

Relaxed Choices in Bottom-Up Asynchronous Multiparty Session Types

Ivan Prokić ✉ 

Faculty of Technical Sciences, University of Novi Sad, Serbia

Simona Prokić ✉ 

Faculty of Technical Sciences, University of Novi Sad, Serbia

Silvia Ghilezan ✉ 

Faculty of Technical Sciences, University of Novi Sad, Serbia

Mathematical Institute of the Serbian Academy of Sciences and Arts, Belgrade, Serbia

Alceste Scalas ✉ 

Technical University of Denmark, DK

Nobuko Yoshida ✉ 

University of Oxford, UK

Abstract

Asynchronous multiparty session types provide a formal model for expressing the behaviour of communicating processes and verifying that they correctly implement desired protocols. In the “bottom-up” approach to session typing, local session types are specified directly, and the properties of their composition (e.g. deadlock freedom and liveness) are checked and transferred to well-typed processes. This method allows expressing and verifying a broad range of protocols, but still has a key limitation: it only supports protocols where every send/receive operation is directed towards strictly one recipient/sender at a time. This makes the technique too restrictive for modelling some classes of protocols, e.g. those used in the field of federated learning.

This paper improves the session typing theory by extending the asynchronous “bottom-up” approach to support protocols where a participant can choose to send or receive messages to/from multiple other participants at the same time, rather than just one at a time. We demonstrate how this extension enables the modeling and verification of real-world protocols, including some used in federated learning. Furthermore, we introduce and formally prove safety, deadlock-freedom, liveness, and session fidelity properties for our session typing system, revealing interesting dependencies between these properties in the presence of a subtyping relation.

2012 ACM Subject Classification Theory of computation → Process calculi; Theory of computation → Type structures

Keywords and phrases Multiparty session types, π -calculus, type systems

1 Introduction

Asynchronous multiparty session types provide a formal approach to the verification of message-passing programs. The key idea is to express *protocols as types*, and use type checking to verify whether one or more communicating processes correctly implement some desired protocols. The original approach [9] is “top-down,” in the sense that it begins by specifying a *global type*, i.e., a choreographic formalisation of all the communications expected between the *participants* in the protocol; then, the global type is *projected* into a set of (local) session types, which can be used to type-check processes. This approach ensures that the projected session types interact correctly with each other — and this property is reflected by processes that are well-typed w.r.t. such projected session types. Many extensions to this “top-down” approach have appeared. For instance, extended models include explicit *connection actions* with a more relaxed form of *internal choice* (performed by a process when it decides which message to send to other participants) [10] — used by

other work [24, 22, 23] to express communication protocols utilized in distributed cloud and edge computing. The work of [15] extends this further by also relaxing the *external choice* in local types (performed by a process when it receives messages from other participants), while using a form of global types similar to previous work. In general, in the “top-down” approach to session typing, global types offer a formal yet intuitive way for describing choreographic behaviours. However, global types can also limit the expressiveness of the theory: many classes of real-world protocols (such as the federated learning protocol described below) do not fit within the constraints of global types languages in literature, and correctly extending such languages often requires considerable effort.

For this reason, an alternative “bottom-up” approach to session typing [21] removes the requirement of global types: instead, local session types are specified directly, and the properties of their composition (e.g. deadlock freedom and liveness) are checked and transferred to well-typed processes. While this method allows for verifying a broader range of protocols, it still inherits a key limitation of the original approach: a participant sending or receiving a message can communicate with strictly one participant at a time. To overcome this limitation, the bottom-up approach in [21] has been extended with the mixed choices in [17] but only for *synchronous* communications — which is still insufficient for several important applications, as we now illustrate.

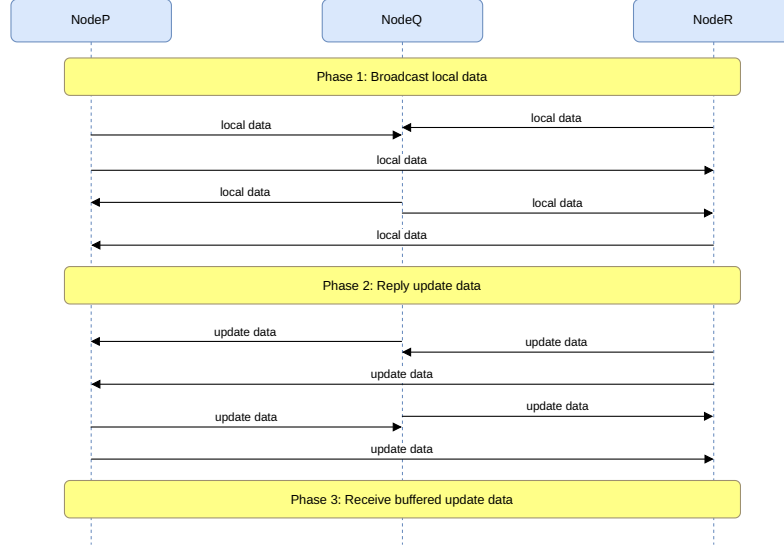
Modelling an asynchronous decentralised federated learning protocol. To motivate our work, let us consider a class of decentralised federated learning (DFL) protocols that rely on a fully connected network topology (in which direct links exist between all pairs of nodes) and use asynchronous communication [5, 28, 19, 20]. Fully connected networks are the most commonly used in DFL [5, 28], as they offer greater reliability and robustness compared to other topologies, such as star, ring, or random. Additionally, asynchronous communication—where no synchronization point exists—tends to converge more quickly and is well suited for DFL scenarios in which nodes have varying computational capabilities [5]. Concretely, we consider a single round of a *generic decentralised one-shot federated learning algorithm (FLA)* [19, 20] having three nodes, where there is no central point of control, depicted in the message sequence chart in Figure 1. Each participant in this algorithm follows the same behaviour divided in three phases. In *phase 1*, each participant sends its local data (i.e., a machine learning model) to all other participants. In *phase 2*, each participant receives the other participants’ local data, trains the algorithm, and sends back the updated data. Finally, in *phase 3*, each participant receives the updated data from other participants and aggregates the updates.

We may model the process for participant p with a value passing labeled π -calculus (following the approach of [17]) as follows — where ld and up are message labels meaning “local data” and “update,” \sum represents a choice, and $q!ld\langle \dots \rangle$ (resp. $q?ld(\dots)$) means “send (resp. receive) the message ld to (resp. from) participant q .”

$$P = \overbrace{q!ld\langle data \rangle . r!ld\langle data \rangle}^{\text{Phase 1}} . \sum \left\{ \overbrace{\left\{ \begin{array}{l} q?ld(x) . q!up\langle data \rangle . r?ld(y) . r!up\langle data \rangle . P_1 \\ r?ld(y) . r!up\langle data \rangle . q?ld(x) . q!up\langle data \rangle . P_1 \end{array} \right\}}^{\text{Phase 2}} \right\}$$

where

$$P_1 = \sum \left\{ \overbrace{\left\{ \begin{array}{l} q?up(x) . r?up(y) \\ r?up(y) . q?up(x) \end{array} \right\}}^{\text{Phase 3}} \right\}$$



■ **Figure 1** An execution of the generic decentralised one-shot federated learning algorithm.

In phase 1, p 's process P above sends its local data to q and then to r ; In phase 2, participant p receives local data and replies the update up to q and also r ; this can happen in two possible orders depending on whether the data is received from q or r first, and this is represented by the two branches of the choice \sum . Finally, in phase 3 (in process P_1), p receives update data from q and r in any order (again, this is represented with two choice branches).

To the best of our knowledge, we may attempt to model this example directly only using the session-type approaches of [15] and [17] — but with either of them, we encounter roadblocks. The model in [15] relies on the existence of a global type — but the global type language in [15] and in existing published work cannot express our decentralised FLA example, in which no participant is guiding the protocol. Instead, [17] does not require global types — but it relies on a process calculus with *synchronous* communication semantics, i.e., where messages can only be sent when their recipient is ready to receive them. Under synchronous semantics, this FLA protocol is immediately stuck, since all participants in the phase 1 are trying to send their local data at the same time, and the recipients are not (yet) ready to receive. In this work we overcome these limitations.

Contributions and outline of the paper. We present the first “bottom-up” asynchronous session typing theory that supports input/output operations directed towards multiple participants. We demonstrate that our approach enables the modelling and verification of processes implementing real-world protocols that, thus far, were not supported by session typing theories — e.g., the federated learning protocol in Figure 1. Furthermore, we formalise and prove safety, deadlock-freedom, liveness, and session fidelity properties for well-typed processes, revealing interesting dependencies between these properties in the presence of a subtyping relation.

The paper is organized as follows: Section 2 introduces our asynchronous session calculus; Section 3 presents the types and the subtyping relation; Section 4 provides the type system and our main results; and finally, Section 5 summarizes the paper with a discussion of related and future work.

$v ::= x, y, z, \dots \mid 1, 2, \dots \mid \text{true}, \text{false}$	(variables, values)
$\mathcal{M} ::= p \triangleleft P \mid p \triangleleft h \mid \mathcal{M} \mid \mathcal{M}$	(participant, parallel)
$h ::= \emptyset \mid (q, \ell(v)) \mid h \cdot h$	(empty, message, concatenation)
$P, Q ::= \sum_{i \in I} p_i ? \ell_i(x_i).P_i \mid \sum_{i \in I} p_i ! \ell_i \langle v_i \rangle . P_i$	(external choice, internal choice)
if e then P else Q	(conditional)
$X \mid \mu X.P \mid 0$	(variable, recursion, inaction)

■ **Figure 2** Syntax of sessions, processes, and queues.

2 The Calculus

In this section we present the syntax (Section 2.1) and the operational semantics (Section 2.2) of our extended asynchronous multiparty sessions calculus. We define a core calculus in the style of [8], omitting session creation and session delegation, and extend it with separate choice constructs from [17].

2.1 Syntax

We define the syntax of our asynchronous calculus in Table 2. Values can be either variables (x, y, z) or constants (positive integers or booleans).

Our **asynchronous multiparty sessions** (ranged over by $\mathcal{M}, \mathcal{M}', \dots$) represent a parallel composition of **participants** (ranged over by p, q, \dots) assigned with their **process** P and **output message queue** h (notation: $p \triangleleft P \mid p \triangleleft h$). Here, $p \triangleleft h$ denotes h is the output message queue of participant p , and the **queued message** $(q, \ell(v))$ represents that participant p has sent message labeled ℓ with payload v to q . In the syntax of processes, the **external choice** $\sum_{i \in I} p_i ? \ell_i(x_i).P_i$ denotes receiving from participant p_i a message labelled ℓ_i with a value that should replace variable x_i in P_i , for any $i \in I$. The **internal choice** $\sum_{i \in I} p_i ! \ell_i \langle v_i \rangle . P_i$ denotes sending to participant p_i a message labelled ℓ_i with value v_i , and then continuing as P_i , for any $i \in I$; in Section 2.2 we will see that, when the internal choice has more than one branch, then one of them is picked nondeterministically. In both external and internal choices, we assume $(p_i, \ell_i) \neq (p_j, \ell_j)$, for all $i, j \in I$, such that $i \neq j$. The **conditional** construct, process **variable**, **recursion**, and **inaction** 0 are standard; we will sometimes omit writing 0 . As usual, we assume that recursion is guarded, i.e., in the process $\mu X.P$, the variable X can occur in P only under an internal or external choice.

Our model allows for representing interesting communication patterns that are not supported in the standard session type theory. This opens the way for extending the session type-like theory for many real-life communication protocols, such as the ones in federated learning. For instance, the standard multiparty session type theory cannot express arbitrary order of message arrivals that take place at an federated learning algorithm (FLA) instance [19, 20]. We show how our calculus can be used to model the two federated learning protocols presented in the paper [19, 20]: one centralized (Example 2) and one decentralized (Example 3). First, we set up some notation.

We define a **concurrent input** macro $\|R_j\|_{j \in \{i\}}.Q$ to represent a process that awaits a series of incoming messages arriving in arbitrary order:

$$R ::= p ? l(x) \mid p ? l(x).R^{?!} \quad R^{?!} ::= p ? l(x) \mid p ! l \langle v \rangle \mid p ? l(x).R^{?!} \mid p ! l \langle v \rangle . R^{?!}$$

$$\|R_i\|_{i \in I}.Q = \sum_{i \in I} R_i. \|R_j\|_{j \in I \setminus \{i\}}.Q \quad \text{and} \quad \|R_j\|_{j \in \{i\}}.Q = R_i.Q$$

where R_i are all process prefixes starting with an input, possibly followed by other inputs or outputs. The concurrent input $\|R_i\|_{i \in I}.Q$ specifies an external choice of any input-prefixed R_i , for $i \in I$, followed by another choice of input-prefixed process with an index from $I \setminus \{i\}$; the composition continues until all $i \in I$ are covered, with process Q added at the end. This way, concurrent input process $\|R_i\|_{i \in I}.Q$ can perform the actions specified by any R_i (for $i \in I$) in an arbitrary order, depending on which inputs become available first; afterwards, process Q is always executed. (See Example 1 below.)

► **Example 1** (Concurrent input). Using our concurrent input macro we may represent the FLA process P_1 from Section 1 with

$$P_1 = \|q?up(y), r?up(x)\|$$

► **Example 2** (Modelling a centralised FL protocol implementation). As specified in [19, 20], the *generic centralized one-shot federated learning protocol* is implemented with one server and $n-1$ clients. The protocol starts with server broadcasting its local data to all the clients. Then the server receives the update data from all the clients. We may model an implementation of this protocol as a session $\mathcal{M} = \prod_{i \in I}(\mathbf{p}_i \triangleleft P_i \mid \mathbf{p}_i \triangleleft \emptyset)$, where $I = \{1, 2, \dots, n\}$, and where \mathbf{p}_1 plays the role of the server, while the rest of participants are clients, defined with:

$$\begin{aligned} P_1 &= \mathbf{p}_2!ld\langle data \rangle \dots \mathbf{p}_n!ld\langle data \rangle. \| \mathbf{p}_2?upd(x_2), \dots, \mathbf{p}_n?upd(x_n) \| \\ P_i &= \mathbf{p}_1?ld(x). \mathbf{p}_1!upd\langle data \rangle \quad \text{for } i = 2, \dots, n \end{aligned}$$

Notice that, after sending the data to all the clients, the server P_1 then receives the updates from all of them in an arbitrary order. The clients first receive the data and then reply the update back to the server.

