtt{p}]) \leqslant G_j\!\restriction\!\mathtt{p}$, $\Delta(s[\mathtt{q}]) \leqslant G_j\!\restriction\!\mathtt{q}$*, and* $\{\mathtt{p},\mathtt{q}\} \cap \{\mathtt{s},\mathtt{t}\} = \emptyset$.

Proof. By combining cases (1) and (2) from Lem. A.25. Note that case 1(a) is incompatible with case 2(b), since 2(b) requires that $\mathtt{p} \neq \mathtt{s}$. Similarly, case 1(b) is incompatible with case 2(a). □

A.6. Completeness of Association

Theorem 3.2 (Completeness of Association). *Given associated global type* G *and typing context* Δ *for session* s: $\Delta \sqsubseteq_s G$. *If* $\Delta \xrightarrow{\alpha} \Delta'$ *where* $\alpha = s[\mathtt{p}][\mathtt{q}]\mathtt{m}$*, then there exists* G' *such that* $\Delta' \sqsubseteq_s G'$ *and* $G \xrightarrow{\alpha} G'$.

Proof. By induction on transitions of typing context $\Delta \xrightarrow{\alpha} \Delta'$. Since α is of the form $s[\mathtt{p}][\mathtt{q}]\mathtt{m}$, we only need to consider the following two cases.

- Case [Δ-⊕&]:

  From the premise, we have:

$$\Delta \sqsubseteq_s G \tag{A.1}$$
$$\alpha = s[\mathtt{p}][\mathtt{q}]\mathtt{m} \tag{A.2}$$
$$\Delta = \Delta_1, \Delta_2 \tag{A.3}$$
$$\Delta_1 \xrightarrow{s[\mathtt{p}]:\mathtt{q}\oplus\mathtt{m}(S)} \Delta'_1 \tag{A.4}$$
$$\Delta_2 \xrightarrow{s[\mathtt{q}]:\mathtt{p}\&\mathtt{m}(S')} \Delta'_2 \tag{A.5}$$
$$\Delta' = \Delta'_1, \Delta'_2 \tag{A.6}$$

  By applying Lem. A.17 on (A.4) and (A.5), we have

$$\mathrm{unf}(\Delta_1(s[\mathtt{p}])) = \mathtt{q}\oplus\{\mathtt{m_i}(S_i).T_i\}_{i\in I_{\mathtt{p}}} \quad \text{and} \quad \mathrm{unf}(\Delta_2(s[\mathtt{q}])) = \mathtt{p}\&\{\mathtt{m_i}(S'_i).T'_i\}_{i\in I_{\mathtt{q}}}$$

  Moreover, by inverting $\Delta \xrightarrow{s[\mathtt{p}][\mathtt{q}]\mathtt{m}} \Delta'$, $\exists k \in (I_{\mathtt{p}} \cap I_{\mathtt{q}})$, such that $\mathtt{m}_k = \mathtt{m}$ and $S'_k \leqslant S_k$.

  We perform case analysis on Lem. A.26.

  – Case 1 of Lem. A.26: we know $\mathrm{unf}(G) = \mathtt{p}\to\mathtt{q}\colon\{\mathtt{m_i}(S''_i).G_i\}_{i\in I}$ where $I_{\mathtt{p}} \subseteq I \subseteq I_{\mathtt{q}}$, $\forall i \in I_{\mathtt{p}} : S''_i \leqslant S_i$, $\forall i \in I : S'_i \leqslant S''_i$, $\forall i \in I_{\mathtt{p}} : T_i \leqslant G_i\!\restriction\!\mathtt{p}$, and $\forall i \in I : T'_i \leqslant G_i\!\restriction\!\mathtt{q}$.

  Since $k \in (I_{\mathtt{p}} \cap I_{\mathtt{q}})$ and $I_{\mathtt{p}} \subseteq I$, we have $k \in I$. By applying [GR-⊕&] (with Prop. A.21), the result becomes G_k.

  We are left to show association. By Lems. A.17 and A.25, we have $\Delta'_1(s[\mathtt{p}]) = T_k \leqslant G_k\!\restriction\!\mathtt{p}$ and $\Delta'_2(s[\mathtt{q}]) = T'_k \leqslant G_k\!\restriction\!\mathtt{q}$. For other roles $\mathtt{r} \in \mathrm{roles}(G)$, we have $\Delta(s[\mathtt{r}]) = \Delta'(s[\mathtt{r}])$ by Lem. A.16, and $G\!\restriction\!\mathtt{r} = \bigsqcap_{i\in I}(G_i\!\restriction\!\mathtt{r}) \leqslant G_k\!\restriction\!\mathtt{r}$ by Lem. A.7. Since $\Delta(s[\mathtt{r}]) \leqslant G\!\restriction\!\mathtt{r}$, we can conclude that $\Delta'(s[\mathtt{r}]) \leqslant G_k\!\restriction\!\mathtt{r}$ by transitivity of subtyping. Additionally, for all other $s[\mathtt{q}'] \in \mathrm{dom}(\Delta)$ where $\mathtt{q}' \notin \mathrm{roles}(G)$, we have $\Delta(s[\mathtt{q}']) = \Delta'(s[\mathtt{q}']) = \mathbf{end}$ by $\Delta \sqsubseteq_s G$ and (3) of Lem. A.19.

  – Case 2 of Lem. A.26: we know $\mathrm{unf}(G) = \mathtt{s}\to\mathtt{t}\colon\{\mathtt{m_j}(S''_j).G_j\}_{j\in J}$, where for all $j \in J$, $\Delta(s[\mathtt{p}]) \leqslant G_j\!\restriction\!\mathtt{p}$ and $\Delta(s[\mathtt{q}]) \leqslant G_j\!\restriction\!\mathtt{q}$, and $\{\mathtt{p},\mathtt{q}\} \cap \{\mathtt{s},\mathtt{t}\} = \emptyset$.

  Take an arbitrary index $j \in J$, we construct a typing context Δ_j such that $\Delta_j \xrightarrow{s[\mathtt{p}][\mathtt{q}]\mathtt{m}} \Delta'_j$ and $\Delta_j \sqsubseteq_s G_j$. To construct Δ_j, we consider sub-cases for all roles, and show that $\Delta_j \sqsubseteq_s G_j$:

* For role $\mathtt{s}$, we know from $\Delta \sqsubseteq_s G$ that $\Delta(s[\mathtt{s}]) \leqslant \mathtt{t}\oplus\{\mathtt{m_j}(S''_j).(G_j\restriction\mathtt{s})\}_{j\in J} = \mathrm{unf}(G)\restriction\mathtt{s}$. By inverting [SUB-⊕] (applying Lem. A.5 where necessary), we have $\mathrm{unf}(\Delta(s[\mathtt{s}])) = \mathtt{t}\oplus\{\mathtt{m_j}(S'''_j).T''_j\}_{j\in J_\mathtt{s}}$, where $J_\mathtt{s} \subseteq J$, and $\forall j \in J_\mathtt{s} : T''_j \leqslant G_j\restriction\mathtt{s}$.
  To construct Δ_j, let $\Delta_j(s[\mathtt{s}]) = T''_j$ if $j \in J_\mathtt{s}$ and $\Delta_j(s[\mathtt{s}]) = G_j\restriction\mathtt{s}$ otherwise. In either case, we have $\Delta_j(s[\mathtt{s}]) \leqslant G_j\restriction\mathtt{s}$, as required.
* For role $\mathtt{t}$, we know from $\Delta \sqsubseteq_s G$ that $\Delta(s[\mathtt{t}]) \leqslant \mathtt{s}\&\{\mathtt{m_j}(S''_j).(G_j\restriction\mathtt{t})\}_{j\in J} = \mathrm{unf}(G)\restriction\mathtt{t}$. By inverting [SUB-&] (applying Lem. A.5 where necessary), we have $\mathrm{unf}(\Delta(s[\mathtt{t}])) = \mathtt{s}\&\{\mathtt{m_j}(S'''_j).T''_j\}_{j\in J_\mathtt{t}}$, where $J \subseteq J_\mathtt{t}$, and $\forall j \in J : T''_j \leqslant G_j\restriction\mathtt{t}$.
  To construct Δ_j, let $\Delta_j(s[\mathtt{t}]) = T''_j$, and we have $\Delta_j(s[\mathtt{t}]) \leqslant G_j\restriction\mathtt{t}$, as required.
* For other roles $\mathtt{r} \in \mathrm{roles}(G)$ with $\mathtt{r} \notin \{\mathtt{s},\mathtt{t}\}$, their typing context entry do not change, i.e. $\Delta_j(s[\mathtt{r}]) = \Delta(s[\mathtt{r}])$. We have $\Delta_j(s[\mathtt{r}]) = \Delta(s[\mathtt{r}]) \leqslant G\restriction\mathtt{r} = \bigsqcap_{j\in J}(G_j\restriction\mathtt{r}) \leqslant G_j\restriction\mathtt{r}$ (applying Lem. A.7). Notice that $\{\mathtt{p},\mathtt{q}\} \in \mathrm{roles}(G)$, so they are still able to perform the communication action $\Delta_j \xrightarrow{s[\mathtt{p}][\mathtt{q}]\mathtt{m}} \Delta'_j$.

We apply inductive hypothesis on Δ_j, and obtain $G_j \xrightarrow{s[\mathtt{p}][\mathtt{q}]\mathtt{m}} G'_j$ and $\Delta'_j \sqsubseteq_s G'_j$.

We can apply [GR-CTX] on $\mathtt{s}\to\mathtt{t}\colon\{\mathtt{m_j}(S''_j).G_j\}_{j\in J} \xrightarrow{s[\mathtt{p}][\mathtt{q}]\mathtt{m}} \mathtt{s}\to\mathtt{t}\colon\{\mathtt{m_j}(S''_j).G'_j\}_{j\in J}$.

We now show $\Delta' \sqsubseteq_s G'$, where $G' = \mathtt{s}\to\mathtt{t}\colon\{\mathtt{m_j}(S''_j).G'_j\}_{j\in J}$.

