Fault-Tolerant Multiparty Session Types with Global Escape Loops

Lukas Bartl Julian Linne Kirstin Peters
Universität Augsburg, Germany

Multiparty session types are designed to abstractly capture the structure of communication protocols and verify behavioural properties. One important such property is progress, i.e., the absence of deadlock. Distributed algorithms often resemble multiparty communication protocols. But proving their properties, in particular termination that is closely related to progress, can be elaborate. Since distributed algorithms are often designed to cope with faults, a first step towards using session types to verify distributed algorithms is to integrate fault-tolerance.

We extend FTMPST—a version of fault-tolerant multiparty session types with failure patterns to represent system requirements for system failures such as unreliable communication and process crashes—by a novel, fault-tolerant loop construct with global escapes that does not require global coordination. Each process runs its own local version of the loop. If a process finds a solution to the considered problem, it does not only terminate its own loop but also informs the other participants via exit-messages. Upon receiving an exit-message, a process immediately terminates its algorithm. To increase efficiency and model standard fault-tolerant algorithms, these messages are non-blocking, i.e., a process may continue until a possibly delayed exit-message is received. To illustrate our approach, we analyse a variant of the well-known rotating coordinator algorithm by Chandra and Toueg.

1 Introduction

Multi-Party Session Types (MPST) are used to statically ensure correctly coordinated behaviour in systems without global control ([18, 13]). One important such property is progress, i.e., the absence of deadlock. Like with every other static typing approach, the main advantage is their efficiency, i.e., they avoid the problem of state space explosion. MPST are designed to abstractly capture the structure of communication protocols. They describe global behaviours as *sessions*, i.e., units of conversations [18, 3, 4]. The participants of such sessions are called *roles*. *Global types* specify protocols from a global point of view. These types are used to reason about processes formulated in a *session calculus*.

Distributed algorithms (DA) very much resemble multiparty communication protocols. An essential behavioural property of DA is termination [27, 21], despite failures, but it is often elaborate to prove. It turns out that progress (as provided by MPST) and termination (as required by DA) are closely related.

Many DA were designed in a fault-tolerant way, in order to work in environments, where they have to cope with system failures—be it links dropping messages or processes crashing. We focus on masking fault-tolerant algorithms (see [16]), i.e., safety and liveness requirements hold despite failures without further intervention by the programmer.

While the detection of conceptual design errors is a standard property of type systems, proving correctness of algorithms despite the occurrence of system failures is not. Likewise, traditional MPST do not cover fault tolerance or failure handling. There are several approaches to integrate explicit failure handling in MPST (e.g. [7, 6, 12, 28, 14, 1]). These approaches are sometimes enhanced with recovery mechanisms such as [8] or even provide algorithms to help find safe states to recover from, as in [22]. Many of these approaches introduce nested TRY-and-CATCH-blocks and a challenge is to

Submitted to:
Express/SOS 2025

ensure that all participants are consistently informed about concurrent THROWS of exceptions. Therefore, exceptions are propagated within the system. Though explicit failure handling makes sense for high-level applications, the required message overhead is too inefficient for many low-level algorithms. Instead, these low-level algorithms are often designed to tolerate a certain amount of failures. Since we focus on the communication structure of systems, additional messages as reaction to faults (e.g. to propagate faults) are considered *non-masking* failure handling. In contrast, we expect masking fault-tolerant algorithms to cope without messages triggered by faults. We study how much unhandled failures a well-typed system can tolerate, while maintaining the typical properties of MPST.

Type systems are usually designed for failure-free scenarios. An exception is [19] that introduces unreliable broadcast, where a transmission can be received by multiple receivers but not necessarily all available receivers. In the latter case, the receiver is deadlocked. In contrast, we consider fault-tolerant interactions, where in the case of a failure the receiver is *not* deadlocked.

The already mentioned systems in [7, 6, 12, 28, 14] extend session types with exceptions thrown by processes within TRY-and-CATCH-blocks, interrupts, or similar syntax. They structurally and semantically encapsulate an unreliable part of a protocol and provide some means to 'detect' a failure and 'react' to it. Here we deliberately do not model how to 'detect' a failure. Different system architectures might provide different mechanisms to do so, for example, by means of time-outs. As is standard for the analysis of DA, our approach allows us to port the verified algorithms on different system architectures that satisfy the necessary system requirements.