► **Example 3** (Modelling a decentralised FL protocol implementation). As specified in [19, 20], an implementation of the *generic decentralized one-shot federated learning protocol* comprises n participant processes acting both as servers and clients. (The protocol with $n = 3$ is given in Figure 1). The protocol starts with each participant acting as a server, and sending their local data to all other participants. Then, each participant acts as a client and receives the data and replies the update back. Finally, each participant receives the updated data from all other participants. We may model an implementation of this protocol with a session $\mathcal{M} = \prod_{i \in I}(\mathbf{p}_i \triangleleft P_i \mid \mathbf{p}_i \triangleleft \emptyset)$, where $I = \{1, 2, \dots, n\}$, and where process P_1 is defined as follows: (the rest of the processes are defined analogously)

$$\begin{aligned} P_1 &= \mathbf{p}_2!ld\langle data \rangle \dots \mathbf{p}_n!ld\langle data \rangle. \| \mathbf{p}_2?ld(x_2). \mathbf{p}_2!upd\langle data \rangle, \dots, \mathbf{p}_n?ld(x_n). \mathbf{p}_n!upd\langle data \rangle \| \\ &\quad \| \mathbf{p}_2?upd(y_2), \dots, \mathbf{p}_n?upd(y_n) \| \end{aligned}$$

Process P_1 specifies that participant \mathbf{p}_1 first sending local data to all other participants. Then, \mathbf{p}_1 receives local data from all other participants and replies the update concurrently (i.e., in an arbitrary order). Finally, \mathbf{p}_1 receives the updates from all other participants, again in an arbitrary order.

2.2 Session Reductions and Properties

The **reduction relation** for our process calculus is defined in Table 3. Rule [R-SEND] specifies sending one of the messages from the internal choice, while the other choices are discarded; the message ℓ_k with payload v_k sent to participant \mathbf{q}_k is appended to the participant \mathbf{p} 's output queue, and \mathbf{p} 's process becomes the continuation P_k . Dually, rule [R-RCV] defines how

$$\begin{aligned}
[\text{R-SEND}] \quad & p \triangleleft \sum_{i \in I} q_i ! \ell_i(v_i).P_i \mid p \triangleleft h_p \mid \mathcal{M} \xrightarrow{p:q_k ! \ell_k} p \triangleleft P_k \mid p \triangleleft h_p \cdot (q_k, \ell_k(v_k)) \mid \mathcal{M} \quad (k \in I) \\
[\text{R-RCV}] \quad & p \triangleleft \sum_{i \in I} q_i ? \ell_i(x_i).P_i \mid p \triangleleft h_p \mid q \triangleleft Q \mid q \triangleleft (p, \ell(v)) \cdot h \mid \mathcal{M} \xrightarrow{p:q ? \ell} p \triangleleft P_k\{v/x_k\} \mid p \triangleleft h_p \mid q \triangleleft Q \mid q \triangleleft h \mid \mathcal{M} \quad (\exists k \in I : (q, \ell) = (q_k, \ell_k)) \\
[\text{R-COND-T}] \quad & p \triangleleft \text{if true then } P \text{ else } Q \mid p \triangleleft h \mid \mathcal{M} \xrightarrow{p:\text{if}} p \triangleleft P \mid p \triangleleft h \mid \mathcal{M} \\
[\text{R-COND-F}] \quad & p \triangleleft \text{if false then } P \text{ else } Q \mid p \triangleleft h \mid \mathcal{M} \xrightarrow{p:\text{if}} p \triangleleft Q \mid p \triangleleft h \mid \mathcal{M} \\
[\text{R-STRUCT}] \quad & \mathcal{M}_1 \Rightarrow \mathcal{M}'_1 \text{ and } \mathcal{M}'_1 \longrightarrow \mathcal{M}'_2 \text{ and } \mathcal{M}'_2 \Rightarrow \mathcal{M}_2 \implies \mathcal{M}_1 \longrightarrow \mathcal{M}_2
\end{aligned}$$

■ **Figure 3** Reduction relation on sessions.

$$\begin{aligned}
& h_1 \cdot (q_1, \ell_1(v_1)) \cdot (q_2, \ell_2(v_2)) \cdot h_2 \equiv h_1 \cdot (q_2, \ell_2(v_2)) \cdot (q_1, \ell_1(v_1)) \cdot h_2 \quad (\text{if } q_1 \neq q_2) \\
& \emptyset \cdot h \equiv h \quad h \cdot \emptyset \equiv h \quad h_1 \cdot (h_2 \cdot h_3) \equiv (h_1 \cdot h_2) \cdot h_3 \quad \mu X.P \Rightarrow P\{\mu X.P/X\} \\
& p \triangleleft 0 \mid p \triangleleft \emptyset \mid \mathcal{M} \equiv \mathcal{M} \quad \mathcal{M}_1 \mid \mathcal{M}_2 \equiv \mathcal{M}_2 \mid \mathcal{M}_1 \quad (\mathcal{M}_1 \mid \mathcal{M}_2) \mid \mathcal{M}_3 \equiv \mathcal{M}_1 \mid (\mathcal{M}_2 \mid \mathcal{M}_3) \\
& P \Rightarrow Q \text{ and } h_1 \equiv h_2 \implies p \triangleleft P \mid p \triangleleft h_1 \mid \mathcal{M} \Rightarrow p \triangleleft Q \mid p \triangleleft h_2 \mid \mathcal{M}
\end{aligned}$$

■ **Figure 4** Structural (pre)congruence rules for queues, processes, and sessions.

a participant p can receive a message, directed towards p with a supported label, from the head of the output queue of the sender q ; after the reduction, the message is removed from the queue and the received message then substitutes the placeholder variable x_k in the continuation process P_k . Observe that, by rules $[\text{R-SEND}]$ and $[\text{R-RCV}]$, output queues are used in FIFO order. Rules $[\text{R-COND-T}]$ and $[\text{R-COND-F}]$ define how to reduce a conditional process: the former when the condition is true, the latter when it is false. Rule $[\text{R-STRUCT}]$ closes the reduction relation under pre-congruence relation, combining the approach of [27] with [8].

The **structural pre-congruence** relation \Rightarrow is defined in a standard way in Table 4. Notice that the congruence \equiv for queues allows the reordering of messages with different receivers [8]: this way, a participant q_2 can always receive the oldest queued message directed to q_2 , even if that message is sent and queued after another message directed to $q_1 \neq q_2$. Also notice that the pre-congruence \Rightarrow for processes allows unfolding recursive processes, while folding is not allowed [27]. The reason is that folding and unfolding are type-agnostic operations (types and sorts are introduced in Section 3); through folding, we may inadvertently identify two variables to which the type system (cf. Section 4) assigns different sorts.

In Definition 4 below we follow the approach of [8] and define how “good” sessions are expected to run by using behavioural properties. We define three behavioural properties. The first one is *safety*, ensuring that a session never has mismatches between the message labels supported by external choices and the labels of incoming messages; since in our sessions one participant can choose to receive messages from multiple senders at once, our definition of safety requires external choices to support all possible message labels from all senders’ queues. The *deadlock freedom* property requires that a session can get stuck (cannot reduce further) only in case it terminates. The *liveness* property ensures all pending inputs eventually receive a message and all queued messages are eventually received, if reduction steps are performed in a *fair* manner. Our fair and live properties for session types with standard choices matches the liveness defined in [8, Definition 2.2].

► **Definition 4** (Session behavioral properties). A *session path* is a (possibly infinite) sequence of sessions $(\mathcal{M}_i)_{i \in I}$, where $I = \{0, 1, 2, \dots, n\}$ (or $I = \{0, 1, 2, \dots\}$) is a set of consecutive natural numbers, and, $\forall i \in I \setminus \{n\}$ (or $\forall i \in I$), $\mathcal{M}_i \longrightarrow \mathcal{M}_{i+1}$. We say that a path $(\mathcal{M}_i)_{i \in I}$

is **safe** iff, $\forall i \in I$:

(SS) if $\mathcal{M}_i \Rightarrow p \triangleleft \sum_{j \in J} q_j ? \ell_j(x_j).P_j \mid p \triangleleft h \mid q \triangleleft Q \mid q \triangleleft h_q \mid \mathcal{M}'$ with $q \in \{q_j\}_{j \in J}$ and $h_q \equiv (p, \ell(v)) \cdot h'_q$, then $(q, \ell) = (q_j, \ell_j)$, for some $j \in J$

We say that a path $(\mathcal{M}_i)_{i \in I}$ is **deadlock-free** iff:

(SD) if $I = \{0, 1, 2, \dots, n\}$ and $\mathcal{M}_n \not\rightarrow$ then $\mathcal{M}_n \Rightarrow p \triangleleft \mathbf{0} \mid p \triangleleft \emptyset$

We say that a path $(\mathcal{M}_i)_{i \in I}$ is **fair** iff, $\forall i \in I$:

(SF1) whenever $\mathcal{M}_i \xrightarrow{p:q!\ell} \mathcal{M}'$, then $\exists k, j$ such that $I \ni k \geq i$ and $\mathcal{M}_k \xrightarrow{p:q_j!\ell_j} \mathcal{M}_{k+1}$

(SF2) whenever $\mathcal{M}_i \xrightarrow{p:q?\ell} \mathcal{M}'$, $\exists k, j$ such that $I \ni k \geq i$ and $\mathcal{M}_k \xrightarrow{p:q_j?\ell_j} \mathcal{M}_{k+1}$

(SF3) whenever $\mathcal{M}_i \xrightarrow{p:\text{if}} \mathcal{M}'$, $\exists k$ such that $I \ni k \geq i$ and $\mathcal{M}_k \xrightarrow{p:\text{if}} \mathcal{M}_{k+1}$

We say that a session path $(\mathcal{M}_i)_{i \in I}$ is **live** iff, $\forall i \in I$:

(SL1) if $\mathcal{M}_i \Rightarrow p \triangleleft \text{if } v \text{ then } P \text{ else } Q \mid p \triangleleft h \mid \mathcal{M}'$, then $\exists k$ such that $I \ni k \geq i$ and, for some \mathcal{M}'' , we have either $\mathcal{M}_k \Rightarrow p \triangleleft P \mid p \triangleleft h \mid \mathcal{M}''$ or $\mathcal{M}_k \Rightarrow p \triangleleft Q \mid p \triangleleft h \mid \mathcal{M}''$

(SL2) if $\mathcal{M}_i \Rightarrow p \triangleleft \sum_{j \in J} q_j ! \ell_j(v_j).P_j \mid p \triangleleft h \mid \mathcal{M}'$, then $\exists k$ such that $I \ni k \geq i$ and $\mathcal{M}_k \xrightarrow{p:q_j!\ell_j} \mathcal{M}_{k+1}$, for some $j \in J$

(SL3) if $\mathcal{M}_i \Rightarrow p \triangleleft P \mid p \triangleleft (q, \ell(v)) \cdot h \mid \mathcal{M}'$, then $\exists k$ such that $I \ni k \geq i$ and $\mathcal{M}_k \xrightarrow{q:p?\ell} \mathcal{M}_{k+1}$

(SL4) if $\mathcal{M}_i \Rightarrow p \triangleleft \sum_{j \in J} q_j ? \ell_j(x_j).P_j \mid p \triangleleft h \mid \mathcal{M}'$, then $\exists k$ such that $I \ni k \geq i$, and $\mathcal{M}_k \xrightarrow{p:q?\ell} \mathcal{M}_{k+1}$, where $(q, \ell) \in \{(q_j, \ell_j)\}_{j \in J}$

We say that a session \mathcal{M} is **safe/deadlock-free** iff all paths beginning with \mathcal{M} are safe/deadlock-free. We say that a session \mathcal{M} is **live** iff all fair paths beginning with \mathcal{M} are live.

Like in standard session types [21], in our calculus safety does not imply liveness nor deadlock-freedom (Example 5). Also, liveness implies deadlock-freedom, since liveness requires that session cannot get stuck before all external choices are performed and queued messages are eventually received. The converse is not true: e.g., a session in which two participants recursively exchange messages is deadlock free, even if a third participant waits forever to receive a message (but this third participant would make the session non-live). Such a session is deadlock-free, since it never gets stuck, but is not live, since in any fair path the third participant never fires its actions.

However, *unlike* standards session types, in our calculus liveness does *not* imply safety: this is illustrated in Example 6 below.

► **Example 5** (A safe but non-live session). Consider session

$$\begin{aligned} \mathcal{M} = & p \triangleleft \sum \{q ? \ell_1(x).r ? \ell_2(y), r ? \ell_2(x), r ? \ell_3(x)\} \mid p \triangleleft \emptyset \\ & \mid q \triangleleft \mathbf{0} \mid q \triangleleft (p, \ell_1(v_1)) \mid r \triangleleft \mathbf{0} \mid r \triangleleft (p, \ell_2(v_2)) \end{aligned}$$

In the session, participant p is ready to receive an input either from q or r , which, in turn, both have enqueued messages for p . The labels of enqueued messages are safely supported in p 's receive. Now, session \mathcal{M} has two possible reductions, where p receives:

1. either from q , where $\mathcal{M} \rightarrow p \triangleleft r ? \ell_2(y) \mid p \triangleleft \emptyset \mid r \triangleleft \mathbf{0} \mid r \triangleleft (p, \ell_2(v_2)) \rightarrow p \triangleleft \mathbf{0} \mid p \triangleleft \emptyset$;
2. or from r , in which case $\mathcal{M} \rightarrow q \triangleleft \mathbf{0} \mid q \triangleleft (p, \ell_1(v_1))$.

Hence, session \mathcal{M} is safe, since in all reductions inputs and matching enqueued messages have matching labels. However, \mathcal{M} is not live, since the second path above starting with \mathcal{M} is not live, for r 's output queue contains an orphan message that cannot be received. Notice also that \mathcal{M} is not deadlock-free since the final session in the second path cannot reduce further but is not equivalent to a terminated session.

► **Example 6** (A live but unsafe session). Consider the following session:

$$\begin{aligned} \mathcal{M}' = & \mathbf{p} \triangleleft \sum \{ \mathbf{q} ? \ell_1(x).r ? \ell_2(y), r ? \ell_3(x) \} \mid \mathbf{p} \triangleleft \emptyset \\ & \mid \mathbf{q} \triangleleft \mathbf{0} \mid \mathbf{q} \triangleleft (\mathbf{p}, \ell_1(v_1)) \mid \mathbf{r} \triangleleft \mathbf{0} \mid \mathbf{r} \triangleleft (\mathbf{p}, \ell_2(v_2)) \end{aligned}$$

The session \mathcal{M}' is not safe since the message in r 's output queue has label ℓ_2 which is not supported in \mathbf{p} 's external choice (that only supports label ℓ_3 for receiving from r). However, \mathcal{M}' has a single reduction path where \mathbf{p} receives from \mathbf{q} , and then \mathbf{p} receives ℓ_2 from r 's output queue; then, the session ends with all processes being $\mathbf{0}$ and all queues empty, which implies \mathcal{M}' is live (and thus, also deadlock-free).