For role $\mathtt{s}$, we know that $\mathrm{unf}(\Delta(s[\mathtt{s}])) = \mathtt{t}\oplus\{\mathtt{m_j}(S'''_j).T''_j\}_{j\in J_\mathtt{s}}$, where $J_\mathtt{s} \subseteq J$, and $\forall j \in J_\mathtt{s} : T''_j \leqslant G_j\restriction\mathtt{s}$ and $S''_j \leqslant S'''_j$. Since $\mathtt{s} \notin \mathrm{subject}(s[\mathtt{p}][\mathtt{q}]\mathtt{m})$, we apply Lem. A.16 on Δ and Δ_j for all $j \in J_\mathtt{s}$. For all $j \in J_\mathtt{s}$, we have $T''_j = \Delta_j(s[\mathtt{s}]) = \Delta'_j(s[\mathtt{s}])$ (from Lem. A.16) and $\Delta'_j(s[\mathtt{s}]) \leqslant G'_j\restriction\mathtt{s}$ (from inductive hypothesis). Therefore, we have $T''_j \leqslant G'_j\restriction\mathtt{s}$. We now apply Lem. A.16 on Δ, which gives $\mathrm{unf}(\Delta(s[\mathtt{s}])) = \mathrm{unf}(\Delta'(s[\mathtt{s}])) = \mathtt{t}\oplus\{\mathtt{m_j}(S'''_j).T''_j\}_{j\in J_\mathtt{s}}$. We can now apply [SUB-⊕] to conclude $\Delta'(s[\mathtt{s}]) \leqslant G'\restriction\mathtt{s}$.

For role $\mathtt{t}$, we know that $\mathrm{unf}(\Delta(s[\mathtt{t}])) = \mathtt{s}\&\{\mathtt{m_j}(S'''_j).T''_j\}_{j\in J_\mathtt{t}}$, where $J \subseteq J_\mathtt{t}$, and $\forall j \in J : T''_j \leqslant G_j\restriction\mathtt{t} \leqslant T''_j$ and $S'''_j \leqslant S''_j$. Since $\mathtt{t} \notin \mathrm{subject}(s[\mathtt{p}][\mathtt{q}]\mathtt{m})$, we apply Lem. A.16 on Δ and Δ_j for all $j \in J$. For all $j \in J$, we have $T''_j = \Delta_j(s[\mathtt{t}]) = \Delta'_j(s[\mathtt{t}])$ (from Lem. A.16) and $\Delta'_j(s[\mathtt{t}]) \leqslant G'_j\restriction\mathtt{t}$ (from inductive hypothesis). Therefore, we have $T''_j \leqslant G'_j\restriction\mathtt{t}$. We now apply Lem. A.16 on Δ, which gives $\mathrm{unf}(\Delta(s[\mathtt{t}])) = \mathrm{unf}(\Delta'(s[\mathtt{t}])) = \mathtt{s}\&\{\mathtt{m_j}(S'''_j).T''_j\}_{j\in J_\mathtt{t}}$. We can now apply [SUB-&] to conclude $\Delta'(s[\mathtt{t}]) \leqslant G'\restriction\mathtt{t}$.

For other roles $\mathtt{r} \in \mathrm{roles}(G)$ (where $\mathtt{r} \notin \{\mathtt{s},\mathtt{t}\}$), we need to show $\Delta'(s[\mathtt{r}]) \leqslant G'\restriction\mathtt{r}$. We know that $G'\restriction\mathtt{r} = \bigsqcap_{j\in J} G'_j\restriction\mathtt{r}$. The inductive hypothesis gives $\Delta'_j(s[\mathtt{r}]) \leqslant G'_j\restriction\mathtt{r}$, and then we apply Lem. A.9 to obtain $\Delta'_j(s[\mathtt{r}]) \leqslant \bigsqcap_{j\in J} G'_j\restriction\mathtt{r} = G'\restriction\mathtt{r}$. Note that $\Delta_j(s[\mathtt{r}]) = \Delta(s[\mathtt{r}])$ by construction. We now apply Lem. A.16 on Δ and all Δ_j, which gives $\Delta'_j(s[\mathtt{r}]) = \Delta_j(s[\mathtt{r}])$ and $\Delta(s[\mathtt{r}]) = \Delta'(s[\mathtt{r}])$, thus $\Delta'_j(s[\mathtt{r}]) = \Delta'(s[\mathtt{r}])$ for all j. Therefore, we have $\Delta'(s[\mathtt{r}]) \leqslant G'\restriction\mathtt{r}$.

- Case [Δ-μ] (possibly with [Δ-,]):
  By inductive hypothesis and Prop. A.21. □

Corollary A.27. *Assume that for any session $s \in \Delta$, there exists a global type G_s such that $\Delta_s \sqsubseteq_s G_s$. If $\Delta \to \Delta'$, then for any $s \in \Delta'$, there exists a global type G'_s such that $G_s \to^* G'_s$ and $\Delta'_s \sqsubseteq_s G'_s$.*

Proof. By Def. 3.6, we have that there exists a label $s[\mathtt{p}][\mathtt{q}]\mathtt{m}$ such that $\Delta \xrightarrow{s[\mathtt{p}][\mathtt{q}]\mathtt{m}} \Delta'$, and hence, by Lem. A.15, $\mathrm{dom}(\Delta) = \mathrm{dom}(\Delta')$. We are left to show that for any $\Delta'_{s'}$ with $s' \in \Delta'$, there exists a global type $G'_{s'}$ such that $G_{s'} \to^* G'_{s'}$ and $\Delta'_{s'} \sqsubseteq_{s'} G'_{s'}$.

- Case Δ'_s: since $\Delta \xrightarrow{s[\mathtt{p}][\mathtt{q}]\mathtt{m}} \Delta'$, it is trivial to have $\Delta_s \xrightarrow{s[\mathtt{p}][\mathtt{q}]\mathtt{m}} \Delta'_s$. Moreover, by $\Delta_s \sqsubseteq_s G_s$ and Thm. 3.2, we have that there exists G'_s such that $G_s \to G'_s$ and $\Delta'_s \sqsubseteq_s G'_s$, as desired.
- Case $\Delta'_{s'}$ with $s' \neq s$: we know from Lem. A.19 that for any $s'[\mathtt{r}] \in \mathrm{dom}(\Delta')$, $\Delta(\mathrm{unf}(s'[\mathtt{r}])) = \Delta'(\mathrm{unf}(s'[\mathtt{r}]))$. Then, along with $G_{s'} \to^* G_{s'}$ and $\Delta_{s'} \sqsubseteq_{s'} G_{s'}$, the thesis holds. □

A.7. Soundness of Association

Theorem 3.1 (Soundness of Association)**.** *Given associated global type G and typing context Δ for session s: $\Delta \sqsubseteq_s G$. If $G \xrightarrow{\alpha} G'$ where $\alpha = s[\mathtt{p}][\mathtt{q}]\mathtt{m}$, then there exist $\mathtt{m}'$, α', Δ', and G'', such that $\alpha' = s[\mathtt{p}][\mathtt{q}]\mathtt{m}'$, $G \xrightarrow{\alpha'} G''$, $\Delta' \sqsubseteq_s G''$, and $\Delta \xrightarrow{\alpha'} \Delta'$.*

Proof. By induction on transitions of global type $G \xrightarrow{\alpha} G'$.

- Case [GR-⊕&]:

  From the premise, we have:

$$\Delta \sqsubseteq_s G \tag{A.7}$$
$$G = \mathtt{p}\to\mathtt{q}\colon \{\mathtt{m_i}(S_i).G_i\}_{i\in I} \tag{A.8}$$
$$\alpha = s[\mathtt{p}][\mathtt{q}]\mathtt{m_j} \tag{A.9}$$
$$j \in I \tag{A.10}$$
$$G' = G_j \tag{A.11}$$

  By association (A.7), we have $\Delta(s[\mathtt{p}]) \leqslant G\restriction\mathtt{p} = \mathtt{q}\oplus\{\mathtt{m_i}(S_i).(G_i\restriction\mathtt{p})\}_{i\in I}$, and $\Delta(s[\mathtt{q}]) \leqslant G\restriction\mathtt{q} = \mathtt{p}\&\{\mathtt{m_i}(S_i).(G_i\restriction\mathtt{p})\}_{i\in I}$. Then by Lem. A.6 and Lem. A.24, we have $\Delta(s[\mathtt{p}]) = \mathtt{q}\oplus\{\mathtt{m'_i}(S'_i).T'_i\}_{i\in I_\mathtt{p}}$ and $\Delta(s[\mathtt{q}]) = \mathtt{p}\&\{\mathtt{m''_i}(S''_i).T''_i\}_{i\in I_\mathtt{q}}$, with $I_\mathtt{p} \subseteq I \subseteq I_\mathtt{q}$, and for all $i \in I_\mathtt{p}$: $\mathtt{m}_i = \mathtt{m}'_i = \mathtt{m}''_i$, $S''_i \leqslant S'_i$, $T'_i \leqslant G_i\restriction\mathtt{p}$, and $T''_i \leqslant G_i\restriction\mathtt{q}$.

  Now let us choose some $k \in I_\mathtt{p}$ such that $\alpha' = s[\mathtt{p}][\mathtt{q}]\mathtt{m_k}$. Furthermore, we have $G \xrightarrow{\alpha'} G''$ with $G'' = G_k$. We are left to show that there exists Δ' such that $\Delta \xrightarrow{s[\mathtt{p}][\mathtt{q}]\mathtt{m_k}} \Delta'$ and $\Delta' \sqsubseteq_s G''$.

  We apply [Δ-⊕] on $s[\mathtt{p}]$ and [Δ-&] on $s[\mathtt{q}]$, which can be combined via [Δ-⊕&]. By applying [Δ-,] and [Δ-,B] when needed, we have $\Delta \xrightarrow{s[\mathtt{p}][\mathtt{q}]\mathtt{m_k}} \Delta'$, with $\Delta'(s[\mathtt{p}]) = T'_k$, $\Delta'(s[\mathtt{q}]) = T''_k$, and $\Delta'(s[\mathtt{r}]) = \Delta(s[\mathtt{r}])$ if $\mathtt{r} \neq \mathtt{p}$ and $\mathtt{r} \neq \mathtt{q}$.

  Finally, we show that $\Delta' \sqsubseteq_s G'' = G_k$. For $\mathtt{p}$, we have $T'_k = \Delta'(s[\mathtt{p}]) \leqslant G_k\restriction\mathtt{p}$, and similar for $\mathtt{q}$. For $\mathtt{r} \neq \mathtt{p} \neq \mathtt{q} \in \mathrm{roles}(G)$, it follows that $\Delta(s[\mathtt{r}]) = \Delta'(s[\mathtt{r}]) \leqslant \bigsqcap_{i\in I} G_i\restriction\mathtt{r} = G\restriction\mathtt{r}$. By Lem. A.7, $\Delta'(s[\mathtt{r}]) \leqslant \bigsqcap_{i\in I} G_i\restriction\mathtt{r} \leqslant G_k\restriction\mathtt{r}$, and hence, $\Delta'(s[\mathtt{r}]) \leqslant G_k\restriction\mathtt{r}$ holds by the transitivity of $\leqslant$. For any $\mathtt{p}' \in \mathrm{roles}(G)$ where $\mathtt{p}' \notin \mathrm{roles}(G_k)$, by Lem. A.14, $G_k\restriction\mathtt{p}' = \mathbf{end}$, implying that $\Delta'(s[\mathtt{p}']) = \mathbf{end}$. Additionally, for all other $s[\mathtt{q}'] \in \mathrm{dom}(\Delta)$ where $\mathtt{q}' \notin \mathrm{roles}(G)$, we have $\Delta(s[\mathtt{q}']) = \Delta'(s[\mathtt{q}']) = \mathbf{end}$ by $\Delta \sqsubseteq_s G$ and (3) of Lem. A.19.

- Case [GR-μ]:

  By inductive hypothesis and [Δ-μ].

- Case [GR-Ctx]:

  From the premise, we have:

$$\Delta \sqsubseteq_s G \tag{A.12}$$
$$G = \mathtt{p}\to\mathtt{q}\colon \{\mathtt{m_i}(S_i).G_i\}_{i\in I} \tag{A.13}$$
$$\forall i \in I : G_i \xrightarrow{\alpha} G'_i \tag{A.14}$$
$$\mathrm{subject}(\alpha) \cap \{\mathtt{p},\mathtt{q}\} = \emptyset \tag{A.15}$$
$$G' = \mathtt{p}\to\mathtt{q}\colon \{\mathtt{m_i}(S_i).G'_i\}_{i\in I} \tag{A.16}$$

  Let $\alpha = s[\mathtt{r}][\mathtt{u}]\mathtt{m}$ with $\{\mathtt{r},\mathtt{u}\} \cap \{\mathtt{p},\mathtt{q}\} = \emptyset$.

  By (A.12) and (A.13), it follows that for any role $\mathtt{t}$ with $\mathtt{t} \neq \mathtt{p}$ and $\mathtt{t} \neq \mathtt{q}$, $\Delta(s[\mathtt{t}]) \leqslant \bigsqcap_{i\in I} G_i\restriction\mathtt{t}$. Let $j \in I$ be arbitrary. By Lem. A.7 and transitivity of subtyping, we obtain $\Delta(s[\mathtt{t}]) \leqslant G_j\restriction\mathtt{t}$.

We define a typing context Δ_j by: $\Delta_j(s[\mathtt{p}]) = G_j \upharpoonright \mathtt{p}$, $\Delta_j(s[\mathtt{q}]) = G_j \upharpoonright \mathtt{q}$, $\Delta_j(s[\mathtt{t}]) = \Delta(s[\mathtt{t}])$ for any other role $\mathtt{t}$. It follows that $\Delta_j \sqsubseteq_s G_j$.

Since $G_j \xrightarrow{\alpha} G'_j$ and $\Delta_j \sqsubseteq_s G_j$, by the induction hypothesis there exist $\alpha' = s[\mathtt{r}][\mathtt{u}]\mathtt{m}'$ and Δ'_j such that $\Delta_j \xrightarrow{\alpha'} \Delta'_j$.

By inversion of typing context transition (Lem. A.17), together with $\Delta_j(s[\mathtt{r}]) = \Delta(s[\mathtt{r}])$ and $\Delta_j(s[\mathtt{u}]) = \Delta(s[\mathtt{u}])$, we obtain that $\mathrm{unf}(\Delta(s[\mathtt{r}])) = \mathtt{u}\oplus\{\mathtt{m}'_k(S'_k).T'_k\}_{k\in K}$ and $\mathrm{unf}(\Delta(s[\mathtt{u}])) = \mathtt{r}\&\{\mathtt{m}''_l(S''_l).T''_l\}_{l\in L}$, where there exist $k \in K$ and $l \in L$ such that $\mathtt{m}'_k = \mathtt{m}''_l = \mathtt{m}'$ and $S''_l \leqslant S'_k$.

Applying typing context transition rule [Δ-⊕&] (and [Δ-μ], when necessary) to Δ, there exists Δ' such that $\Delta \xrightarrow{\alpha'} \Delta'$.

Finally, by completeness of association (Thm. 3.2), there exists G'' such that $G \xrightarrow{\alpha'} G''$ and $\Delta' \sqsubseteq_s G''$, as desired. □

A.8. Behavioural Properties Guaranteed by Association

A.8.1. Safety by Association

Lemma A.28. *If $\Delta \sqsubseteq_s G$, then Δ is s-safe.*

Proof. Let $\varphi = \{\Delta' \mid \exists G' : G \rightarrow^* G' \text{ and } \Delta' \sqsubseteq_s G'\}$.

Take any $\Delta \in \varphi$, we show that Δ satisfies all safety properties. By definition of φ, there exists G' with $G \rightarrow^* G'$ and $\Delta \sqsubseteq_s G'$. We only detail the case that $G' \neq \mathbf{end}$ as if $G' = \mathbf{end}$, by Lem. A.22, we have that $\Delta = \Delta_{\mathbf{end}}$, which satisfies all clauses of Def. 3.8.

[S-⊕&] Since (by hypothesis) $\Delta \xrightarrow{s[\mathtt{p}]:\mathtt{q}\oplus\mathtt{m}(S)}$ and $\Delta \xrightarrow{s[\mathtt{q}]:\mathtt{p}\&\mathtt{m}'(S')}$, by Lem. A.17, we have $\mathrm{unf}(\Delta(s[\mathtt{p}])) = \mathtt{q}\oplus\{\mathtt{m}_i(S_i).T_i\}_{i\in I}$, $\exists k \in I$ such that $\mathtt{m} = \mathtt{m}_k$, and $\mathrm{unf}(\Delta(s[\mathtt{q}])) = \mathtt{p}\&\{\mathtt{m}'_j(S'_j).T'_j\}_{j\in J}$. Then we apply Lem. A.24 on $\Delta(s[\mathtt{p}]) \leqslant G' \upharpoonright \mathtt{p}$ and $\Delta(s[\mathtt{q}]) \leqslant G' \upharpoonright \mathtt{q}$ to get $I \subseteq J$, and $\forall i \in I : \mathtt{m}_i = \mathtt{m}'_i$ and $S'_i \leqslant S_i$. Consequently, along with $\mathtt{m} = \mathtt{m}_k$, by applying [Δ-⊕&] (and [Δ-μ] as needed), $\Delta \xrightarrow{s[\mathtt{p}][\mathtt{q}]\mathtt{m}}$, which is the thesis.

[S-μ] Let Δ' be constructed from Δ with $\Delta'(s[\mathtt{p}]) = T\{^{\mu \mathbf{t}.T}/_{\mathbf{t}}\}$. By Lem. A.5, we know $\Delta'(s[\mathtt{p}]) \leqslant \Delta(s[\mathtt{p}])$, and thus $\Delta' \leqslant \Delta$. By Def. 3.7 and transitivity of subtyping, we have $\Delta' \sqsubseteq_s G'$, which means that $\Delta' \in \varphi$.

[S-$\rightarrow_s$] Let $\Delta \xrightarrow{\alpha} \Delta'$ with $\alpha = s[\mathtt{p}][\mathtt{q}]\mathtt{m}$, meaning that $\Delta \rightarrow_s \Delta'$. By Thm. 3.2, there exists G'' with $G' \xrightarrow{\alpha} G''$ and $\Delta' \sqsubseteq_s G''$. By definition of φ, the typing context Δ' after transition Δ on session s is in φ. □

A.8.2. Deadlock-Freedom by Association

Lemma A.29. *If $\Delta \sqsubseteq_s G$, then Δ is s-deadlock-free.*

Proof. By operational correspondence of global type G and typing context Δ (Thms. 3.1 and 3.2), there exists a global type G' such that $G \rightarrow^* G' \not\rightarrow$, with associated typing contexts $\Delta \rightarrow_s^* \Delta' \not\rightarrow_s$. Since no further reductions are possible for the global type G', it must be in the form of **end** (Lem. A.13). Therefore, the thesis holds by Lem. A.22. □

A.8.3. Liveness by Association

Lemma A.30. *If $\Delta \sqsubseteq_s G$, then Δ is s-live.*

Proof. We want to show that any path starting with Δ which is fair for session s is also live for s. We proceed by contradiction, assuming that there is a fair path for session s: $(\Delta_n)_{n\in N}$ where $N = \{0, 1, 2, \ldots\}$, $\Delta_0 = \Delta$, and $\forall n, n+1 \in N : \Delta_n \rightarrow_s \Delta_{n+1}$, which is not live for s. We consider the following two cases.

- Case $\Delta_j \xrightarrow{s[\mathtt{p}]:\mathtt{q}\oplus\mathtt{m}(S)}$ with $j \in N$, and for any $k \in N$ with $k \geq j$, there does not exist $\mathtt{m}'$ such that $\Delta_k \xrightarrow{s[\mathtt{p}][\mathtt{q}]\mathtt{m}'} \Delta_{k+1}$: by operational correspondence of global type G and typing context Δ (Thms. 3.1 and 3.2), there exists a global type G_j such that $G \to^* G_j$ and $\Delta_j \sqsubseteq_s G_j$. Moreover, since $\Delta_j \xrightarrow{s[\mathtt{p}]:\mathtt{q}\oplus\mathtt{m}(S)}$, by Lems. A.17 and A.25, we have $\mathrm{unf}(\Delta_j(s[\mathtt{p}])) = \mathtt{q}\oplus\{\mathtt{m_i}(S_i).T_i\}_{i\in I}$, and
  - either $\mathrm{unf}(G_j) = \mathtt{p}\to\mathtt{q}\colon\{\mathtt{m_i}(S'_i).G''_i\}_{i\in I'}$, where $I \subseteq I'$, and for all $i \in I : \mathtt{m}_i = \mathtt{m}_i$, $S'_i \leqslant S_i$, and $T_i \leqslant G''_i\restriction\mathtt{p}$. It follows directly that $\Delta_j(s[\mathtt{q}]) \leqslant G_j\restriction\mathtt{q} \leqslant \mathrm{unf}(G_j)\restriction\mathtt{q} = \mathtt{p}\&\{\mathtt{m_i}(S'_i).G''_i\restriction\mathtt{q}\}_{i\in I'}$. Hence, we have $\Delta_j \xrightarrow{s[\mathtt{q}]:\mathtt{p}\&\mathtt{m}(S')}$ with $S' \leqslant S$, and therefore, $\Delta_j \xrightarrow{s[\mathtt{p}][\mathtt{q}]\mathtt{m}}$. With the fact that $(\Delta_n)_{n\in N}$ is a fair path for session s, we know that there exist some k and $\mathtt{m}'$ such that $k \in N$, $k \geq j$, and $\Delta_k \xrightarrow{s[\mathtt{p}][\mathtt{q}]\mathtt{m}'} \Delta_{k+1}$, a desired contradiction.
  - or $\mathrm{unf}(G_j) = \mathtt{s}\to\mathtt{t}\colon\{\mathtt{m_l}(S'_l).G''_l\}_{l\in L}$, where for all $l \in L : \Delta_j(s[\mathtt{p}]) \leqslant G''_l\restriction\mathtt{p}$, with $\mathtt{p} \neq \mathtt{s}$ and $\mathtt{p} \neq \mathtt{t}$. By the assumption that for any $k \in N$ with $k \geq j$, there does not exist $\mathtt{m}'$ such that $\Delta_k \xrightarrow{s[\mathtt{p}][\mathtt{q}]\mathtt{m}'} \Delta_{k+1}$, $\mathrm{unf}(\Delta_j(s[\mathtt{p}])) = \mathtt{q}\oplus\{\mathtt{m_i}(S_i).T_i\}_{i\in I}$, we know that transmission between $\mathtt{p}$ and $\mathtt{q}$ will never occur from Δ_j. By operational correspondence, we also conclude that such transmission will never be triggered from G_j. This is a desired contradiction, as by Lem. A.23, each continuation G''_l of G_j must involve transmission between $\mathtt{p}$ and $\mathtt{q}$, and therefore, the previous subcase scenario should inevitably occur.