3 Types and Typing Environments

This section introduces syntax of our (local) session types (Definition 7) which are a blend of asynchronous multiparty session types [8] and the separated choice multiparty sessions (SCMP) types from [17]. Then, we formalise typing contexts and their properties (Section 3.1), and subtyping (Section 3.2). These definitions and results will be the basis of our typing system (Section 4).

► **Definition 7.** The *sorts* S and (*local*) *session types* T are defined as:

$$S ::= \mathbf{nat} \mid \mathbf{bool} \quad T ::= \sum_{i \in I} \mathbf{p}_i ! \ell_i \langle S_i \rangle . T_i \mid \sum_{i \in I} \mathbf{p}_i ? \ell_i \langle S_i \rangle . T_i \mid \mathbf{end} \mid \mu \mathbf{t} . T \mid \mathbf{t}$$

where $(\mathbf{p}_i, \ell_i) \neq (\mathbf{p}_j, \ell_j)$, for all $i, j \in I$ such that $i \neq j$. The *queue types* σ and *typing environments* Γ are defined as:

$$\sigma ::= \epsilon \mid \mathbf{p} ! \ell \langle S \rangle \mid \sigma \cdot \sigma \quad \Gamma ::= \emptyset \mid \Gamma, \mathbf{p} : (\sigma, T)$$

Sorts are the types of values, which can be natural numbers (\mathbf{nat}) or booleans (\mathbf{bool}). The *internal choice* session type $\sum_{i \in I} \mathbf{p}_i ! \ell_i \langle S_i \rangle . T_i$ describes an output of message ℓ_i with sort S_i towards participant \mathbf{p}_i and then evolving to type T_i , for some $i \in I$. Similarly, $\sum_{i \in I} \mathbf{p}_i ? \ell_i \langle S_i \rangle . T_i$ stands for *external choice*, i.e., receiving a message ℓ_i with sort S_i from participant \mathbf{p}_i , for some $i \in I$. The type \mathbf{end} denotes the terminated session type, $\mu \mathbf{t} . T$ is a recursive type, and \mathbf{t} is a recursive type variable. We assume all recursions being guarded and a form of Barendregt convention: every $\mu \mathbf{t} . T$ binds a syntactically distinct \mathbf{t} .

The *queue type* represents the type of the messages contained in an output queue; it can be empty (ϵ), or it can contain message ℓ with sort S for participant \mathbf{p} ($\mathbf{p} ! \ell \langle S \rangle$), or it can be a concatenation of two queue types ($\sigma \cdot \sigma$). The *typing environment* assigns a pair of queue session type to a participant ($\mathbf{p} : (\sigma, T)$). We use $\Gamma(\mathbf{p})$ to denote the type that Γ assigns to \mathbf{p} .

3.1 Typing Environment Reductions and Properties

Now we define typing environment reductions, which relies on a structural congruence relation \equiv over session types and queue types. For the session types, we define \equiv as the least congruence such that $\mu \mathbf{t} . T \equiv T\{\mu \mathbf{t} . T / \mathbf{t}\}$. For the queue types, we define \equiv to allow reordering

messages that have different recipients (similarly to the congruence for message queues in Figure 4). More precisely, \equiv is defined as the least congruence satisfying:

$$\begin{aligned} \sigma \cdot \epsilon &\equiv \epsilon \cdot \sigma \equiv \sigma & \sigma_1 \cdot (\sigma_2 \cdot \sigma_3) &\equiv (\sigma_1 \cdot \sigma_2) \cdot \sigma_3 \\ \sigma \cdot p_1!l_1\langle S_1 \rangle \cdot p_2!l_2\langle S_2 \rangle \cdot \sigma' &\equiv \sigma \cdot p_2!l_2\langle S_2 \rangle \cdot p_1!l_1\langle S_1 \rangle \cdot \sigma' \quad (\text{if } p_1 \neq p_2) \end{aligned}$$

Pairs of queue/session types are related via structural congruence $(\sigma_1, T_1) \equiv (\sigma_2, T_2)$ iff $\sigma_1 \equiv \sigma_2$ and $T_1 \equiv T_2$. By extension, we also define congruence for typing environments (allowing additional entries being pairs of types of empty queues and terminated sessions): $\Gamma \equiv \Gamma'$ iff $\forall p \in \text{dom}(\Gamma) \cap \text{dom}(\Gamma') : \Gamma(p) \equiv \Gamma'(p)$ and $\forall p \in \text{dom}(\Gamma) \setminus \text{dom}(\Gamma'), q \in \text{dom}(\Gamma') \setminus \text{dom}(\Gamma) : \Gamma(p) \equiv (\epsilon, \text{end}) \equiv \Gamma'(q)$.

► **Definition 8** (Typing environment reduction). *The typing environment reduction $\xrightarrow{\alpha}$, with α being either $p:q?\ell$ or $p:q!\ell$ (for some p, q, ℓ), is inductively defined as follows:*

$$\begin{aligned} [\text{E-RCV}] \quad p_k : (q!l_k\langle S_k \rangle \cdot \sigma, T_p), q : (\sigma_q, \sum_{i \in I} p_i?l_i\langle S_i \rangle \cdot T_i), \Gamma &\xrightarrow{q:p_k?\ell_k} p_k : (\sigma, T_p), q : (\sigma_q, T_k), \Gamma \quad (k \in I) \\ [\text{E-SEND}] \quad p : (\sigma, \sum_{i \in I} q_i!l_i\langle S_i \rangle \cdot T_i), \Gamma &\xrightarrow{p:q_k!\ell_k} p : (\sigma \cdot q_k!l_k\langle S_k \rangle, T_k), \Gamma \quad (k \in I) \\ [\text{E-STRUCT}] \quad \Gamma \equiv \Gamma_1 \xrightarrow{\alpha} \Gamma'_1 \equiv \Gamma' &\implies \Gamma \xrightarrow{\alpha} \Gamma' \end{aligned}$$

We use $\Gamma \longrightarrow \Gamma'$ instead of $\Gamma \xrightarrow{\alpha} \Gamma'$ when α is not relevant, and we use \longrightarrow^* for the reflexive and transitive closure of \longrightarrow .

In rule [E-RCV] an environment has a reduction step labeled $q:p_k?\ell_k$ if participant p_k has at the head of its queue a message for q with label ℓ_k and payload sort S_k , and q has an external choice type that includes participant p_k with label ℓ_k and a corresponding sort S_k ; the environment evolves by consuming p_k 's message and activating the continuation T_k in q 's type. Rule [E-SEND] specifies reduction if p has an internal choice type where p sends a message toward q_k , for some $k \in I$; the reduction is labelled $p:q_k!\ell_k$ and the message is placed at the end of p 's queue. Rule [E-STRUCT] is standard closure of the reduction relation under structural congruence.

Similarly to [21], we define behavioral properties also for typing environments (and their reductions). As for processes, we use three properties for typing environments. *Safety* ensures that the typing environment never has label or sort mismatches. In our setting, where one participant can receive from more than one queue, this property ensures safe receptions of queued messages. The second property, called *deadlock freedom*, ensures that typing environment which can not reduce further must be terminated. The third property is *liveness*, which, for the case of standard session types (as in [8]), matches the liveness defined in [8, Definition 4.7]. Liveness ensures all pending inputs eventually receive a message and all queued messages are eventually received, if reduction steps are performed in a fair manner.

Notably, our definition of deadlock-freedom and liveness also require safety. The reason for this is that liveness and deadlock-freedom properties for typing environments if defined without assuming safety are not preserved by the subtyping (introduced in the next section, see Example 22). We use the predicate over typing environments $\text{end}(\Gamma)$ (read “ Γ is terminated”) that holds iff, for all $p \in \text{dom}(\Gamma)$, we have $\Gamma(p) \equiv (\epsilon, \text{end})$.

► **Definition 9** (Typing environment behavioral properties). *A typing environment path is a (possibly infinite) sequence of typing environments $(\Gamma_i)_{i \in I}$, where $I = \{0, 1, 2, \dots, n\}$ (or $I = \{0, 1, 2, \dots\}$) is a set of consecutive natural numbers, and, $\forall i \in I \setminus \{n\}$ (or $\forall i \in I$), $\Gamma_i \longrightarrow \Gamma_{i+1}$. We say that a path $(\Gamma_i)_{i \in I}$ is **safe** iff, $\forall i \in I$:*

(TS) *if $\Gamma_i(p) \equiv (\sigma_p, \sum_{j \in J} q_j?l_j\langle S_j \rangle \cdot T_j)$ and $\Gamma_i(q_k) \equiv (p!l_k\langle S_k \rangle \cdot \sigma_q, T_q)$, with $q_k \in \{q_j\}_{j \in J}$, then $\exists j \in J : (q_j, \ell_j, S_j) = (q_k, \ell_k, S_k)$*

We say that a path $(\Gamma_i)_{i \in I}$ is **deadlock free** iff:

(TD) if $I = \{0, 1, \dots, n\}$ and $\Gamma_n \not\rightarrow$ then $\text{end}(\Gamma_n)$

We say that a path $(\Gamma_i)_{i \in I}$ is **fair** iff, $\forall i \in I$:

(TF1) whenever $\Gamma_i \xrightarrow{p:q!\ell} \Gamma'$, then $\exists k, q', \ell'$ such that $I \ni k \geq i$ and $\Gamma_k \xrightarrow{p:q'!\ell'} \Gamma_{k+1}$

(TF2) whenever $\Gamma_i \xrightarrow{p:q?\ell} \Gamma'$, then $\exists k, q', \ell'$ such that $I \ni k \geq i$ and $\Gamma_k \xrightarrow{p:q'?\ell'} \Gamma_{k+1}$

We say that a path $(\Gamma_i)_{i \in I}$ is **live** iff, $\forall i \in I$:

(TL1) if $\Gamma_i(p) \equiv (q!\ell(S) \cdot \sigma, T)$, then $\exists k$ such that $I \ni k \geq i$ and $\Gamma_k \xrightarrow{q:p?\ell} \Gamma_{k+1}$

(TL2) if $\Gamma_i(p) \equiv \left(\sigma_p, \sum_{j \in J} q_j?\ell_j(S_j) \cdot T_j \right)$, then $\exists k, q, \ell$ such that $I \ni k \geq i$, $(q, \ell) \in \{(q_j, \ell_j)\}_{j \in J}$, and $\Gamma_k \xrightarrow{p:q?\ell} \Gamma_{k+1}$

A typing environment Γ is **safe** iff all paths starting with Γ are safe. A typing environment Γ is **deadlock-free** iff all paths starting with Γ are safe and deadlock-free. We say that a typing environment Γ is **live** iff it is safe and all fair paths beginning with Γ are live.

Since our deadlock-freeness and liveness for typing environments subsumes safety, we do not have the situation in which typing environment is deadlock-free and/or live but not safe. Still, we can have typing environments that are safe but not deadlock-free or live.

► **Example 10.** Consider typing environment

$$\Gamma = \{p: (\epsilon, \sum \{q?\ell_1(S_1) \cdot r?\ell_2(S_2), r?\ell_2(S_2), r?\ell_3(S_3)\}, q: (p!\ell_1(S_1), \text{end}), r: (p!\ell_2(S_2), \text{end})\}$$

Here, Γ is safe by Definition 9, but is not live nor deadlock-free, because the path in which p receives from r message ℓ_2 leads to a typing environment that cannot reduce further but is not terminated. Now consider typing environment

$$\Gamma' = \{p: (\epsilon, \sum \{q?\ell_1(S_1) \cdot r?\ell_2(S_2), r?\ell_3(S_3)\}, q: (p!\ell_1(S_1), \text{end}), r: (p!\ell_2(S_2), \text{end})\}$$

Here, Γ' is not safe by Definition 9 (hence, also not deadlock-free nor live) because there is a mismatch between r sending ℓ_2 to p , while p can only receive ℓ_3 from r .

We now prove that all three behavioral properties are closed under structural congruence (in Appendix A) and reductions.

► **Proposition 11.** If Γ is safe/deadlock-free/live and $\Gamma \longrightarrow \Gamma'$, then Γ' is safe/deadlock-free/live.

Proof. Consider all paths $(\Gamma_i)_{i \in I}$ starting with $\Gamma_0 = \Gamma$ and $\Gamma_1 = \Gamma'$. If any of the properties of Definition 9 holds for $i \geq 0$ (i.e. starting with Γ) for any of considered paths, then it also must hold for $i \geq 1$ (i.e. paths starting with Γ'). ◀

3.2 Subtyping

In Definition 12 below we formalise a standard “synchronous” subtyping relation as in [17], not dealing with possible asynchronous permutations [8].

► **Definition 12.** The subtyping \leq is coinductively defined as:

$$\begin{array}{c} \frac{\forall i \in I \quad T_i \leq T'_i \quad \{p_i\}_{i \in I} = \{p_i\}_{i \in I \cup J}}{\sum_{i \in I} p_i!\ell_i(S_i) \cdot T_i \leq \sum_{i \in I \cup J} p_i!\ell_i(S_i) \cdot T'_i} [s\text{-OUT}] \quad \frac{\forall i \in I \quad T_i \leq T'_i \quad \{p_i\}_{i \in I} = \{p_i\}_{i \in I \cup J}}{\sum_{i \in I \cup J} p_i?\ell_i(S_i) \cdot T_i \leq \sum_{i \in I} p_i?\ell_i(S_i) \cdot T'_i} [s\text{-IN}] \\ \frac{T_1\{\mu t, T_1/t\} \leq T_2}{\mu t. T_1 \leq T_2} [s\text{-MUL}] \quad \frac{T_1 \leq T_2\{\mu t, T_2/t\}}{T_1 \leq \mu t. T_2} [s\text{-MUR}] \quad \frac{}{\text{end} \leq \text{end}} [s\text{-END}] \end{array}$$

Pair of queue/session types are related via subtyping $(\sigma_1, T_1) \leq (\sigma_2, T_2)$ iff $\sigma_1 = \sigma_2$ and $T_1 \leq T_2$. We define $\Gamma \leq \Gamma'$ iff $\text{dom}(\Gamma) = \text{dom}(\Gamma')$ and $\forall p \in \text{dom}(\Gamma) : \Gamma(p) \leq \Gamma'(p)$.