- Case $\Delta_j \xrightarrow{s[\mathtt{q}]:\mathtt{p}\&\mathtt{m}(S)}$ with $j \in N$, and for any $k \in N$ with $k \geq j$, there does not exist $\mathtt{m}'$ such that $\Delta_k \xrightarrow{s[\mathtt{p}][\mathtt{q}]\mathtt{m}'} \Delta_{k+1}$: similar to the previous case. □

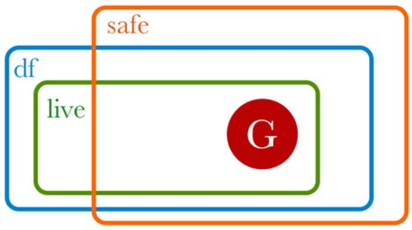
Theorem 3.3 (Safety, Deadlock-Freedom, and Liveness by Association). *Let G be a global type, Δ a typing context, and s a session. If Δ is associated with G for s: $\Delta \sqsubseteq_s G$, then Δ is s-safe, s-deadlock-free, and s-live.*

Proof. Apply Lems. A.28 to A.30. □

A.9. Relating Typing Context Properties

Theorem 3.4. *For any typing context Δ and session s, the following statements are valid:*

(1) $\mathrm{df}(s,\Delta) \not\Leftarrow \not\Rightarrow \mathrm{safe}(s,\Delta)$;
(2) $\mathrm{live}(s,\Delta) \not\Leftarrow \not\Rightarrow \mathrm{safe}(s,\Delta)$;
(3) $\mathrm{live}(s,\Delta) \not\Leftarrow \Longrightarrow \mathrm{df}(s,\Delta)$;
(4) $\exists G : \Delta \sqsubseteq_s G \not\Leftarrow \Longrightarrow \mathrm{safe}(s,\Delta)$;
(5) $\exists G : \Delta \sqsubseteq_s G \not\Leftarrow \Longrightarrow \mathrm{df}(s,\Delta)$;
(6) $\exists G : \Delta \sqsubseteq_s G \not\Leftarrow \Longrightarrow \mathrm{live}(s,\Delta)$.



Proof. The negated implications in the statement are demonstrated in Exs. 3.7, 3.8 and 3.10. Let's now consider the remaining implications.

(3). Assume $\mathrm{live}(s,\Delta)$. We need to prove $\mathrm{df}(s,\Delta)$, i.e. for any path $\Delta_0 \to_s \Delta_1 \to_s \cdots \to_s \Delta_n$ with $n > 0$, $\Delta_0 = \Delta$, and $\Delta_n \not\to_s$, it holds that $\forall s[\mathtt{p}] \in \mathrm{dom}(\Delta_n) : \Delta_n(s[\mathtt{p}]) = \mathbf{end}$.

We first show that any such path $\Delta_0 \to_s \Delta_1 \to_s \cdots \to_s \Delta_n$ is fair for session s by contradiction, assuming it is not fair for s. In this case, there exists k such that $0 \leq k \leq n$ and $\Delta_k \xrightarrow{s[\mathtt{p}][\mathtt{q}]\mathtt{m}}$, while for all j and $\mathtt{m}'$ with $k \leq j \leq n$, $\Delta_j \xrightarrow{s[\mathtt{p}][\mathtt{q}]\mathtt{m}'} \Delta_{j+1}$ does not occur in the path. Hence, by Lem. A.16, we have that $\Delta_n(s[\mathtt{p}]) = \Delta_k(s[\mathtt{p}])$ and $\Delta_n(s[\mathtt{q}]) = \Delta_k(s[\mathtt{q}])$. It follows directly that $\Delta_n \xrightarrow{s[\mathtt{p}][\mathtt{q}]\mathtt{m}}$, contradicting $\Delta_n \not\to_s$.

We proceed by contradiction again, assuming that there exists $s[\mathtt{p}] \in \mathrm{dom}(\Delta_n)$ such that $\Delta_n(s[\mathtt{p}]) \not\leqslant \mathbf{end}$, i.e. $\mathrm{unf}(\Delta_n(s[\mathtt{p}])) = \mathtt{q}\&\{\mathtt{m_i}(S_i).T_i\}_{i\in I}$ or $\mathrm{unf}(\Delta_n(s[\mathtt{p}])) = \mathtt{q}\oplus\{\mathtt{m_i}(S_i).T_i\}_{i\in I}$.

We now detail the case where $\mathrm{unf}(\Delta_n(s[\mathrm{p}])) = \mathrm{q}\&\{\mathtt{m_i}(S_i).T_i\}_{i\in I}$, with the other case following similarly.

Since $\mathrm{unf}(\Delta_n(s[\mathrm{p}])) = \mathrm{q}\&\{\mathtt{m_i}(S_i).T_i\}_{i\in I}$, we have $\Delta_n \xrightarrow{s[\mathrm{p}]:\mathrm{q}\&\mathtt{m_k}(S_k)}$ with $k \in I$. Given that Δ_0 is s-live and $\Delta_0 \to_s \Delta_1 \to_s \cdots \to_s \Delta_n$ is fair for s, there exist j and $\mathtt{m}'$ such that $n \leq j \leq n$ and $\Delta_j \xrightarrow{s[\mathrm{p}][\mathrm{q}]\mathtt{m}'} \Delta_{j+1}$. This follows $\Delta_n \to_s$, contradicting $\Delta_n \not\to_s$.

(4). Straightforward from Lem. A.28.

(5). Straightforward from Lem. A.29.

(6). Straightforward from Lem. A.30. □

B. Subtyping Properties Regarding Association

Lemma B.1. *If $\Delta \sqsubseteq_s G$ and $\Delta' \leqslant \Delta$, then $\Delta' \sqsubseteq_s G$.*

Proof. By the definition of association (Def. 3.7), the definition of $\Delta' \leqslant \Delta$ (Def. 3.5), and the transitivity of subtyping (Lem. A.4). □

Lemma B.2. *Assume that $\Delta \sqsubseteq_s G$, $\Delta' \leqslant \Delta$ and $\Delta' \xrightarrow{\alpha} \Delta''$ with $\alpha \in \{s[\mathrm{p}][\mathrm{q}]\mathtt{m} \mid \mathrm{p},\mathrm{q} \in \mathcal{R}\}$. Then, there is Δ''' such that $\Delta \xrightarrow{\alpha} \Delta'''$ and $\Delta'' \leqslant \Delta'''$.*

Proof. Since $\Delta' \xrightarrow{s[\mathrm{p}][\mathrm{q}]\mathtt{m}} \Delta''$, by applying and inverting [Δ-⊕&] (and [Δ-μ] when necessary), and also using Lem. A.17, we have $\mathrm{unf}(\Delta'(s[\mathrm{p}])) = \mathrm{q}\oplus\{\mathtt{m_i}(S_i).T_i\}_{i\in I}$, $\mathrm{unf}(\Delta'(s[\mathrm{q}])) = \mathrm{p}\&\{\mathtt{m'_j}(S'_j).T'_j\}_{j\in J}$, $\exists k : k \in I$, $k \in J$, $\mathtt{m}_k = \mathtt{m'}_k = \mathtt{m}$, $\Delta''(s[\mathrm{p}]) = T_k$, and $\Delta''(s[\mathrm{q}]) = T'_k$. Furthermore, by Lems. A.15 and A.16, it also holds that for all $s[\mathrm{r}] \in \mathrm{dom}(\Delta') = \mathrm{dom}(\Delta'')$ with $\mathrm{r} \neq \mathrm{p}$ and $\mathrm{r} \neq \mathrm{q}$, $\Delta'(s[\mathrm{r}]) = \Delta''(s[\mathrm{r}])$. Observe that by $\Delta' \leqslant \Delta$, $\mathrm{unf}(\Delta(s[\mathrm{p}])) = \mathrm{q}\oplus\{\mathtt{m_i}(S''_i).T''_i\}_{i\in I_\mathrm{p}}$ where $I \subseteq I_\mathrm{p}$ and $\forall i \in I : S''_i \leqslant S_i$ and $T_i \leqslant T''_i$, and $\mathrm{unf}(\Delta(s[\mathrm{q}])) = \mathrm{p}\&\{\mathtt{m'_j}(S'''_j).T'''_j\}_{j\in J_\mathrm{q}}$ where $J_\mathrm{q} \subseteq J$ and $\forall j \in J_\mathrm{q} : S'_j \leqslant S'''_j$ and $T'_j \leqslant T'''_j$.

Now we apply Lem. A.24 on $\Delta \sqsubseteq_s G$, $\mathrm{unf}(\Delta(s[\mathrm{p}]))$, and $\mathrm{unf}(\Delta(s[\mathrm{q}]))$, to get $I \subseteq I_\mathrm{p} \subseteq J_\mathrm{q} \subseteq J$, and $\forall i \in I_\mathrm{p} : \mathtt{m}_i = \mathtt{m'}_i$ and $S'''_i \leqslant S''_i$. Consequently, we have $k \in I_\mathrm{p}$, $\mathtt{m}_k = \mathtt{m'}_k = \mathtt{m}$, and $S'''_k \leqslant S''_k$, which follows that there exists Δ''' such that $\Delta \xrightarrow{s[\mathrm{p}][\mathrm{q}]\mathtt{m}} \Delta'''$, $\Delta'''(s[\mathrm{p}]) = T''_k$, $\Delta'''(s[\mathrm{q}]) = T'''_k$, and for all $s[\mathrm{r}] \in \mathrm{dom}(\Delta) = \mathrm{dom}(\Delta') = \mathrm{dom}(\Delta'') = \mathrm{dom}(\Delta''')$ with $\mathrm{r} \neq \mathrm{p}$ and $\mathrm{r} \neq \mathrm{q}$, $\Delta(s[\mathrm{r}]) = \Delta'''(s[\mathrm{r}])$.

We are left to show that $\Delta'' \leqslant \Delta'''$, which is straightforward from $\Delta''(s[\mathrm{p}]) = T_k \leqslant T''_k = \Delta'''(s[\mathrm{p}])$, $\Delta''(s[\mathrm{q}]) = T'_k \leqslant T'''_k = \Delta'''(s[\mathrm{q}])$, and for all $s[\mathrm{r}] \in \mathrm{dom}(\Delta''') = \mathrm{dom}(\Delta'')$ with $\mathrm{r} \neq \mathrm{p}$ and $\mathrm{r} \neq \mathrm{q}$, $\Delta''(s[\mathrm{r}]) = \Delta'(s[\mathrm{r}]) \leqslant \Delta(s[\mathrm{r}]) = \Delta'''(s[\mathrm{r}])$. □

Lemma B.3. *Assume that $\forall s \in \Delta : \exists G_s : \Delta_s \sqsubseteq_s G_s$, $\Delta' \leqslant \Delta$ and $\Delta' \xrightarrow{\alpha} \Delta''$ with: $\alpha \in \{s[\mathrm{p}][\mathrm{q}]\mathtt{m} \mid \mathrm{p},\mathrm{q} \in \mathcal{R}\}$. Then, there is Δ''' such that $\Delta \xrightarrow{\alpha} \Delta'''$ and $\Delta'' \leqslant \Delta'''$.*

Proof. Apply Def. 3.5 and Lems. A.19 and B.2. □

Proposition B.4. *Assume that $\Delta \sqsubseteq_s G$, $\Delta' \leqslant \Delta$ and $\Delta' \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} \Delta''$, with: $\forall i \in 1...n : \alpha_i \in \{s[\mathrm{p}][\mathrm{q}]\mathtt{m_i} \mid \mathrm{p},\mathrm{q} \in \mathcal{R}\}$. Then, there is Δ''' such that $\Delta \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} \Delta'''$ and $\Delta'' \leqslant \Delta'''$.*

Proof. By induction on the number of transitions n in $\Delta' \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} \Delta''$. The base case ($n = 0$ transitions) is immediate: we have $\Delta' = \Delta''$, hence we conclude by taking $\Delta''' = \Delta$. In the inductive case with $n = m+1$ transitions, there is Δ''_0 such that $\Delta' \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_m} \Delta''_0 \xrightarrow{\alpha_n} \Delta''$. By the induction hypothesis, there is Δ'''_0 such that $\Delta \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_m} \Delta'''_0$ and $\Delta''_0 \leqslant \Delta'''_0$. By $\Delta \sqsubseteq_s G$, $\Delta \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_m} \Delta'''_0$, and applying completeness of association (Thm. 3.2) m times, there exists some G''' such that $\Delta'''_0 \sqsubseteq_s G'''$. Hence, by $\Delta'''_0 \sqsubseteq_s G'''$, $\Delta''_0 \xrightarrow{\alpha_n} \Delta''$, $\Delta''_0 \leqslant \Delta'''_0$, and Lem. B.2, there exists Δ''' such that $\Delta'''_0 \xrightarrow{\alpha_n} \Delta'''$ and $\Delta'' \leqslant \Delta'''$. Therefore, we have $\Delta \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} \Delta'''$ and $\Delta'' \leqslant \Delta'''$, which is the thesis. □

C. Proofs for § 4

C.1. Type System Properties

Lemma C.1 (Broadening). *If* $\Gamma \vdash P \triangleright \Delta$ *and* $\Delta \leqslant \Delta'$*, then* $\Gamma \vdash P \triangleright \Delta'$.

Proof. By induction on the derivation of $\Gamma \vdash P \triangleright \Delta$, we obtain a derivation that concludes $\Gamma \vdash P \triangleright \Delta'$ by inserting (possibly vacuous) instances of typing rule [T-Sub] (Fig. 8). □

Lemma C.2 (Terminated Typing Context). *If* $\mathrm{end}(\Delta)$*, then either* $\Delta = \emptyset$ *or* $\forall c \in \mathrm{dom}(\Delta) : \Delta(c) = \mathbf{end}$.

Proof. By inversion of [T-end] and $\mathrm{end}(\emptyset)$. □

Lemma C.3 (Typing Inversion). *Assume* $\Gamma \vdash P \triangleright \Delta'$. *Then there exists* $\Delta \leqslant \Delta'$ *such that*

(1) $P = \mathbf{0}$ *implies* $\mathrm{end}(\Delta)$;

(2) $P = \mathbf{def}\ X(x_1 : B_1, \ldots, x_n : B_n, y_1 : T_1, \ldots, y_m : T_m) = P'\ \mathbf{in}\ Q$ *implies:*

- *(i)* $\Gamma, X : B_1, \ldots, B_n, T_1, \ldots, T_m, x_1 : B_1, \ldots, x_n : B_n \vdash P' \triangleright \Delta, y_1 : T_1, \ldots, y_m : T_m$, *and*
- *(ii)* $\Gamma, X : B_1, \ldots, B_n, T_1, \ldots, T_m \vdash Q \triangleright \Delta$;

(3) $P = X\langle e_1, \ldots, e_n, c_1, \ldots, c_m \rangle$ *implies:*

- *(i)* $\Delta = \Delta'', c_1 : T_1, \ldots, c_m : T_m$, *and*
- *(ii)* $\Gamma \vdash e_1 : B_1, \ldots, e_n : B_n$, *and*
- *(iii)* $\mathrm{end}(\Delta'')$;

(4) $P = (\nu s : \Delta'')\, P'$ *implies:*

- *(i)* $s \notin \Delta$, *and*
- *(ii)* $\Delta'' = \{s[\mathtt{p}] : G \upharpoonright \mathtt{p}\}_{\mathtt{p} \in \mathrm{roles}(G)}$ *for some* G, *and*
- *(iii)* $\Gamma \vdash P' \triangleright \Delta, \Delta''$;

(5) $P = P_1 \mid P_2$ *implies:*

- *(i)* $\Delta = \Delta_1, \Delta_2$ *such that*
- *(ii)* $\Gamma \vdash P_1 \triangleright \Delta_1$, *and*
- *(iii)* $\Gamma \vdash P_2 \triangleright \Delta_2$;

(6) $P = c[\mathtt{q}] \& \{\mathtt{m}_i(z_i).P_i\}_{i \in I}$ *implies:*

- *(i)* $\Delta = \Delta'', c : \mathtt{q} \& \{\mathtt{m_i}(S_i).T_i\}_{i \in I}$ *such that*
- *(ii)* $\forall i \in I : \left\{ \begin{array}{ll} \Gamma, z_i : B \vdash P_i \triangleright \Delta'', c : T_i & \text{if } S_i = B \\ \Gamma \vdash P_i \triangleright \Delta'', z_i : T, c : T_i & \text{if } S_i = T \end{array} \right\}$;

(7) $P = c[\mathtt{q}] \oplus \mathtt{m}\langle e \rangle.P'$ *implies:*

- *(i)* $\Delta = \Delta'', c : \mathtt{q} \oplus \{\mathtt{m}(B).T\}$ *such that*
- *(ii)* $\Gamma \vdash e : B$, *and*
- *(iii)* $\Gamma \vdash P' \triangleright \Delta'', c : T$;

(8) $P = c[\mathtt{q}] \oplus \mathtt{m}\langle c' \rangle.P'$ *implies:*

- *(i)* $\Delta = \Delta'', c : \mathtt{q} \oplus \{\mathtt{m}(T').T\}, c' : T'$ *such that*

(ii) $\Gamma \vdash P' \triangleright \Delta'', c : T$;

(9) $P = \text{if } e \text{ then } Q_1 \text{ else } Q_2$ *implies:*

(i) $\Gamma \vdash e : \text{bool}$, *and*

(ii) $\Gamma \vdash Q_1 \triangleright \Delta$, *and*

(iii) $\Gamma \vdash Q_2 \triangleright \Delta$.

Proof. By inverting [T-Sub] and then applying induction on typing rules in Fig. 8. □

Lemma C.4 (Substitution)**.**

- *If* $\Gamma, x : B \vdash P \triangleright \Delta$ *and* $\Gamma \vdash v : B$, *then* $\Gamma \vdash P\{v/x\} \triangleright \Delta$.
- *If* $\Gamma \vdash P \triangleright \Delta, y : T$, *then* $\Gamma \vdash P\{s[\mathtt{p}]/y\} \triangleright \Delta, s[\mathtt{p}] : T$.

Proof. By induction on typing rules in Fig. 8. □

Lemma C.5 (Subject Congruence)**.** *Assume* $\Gamma \vdash P \triangleright \Delta$ *and* $P \equiv P'$. *Then,* $\Gamma \vdash P' \triangleright \Delta$.

Proof. By analysing the cases where $P \equiv P'$ holds, and by applying the inversion of the typing judgements $\Gamma \vdash P \triangleright \Delta$ and $\Gamma \vdash P' \triangleright \Delta$. □

Lemma C.6. *If* $\Gamma \vdash P \triangleright \Delta, \Delta'$, $\text{end}(\Delta')$, *and* $\Delta \leqslant \Delta''$, *then* $\Gamma \vdash P \triangleright \Delta''$.

Proof. By $P \equiv P \mid \mathbf{0}$ and Lem. C.5, we have $\Gamma \vdash P \mid \mathbf{0} \triangleright \Delta, \Delta'$. Inverting [T-**0**] and [T-|], we obtain $\Gamma \vdash P \triangleright \Delta$. Hence, the thesis follows by applying [T-Sub]. □

C.2. Subject Reduction

Theorem 4.1 (Subject Reduction via Association)**.** *Assume* $\Gamma \vdash P \triangleright \Delta$ *where* $\forall s \in \Delta : \exists G_s : \Delta_s \sqsubseteq_s G_s$. *If* $P \to P'$, *then* $\exists \Delta'$ *such that* $\Delta \to^* \Delta'$, $\Gamma \vdash P' \triangleright \Delta'$, *and* $\forall s \in \Delta' : \exists G'_s : G_s \to^* G'_s$ *and* $\Delta'_s \sqsubseteq_s G'_s$.