The subtyping rules are standard: they allow less branches for the subtype in the internal choice (in $[\text{S-OUT}]$), and more branches in the external choice (in $[\text{S-IN}]$). The side condition in both rules ensures the set of participants specified in the subtype and supertype choices is the same. Hence, the subtyping allows flexibility in the set of labels, not in the set of participants. Subtyping holds up to unfolding (by $[\text{S-MUL}]$ and $[\text{S-MUR}]$), as usual for coinductive subtyping [18, Chapter 21]. By rule $[\text{S-END}]$, **end** is related via subtyping to itself.

► **Example 13.** Consider the typing environments Γ and Γ' from Example 10. We have:

$$\sum \{q?\ell_1(S_1).r?\ell_2(S_2), r?\ell_2(S_2), r?\ell_3(S_3)\} \leq \sum \{q?\ell_1(S_1).r?\ell_2(S_2), r?\ell_3(S_3)\} \text{ (by } [\text{S-IN}])$$

and therefore, we also have that $\Gamma \leq \Gamma'$ holds.

The following two lemmas show that subtyping of safe typing environments is a simulation and that subtyping preserves all typing environment properties. (*Proofs in Appendix A.*)

► **Lemma 14.** If $\Gamma' \leq \Gamma$, Γ is safe, and $\Gamma' \xrightarrow{\alpha} \Gamma'_1$, then there is Γ_1 such that $\Gamma'_1 \leq \Gamma_1$ and $\Gamma \xrightarrow{\alpha} \Gamma_1$.

► **Lemma 15.** If Γ is safe/deadlock-free/live and $\Gamma' \leq \Gamma$, then Γ' is safe/deadlock-free/live.

► **Remark 16.** Notice that if in Definition 9 safety is not assumed in deadlock-freedom and liveness, then Lemma 15 does not hold - see Example 13. The same observation also holds for deadlock-freedom in the synchronous case [17, Remark 3.2].

4 The Typing System and Its Properties

In this section we introduce the type system (Definition 17) that assigns types (introduced in Section 3) to processes, queues, and sessions (introduced in Section 2). Our typing system is an extension and adaptation of the one in [8]. Then, we prove its key properties, including subject reduction (Theorem 23), type safety (Theorem 24), session fidelity (Theorem 25), deadlock freedom (Theorem 26), and liveness (Theorem 27).

► **Definition 17 (Typing system).** A shared typing environment Θ , which assigns sorts to expression variables, and (recursive) session types to process variables, is defined as:

$$\Theta ::= \emptyset \mid \Theta, X : T \mid \Theta, x : S$$

Our type system is inductively defined by the rules in Figure 5, with 4 judgements — which cover respectively values and variables v , processes P , message queues h , and sessions \mathcal{M} :

$$\Theta \vdash v : S \qquad \Theta \vdash P : T \qquad \vdash h : \sigma \qquad \Gamma \vdash \mathcal{M}$$

By Definition 17, the typing judgment $\Theta \vdash v : S$ means that value v is of sort S under environment Θ . The judgement $\Theta \vdash P : T$ says that process P behaves as prescribed with type T and uses its variables as given in Θ . The judgement $\vdash h : \sigma$ says that the messages in the queue h correspond to the queue type σ . Finally, $\Gamma \vdash \mathcal{M}$ means that all participant processes in session \mathcal{M} behave as prescribed by the session types in Γ .

We now illustrate the typing rules in Figure 5. The first row gives rules for typing natural and boolean values and expression variables. The second row provides rules for

$$\begin{array}{c}
\overline{\Theta \vdash 1, 2, \dots : \mathbf{nat}}^{[T\text{-NAT}]} \quad \overline{\Theta \vdash \mathbf{true}, \mathbf{false} : \mathbf{bool}}^{[T\text{-BOOL}]} \quad \overline{\Theta, x : S \vdash x : S}^{[T\text{-VAR}]} \\
\\
\overline{\vdash \emptyset : \epsilon}^{[T\text{-NUL}]} \quad \overline{\vdash v : S}^{[T\text{-ELM}]} \quad \overline{\vdash h_1 : \sigma_1 \quad \vdash h_2 : \sigma_2}{\vdash h_1 \cdot h_2 : \sigma_1 \cdot \sigma_2}^{[T\text{-QUEUE}]} \\
\\
\overline{\Theta \vdash \mathbf{0} : \mathbf{end}}^{[T\text{-0}]} \quad \overline{\forall i \in I \quad \Theta \vdash v_i : S_i \quad \Theta \vdash P_i : T_i}{\Theta \vdash \sum_{i \in I} p_i! \ell_i \langle v_i \rangle . P_i : \sum_{i \in I} p_i! \ell_i \langle S_i \rangle . T_i}^{[T\text{-OUT}]} \\
\\
\overline{\forall i \in I \quad \Theta, x_i : S_i \vdash P_i : T_i}{\Theta \vdash \sum_{i \in I} p_i? \ell_i \langle x_i \rangle . P_i : \sum_{i \in I} p_i? \ell_i \langle S_i \rangle . T_i}^{[T\text{-IN}]} \quad \overline{\Theta \vdash v : \mathbf{bool} \quad \Theta \vdash P_i : T \quad (i = 1, 2)}{\Theta \vdash \text{if } v \text{ then } P_1 \text{ else } P_2 : T}^{[T\text{-COND}]} \\
\\
\overline{\Theta, X : T \vdash P : T}{\Theta \vdash \mu X. P : T}^{[T\text{-REC}]} \quad \overline{\Theta, X : T \vdash X : T}^{[T\text{-VAR}]} \quad \overline{\Theta \vdash P : T \quad T \leq T'}{\Theta \vdash P : T'}^{[T\text{-SUB}]} \\
\\
\overline{\Gamma = \{p_i : (\sigma_i, T_i) \mid i \in I\} \quad \forall i \in I \quad \emptyset \vdash P_i : T_i \quad \vdash h_i : \sigma_i}{\Gamma \vdash \prod_{i \in I} (p_i \triangleleft P_i \mid p_i \triangleleft h_i)}^{[T\text{-SESS}]}
\end{array}$$

■ **Figure 5** Typing rules for values, queues, and for processes and sessions.

typing message queues: an empty queue has the empty queue type (by $[T\text{-NUL}]$) while a queued message is typed if its payload has the correct sort (by $[T\text{-ELM}]$), and a queue obtained with concatenating two message queues is typed by concatenating their queue types (by $[T\text{-QUEUE}]$).

Rule $[T\text{-0}]$ types a terminated process $\mathbf{0}$ with \mathbf{end} . Rules $[T\text{-OUT}]/[T\text{-IN}]$ type internal/external choice processes with the corresponding internal/external choice types: in each branch the payload v_i/x_i and the continuation process P_i have the corresponding sort S_i and continuation type T_i ; observe that in $[T\text{-IN}]$, each continuation process P_i is typed under environment Θ extended with the input-bound variable x_i having sort S_i . Rule $[T\text{-COND}]$ types conditional: both branches must have the same type, and the condition must be boolean. Rule $[T\text{-REC}]$ types a recursive process $\mu X.P$ with T if P has type T in an environment where that X has type T . Rule $[T\text{-VAR}]$ types recursive process variables. Finally, $[T\text{-SUB}]$ is the *subsumption rule*: a process P of type T can be typed with any supertype T' . This rule makes our type system equi-recursive [18], as \leq relates types up-to unfolding (by Definition 12). Rule $[T\text{-SESS}]$ types a session under environment Γ if each participant's queue and process have corresponding queue and process types in an empty Θ (which forces processes to be closed).

► **Example 18** (Types for a centralised FL protocol implementation). Consider session \mathcal{M} from Example 2. Take that value **data** and variables x, x_2, \dots, x_n are of sort S . We may derive $\vdash P_1 : T_1$ and $\vdash P_i : T_2$ (for $i = 1, 2, \dots, n$), where (with a slight abuse of notation using the concurrent input macro for types):

$$\begin{aligned}
T_1 &= p_2!ld\langle S \rangle \dots p_n!ld\langle S \rangle . \| p_2?upd(S), \dots, p_n?upd(S) \| \\
T_2 &= p_1?ld\langle S \rangle . p_1!upd\langle S \rangle
\end{aligned}$$

Hence, with $\Gamma = \{p_1 : (\epsilon, T_1)\} \cup \{p_i : (\epsilon, T_2) \mid i = 2, \dots, n\}$ we obtain $\Gamma \vdash \mathcal{M}$. Observe that Γ is deadlock-free and live: after p_1 sends the data to all other participants, each participant receives and replies with an update, that p_1 finally receives in arbitrary order.

► **Example 19** (Types for a decentralised FL protocol implementation). Consider session \mathcal{M} from Example 3. Take that value **data** and variables $x_2, \dots, x_n, y_2, \dots, y_n$ are of sort S . We may derive $\vdash P_1 : T_1$ where (again with a slight abuse of notation using the concurrent input

macro for types):

$$\begin{aligned} T_1 = & p_2!ld\langle S \rangle \dots p_n!ld\langle S \rangle . \| p_2?ld\langle S \rangle . p_2!upd\langle S \rangle , \dots , p_n?ld\langle S \rangle . p_n!upd\langle S \rangle \| . \\ & \| p_2?upd\langle S \rangle , \dots , p_n?upd\langle S \rangle \| \end{aligned}$$

and similarly for other process we may derive the corresponding types so that $\vdash P_i : T_i$, for $i = 2, \dots, n$. Thus, with $\Gamma = \{p_i : (\epsilon, T_i) \mid i = 1, \dots, n\}$ we obtain $\Gamma \vdash \mathcal{M}$. Notice that Γ is safe, i.e., has no label or sort mismatches: participant p_i always receives from p_j message ld before up (as they are enqueued always in this order). This also implies Γ is deadlock-free and live: a participant first asynchronously sends all its ld messages (phase 1), then awaits to receive ld messages from queues of all other participants (phase 2), and only then awaits to receive up messages (phase 3). Hence, all $n(n-1)$ of ld and the same number of up messages are sent/received, and a reduction path always ends with a terminated typing environment.

► **Remark 20 (On the decidability of typing environment properties).** Under the “bottom-up approach” to session typing, typing environment properties such as deadlock freedom and liveness are generally undecidable, since two session types with unbounded message queues are sufficient to encode a Turing machine [1, Theorem 2.5]. Still, many practical protocols have bounded buffer sizes — and in particular, the buffer sizes for the FL protocols in [20] are given in the same paper. Therefore, Examples 18 and 19 yield finite-state typing environment whose behavioural properties are decidable and easily verified, e.g., via model checking.

Properties of our typing system. We now illustrate and prove the properties of our typing system in Definition 17. We aim to prove that if a session \mathcal{M} is typed with safe/deadlock-free/live typing environment Γ , then \mathcal{M} is also safe/deadlock-free/live. Along the way, we illustrate the subtle interplay between our deadlock freedom, liveness, and safety properties (Definition 9) and subtyping (Definition 12).

First, we introduce the subtyping-related Lemma 21 below, which holds directly rule by $[T\text{-SUB}]$ and is important for proving Subject Reduction later on (Theorem 23). Based on this, in Example 22 we show why our definitions of deadlock-free and live typing environments (Definition 9) also require safety.

► **Lemma 21.** *If $\Gamma \vdash \mathcal{M}$ and $\Gamma \leq \Gamma'$, then $\Gamma' \vdash \mathcal{M}$.*

Example 22 below shows that live and deadlock freedom properties for typing environments defined without requiring safety (as for sessions in Definition 4) would not be preserved by subtyping (Definition 12). For deadlock-freedom, this is already known in the synchronous calculus [17, Remark 3.2].

► **Example 22.** Consider the safe but not deadlock-free nor live session \mathcal{M} from Example 5, and the typing environments Γ and Γ' from Example 10. It is straightforward to show that $\Gamma \vdash \mathcal{M}$, by Definition 17. Observe that, as shown in Example 13, we have $\Gamma \leq \Gamma'$; therefore, by Lemma 21 we also have $\Gamma' \vdash \mathcal{M}$. As noted in Example 10, the typing environment Γ' is not safe. Now, suppose that in Definition 9, deadlock-freedom and liveness of typing environments were defined without requiring safety (as for deadlock-freedom and liveness of sessions in Definition 4); also notice that Γ' has a single path that is deadlock-free and live, but not safe. Therefore, this change in Definition 9 would cause Γ' , an unsafe but deadlock-free and live typing environment, to type \mathcal{M} , a session that is not deadlock-free nor live. This would hamper our goal of showing that if a session is typed by a deadlock-free/live typing environment, then the session is deadlock free/live.

We can now show the subject reduction result (Theorem 23 below): if a session \mathcal{M} , typed with a safe/deadlock-free/live typing environment Γ , reduces to \mathcal{M}' , then \mathcal{M}' is also typed with a safe/deadlock-free/live typing environment Γ' such that $\Gamma \longrightarrow \Gamma'$. (*Proofs of this and the following results are in Appendix B.*)

► **Theorem 23** (Subject Reduction). *Assume $\Gamma \vdash \mathcal{M}$ with Γ safe/deadlock-free/live and $\mathcal{M} \longrightarrow \mathcal{M}'$. Then, there is safe/deadlock-free/live type environment Γ' such that $\Gamma \longrightarrow^* \Gamma'$ and $\Gamma' \vdash \mathcal{M}'$.*

A direct consequence of Theorem 23 is that session typed with safe typing environments must also be safe.

► **Theorem 24** (Type Safety). *If $\Gamma \vdash \mathcal{M}$ and Γ is safe, then \mathcal{M} is safe.*

Session fidelity (Theorem 25 below) states that if a typing environment Γ can reduce, then a session \mathcal{M} typed by Γ can reduce accordingly.

► **Theorem 25** (Session Fidelity). *Let $\Gamma \vdash \mathcal{M}$. If $\Gamma \longrightarrow$, then $\exists \Gamma', P'$ such that $\Gamma \longrightarrow \Gamma'$ and $\mathcal{M} \longrightarrow^+ \mathcal{M}'$ and $\Gamma' \vdash \mathcal{M}'$.*

The deadlock freedom Theorem 26 below is a consequence of subject reduction (Theorem 23) and session fidelity (Theorem 25): it states that if a session \mathcal{M} is typed with deadlock-free typing environment, then \mathcal{M} is also deadlock-free. (Notice here that deadlock-freedom for typing environments is somewhat stronger than the one for sessions: the former also requires safety, and the reason is explained in Example 22 above.)

► **Theorem 26** (Deadlock freedom). *If $\Gamma \vdash \mathcal{M}$ and Γ is deadlock-free, then \mathcal{M} is deadlock-free.*

Our last result is the liveness Theorem 27 below, which is a consequence of subject reduction (Theorem 23). It ensures that if a session \mathcal{M} is typed with live typing environment, then \mathcal{M} is also live. (Again, notice that liveness for typing environments is stronger than liveness for processes, since the former also requires safety, as motivated in Example 22.)