Proof. Let us recap the assumptions:

$$\Gamma \vdash P \triangleright \Delta \tag{C.1}$$
$$\forall s \in \Delta : \exists G_s : \Delta_s \sqsubseteq_s G_s \tag{C.2}$$
$$P \to P' \tag{C.3}$$

The proof proceeds by induction of the derivation of $P \to P'$, and when the reduction holds by rule [R-Ctx], with a further structural induction on the reduction context $\mathbb{C}$. Most cases hold by inverting the typing $\Gamma \vdash P \triangleright \Delta$ (Lem. C.3), and applying the induction hypothesis along with Lem. C.1.

- Case [R-⊕&-V]:

$$P = s[\mathtt{p}][\mathtt{q}]\&\{\mathtt{m}_i(z_i).P_i\}_{i \in I} \mid s[\mathtt{q}][\mathtt{p}]\oplus\mathtt{m}_k\langle e\rangle.Q$$
$$P' = P_k\{v/z_k\} \mid Q \quad (k \in I, e \downarrow v)$$
$$\text{(by inversion of [R-}\oplus\&\text{-V]}) \tag{C.4}$$

$$\Delta_\&, \Delta_\oplus \leqslant \Delta \quad \text{such that}$$
$$\Gamma \vdash s[\mathtt{p}][\mathtt{q}]\&\{\mathtt{m}_i(z_i).P_i\}_{i \in I} \triangleright \Delta_\&$$
$$\Gamma \vdash s[\mathtt{q}][\mathtt{p}]\oplus\mathtt{m}_k\langle e\rangle.Q \triangleright \Delta_\oplus$$
$$\text{(by (C.4) and Lem. C.3(5))} \tag{C.5}$$

$$\Delta_1, s[\mathtt{p}] : \mathtt{q}\&\{\mathtt{m_i}(S_i).T_i\}_{i\in I} \leqslant \Delta_\& \quad \text{such that}$$
$$\forall i \in I \begin{cases} \Gamma, z_i : B \vdash P_i \triangleright \Delta_1, s[\mathtt{p}] : T_i \text{ if } S_i = B \\ \Gamma \vdash P_i \triangleright \Delta_1, z_i : T, s[\mathtt{p}] : T_i \text{ if } S_i = T \end{cases}$$

(by (C.5) and Lem. C.3(6)) (C.6)

$$\Delta_2, s[\mathtt{q}] : \mathtt{p}\oplus\{\mathtt{m}(B).T\} \leqslant \Delta_\oplus \quad \text{such that}$$
$$\Gamma \vdash e : B \text{ and } \Gamma \vdash Q \triangleright \Delta_2, s[\mathtt{q}] : T$$

(by (C.5) and Lem. C.3(7)) (C.7)

Now, notice that:

$$\Delta_1, s[\mathtt{p}] : \mathtt{q}\&\{\mathtt{m_i}(S_i).T_i\}_{i\in I}, \Delta_2, s[\mathtt{q}] : \mathtt{p}\oplus\{\mathtt{m}(B).T\} = \Delta'' \leqslant \Delta$$

(by (C.5), (C.6), and (C.7)) (C.8)

$\forall s \in \Delta : \Delta''_s \sqsubseteq_s G_s$ (by (C.2), (C.8), and Lem. B.1) (C.9)

$k \in I$ and $S_k \leqslant B$ (by (C.8), (C.9) and Lem. A.24) (C.10)

$\Delta'' \to \Delta' = \Delta_1, s[\mathtt{p}] : T_k, \Delta_2, s[\mathtt{q}] : T$ (by (C.8), (C.10), and Def. 3.6) (C.11)

$\forall s \in \Delta : \exists G'_s : G_s \rightarrow^* G'_s$ and $\Delta'_s \sqsubseteq_s G'_s$ (by (C.9), (C.11) and Cor. A.27) (C.12)

We can now use Δ' to type P':

$\Gamma, z_k : B \vdash P_k \triangleright \Delta_1, s[\mathtt{p}] : T_k$ (by (C.10), and (C.6)) (C.13)

$\Gamma \vdash v : B$ (by (C.4) and (C.7)) (C.14)

$\Gamma \vdash P_k\{v/z_k\} \triangleright \Delta_1, s[\mathtt{p}] : T_k$ (by (C.13), (C.14) and Lem. C.4) (C.15)

$$\dfrac{\Gamma \vdash P_k\{v/z_k\} \triangleright \Delta_1, s[\mathtt{p}] : T_k \qquad \Gamma \vdash Q \triangleright \Delta_2, s[\mathtt{q}] : T}{\Gamma \vdash P' \triangleright \Delta'}\ [\text{T-}|]$$

(by (C.15), (C.7), (C.11), (C.12), and (C.4)) (C.16)

We conclude this case by showing that there exists some Δ''' that satisfies the statement:

$\Delta \to \Delta'''$ and $\Delta' \leqslant \Delta'''$ (by (C.2), (C.8), (C.11), and Lem. B.3) (C.17)

$\Gamma \vdash P' \triangleright \Delta'''$ (by (C.16) (C.17), and Lem. C.1) (C.18)

$\forall s \in \Delta''' : \exists G'''_s : G_s \rightarrow^* G'''_s$ and $\Delta'''_s \sqsubseteq_s G'''_s$ (by (C.2), (C.17), and Cor. A.27) (C.19)

- Case [R-⊕&-D]:

$$P = s[\mathtt{p}][\mathtt{q}]\&\{\mathtt{m}_i(z_i).P_i\}_{i\in I} \mid s[\mathtt{q}][\mathtt{p}]\oplus\mathtt{m}_k\langle s'[\mathtt{r}]\rangle.Q$$
$$P' = P_k\{s'[\mathtt{r}]/z_k\} \mid Q \quad (k \in I)$$

(by inversion of [R-⊕&-D]) (C.20)

$$\Delta_\&, \Delta_\oplus \leqslant \Delta \quad \text{such that}$$
$$\Gamma \vdash s[\mathtt{p}][\mathtt{q}]\&\{\mathtt{m}_i(z_i).P_i\}_{i\in I} \triangleright \Delta_\&$$
$$\Gamma \vdash s[\mathtt{q}][\mathtt{p}]\oplus\mathtt{m}_k\langle s'[\mathtt{r}]\rangle.Q \triangleright \Delta_\oplus$$

(by (C.20) and Lem. C.3(5)) (C.21)

$$\Delta_1, s[\mathtt{p}] : \mathtt{q}\&\{\mathtt{m_i}(S_i).T_i\}_{i\in I} \leqslant \Delta_\& \quad \text{such that} \quad \forall i \in I \begin{Bmatrix} \Gamma, z_i : B \vdash P_i \triangleright \Delta_1, s[\mathtt{p}] : T_i \text{ if } S_i = B \\ \Gamma \vdash P_i \triangleright \Delta_1, z_i : T, s[\mathtt{p}] : T_i \text{ if } S_i = T \end{Bmatrix}$$

(by (C.21) and Lem. C.3(6)) (C.22)

$$\Delta_2, s[\mathtt{q}] : \mathtt{p}\oplus\{\mathtt{m}(T_d).T\}, s'[\mathtt{r}] : T_d \leqslant \Delta_\oplus \quad \text{such that} \quad \Gamma \vdash Q \triangleright \Delta_2, s[\mathtt{q}] : T$$

(by (C.21) and Lem. C.3(8)) (C.23)

Now, notice that:

$$\Delta_1, s[\mathtt{p}] : \mathtt{q}\&\{\mathtt{m_i}(S_i).T_i\}_{i\in I}, \Delta_2, s[\mathtt{q}] : \mathtt{p}\oplus\{\mathtt{m}(T_d).T\}, s'[\mathtt{r}] : T_d = \Delta'' \leqslant \Delta$$

(by (C.21), (C.22), and (C.23)) (C.24)

$\forall s \in \Delta : \Delta''_s \sqsubseteq_s G_s$ (by (C.2), (C.24) and Lem. B.1) (C.25)

$k \in I$ and $S_k \leqslant T_d$ (by (C.24), (C.25) and Lem. A.24) (C.26)

$$\Delta \to \Delta' = \Delta_1, s[\mathtt{p}] : T_k, \Delta_2, s[\mathtt{q}] : T, s'[\mathtt{r}] : T_d$$

(by (C.24), (C.26) and Def. 3.6) (C.27)

$\forall s \in \Delta : \exists G'_s : G_s \to^* G'_s$ and $\Delta'_s \sqsubseteq_s G'_s$ (by (C.27), (C.25) and Cor. A.27) (C.28)

We can now use Δ' to type P':

$\Gamma \vdash P_k \triangleright \Delta_1, z_k : S_k, s[\mathtt{p}] : T_k$ (by (C.26), and (C.22)) (C.29)

$\Gamma \vdash P_k \triangleright \Delta_1, z_k : T_d, s[\mathtt{p}] : T_k$ (by (C.26), (C.29), and [T-Sub]) (C.30)

$\Delta_1, s[\mathtt{p}] : T_k, s'[\mathtt{r}] : T_d$ defined (by (C.23), (C.22), and (C.21)) (C.31)

$\Gamma \vdash P_k\{s'[\mathtt{r}]/z_k\} \triangleright \Delta_1, s[\mathtt{p}] : T_k, s'[\mathtt{r}] : T_d$ (by (C.30), (C.31), and Lem. C.4) (C.32)

$$\dfrac{\Gamma \vdash P_k\{s'[\mathtt{r}]/z_k\} \triangleright \Delta_1, s[\mathtt{p}] : T_k, s'[\mathtt{r}] : T_d \qquad \Gamma \vdash Q \triangleright \Delta_2, s[\mathtt{q}] : T}{\Gamma \vdash P' \triangleright \Delta'}\,[\text{T-}|]$$

(by (C.32), (C.23), (C.27), (C.28), and (C.20)) (C.33)

We conclude this case by showing that there exists some Δ''' that satisfies the statement:

$\exists \Delta''' : \Delta \to \Delta'''$ and $\Delta' \leqslant \Delta'''$ (by (C.2), (C.24), (C.27), and Lem. B.3) (C.34)

$\Gamma \vdash P' \triangleright \Delta'''$ (by (C.33) (C.34), and Lem. C.1) (C.35)

$\forall s \in \Delta''' : \exists G'''_s : G_s \to^* G'''_s$ and $\Delta'''_s \sqsubseteq_s G'''_s$ (by (C.2), (C.34), and Cor. A.27) (C.36)

- Case [R-Ctx]: By inversion of the rule (Lem. C.3) and Def. 2.2, we have to prove the statement in the following sub-cases:

1. $P = Q \mid R$ and $P' = Q' \mid R$ and $Q \to Q'$
2. $P = (\nu s')\, Q$ and $P' = (\nu s')\, Q'$ and $Q \to Q'$

3. $P = \mathbf{def}\ D\ \mathbf{in}\ Q$ and $P' = \mathbf{def}\ D\ \mathbf{in}\ Q'$ and $Q \to Q'$

Cases 1 and 3 are easily proved using the induction hypothesis. Therefore, here we focus on case 2.

$$\exists \Delta_1 \leqslant \Delta, G \text{ such that } \Delta_2 = \{s'[\mathtt{p}] : G \upharpoonright \mathtt{p}\}_{\mathtt{p} \in \text{roles}(G)},\ s' \notin \Delta_1, \text{ and } \Gamma \vdash Q \triangleright \Delta_1, \Delta_2 \quad \text{(by 2 and Lem. C.3(4))} \quad \text{(C.37)}$$

$$\Delta_2 \sqsubseteq_{s'} G \quad \text{(by (C.37), Def. 3.7, and Lem. A.3)} \quad \text{(C.38)}$$

$$\exists \Delta_3, \Delta_4, G' \text{ such that } \left\{\begin{array}{l} s' \notin \Delta_3 \\ \Delta_1 \to^* \Delta_3 \\ \Delta_2 \to^* \Delta_4 \\ \forall s \in \Delta_3 : \exists G'''_s : G_s \to^* G'''_s \text{ and } \Delta_{3s} \sqsubseteq_s G'''_s \\ \Delta_4 \sqsubseteq_{s'} G' \\ \Gamma \vdash Q' \triangleright \Delta_3, \Delta_4 \end{array}\right\}$$
$$\text{(by (C.37), (C.38), and inductive hypothesis)} \quad \text{(C.39)}$$

$$\exists \Delta_5 \text{ such that } \Delta_5 = \{s'[\mathtt{p}] : G' \upharpoonright \mathtt{p}\}_{\mathtt{p} \in \text{roles}(G')} \text{ and } \Gamma \vdash Q' \triangleright \Delta_3, \Delta_5 \quad \text{(by (C.39), Def. 3.7, and Lem. C.6)} \quad \text{(C.40)}$$

$$\dfrac{\Delta_5 = \{s'[\mathtt{p}] : G' \upharpoonright \mathtt{p}\}_{\mathtt{p} \in \text{roles}(G')} \qquad s' \notin \Delta_3 \qquad \Gamma \vdash Q' \triangleright \Delta_3, \Delta_5}{\Gamma \vdash P' \triangleright \Delta_3}\ [\text{T-}G\text{-}\nu]$$
$$\text{(by (C.39), (C.40) and 2)} \quad \text{(C.41)}$$

Hence, we obtain the thesis by (C.37), (C.39) and (C.41), Lem. B.3, Lem. C.1, and Cor. A.27.

- Case [R-X]: Follows from Lem. C.3(2), Lem. C.3(3), Lem. C.4, and Lem. C.1.
- Case [R-Err]: Trivial, as P is not typable.
- Case [R-≡]: Follows directly from induction hypothesis and Lem. C.5
- Cases [R-Cond-T] and [R-Cond-F]: Follow directly from Lem. C.3(9) and Lem. C.1. □

Theorem 4.2 (Subject Reduction)**.** *Assume* $\Gamma \vdash P \triangleright \Delta$*, where* $\forall s \in \Delta : \exists G_s : \Delta_s = \{s[\mathtt{p}] : G_s \upharpoonright \mathtt{p}\}_{\mathtt{p} \in \text{roles}(G_s)}$*. If* $P \to P'$*, then* $\exists \Delta'$ *such that* $\Gamma \vdash P' \triangleright \Delta'$*, and* $\forall s \in \Delta' : \exists G'_s : G_s \to^* G'_s$ *and* $\Delta'_s = \{s[\mathtt{p}] : G'_s \upharpoonright \mathtt{p}\}_{\mathtt{p} \in \text{roles}(G'_s)}$*.*

Proof. By Def. 3.7 and Lem. A.3, we have $\forall s \in \Delta : \exists G_s : \Delta_s \sqsubseteq_s G_s$. Then, by Thm. 4.1, it follows that $\exists \Delta'$ such that $\Delta \to^* \Delta'$, $\Gamma \vdash P' \triangleright \Delta'$, and $\forall s \in \Delta' : \exists G'_s : G_s \to^* G'_s$ and $\Delta'_s \sqsubseteq_s G'_s$.

Let Δ'' be the typing context defined by $\forall s \in \Delta' : \Delta''_s = \{s[\mathtt{p}] : G'_s \upharpoonright \mathtt{p}\}_{\mathtt{p} \in \text{roles}(G'_s)}$. It remains to show that $\Gamma \vdash P' \triangleright \Delta''$, which follows from $\Gamma \vdash P' \triangleright \Delta'$, Def. 3.7, and Lem. C.6. □

Corollary 4.3 (Type Safety)**.** *Assume* $\emptyset \vdash P \triangleright \emptyset$*. If* $P \to^* P'$*, then* P' *has no error.*

Proof. From the hypothesis $P \to^* P'$, we know that $P = P_0 \to P_1 \to \cdots \to P_n = P'$ (for some n). The proof proceeds by induction on n.

The base case for n=0 is straightforward: we have $P = P'$, thus P' is well-typed. Furthermore, since the term `err` is not typeable, P' cannot contain such a term.

In the inductive case for n = m+1, we already know (by the induction hypothesis) that P_m is well-typed. By applying Thm. 4.2, we can conclude that $P_{m+1} = P'$ is also well-typed and does not contain any `err` subterms. □

C.3. Session Fidelity

Theorem 4.4 (Session Fidelity via Association)**.** *Assume $\emptyset \vdash P \triangleright \Delta$, with $\Delta \sqsubseteq_s G$, $P \equiv \Pi_{p \in I} P_p$, and $\Delta = \bigcup_{p \in I} \Delta_p$ such that for each P_p: (1) $\emptyset \vdash P_p \triangleright \Delta_p$, and (2) either $P_p \equiv \mathbf{0}$, or P_p only plays p in s, by Δ_p. Then, $\Delta \rightarrow_s$ implies $\exists \Delta', G', P'$ such that $\Delta \rightarrow_s \Delta'$, $G \rightarrow G'$, $P \rightarrow^* P'$, and $\emptyset \vdash P' \triangleright \Delta'$, with $\Delta' \sqsubseteq_s G'$, $P' \equiv \Pi_{p \in I} P'_p$, and $\Delta' = \bigcup_{p \in I} \Delta'_p$ such that for each P'_p: (1) $\emptyset \vdash P'_p \triangleright \Delta'_p$, and (2) either $P'_p \equiv \mathbf{0}$, or P'_p only plays p in s, by Δ'_p.*

Proof. The proof structure is based on induction on the derivation of the reduction of Δ. We infer the contents of Δ, as well as the shape of P and its sub-processes P_p, showing that they can mimic the reduction of Δ. We focus on the case of $\Delta \xrightarrow{s[p][q]m} \Delta'$, while the other cases follow by applying the induction hypothesis.

- Case $\Delta \xrightarrow{s[p][q]m} \Delta'$: in this case, the process P_p playing role p in session s is a selection on $s[p]$ towards q (possibly within a process definition); while the process P_q playing role q in session s is a branching on $s[q]$ from p (possibly within a process definition). Therefore, by [R-⊕&-V] or [R-⊕&-D] in Fig. 3, P can correspondingly reduce to P' by transmitting either a value v or a channel endpoint $s'[p']$ from p to q in session s (possibly after a finite number of transitions under rule [R-X]). The resulting continuation process P' is typed by Δ'. The assertion that there exists G' such that $G \rightarrow G'$ and $\Delta' \sqsubseteq_s G'$ follows from $\Delta \xrightarrow{s[p][q]m} \Delta'$, $\Delta \sqsubseteq_s G$, and Thm. 3.2. □

Prop. C.7 below states that if a process P satisfies the assumptions of session fidelity (Thm. 4.4), then all its reductums will also satisfy these assumptions. In other words, if P enjoys session fidelity, so will all of its reductums.

Proposition C.7. *Assume $\emptyset \vdash P \triangleright \Delta$, where $\Delta \sqsubseteq_s G$, $P \equiv \Pi_{p \in I} P_p$, and $\Delta = \bigcup_{p \in I} \Delta_p$ such that, for each P_p, we have $\emptyset \vdash P_p \triangleright \Delta_p$. Further, assume that each P_p is either $\mathbf{0}$ (up to $\equiv$), or only plays p in s, by Δ_p. Then, $P \rightarrow P'$ implies $\exists \Delta', G'$ such that $\Delta \rightarrow_s^* \Delta'$ and $\emptyset \vdash P' \triangleright \Delta'$, with $\Delta' \sqsubseteq_s G'$, $P' \equiv \Pi_{p \in I} P'_p$, and $\Delta' = \bigcup_{p \in I} \Delta'_p$ such that, for each P'_p, we have $\emptyset \vdash P'_p \triangleright \Delta'_p$; furthermore, each P'_p is $\mathbf{0}$ (up to $\equiv$), or only plays p in s, by Δ'_p.*

Proof. Straightforward from the proof of Thm. 4.1, which accounts for all possible transitions from P to P', and in all cases yields the desired properties for its typing context Δ'. □

Theorem 4.5 (Session Fidelity)**.** *Assume $\emptyset \vdash P \triangleright \{s[p] : G\!\upharpoonright\! p\}_{p \in I}$, with $\mathrm{roles}(G) \subseteq I$ and $P \equiv \Pi_{p \in I} P_p$ such that for each P_p: (1) $\emptyset \vdash P_p \triangleright s[p] : G\!\upharpoonright\! p$, and (2) either $P_p \equiv \mathbf{0}$, or P_p only plays p in s, by $s[p] : G\!\upharpoonright\! p$. Then, $G \rightarrow$ implies $\exists G', P'$ such that $G \rightarrow G'$, $P \rightarrow^* P'$, and $\emptyset \vdash P' \triangleright \{s[p] : G'\!\upharpoonright\! p\}_{p \in I}$, with $\mathrm{roles}(G') \subseteq I$ and $P' \equiv \Pi_{p \in I} P'_p$ such that for each P'_p: (1) $\emptyset \vdash P'_p \triangleright s[p] : G'\!\upharpoonright\! p$, and (2) either $P'_p \equiv \mathbf{0}$, or P'_p only plays p in s, by $s[p] : G'\!\upharpoonright\! p$.*

Proof. Let $\Delta = \{s[p] : G\!\upharpoonright\! p\}_{p \in I}$ with $\mathrm{roles}(G) \subseteq I$. It follows directly that $\Delta = \bigcup_{p \in I} \Delta_p$, where, by Lem. A.14, $\Delta_p = s[p] : \mathbf{end}$ if $p \notin \mathrm{roles}(G)$. Then, by Def. 3.7, $\Delta \sqsubseteq_s G$.