► **Theorem 27** (Liveness). *If $\Gamma \vdash \mathcal{M}$ and Γ is live, then \mathcal{M} is live.*

5 Related and Future Work

Related work. The original “top-down” model of asynchronous multiparty session types [9] has been extended to enable the modeling and verification of a broader class of protocols: we discuss those most closely related to our work. [10, 7] extend multiparty sessions to support protocols with optional and dynamic participants, allowing sender-driven choices directed toward multiple participants. The models in [15, 25, 13, 14] generalize the notion of projection in asynchronous multiparty session types. Building on the global types of [10], they allow local types with internal and external choices directed at different participants, as we do in this paper. Nevertheless, these approaches are constrained by their reliance on global types and a projection function: the global type couples both sending and receiving in a single construct and enforces a protocol structure where a single participant drives the interaction. This results in at least one projected local type beginning with a receive action — unlike the local types for our decentralised FL protocol implementation (see Example 19).

This limitation could potentially be addressed through a subtyping relation that allows message reordering. However, existing asynchronous subtyping relations handle only standard internal and external choices [16, 8]. A recent paper introduces a subtyping relation for

extended choices [12], but it does not permit message reordering, as “this notion of subtyping does not satisfy subprotocol fidelity.”

During the submission of this paper, a new automata-theoretic session typing model was introduced in an online technical report [26] for a paper to be published in May 2025. Our initial assessment suggests that this model could potentially express our running example: its global types feature decoupled send-receive operations, and its typing system can accommodate a “bottom-up” approach with asynchronous multiparty session types having relaxed choices [26, Section 5]. However, unlike our work, [26] does not address liveness; moreover, it explicitly notes that “there are subtleties for subtyping as one cannot simply add receives” [26, Appendix C.3] — whereas our subtyping (Definition 12) does not have such restrictions. We plan to conduct a more detailed comparison with this work in the near future.

The approach in [17] extends the “bottom-up” synchronous multiparty session model of [21], not only by introducing more flexible choices but also by supporting mixed choices — i.e., choices that combine inputs and outputs. In contrast to [17], which assumes synchronous communication, our theory supports asynchronous communication, which is essential for our motivating scenarios and running examples; moreover, [17] does not prove process liveness nor session fidelity. The subtle interplay between deadlock-freedom, safety, and subtyping (cf. Example 22) is also acknowledged in [17, Remark 3.2].

The correctness of the decentralized FL algorithm has been formally verified for deadlock-freedom and termination in [20, 6] by using the Communicating Sequential Processes calculus (CSP) and the Process Analysis Toolkit (PAT) model checker. Our asynchronous multiparty session typing model abstracts protocol behavior at the type level, paving the way for more scalable and efficient techniques — not only for model checking, but also for broader verification and analysis applications.

Conclusions and future work. We present the first “bottom-up” asynchronous multiparty session typing theory that supports internal and external choices targeting multiple distinct participants. We introduce a process and type calculus, which we relate through a type system. We formally prove safety, deadlock-freedom, liveness, and session fidelity, highlighting interesting dependencies between these properties in the presence of a subtyping relation. Finally, we demonstrate how our model can represent a wide range of communication protocols, including those used in asynchronous decentralized federated learning.

For future work, we identify two main research directions. The first focuses on the fundamental properties of our approach, including: verifying typing environment properties via model checking, in the style of [21]; investigating decidable approximations of deadlock freedom and liveness of typing contexts (see Remark 20), e.g. with the approach of [11]; and investigating the expressiveness of our model based on the framework in [17]. The second direction explores the application of our model as a foundation for reasoning about federated-learning-specific properties, e.g.: handling crashes of arbitrary participants [3, 2, 4]; supporting optional participation [10, 7]; ensuring that participants only receive data they are able to process; statically enforcing that only model parameters—not raw data—are exchanged to preserve privacy; guaranteeing sufficiently large server buffers to receive messages from all clients; and ensuring that all clients contribute equally to the algorithm.

References

- 1 Massimo Bartoletti, Alceste Scalas, Emilio Tuosto, and Roberto Zunino. Honesty by typing. *Log. Methods Comput. Sci.*, 12(4), 2016. doi:10.2168/LMCS-12(4:7)2016.

- 2 Adam D. Barwell, Ping Hou, Nobuko Yoshida, and Fangyi Zhou. Designing asynchronous multiparty protocols with crash-stop failures. In Karim Ali and Guido Salvaneschi, editors, *37th European Conference on Object-Oriented Programming, ECOOP 2023, July 17-21, 2023, Seattle, Washington, United States*, volume 263 of *LIPIcs*, pages 1:1–1:30. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. URL: <https://doi.org/10.4230/LIPIcs.ECOOP.2023.1>, doi:10.4230/LIPIcs.ECOOP.2023.1.
- 3 Adam D. Barwell, Ping Hou, Nobuko Yoshida, and Fangyi Zhou. Crash-Stop Failures in Asynchronous Multiparty Session Types. *Logical Methods in Computer Science*, pages –, 2025.
- 4 Adam D. Barwell, Alceste Scalas, Nobuko Yoshida, and Fangyi Zhou. Generalised multiparty session types with crash-stop failures. In Bartek Klin, Slawomir Lasota, and Anca Muscholl, editors, *33rd International Conference on Concurrency Theory, CONCUR 2022, September 12-16, 2022, Warsaw, Poland*, volume 243 of *LIPIcs*, pages 35:1–35:25. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. URL: <https://doi.org/10.4230/LIPIcs.CONCUR.2022.35>, doi:10.4230/LIPIcs.CONCUR.2022.35.
- 5 Enrique Tomás Martínez Beltrán, Mario Quiles Pérez, Pedro Miguel Sánchez Sánchez, Sergio López Bernal, Jérôme Bovet, Manuel Gil Pérez, Gregorio Martínez Pérez, and Alberto Huertas Celdrán. Decentralized federated learning: Fundamentals, state of the art, frameworks, trends, and challenges. *IEEE Commun. Surv. Tutorials*, 25(4):2983–3013, 2023. doi:10.1109/COMST.2023.3315746.
- 6 Miodrag Djukic, Ivan Prokić, Miroslav Popovic, Silvia Ghilezan, Marko Popovic, and Simona Prokić. Correct orchestration of federated learning generic algorithms: Python translation to CSP and verification by PAT. *Journal on Software Tools for Technology Transfer (to appear)*, 2025.
- 7 Lorenzo Gheri, Ivan Lanese, Neil Sayers, Emilio Tuosto, and Nobuko Yoshida. Design-by-contract for flexible multiparty session protocols. In Karim Ali and Jan Vitek, editors, *36th European Conference on Object-Oriented Programming, ECOOP 2022, June 6-10, 2022, Berlin, Germany*, volume 222 of *LIPIcs*, pages 8:1–8:28. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. URL: <https://doi.org/10.4230/LIPIcs.ECOOP.2022.8>, doi:10.4230/LIPIcs.ECOOP.2022.8.
- 8 Silvia Ghilezan, Jovanka Pantovic, Ivan Prokic, Alceste Scalas, and Nobuko Yoshida. Precise subtyping for asynchronous multiparty sessions. *ACM Trans. Comput. Log.*, 24(2):14:1–14:73, 2023. doi:10.1145/3568422.
- 9 Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. *J. ACM*, 63(1):9:1–9:67, 2016. URL: <http://doi.acm.org/10.1145/2827695>, doi:10.1145/2827695.
- 10 Raymond Hu and Nobuko Yoshida. Explicit connection actions in multiparty session types. In Marieke Huisman and Julia Rubin, editors, *Fundamental Approaches to Software Engineering - 20th International Conference, FASE 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, volume 10202 of *Lecture Notes in Computer Science*, pages 116–133. Springer, 2017. doi:10.1007/978-3-662-54494-5_7.
- 11 Julien Lange and Nobuko Yoshida. Verifying asynchronous interactions via communicating session automata. In *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I*, pages 97–117, 2019. doi:10.1007/978-3-030-25540-4_6.
- 12 Elaine Li, Felix Stutz, and Thomas Wies. Deciding subtyping for asynchronous multiparty sessions. In Stephanie Weirich, editor, *Programming Languages and Systems - 33rd European Symposium on Programming, ESOP 2024, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2024, Luxembourg City, Luxembourg, April 6-11, 2024, Proceedings, Part I*, volume 14576 of *Lecture Notes in Computer Science*, pages 176–205. Springer, 2024. doi:10.1007/978-3-031-57262-3_8.

- 13 Elaine Li, Felix Stutz, Thomas Wies, and Damien Zufferey. Complete multiparty session type projection with automata. In Constantin Enea and Akash Lal, editors, *Computer Aided Verification - 35th International Conference, CAV 2023, Paris, France, July 17-22, 2023, Proceedings, Part III*, volume 13966 of *Lecture Notes in Computer Science*, pages 350–373. Springer, 2023. doi:10.1007/978-3-031-37709-9_17.
- 14 Elaine Li, Felix Stutz, Thomas Wies, and Damien Zufferey. Characterizing implementability of global protocols with infinite states and data. *CoRR*, abs/2411.05722, 2024. URL: <https://doi.org/10.48550/arXiv.2411.05722>, arXiv:2411.05722, doi:10.48550/ARXIV.2411.05722.
- 15 Rupak Majumdar, Madhavan Mukund, Felix Stutz, and Damien Zufferey. Generalising projection in asynchronous multiparty session types. In Serge Haddad and Daniele Varacca, editors, *32nd International Conference on Concurrency Theory, CONCUR 2021, August 24-27, 2021, Virtual Conference*, volume 203 of *LIPIcs*, pages 35:1–35:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. URL: <https://doi.org/10.4230/LIPIcs.CONCUR.2021.35>, doi:10.4230/LIPIcs.CONCUR.2021.35.
- 16 Dimitris Mostrous, Nobuko Yoshida, and Kohei Honda. Global principal typing in partially commutative asynchronous sessions. In Giuseppe Castagna, editor, *Programming Languages and Systems, 18th European Symposium on Programming, ESOP 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*, volume 5502 of *Lecture Notes in Computer Science*, pages 316–332. Springer, 2009. doi:10.1007/978-3-642-00590-9_23.
- 17 Kirstin Peters and Nobuko Yoshida. Separation and encodability in mixed choice multiparty sessions. In Pawel Sobocinski, Ugo Dal Lago, and Javier Esparza, editors, *Proceedings of the 39th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2024, Tallinn, Estonia, July 8-11, 2024*, pages 62:1–62:15. ACM, 2024. doi:10.1145/3661814.3662085.
- 18 Benjamin C. Pierce. *Types and programming languages*. MIT Press, 2002.
- 19 Miroslav Popovic, Marko Popovic, Ivan Kastelan, Miodrag Djukic, and Silvia Ghilezan. A simple Python testbed for federated learning algorithms. In *2023 Zooming Innovation in Consumer Technologies Conference (ZINC)*, pages 148–153. IEEE, 2023.
- 20 Ivan Prokic, Silvia Ghilezan, Simona Kasterovic, Miroslav Popovic, Marko Popovic, and Ivan Kastelan. Correct orchestration of federated learning generic algorithms: Formalisation and verification in CSP. In Jan Kofron, Tiziana Margaria, and Cristina Seceleanu, editors, *Engineering of Computer-Based Systems - 8th International Conference, ECBS 2023, Västerås, Sweden, October 16-18, 2023, Proceedings*, volume 14390 of *Lecture Notes in Computer Science*, pages 274–288. Springer, 2023. doi:10.1007/978-3-031-49252-5_25.
- 21 Alceste Scalas and Nobuko Yoshida. Less is more: multiparty session types revisited. *PACMPL*, 3(POPL):30:1–30:29, 2019. doi:10.1145/3290343.
- 22 Milos Simic, Jovana Dedeic, Milan Stojkov, and Ivan Prokic. A hierarchical namespace approach for multi-tenancy in distributed clouds. *IEEE Access*, 12:32597–32617, 2024. doi:10.1109/ACCESS.2024.3369031.
- 23 Milos Simic, Jovana Dedeic, Milan Stojkov, and Ivan Prokic. Data overlay mesh in distributed clouds allowing collaborative applications. *IEEE Access*, 13:6180–6203, 2025. doi:10.1109/ACCESS.2024.3525336.
- 24 Milos Simic, Ivan Prokic, Jovana Dedeic, Goran Sladic, and Branko Milosavljevic. Towards edge computing as a service: Dynamic formation of the micro data-centers. *IEEE Access*, 9:114468–114484, 2021. doi:10.1109/ACCESS.2021.3104475.
- 25 Felix Stutz. Asynchronous multiparty session type implementability is decidable - lessons learned from message sequence charts. In Karim Ali and Guido Salvaneschi, editors, *37th European Conference on Object-Oriented Programming, ECOOP 2023, July 17-21, 2023, Seattle, Washington, United States*, volume 263 of *LIPIcs*, pages 32:1–32:31. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. URL: <https://doi.org/10.4230/LIPIcs.ECOOP.2023.32>, doi:10.4230/LIPIcs.ECOOP.2023.32.

- 26 Felix Stutz and Emanuele D’Osualdo. An automata-theoretic basis for specification and type checking of multiparty protocols. *CoRR*, abs/2501.16977, 2025. [arXiv:2501.16977](#), doi:10.48550/ARXIV.2501.16977.
- 27 Thien Udomsrirungruang and Nobuko Yoshida. Top-down or bottom-up? complexity analyses of synchronous multiparty session types. *Proc. ACM Program. Lang.*, 9(POPL):1040–1071, 2025. doi:10.1145/3704872.
- 28 Liangqi Yuan, Ziran Wang, Lichao Sun, Philip S. Yu, and Christopher G. Brinton. Decentralized federated learning: A survey and perspective. *IEEE Internet Things J.*, 11(21):34617–34638, 2024. doi:10.1109/JIOT.2024.3407584.

A

 Proofs of Section 3

► **Proposition 28.** *If Γ is safe/deadlock-free/live and $\Gamma \equiv \Gamma'$, then Γ' is safe/deadlock-free/live.*

Proof. Since the typing environment reductions are closed under structural congruence, paths $(\Gamma_i)_{i \in I}$ starting with $\Gamma_0 = \Gamma$ are exactly those starting with $\Gamma_0 = \Gamma'$. ◀

► **Lemma 29.** *Let $\mathsf{T}' \leq \mathsf{T}$.*

1. $\mathsf{T}' \equiv \sum_{i \in I \cup J} \mathbf{q}_i ? \ell_i \langle \mathsf{S}_i \rangle . \mathsf{T}'_i$ iff $\mathsf{T} \equiv \sum_{i \in I} \mathbf{q}_i ? \ell_i \langle \mathsf{S}_i \rangle . \mathsf{T}_i$, where for all $i \in I$ $\mathsf{T}'_i \leq \mathsf{T}_i$, and $\{\mathbf{q}_i\}_{i \in I \cup J} = \{\mathbf{q}_i\}_{i \in I}$.
2. $\mathsf{T}' \equiv \sum_{i \in I} \mathbf{q}_i ! \ell_i \langle \mathsf{S}_i \rangle . \mathsf{T}'_i$ iff $\mathsf{T} \equiv \sum_{i \in I \cup J} \mathbf{q}_i ! \ell_i \langle \mathsf{S}_i \rangle . \mathsf{T}_i$, where for all $i \in I$ $\mathsf{T}'_i \leq \mathsf{T}_i$, and $\{\mathbf{q}_i\}_{i \in I} = \{\mathbf{q}_i\}_{i \in I \cup J}$.

► **Lemma 14.** *If $\Gamma' \leq \Gamma$, Γ is safe, and $\Gamma' \xrightarrow{\alpha} \Gamma'_1$, then there is Γ_1 such that $\Gamma'_1 \leq \Gamma_1$ and $\Gamma \xrightarrow{\alpha} \Gamma_1$.*

Proof. We distinguish two cases for α .

Case $\alpha = \mathbf{q} : \mathbf{p}_k ? \ell_k$: In this case we have:

$$\Gamma' \equiv \mathbf{p}_k : (\mathbf{q} ! \ell_k \langle \mathsf{S}_k \rangle . \sigma, \mathsf{T}'_p), \mathbf{q} : (\sigma_{\mathbf{q}}, \sum_{i \in I \cup J} \mathbf{p}_i ? \ell_i \langle \mathsf{S}_i \rangle . \mathsf{T}'_i), \Gamma'_2 \quad (1)$$

$$\Gamma'_1 \equiv \mathbf{p}_k : (\sigma, \mathsf{T}'_p), \mathbf{q} : (\sigma_{\mathbf{q}}, \mathsf{T}'_k), \Gamma'_2 \quad (2)$$

where $\exists i \in I \cup J : (\mathbf{p}_i, \ell_i, \mathsf{S}_i) = (\mathbf{p}_k, \ell_k, \mathsf{S}_k)$. Since $\Gamma' \leq \Gamma$ we have $\forall \mathbf{p} \in \text{dom}(\Gamma) = \text{dom}(\Gamma')$: $\Gamma'(\mathbf{p}) \leq \Gamma(\mathbf{p})$. Hence, by Lemma 29 we obtain

$$\Gamma \equiv \mathbf{p}_k : (\mathbf{q} ! \ell_k \langle \mathsf{S}_k \rangle . \sigma, \mathsf{T}_p), \mathbf{q} : (\sigma_{\mathbf{q}}, \sum_{i \in I} \mathbf{p}_i ? \ell_i \langle \mathsf{S}_i \rangle . \mathsf{T}_i), \Gamma_2 \quad (3)$$

where $\mathsf{T}'_p \leq \mathsf{T}_p$, $\forall i \in I : \mathsf{T}'_i \leq \mathsf{T}_i$, $\{\mathbf{p}_i\}_{i \in I \cup J} = \{\mathbf{p}_i\}_{i \in I}$, and $\Gamma'_2 \leq \Gamma_2$. Since Γ is safe, we have $\exists i \in I : (\mathbf{p}_i, \ell_i, \mathsf{S}_i) = (\mathbf{p}_k, \ell_k, \mathsf{S}_k)$, and hence $\Gamma \xrightarrow{\alpha} \Gamma_1$ and $\Gamma'_1 \leq \Gamma_1$ for

$$\Gamma_1 \equiv \mathbf{p}_k : (\sigma, \mathsf{T}_p), \mathbf{q} : (\sigma_{\mathbf{q}}, \mathsf{T}_k), \Gamma_2 \quad (4)$$

Case $\alpha = \mathbf{p} : \mathbf{q}_k ! \ell_k$: In this case we have:

$$\Gamma' \equiv \mathbf{p} : (\sigma, \sum_{i \in I} \mathbf{q}_i ! \ell_i \langle \mathsf{S}_i \rangle . \mathsf{T}'_i), \Gamma'_2 \quad (5)$$

$$\Gamma'_1 \equiv \mathbf{p} : (\sigma \cdot \mathbf{q}_k ! \ell_k \langle \mathsf{S}_k \rangle, \mathsf{T}'_k), \Gamma'_2 \quad (6)$$

where $k \in I$. Since $\Gamma' \leq \Gamma$, by Lemma 29 and we have

$$\Gamma \equiv \mathbf{p} : (\sigma, \sum_{i \in I \cup J} \mathbf{q}_i ! \ell_i \langle \mathsf{S}_i \rangle . \mathsf{T}_i), \Gamma_2 \quad (7)$$

where $\forall i \in I : \mathsf{T}'_i \leq \mathsf{T}_i$, $\{\mathbf{q}_i\}_{i \in I} = \{\mathbf{q}_i\}_{i \in I \cup J}$, and $\Gamma'_2 \leq \Gamma_2$. Hence, $\Gamma \xrightarrow{\alpha} \Gamma_1$ and $\Gamma'_1 \leq \Gamma_1$ for

$$\Gamma_1 \equiv \mathbf{p} : (\sigma \cdot \mathbf{q}_k ! \ell_k \langle \mathsf{S}_k \rangle, \mathsf{T}_k), \Gamma_2 \quad (8)$$

► **Lemma 30.** *Let $\Gamma' \leq \Gamma$.*

1. If $\Gamma \xrightarrow{q:p_k?\ell_k} \Gamma_1$, then $\Gamma' \xrightarrow{q:p_k?\ell_k} \Gamma'_1$ and $\Gamma'_1 \leq \Gamma_1$.
2. If $\Gamma \xrightarrow{q:p_k!\ell_k} \Gamma_k$, for $k \in I \cup J$, then $\Gamma' \xrightarrow{q:p_i!\ell_i} \Gamma'_i$ and $\Gamma'_i \leq \Gamma_i$, for $i \in I$, where $\{p_k\}_{k \in I \cup J} = \{p_i\}_{i \in I}$.

Proof. The proof is analogous to the proof of Lemma 14. \blacktriangleleft

► **Lemma 31.** *If Γ is safe and $\Gamma' \leq \Gamma$, then Γ' satisfies clause **TS** of Definition 9.*

Proof. Let $\Gamma'(p) \equiv (\sigma_p, \sum_{j \in J \cup K} q_j?\ell_j(S_j).T'_j)$ and $\Gamma'(q_k) \equiv (p!\ell_k(S_k).\sigma_q, T'_q)$, with $q_k \in \{q_j\}_{j \in J \cup K}$. We need to prove that then $\exists j \in J \cup K : (q_j, \ell_j, S_j) = (q_k, \ell_k, S_k)$. If $\Gamma'(p) = (\sigma_p, T'_p)$, then we have

$$T'_p \equiv \sum_{j \in J \cup K} q_j?\ell_j(S_j).T'_j \quad (9)$$

Let $\Gamma(p) = (\sigma_p, T_p)$ and $\Gamma(q_k) = (p!\ell_k(S_k).\sigma_q, T_q)$. Since $\Gamma' \leq \Gamma$, we have $T'_p \leq T_p$, and $T'_q \leq T_q$. By Lemma 29 and (9), we have

$$T_p \equiv \sum_{j \in J} q_j?\ell_j(S_j).T_j \quad \forall j \in J \quad T'_j \leq T_j \quad \text{and} \quad \{q_j\}_{j \in J \cup K} = \{q_j\}_{j \in J} \quad (10)$$

Now we have

$$\Gamma(p) \equiv (\sigma_p, \sum_{j \in J} q_j?\ell_j(S_j).T_j) \quad \text{and} \quad \Gamma(q_k) \equiv (p!\ell_k(S_k).\sigma_q, T_q) \quad (11)$$

with $q_k \in \{q_j\}_{j \in J \cup K} = \{q_j\}_{j \in J}$. Since Γ is safe, we have $\exists j \in J (\subseteq J \cup K) : (q_j, \ell_j, S_j) = (q_k, \ell_k, S_k)$. \blacktriangleleft

► **Lemma 15.** *If Γ is safe/deadlock-free/live and $\Gamma' \leq \Gamma$, then Γ' is safe/deadlock-free/live.*

Proof. Let us first assume Γ is safe, $\Gamma' \leq \Gamma$ and that Γ' are safe. Now let $(\Gamma'_i)_{i \in I}$ be a path starting with Γ' , i.e., $\Gamma' = \Gamma'_0$ and

$$\Gamma'_0 \xrightarrow{\alpha_1} \Gamma'_1 \xrightarrow{\alpha_2} \dots \quad \text{or} \quad \Gamma'_0 \xrightarrow{\alpha_1} \Gamma'_1 \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_n} \Gamma'_n \quad (12)$$

Let $\Gamma = \Gamma_0$. Since $\Gamma'_0 \leq \Gamma_0$, by application of Lemma 14 we have $\exists \Gamma_1$ such that $\Gamma'_1 \leq \Gamma_1$ and $\Gamma_0 \rightarrow \Gamma_1$. Since $\Gamma = \Gamma_0$ is safe, by Proposition 11, Γ_1 is also safe. By consecutive application of the above arguments, we obtain that for all $i \in I$ there is Γ_i such that $\Gamma'_i \leq \Gamma_i$, Γ_i is safe, and

$$\Gamma_0 \xrightarrow{\alpha_1} \Gamma_1 \xrightarrow{\alpha_2} \dots \quad \text{or} \quad \Gamma_0 \xrightarrow{\alpha_1} \Gamma_1 \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_n} \Gamma_n \quad (13)$$

Now we can prove the main statement.

1. Safety: Since Γ_i is safe and $\Gamma'_i \leq \Gamma_i$, by Lemma 31, Γ'_i satisfies the clause **TS** of Definition 9, for all $i \in I$. Hence, by Definition 9, path $(\Gamma'_i)_{i \in I}$ is safe. This implies Γ' is safe.
2. Deadlock-freedom: Assume Γ is deadlock-free (which subsumes safe). We need to prove that if $I = \{0, 1, \dots, n\}$ and $\Gamma'_n \not\rightarrow$, then $\text{end}(\Gamma'_n)$. Assume $I = \{0, 1, \dots, n\}$ and $\Gamma'_n \not\rightarrow$. Then, since $\Gamma'_n \leq \Gamma_n$, by Lemma 30, we obtain $\Gamma_n \not\rightarrow$. Since Γ is deadlock-free, this implies $\text{end}(\Gamma_n)$. From $\Gamma'_n \leq \Gamma_n$ and $\text{end}(\Gamma_n)$ we can show $\text{end}(\Gamma'_n)$. Hence, by Definition 9, Γ' is deadlock-free.

3. Liveness: Assume Γ is live (which subsumes safe). Assume $(\Gamma'_i)_{i \in I}$ from (12) is fair. Let us prove that it is also live.

Now we prove $(\Gamma_i)_{i \in I}$ from (13) is fair. Assume $\Gamma_i \xrightarrow{p:q!\ell}$ or $\Gamma_i \xrightarrow{p:q?\ell}$. Since $\Gamma'_i \leq \Gamma_i$, by Lemma 30, we have $\Gamma'_i \xrightarrow{p:q_1!\ell_1}$, for some q_1 and ℓ_1 , or $\Gamma'_i \xrightarrow{p:q?\ell}$. From Γ' being fair we conclude that $\exists k, q', \ell'$ such that $I \ni k \geq i$ and $\Gamma'_k \xrightarrow{p:q'!\ell'}$ or $\Gamma'_k \xrightarrow{p:q'?\ell'}$. Therefore, $\alpha_{k+1} = p:q'!\ell'$ or $\alpha_{k+1} = p:q'?\ell'$ in both (12) and (13), i.e., we obtain $\Gamma_k \xrightarrow{p:q'!\ell'} \Gamma_{k+1}$ or $\Gamma_k \xrightarrow{p:q'?\ell'} \Gamma_{k+1}$. Hence, $(\Gamma_i)_{i \in I}$ from (13) is fair.

Since $(\Gamma_i)_{i \in I}$ is fair and Γ is live, $(\Gamma_i)_{i \in I}$ is also live.

We now show that $(\Gamma'_i)_{i \in I}$ is live, which completes the proof. We have two cases.

- **TL1**: $\Gamma'_i(p) \equiv (q!\ell(S) \cdot \sigma, T')$. Since $\Gamma'_i \leq \Gamma_i$, we have $\Gamma_i(p) \equiv (q!\ell(S) \cdot \sigma, T)$, where $T' \leq T$. Since Γ is live we have there $\exists k$ such that $I \ni k \geq i$ and $\Gamma_k \xrightarrow{q:p?\ell} \Gamma_{k+1}$. Hence, $\alpha_{k+1} = q:p!\ell$ in both (13) and (12), i.e., we obtain $\Gamma'_k \xrightarrow{q:p?\ell} \Gamma'_{k+1}$. Thus, path $(\Gamma'_i)_{i \in I}$ satisfies clause **TL1** of definition of live path.
- **TL2**: $\Gamma'_i(p) \equiv \left(\sigma_p, \sum_{j \in J \cup L} q_j ? \ell_j (S_j). T'_j \right)$. Since $\Gamma'_i \leq \Gamma_i$, by Lemma 29, we have $\Gamma_i(p) \equiv \left(\sigma_p, \sum_{j \in J} q_j ? \ell_j (S_j). T_j \right)$, where $T'_j \leq T_j$, for $j \in J$, and $\{q_j\}_{j \in J \cup L} = \{q_j\}_{j \in J}$. From Γ being live, we obtain there $\exists k, q, \ell$ such that $I \ni k \geq i$, $(q, \ell) \in \{(q_j, \ell_j)\}_{j \in J}$, and $\Gamma_k \xrightarrow{p:q?\ell} \Gamma_{k+1}$. Hence, $\alpha_{k+1} = p:q?\ell$ in both (13) and (12), i.e., we obtain there $\exists k, q, \ell$ such that $I \ni k \geq i$, $(q, \ell) \in \{(q_j, \ell_j)\}_{j \in J \cup L}$, and $\Gamma'_k \xrightarrow{p:q?\ell} \Gamma'_{k+1}$. Thus, path $(\Gamma'_i)_{i \in I}$ satisfies clause **TL2** of definition of live path.