Since $G \rightarrow$, by Thm. 3.1, $\Delta \rightarrow_s$. Moreover, by the assumptions and Thm. 4.4, $\exists \Delta', G', P'$ such that $\Delta \rightarrow_s \Delta'$, $G \rightarrow G'$, $P \rightarrow^* P'$, and $\emptyset \vdash P' \triangleright \Delta'$, with $\Delta' \sqsubseteq_s G'$, $P' \equiv \Pi_{p \in I} P'_p$, and $\Delta' = \bigcup_{p \in I} \Delta'_p$ such that for each P'_p: (1) $\emptyset \vdash P'_p \triangleright \Delta'_p$, and (2) either $P'_p \equiv \mathbf{0}$, or P'_p only plays p in s, by Δ'_p.

We are left to show that:

- $\mathrm{roles}(G') \subseteq I$: By $\Delta \rightarrow_s \Delta'$ and Lem. A.15, $\mathrm{dom}(\Delta') = \mathrm{dom}(\Delta) = \{s[p]\}_{p \in I}$. Moreover, since $\Delta' \sqsubseteq_s G'$, $\{s[p]\}_{p \in \mathrm{roles}(G')} \subseteq \mathrm{dom}(\Delta')$, which implies that $\mathrm{roles}(G') \subseteq I$.
- $\emptyset \vdash P' \triangleright \{s[p] : G'\!\upharpoonright\! p\}_{p \in I}$: By $\Delta' \sqsubseteq_s G'$, $\mathrm{dom}(\Delta') = \{s[p]\}_{p \in I}$, and Lem. A.14, $\Delta' \leqslant \{s[p] : G'\!\upharpoonright\! p\}_{p \in I}$. Moreover, by $\emptyset \vdash P' \triangleright \Delta'$ and [T-Sub], it follows that $\emptyset \vdash P' \triangleright \{s[p] : G'\!\upharpoonright\! p\}_{p \in I}$.
- $\emptyset \vdash P'_p \triangleright s[p] : G'\!\upharpoonright\! p$: Follows directly from $\Delta' = \bigcup_{p \in I} \Delta'_p \leqslant \{s[p] : G'\!\upharpoonright\! p\}_{p \in I}$, $\emptyset \vdash P'_p \triangleright \Delta'_p$, and [T-Sub].
- Either $P'_p \equiv \mathbf{0}$, or P'_p only plays p in s, by $s[p] : G'\!\upharpoonright\! p$: Follows directly from $\emptyset \vdash P'_p \triangleright s[p] : G'\!\upharpoonright\! p$, $\Delta'_p \leqslant s[p] : G'\!\upharpoonright\! p$, and the fact that either $P'_p \equiv \mathbf{0}$, or P'_p only plays p in s, by Δ'_p. □

C.4. Process Properties

Lemma C.8 (Process Deadlock-Freedom)**.** *Assume* $\emptyset \vdash P \triangleright \{s[\mathtt{p}] : G\!\upharpoonright\!\mathtt{p}\}_{\mathtt{p}\in\mathrm{roles}(G)}$*, where* $P \equiv \Pi_{\mathtt{p}\in\mathrm{roles}(G)} P_{\mathtt{p}}$ *and for each* $P_{\mathtt{p}}$*,* $\emptyset \vdash P_{\mathtt{p}} \triangleright s[\mathtt{p}] : G\!\upharpoonright\!\mathtt{p}$*. Further, assume that each* $P_{\mathtt{p}}$ *is either* $\mathbf{0}$ *(up to* $\equiv$*), or only plays* $\mathtt{p}$ *in* s*, by* $s[\mathtt{p}] : G\!\upharpoonright\!\mathtt{p}$*. Then,* P *is deadlock-free.*

Proof. Let $\Delta = \{s[\mathtt{p}] : G\!\upharpoonright\!\mathtt{p}\}_{\mathtt{p}\in\mathrm{roles}(G)}$. It follows directly that $\Delta = \bigcup_{\mathtt{p}\in\mathrm{roles}(G)} \Delta_{\mathtt{p}}$, where $\Delta_{\mathtt{p}} = s[\mathtt{p}] : G\!\upharpoonright\!\mathtt{p}$. By Def. 3.7, $\Delta \sqsubseteq_s G$. Moreover, by Thm. 3.3, Δ is s-deadlock-free.

Consider any P' such that $P \to^* P' \not\to$, with $P = P_0 \to P_1 \to \cdots \to P_n = P' \not\to$ (for some n), where each reduction $P_i \to P_{i+1}$ ($i \in 0...n-1$). By Prop. C.7, we know that each P_i is well-typed and its typing context Δ_i satisfies $\Delta \to_s^* \Delta_i$; moreover, P_i adheres to the single-session requirements of Thm. 4.4. Now, observe that since the process $P_n = P' \not\to$ cannot reduce further, by the contrapositive of Thm. 4.4, we obtain $\Delta_n \not\to_s$. Furthermore, since Δ is s-deadlock-free, by Def. 3.9, we have $\forall s[\mathtt{p}] \in \Delta_n$: $\Delta_n(s[\mathtt{p}]) = \mathbf{end}$. Therefore, by [T-$\mathbf{0}$], we have $P' \equiv \mathbf{0}$, which (by Def. 4.2(1)) is the statement. □

Lemma C.9 (Process Liveness)**.** *Assume* $\emptyset \vdash P \triangleright \{s[\mathtt{p}] : G\!\upharpoonright\!\mathtt{p}\}_{\mathtt{p}\in\mathrm{roles}(G)}$*, where* $P \equiv \Pi_{\mathtt{p}\in\mathrm{roles}(G)} P_{\mathtt{p}}$ *and for each* $P_{\mathtt{p}}$*,* $\emptyset \vdash P_{\mathtt{p}} \triangleright s[\mathtt{p}] : G\!\upharpoonright\!\mathtt{p}$*. Further, assume that each* $P_{\mathtt{p}}$ *is either* $\mathbf{0}$ *(up to* $\equiv$*), or only plays* $\mathtt{p}$ *in* s*, by* $s[\mathtt{p}] : G\!\upharpoonright\!\mathtt{p}$*. Then,* P *is live.*

Proof. Let $\Delta = \{s[\mathtt{p}] : G\!\upharpoonright\!\mathtt{p}\}_{\mathtt{p}\in\mathrm{roles}(G)}$. It follows directly that $\Delta = \bigcup_{\mathtt{p}\in\mathrm{roles}(G)} \Delta_{\mathtt{p}}$, where $\Delta_{\mathtt{p}} = s[\mathtt{p}] : G\!\upharpoonright\!\mathtt{p}$. By Def. 3.7, $\Delta \sqsubseteq_s G$. Moreover, by Thm. 3.3, Δ is s-live.

The proof proceeds by contradiction: assume that P is *not* live. Since (by hypothesis) each parallel component of P only plays one role $\mathtt{p}$ in session s, there are $P', \mathbb{C}, Q$ such that $P = P_0 \to P_1 \to \cdots \to P_n = P' \equiv \mathbb{C}[Q]$ where either:

- $Q = s[\mathtt{p}][\mathtt{q}] \oplus \mathtt{m}\langle w \rangle.Q'$ (for some $\mathtt{m}, w, Q'$), and $\nexists \mathbb{C}'$: $\mathbb{C} \rightsquigarrow^* \mathbb{C}'$ and $P' \to^* \mathbb{C}'[Q']$. By Prop. C.7, we know that each P_i is well-typed and its typing context Δ_i is such that $\Delta \to_s^* \Delta_i$; moreover, each P_i satisfies the single-session requirements of Thm. 4.4. Therefore, P' satisfies the single-session requirements of Thm. 4.4, and is typed by some Δ' such that $\Delta \to_s^* \Delta'$. Hence, by inversion of typing, Q is typed by some $\Delta'_{\mathtt{p}}$ (part of Δ') where $\Delta'_{\mathtt{p}}(s[\mathtt{p}])$ is a (possibly recursive) internal choice towards $\mathtt{q}$, including a choice $\mathtt{m}(S)$ (where S types the message payload w). Therefore, we have $\Delta' \xrightarrow{s[\mathtt{p}]:\mathtt{q}\oplus\mathtt{m}(S)}$.

  Now, recall that (for the sake of the proof by contradiction) there is no $\mathbb{C}'$ with $\mathbb{C} \rightsquigarrow^* \mathbb{C}'$ such that a reduction of P' reaches $\mathbb{C}'[Q']$; that is, the top-level selection of Q cannot be fired. Hence, there is at least one fair path beginning with Δ' that never fires a transmission label $s[\mathtt{p}][\mathtt{q}]\mathtt{m}'$ (for any $\mathtt{m}'$). But then, such a fair path starting from Δ' is not live, and furthermore, (by Def. 3.11) we obtain that Δ is *not* live, a desired contradiction;

- $Q = s[\mathtt{p}][\mathtt{q}] \& \{\mathtt{m}_i(x_i).Q'_i\}_{i\in I}$ (for some $I, \mathtt{m}_i, x_i, Q'_i$), and $\nexists \mathbb{C}', k \in I, u$: $\mathbb{C} \rightsquigarrow^* \mathbb{C}'$ and $P' \to^* \mathbb{C}'[Q'_k\{u/x_k\}]$. The proof is similar to the previous case, and reaches a similar contradiction. □

Theorem 4.6 (Process Deadlock-Freedom, Liveness)**.** *Assume* $\emptyset \vdash P \triangleright \{s[\mathtt{p}] : G\!\upharpoonright\!\mathtt{p}\}_{\mathtt{p}\in\mathrm{roles}(G)}$*, where* $P \equiv \Pi_{\mathtt{p}\in\mathrm{roles}(G)} P_{\mathtt{p}}$ *and for each* $P_{\mathtt{p}}$*,* $\emptyset \vdash P_{\mathtt{p}} \triangleright s[\mathtt{p}] : G\!\upharpoonright\!\mathtt{p}$*. Further, assume that each* $P_{\mathtt{p}}$ *is either* $\mathbf{0}$ *(up to* $\equiv$*), or only plays* $\mathtt{p}$ *in* s*, by* $s[\mathtt{p}] : G\!\upharpoonright\!\mathtt{p}$*. Then,* P *is deadlock-free and live.*

Proof. Directly from Lems. C.8 and C.9. □