◀

B Proofs of Section 4

- **Lemma 32** (Typing congruence). 1. If $\Theta \vdash P : T$ and $P \Rightarrow Q$, then $\Theta \vdash Q : T$.
 2. If $\vdash h_1 : \sigma_1$ and $h_1 \equiv h_2$, then there is σ_2 such that $\sigma_1 \equiv \sigma_2$ and $\vdash h_2 : \sigma_2$.
 3. If $\Gamma \vdash M$ and $M \Rightarrow M'$, then there is Γ' such that $\Gamma \equiv \Gamma'$ and $\Gamma' \vdash M'$.

Proof. The proof is by case analysis. ◀

- **Lemma 33** (Substitution). If $\Theta, x : S \vdash P : T$ and $\Theta \vdash v : S$, then $\Theta \vdash P\{v/x\} : T$.

Proof. By structural induction on P . ◀

- **Theorem 23** (Subject Reduction). Assume $\Gamma \vdash M$ with Γ safe/deadlock-free/live and $M \longrightarrow M'$. Then, there is safe/deadlock-free/live type environment Γ' such that $\Gamma \longrightarrow^* \Gamma'$ and $\Gamma' \vdash M'$.

Proof. Assume:

$$\Gamma \vdash M \quad \text{where} \quad M = \prod_{j \in J} (p_j \triangleleft P_j \mid p_j \triangleleft h_j) \quad (\text{for some } J) \quad (\text{by hypothesis}) \quad (14)$$

$$\Gamma \text{ is safe/deadlock-free/live} \quad (\text{by hypothesis}) \quad (15)$$

From (14), we know that there is a typing derivation for M starting with rule [T-SESS]:

$$\frac{\Gamma = \{p_j : (\sigma_j, T_j) \mid j \in J\} \quad \forall j \in J \quad \vdash P_j : T_j \quad \vdash h_j : \sigma_j}{\Gamma \vdash M} [\text{T-SESS}] \quad (16)$$

We proceed by induction on the derivation of $M \longrightarrow M'$.

Base cases:

[R-SEND]: We have:

$$\mathcal{M} = \mathbf{p}_n \triangleleft \sum_{i \in I} \mathbf{q}_i ! \ell_i \langle \mathbf{v}_i \rangle . P_i \mid \mathbf{p}_n \triangleleft h \mid \mathcal{M}_1 \quad (17)$$

$$\mathcal{M}' = \mathbf{p}_n \triangleleft P_k \mid \mathbf{p}_n \triangleleft h \cdot (\mathbf{q}_k, \ell_k(\mathbf{v}_k)) \mid \mathcal{M}_1 \quad (18)$$

$$\mathcal{M}_1 = \prod_{j \in J \setminus \{n\}} (\mathbf{p}_j \triangleleft P_j \mid \mathbf{p}_j \triangleleft h_j) \quad (19)$$

By inversion on the typing rules we have:

$$\vdash \sum_{i \in I} \mathbf{q}_i ! \ell_i \langle \mathbf{v}_i \rangle . P_i : \mathsf{T}_{\mathbf{p}_n} \quad \text{where} \quad \sum_{i \in I} \mathbf{q}_i ! \ell_i \langle \mathsf{S}_i \rangle . \mathsf{T}_i \leq \mathsf{T}_{\mathbf{p}_n} \quad (20)$$

$$\forall i \in I \quad \vdash \mathbf{v}_i : \mathsf{S}_i \quad (21)$$

$$\forall i \in I \quad \vdash P_i : \mathsf{T}_i \quad (22)$$

$$\vdash h : \sigma \quad (23)$$

$$\forall j \in J \setminus \{n\} \quad \vdash P_j : \mathsf{T}_j \quad (24)$$

$$\forall j \in J \setminus \{n\} \quad \vdash h_j : \sigma_j \quad (25)$$

$$\Gamma_1 = \{\mathbf{p}_n : (\sigma, \sum_{i \in I} \mathbf{q}_i ! \ell_i \langle \mathsf{S}_i \rangle . \mathsf{T}_i)\} \cup \{\mathbf{p}_j : (\sigma_j, \mathsf{T}_j) : j \in J \setminus \{n\}\} \quad (26)$$

$$\Gamma_1 \leq \Gamma \quad \text{and} \quad \Gamma_1 \text{ is safe/deadlock-free/live by Lemma 15} \quad (27)$$

Now, let:

$$\Gamma'_1 = \{\mathbf{p}_n : (\sigma \cdot \mathbf{q}_k ! \ell_k \langle \mathsf{S}_k \rangle, \mathsf{T}_k)\} \cup \{\mathbf{p}_j : (\sigma_j, \mathsf{T}_j) : j \in J \setminus \{n\}\} \quad (28)$$

Then, we conclude:

$$\Gamma_1 \longrightarrow \Gamma'_1 \quad (\text{by (26), (28), and [E-SEND] of Def. 8}) \quad (29)$$

$$\Gamma'_1 \text{ is safe/deadlock-free/live} \quad (\text{by (27), (29), and Proposition 11}) \quad (30)$$

$$\Gamma'_1 \vdash \mathcal{M}' \quad (\text{by (20)–(25), and Def. 17}) \quad (31)$$

$$\exists \Gamma' : \Gamma'_1 \leq \Gamma' \quad \Gamma \longrightarrow \Gamma' \quad (\text{by (14), (27), (29), and Lemma 14}) \quad (32)$$

$$\Gamma' \vdash \mathcal{M}' \quad (\text{by (31), (32), and Lemma 21}) \quad (33)$$

$$\Gamma' \text{ is safe/deadlock-free/live} \quad (\text{by (14), (32) and Proposition 11}) \quad (34)$$

[R-RCV]: We have:

$$\mathcal{M} = \mathbf{p}_n \triangleleft \sum_{i \in I} \mathbf{q}_i ? \ell_i (x_i) . P_i \mid \mathbf{p}_n \triangleleft h_{\mathbf{p}} \mid \mathbf{p}_m \triangleleft P_{\mathbf{q}} \mid \mathbf{p}_m \triangleleft (\mathbf{p}_n, \ell(\mathbf{v})) \cdot h \mid \mathcal{M}_1$$

$$(\exists k \in I : (\mathbf{q}_k, \ell_k) = (\mathbf{p}_m, \ell)) \quad (35)$$

$$\mathcal{M}' = \mathbf{p}_n \triangleleft P_k \{ \mathbf{v} / x_k \} \mid \mathbf{p}_n \triangleleft h_{\mathbf{p}} \mid \mathbf{p}_m \triangleleft P_{\mathbf{q}} \mid \mathbf{p}_m \triangleleft h \mid \mathcal{M}_1 \quad (36)$$

$$\mathcal{M}_1 = \prod_{j \in J \setminus \{n, m\}} (\mathbf{p}_j \triangleleft P_j \mid \mathbf{p}_j \triangleleft h_j) \quad (37)$$

By inversion on the typing rules we have:

$$\vdash \sum_{i \in I} q_i ? \ell_i(x_i).P_i : T_{p_n} \quad \text{where} \quad \sum_{i \in I} q_i ? \ell_i(S_i).T_i \leq T_{p_n} \quad (38)$$

$$\vdash h_p : \sigma_p \quad (39)$$

$$\vdash P_q : T_q \quad (40)$$

$$\vdash (p_n, \ell(v)) \cdot h : p_n ! \ell \langle S \rangle \cdot \sigma \quad (41)$$

$$\forall j \in J \setminus \{n, m\} : \vdash P_j : T_j \quad (42)$$

$$\forall j \in J \setminus \{n, m\} : \vdash h_j : \sigma_j \quad (43)$$

$$\Gamma_1 = \{p_n : (\sigma_p, \sum_{i \in I} q_i ? \ell_i(S_i).T_i), p_m : (p_n ! \ell \langle S \rangle \cdot \sigma, T_q)\} \cup \{p_j : (\sigma_j, T_j) : j \in J \setminus \{n, m\}\} \quad (44)$$

$$\Gamma_1 \leq \Gamma \quad \text{and} \quad \Gamma_1 \text{ is safe/deadlock-free/live by Lemma 15} \quad (45)$$

From (38) and (41) by inversion on the typing rules we have

$$\forall i \in I : x_i : S_i \vdash P_i : T_i \quad (46)$$

$$\vdash h : \sigma \quad \text{and} \quad \vdash v : S \quad (47)$$

By (35) and Γ_1 is safe (by (45)), we obtain

$$(q_k, \ell_k, S_k) = (p_m, \ell, S) \quad (48)$$

By Lemma 33, (46), (47), and (48) we have

$$\vdash P_k \{v/x_k\} : T_k \quad (49)$$

Now let:

$$\Gamma'_1 = \{p_n : (\sigma_p, T_k), p_m : (\sigma, T_q)\} \cup \{p_j : (\sigma_j, T_j) : j \in J \setminus \{n, m\}\} \quad (50)$$

And we conclude:

$$\Gamma_1 \longrightarrow \Gamma'_1 \quad (\text{by (44), (48), (50), and rule [E-RCV] of Def. 8}) \quad (51)$$

$$\Gamma'_1 \text{ is safe/deadlock-free/live} \quad (\text{by (45), (51), and Proposition 11}) \quad (52)$$

$$\Gamma'_1 \vdash \mathcal{M}' \quad (\text{by (38)-(43), (49), and Definition 17}) \quad (53)$$

$$\exists \Gamma' : \Gamma'_1 \leq \Gamma' \quad \Gamma \longrightarrow \Gamma' \quad (\text{by (45), (51), and Lemma 14}) \quad (54)$$

$$\Gamma' \vdash \mathcal{M}' \quad (\text{by (53), (54), and Lemma 21)) \quad (55)$$

$$\Gamma' \text{ is safe/deadlock-free/live} \quad (\text{by (15), (51), and Proposition 11}) \quad (56)$$

[R-COND-T] ([R-COND-F]): We have:

$$\mathcal{M} = p_n \triangleleft \text{if } v \text{ then } P \text{ else } Q \mid p_n \triangleleft h \mid \mathcal{M}_1 \quad (57)$$

$$\mathcal{M}' = p_n \triangleleft P \mid p_n \triangleleft h \mid \mathcal{M}_1 \quad (\mathcal{M}' = p_n \triangleleft Q \mid p_n \triangleleft h \mid \mathcal{M}_1) \quad (58)$$

$$\mathcal{M}_1 = \prod_{j \in J \setminus \{n\}} (p_j \triangleleft P_j \mid p_j \triangleleft h_j) \quad (59)$$

By inversion on the typing rules we have:

$$\vdash \text{if } v \text{ then } P \text{ else } Q : T \quad (60)$$

$$\vdash h : \sigma \quad (61)$$

$$\forall j \in J \setminus \{n\} : \vdash P_j : T_j \quad (62)$$

$$\forall j \in J \setminus \{n\} : \vdash h_j : \sigma_j \quad (63)$$

$$\Gamma = \{p_n : (\sigma, T)\} \cup \{p_j : (\sigma_j, T_j) : j \in J \setminus \{n\}\} \quad (64)$$

By inversion on the typing rules we have $\exists T' : T' \leq T$ such that:

$$\vdash P : T' \quad (65)$$

$$\vdash Q : T' \quad (66)$$

$$\vdash v : \text{bool} \quad (67)$$

But then, by rule $[T\text{-SUB}]$ we have:

$$\vdash P : T \quad (68)$$

$$\vdash Q : T \quad (69)$$

Then, letting $\Gamma' = \Gamma$, we have:

$$\Gamma' \text{ is safe/deadlock-free/live} \quad (\text{by (15)}) \quad (70)$$

$$\Gamma' \vdash \mathcal{M}' \quad (\text{by (58), (64), and (68) (or (69))}) \quad (71)$$

Inductive step:

$[R\text{-STRUCT}]$ Assume that $\mathcal{M} \longrightarrow \mathcal{M}'$ is derived from:

$$\mathcal{M} \Rightarrow \mathcal{M}_1 \quad (72)$$

$$\mathcal{M}_1 \longrightarrow \mathcal{M}'_1 \quad (73)$$

$$\mathcal{M}'_1 \Rightarrow \mathcal{M}' \quad (74)$$

By (72), (14), and Lemma 32, there is Γ_1 such that

$$\Gamma_1 \equiv \Gamma \quad (75)$$

$$\Gamma_1 \vdash \mathcal{M}_1 \quad (76)$$

Since Γ is safe/deadlock-free/live and $\Gamma \equiv \Gamma_1$, by Proposition 28, Γ_1 is also safe/deadlock-free/live. Since $\mathcal{M}_1 \longrightarrow \mathcal{M}'_1$, by induction hypothesis there is a safe/deadlock-free/live type environment Γ'_1 such that:

$$\Gamma_1 \longrightarrow \Gamma'_1 \text{ or } \Gamma_1 \equiv \Gamma'_1 \quad (77)$$

$$\Gamma'_1 \vdash \mathcal{M}'_1 \quad (78)$$

Since Γ_1 is safe/deadlock-free/live, by Proposition 11 or Proposition 28, and (77), we have Γ'_1 is safe/deadlock-free/live. Now, by (74), (78), and Lemma 32, there is a environment Γ' such that

$$\Gamma' \equiv \Gamma'_1 \quad (79)$$

$$\Gamma' \vdash \mathcal{M}' \quad (80)$$

Since Γ'_1 is safe/deadlock-free/live and $\Gamma' \equiv \Gamma'_1$, by Proposition 28, Γ' is also safe/deadlock-free/live. We now may conclude with:

$$\Gamma \longrightarrow \Gamma' \text{ or } \Gamma \equiv \Gamma' \quad (\text{by (75), (77), (79) and rule [E-STRUCT] of Def. 8}) \quad (81)$$

◀

► **Theorem 24** (Type Safety). *If $\Gamma \vdash \mathcal{M}$ and Γ is safe, then \mathcal{M} is safe.*

Proof. Assume \mathcal{M} is not safe. Then, there is a session path $(\mathcal{M}_i)_{i \in I}$, starting with $\mathcal{M}_0 = \mathcal{M}$, such that for some $i \in I$

$$\begin{aligned} \mathcal{M}_i &\Rightarrow \mathbf{p} \triangleleft \sum_{j \in J} \mathbf{q}_j ? \ell_j(x_j).P_j \mid \mathbf{p} \triangleleft h \mid \mathbf{q} \triangleleft Q \mid \mathbf{q} \triangleleft (\mathbf{p}, \ell(\mathbf{v})) \cdot h_{\mathbf{q}} \mid \mathcal{M}' \\ \text{with } \mathbf{q} &\in \{\mathbf{q}_j\}_{j \in J} \text{ and } \forall j \in J \quad (\mathbf{q}, \ell) \neq (\mathbf{q}_j, \ell_j) \end{aligned} \quad (82)$$

Since $\mathcal{M} = \mathcal{M}_0 \longrightarrow \dots \longrightarrow \mathcal{M}_i$ and Γ is safe, by consecutive application of Theorem 23, we obtain $\Gamma = \Gamma_0 \longrightarrow^* \dots \longrightarrow^* \Gamma_i$, where Γ_i is safe and $\Gamma_i \vdash \mathcal{M}_i$.

By Lemma 32 there is Γ'_i such that $\Gamma_i \equiv \Gamma'_i$ and

$$\Gamma'_i \vdash \mathbf{p} \triangleleft \sum_{j \in J} \mathbf{q}_j ? \ell_j(x_j).P_j \mid \mathbf{p} \triangleleft h \mid \mathbf{q} \triangleleft Q \mid \mathbf{q} \triangleleft (\mathbf{p}, \ell(\mathbf{v})) \cdot h_{\mathbf{q}} \mid \mathcal{M}'$$

Since Γ_i is safe, by Proposition 28, we have that Γ'_i is safe. Same as in the proof of Theorem 23 (see (44) and (45)) we may conclude there is Γ''_i , such that $\Gamma''_i \leq \Gamma'_i$ and

$$\Gamma''_i = \{\mathbf{p} : (\sigma_{\mathbf{p}}, \sum_{j \in J} \mathbf{q}_j ? \ell_j(S_j).T_j), \mathbf{q} : (\mathbf{p} ! \ell(S) \cdot \sigma, T_{\mathbf{q}})\} \cup \Gamma'''_i$$

with $\mathbf{q} \in \{\mathbf{q}_j\}_{j \in J}$, and Γ''_i is safe by Lemma 15. Since Γ''_i is safe, by **TS**, $\exists j \in J : (\mathbf{q}_j, \ell_j, S_j) = (\mathbf{q}_k, \ell_k, S_k)$, which contradicts (82). ◀

► **Theorem 25** (Session Fidelity). *Let $\Gamma \vdash \mathcal{M}$. If $\Gamma \longrightarrow$, then $\exists \Gamma', P'$ such that $\Gamma \longrightarrow \Gamma'$ and $\mathcal{M} \longrightarrow^+ \mathcal{M}'$ and $\Gamma' \vdash \mathcal{M}'$.*

Proof. We distinguish two cases for deriving $\Gamma \longrightarrow$.

Case 1: $\Gamma \xrightarrow{\mathbf{q}:\mathbf{p}_k ? \ell_k} \Gamma$. Then:

$$\Gamma \equiv \mathbf{p}_k : (\mathbf{q} ! \ell_k(S_k) \cdot \sigma, T_{\mathbf{p}}), \mathbf{q} : (\sigma_{\mathbf{q}}, \sum_{i \in I} \mathbf{p}_i ? \ell_i(S_i).T_i), \Gamma_1 \xrightarrow{\mathbf{q}:\mathbf{p}_k ? \ell_k} \mathbf{p}_k : (\sigma, T_{\mathbf{p}}), \mathbf{q} : (\sigma_{\mathbf{q}}, T_k), \Gamma_1 = \Gamma'$$

where $k \in I$, and where in structural congruence (starting with Γ) we can assume folding of types is not used (since only the unfolding is necessary). By inversion on the typing rules we derive

$$\mathcal{M} \Rightarrow \mathbf{p}_k \triangleleft P \mid \mathbf{p}_k \triangleleft (\mathbf{q}, \ell_k(\mathbf{v})) \cdot h \mid \mathbf{q} \triangleleft \sum_{i \in I \cup J} \mathbf{p}_i ? \ell_i(x_i).P_i \mid \mathbf{p} \triangleleft h_{\mathbf{q}} \mid \mathcal{M}_1$$

for some P, \mathbf{v}, h, P_i (for $i \in I \cup J$), $h_{\mathbf{q}}$, and \mathcal{M}_1 , such that

$$\vdash P : T_{\mathbf{p}} \quad \vdash \mathbf{v} : S_k \quad \vdash h : \sigma \quad x_i : S_i \vdash P_i : T_i \quad \vdash h_{\mathbf{q}} : \sigma_{\mathbf{q}} \quad \Gamma_1 \vdash \mathcal{M}_1$$

Since $k \in I (\subseteq I \cup J)$, we can show that for

$$\mathcal{M}' = \mathbf{p}_k \triangleleft P \mid \mathbf{p}_k \triangleleft h \mid \mathbf{q} \triangleleft P_k\{\mathbf{v}/x_k\} \mid \mathbf{q} \triangleleft h_{\mathbf{q}} \mid \mathcal{M}_1$$

we have $\mathcal{M} \longrightarrow \mathcal{M}'$ and $\Gamma' \vdash \mathcal{M}'$.

Case 2: $\Gamma \xrightarrow{p:q_k!\ell_k}$. We have:

$$\Gamma \equiv p : (\sigma, \sum_{i \in I \cup J} q_i! \ell_i \langle S_i \rangle . T_i), \Gamma_1 \xrightarrow{p:q_k!\ell_k}$$

where $k \in I \cup J$, and where in structural congruence (starting with Γ) we can assume folding of types is not used (since only the unfolding is necessary). By inversion on the typing rules, we have

$$\mathcal{M} \Rightarrow p \triangleleft \sum_{i \in I} q_i! \ell_i \langle v_i \rangle . P_i \mid p \triangleleft h \mid \mathcal{M}_1$$

for some v_i, P_i (for $i \in I$), h , and \mathcal{M}_1 , such that

$$\vdash v_i : S_i \quad \vdash P_i : T_i \quad \vdash h : \sigma \quad \Gamma_1 \vdash \mathcal{M}_1$$

Now for $k \in I$ defining

$$\Gamma'_k = p : (\sigma \cdot q_k! \ell_k \langle S_k \rangle, T_k), \Gamma_1 \quad \text{and} \quad \mathcal{M}'_k = p \triangleleft P_k \mid p \triangleleft h \cdot (q_k, \ell_k(v_k)) \mid \mathcal{M}_1$$

we obtain $\Gamma \longrightarrow \Gamma'_k$, $\mathcal{M} \longrightarrow \mathcal{M}'_k$ and $\Gamma'_k \vdash \mathcal{M}'_k$. \blacktriangleleft

► **Theorem 26** (Deadlock freedom). *If $\Gamma \vdash \mathcal{M}$ and Γ is deadlock-free, then \mathcal{M} is deadlock-free.*

Proof. Take a path $(\mathcal{M}_i)_{i \in I}$, starting with $\mathcal{M}_0 = \mathcal{M}$, such that $I = \{0, 1, \dots, n\}$ and $\mathcal{M}_n \not\rightarrow$. Since $\mathcal{M} = \mathcal{M}_0 \longrightarrow \dots \longrightarrow \mathcal{M}_n$ and Γ is deadlock-free, by consecutive application of Theorem 23, we obtain $\Gamma = \Gamma_0 \longrightarrow^* \dots \longrightarrow^* \Gamma_n$, where $\Gamma_i \vdash \mathcal{M}_i$. Since $\mathcal{M}_n \not\rightarrow$, by contrapositive of Theorem 25, we obtain $\Gamma_n \not\rightarrow$. Since Γ is deadlock-free, we have $\text{end}(\Gamma_n)$, and with $\Gamma_n \vdash \mathcal{M}_n$ and by inversion on the typing rules we conclude $\mathcal{M}_n \Rightarrow p \triangleleft \mathbf{0} \mid p \triangleleft \emptyset$. This proves \mathcal{M} is deadlock-free. \blacktriangleleft

► **Remark 34.** The proof of deadlock-freedom [17, Theorem 5.2] uses reasoning that by subject reduction [17, Theorem 5.1] from $\Gamma' \vdash \mathcal{M}'$ and $\mathcal{M}' \not\rightarrow$ one can conclude $\Gamma' \not\rightarrow$. Such conclusion can be a consequence of session fidelity, not subject reduction.

► **Theorem 27** (Liveness). *If $\Gamma \vdash \mathcal{M}$ and Γ is live, then \mathcal{M} is live.*

Proof. Assume \mathcal{M} is not live. Then, there is a fair session path $(\mathcal{M}_i)_{i \in I}$, starting with $\mathcal{M}_0 = \mathcal{M}$, that violate one of the clauses:

- **SL1:** There is $i \in I$ such that $\mathcal{M}_i \Rightarrow p \triangleleft \text{if } v \text{ then } P \text{ else } Q \mid p \triangleleft h \mid \mathcal{M}'$, and $\forall k$ such that $I \ni k \geq i$ we have that $\mathcal{M}_k \Rightarrow p \triangleleft P \mid p \triangleleft h \mid \mathcal{M}''$ and $\mathcal{M}_k \Rightarrow p \triangleleft Q \mid p \triangleleft h \mid \mathcal{M}''$ do not hold. By Theorem 23 we obtain that \mathcal{M}_i is also typable, which by inspection of the typing rules implies that v is a boolean (true or false). Hence, we have that \mathcal{M}_i can reduce the p 's conditional process, but along path $(\mathcal{M}_i)_{i \in I}$ this is never scheduled. This implies path $(\mathcal{M}_i)_{i \in I}$ also violates clause **SF3**, and it is not fair - which is a contradiction.
- **SL2:** There is $i \in I$ such that $\mathcal{M}_i \Rightarrow p \triangleleft \sum_{j \in J} q_j! \ell_j \langle v_j \rangle . P_j \mid p \triangleleft h \mid \mathcal{M}'$, and $\forall k$ such that $I \ni k \geq i$ we have that $\mathcal{M}_k \xrightarrow{p:q_j!\ell_j} \mathcal{M}_{k+1}$, for any $j \in J$ does not hold. By Theorem 23 we obtain that \mathcal{M}_i is also typable, which by inspection of the typing rules implies that v_j is a value (a number or a boolean) and not a variable, for all $j \in J$. This is a consequence of the fact that we can only type closed processes (by $[\text{T-SESS}]$). Hence, we have that \mathcal{M}_i can reduce the p 's internal choice process, but along path $(\mathcal{M}_i)_{i \in I}$ this is never scheduled. This implies path $(\mathcal{M}_i)_{i \in I}$ also violates clause **SF1**, and it is not fair - which is a contradiction.

- **SL3:** There is $i \in I$ such that $\mathcal{M}_i \Rightarrow \mathbf{p} \triangleleft P \mid \mathbf{p} \triangleleft (q, \ell(v)) \cdot h \mid \mathcal{M}'$, and we have that $\forall k$ such that $I \ni k \geq i$, $\mathcal{M}_k \xrightarrow{\mathbf{q}:\mathbf{p}?\ell} \mathcal{M}_{k+1}$ does not hold. Since $\Gamma \vdash \mathcal{M}$ and Γ is live, following the proof of Theorem 23, we can construct typing environment path $(\Gamma_j)_{j \in J}$, with $\Gamma_0 = \Gamma$, where for $i \in I$, if

$$\begin{aligned} \mathcal{M}_i \xrightarrow{\mathbf{p}:\mathbf{q}!\ell} \mathcal{M}_{i+1} \quad \text{or} \quad \mathcal{M}_i \xrightarrow{\mathbf{p}:\mathbf{q}?\ell} \mathcal{M}_{i+1} \quad \text{or} \quad \mathcal{M}_i \xrightarrow{\mathbf{p}:\text{if}} \mathcal{M}_{i+1} \quad \text{then} \\ \Gamma_i \xrightarrow{\mathbf{p}:\mathbf{q}!\ell} \Gamma_{i+1} \quad \text{or} \quad \Gamma_i \xrightarrow{\mathbf{p}:\mathbf{q}?\ell} \Gamma_{i+1} \quad \text{or} \quad \Gamma_i = \Gamma_{i+1} \quad \text{respectively} \end{aligned} \quad (83)$$

Notice that trace of $(\Gamma_j)_{j \in J}$ is exactly the same as $(\mathcal{M}_i)_{i \in I}$ excluding the if reductions. Since $(\mathcal{M}_i)_{i \in I}$ is fair (satisfies clauses **SF1**, **SF2**, and **SF3**), we conclude $(\Gamma_j)_{j \in J}$ is also fair (satisfies clauses **TF1** and **TF2**). From $\Gamma_i \vdash \mathcal{M}_i$, we may conclude $\Gamma_i(\mathbf{p}) \equiv (\mathbf{q}!\ell\langle S \rangle \cdot \sigma, \top)$, for some σ and \top . But then, $\forall k$ such that $J \ni k \geq i$ we have that $\Gamma_k \xrightarrow{\mathbf{q}:\mathbf{p}?\ell} \Gamma_{k+1}$ does not hold. This implies $(\Gamma_j)_{j \in J}$ is fair but not live - a contradiction with assumption that Γ is live.

- **SL4:** There is $i \in I$ such that $\mathcal{M}_i \Rightarrow \mathbf{p} \triangleleft \sum_{s \in S \cup N} \mathbf{q}_s ? \ell_s(x_s) \cdot P_s \mid \mathbf{p} \triangleleft h \mid \mathcal{M}'$, and $\forall k$ such that $I \ni k \geq i$, we have that $\mathcal{M}_k \xrightarrow{\mathbf{p}:\mathbf{q}?\ell} \mathcal{M}_{k+1}$, where $(\mathbf{q}, \ell) \in \{(\mathbf{q}_s, \ell_s)\}_{s \in S \cup N}$, does not hold. As in the previous case in (83), for fair path $(\mathcal{M}_i)_{i \in I}$ we may construct fair path $(\Gamma_j)_{j \in J}$. From $\Gamma_i \vdash \mathcal{M}_i$ we may conclude $\Gamma_i(\mathbf{p}) \equiv (\sigma_{\mathbf{p}}, \sum_{s \in S} \mathbf{q}_s ? \ell_s(S_s) \cdot \top_s)$, for some $\sigma_{\mathbf{p}}$, S_s , and \top_s , for $s \in S$. But then, $\forall k, \mathbf{q}, \ell$ such that $J \ni k \geq i$ and $(\mathbf{q}, \ell) \in \{(\mathbf{q}_s, \ell_s)\}_{s \in S}$, we have that $\Gamma_k \xrightarrow{\mathbf{p}:\mathbf{q}?\ell} \Gamma_{k+1}$ does not hold. This implies $(\Gamma_j)_{j \in J}$ is fair but not live - a contradiction with assumption that Γ is live.

◀