Another essential difference is how systems react to faults. In [6], THROW-messages are propagated among nested TRY-and-CATCH-blocks to ensure that all participants are *consistently* informed about concurrent THROWS of exceptions. Fault-tolerant DA, however, have to deal with the problem of inconsistency; one of their most challenging problems. *Distributed* processes usually cannot reliably observe an error on another system part, unless they are informed by some system "device" (like the "coordinator" of [28] or the "oracle" of [6]). Therefore, abstractions like unreliable failure detectors are used to model this restricted observability which can, for example, be implemented by time-outs.

We extend FTMPST [23, 24], a version of fault-tolerant multiparty session types with failure patterns to represent system requirements for system failures such as unreliable communication and process crashes. We add a novel, fault-tolerant loop construct with global escapes but without a need for global coordination. Thereby, we tackle an open question of [24], namely how to conveniently type unreliable recursive parts of protocols. Distributed algorithms are often recursive and exit this recursion if a result was successfully computed. In [24], weakly reliable branching was used to exit a standard recursion. Unfortunately, this operation temporarily blocks some processes. Our novel loop construct overcomes this problem.

Each loop of an algorithm has a unique identifier, where unique means from a global point of view. Each process runs its own local version of the loop, but the local loops that jointly define a recursive routine of the algorithm have the same identifier. If a process finds a solution to the considered problem, it does not only terminate its own loop but also informs the other participants via exit-messages that may carry a solution value. Upon receiving an exit-message, a process immediately terminates its algorithm. To increase efficiency and model standard fault-tolerant algorithms, these messages are non-blocking, i.e., a process may continue until a possibly delayed exit-message is received. Since communication in the system is asynchronous and because of faults such as message delays, many algorithms do not forbid that different participants terminate the protocol concurrently. Hence, there may be several concurrent exit-messages for the same local loop. The algorithm then has to ensure, that all of them carry the same solution value—usually called *agreement*.

To guide the behaviour of unreliable communication, we inherit from [24] the *failure pattern* used in the semantics of processes. Note that these pattern are not defined but *could* be instantiated by an application. This allows us to cover requirements on the system—as, e.g., a bound on the number of faulty processes—as well as more abstract concepts like failure detectors. It is beyond the scope of this paper to discuss *how* failure patterns could be implemented. To illustrate our approach we analyse a variant of the well-known rotating coordinator algorithm by Chandra and Toueg.

2 Fault-Tolerant Types and Processes

Following [24], we consider three levels of failures in interactions:

Strongly Reliable (r) Neither the sender nor the receiver can crash as long as they are involved in this interaction. The message cannot be lost by the communication medium. This form corresponds to reliable communication as it was described in [2] in the context of distributed algorithms. This is the standard, failure-free case.

Weakly Reliable (w) Both the sender and the receiver might crash at every possible point during this interaction. But the communication medium cannot lose the message.

Unreliable (u) Both the sender and the receiver might crash at every possible point during this interaction and the communication medium might lose the message. There are no guarantees that this interaction—or any part of it—takes place. Here, it is difficult to ensure interesting properties in branching.

We use the subscripts or superscripts r, w, or u to indicate actions of the respective kind. Our new loop construct relies on unreliable interactions for the loop body such that the termination of the loop does not cause any blocking of the interaction partners. However, the exit-messages should not be dropped before the loop is terminated and are thus weakly reliable.

For clarity, we often distinguish names into *values*, i.e., the payload of messages, *shared channels*, or *session channels* according to their usage; there is, however, no need to formally distinguish between different kinds of names.

We assume that the sets $\mathcal{N}$ of names $a, s, x \ldots$; $\mathcal{R}$ of roles $\mathsf{n}, \mathsf{r}, \ldots$; $\mathcal{L}$ of labels $l, l_{\mathsf{d}}, \ldots$; $\mathcal{V}_{\mathrm{T}}$ of type variables t; and $\mathcal{V}_{\mathrm{P}}$ of process variables X are pairwise distinct. To simplify the reduction semantics of our session calculus, we use natural numbers as roles (compare to [18]). Sorts S range over $\mathbb{B}, \mathbb{N}, \ldots$. The set $\mathcal{E}$ of expressions $e, v, b, \ldots$ is constructed from the standard Boolean operations, natural numbers, standard arithmetic operators, tuples, names, and (in)equalities. We assume an evaluation function $\mathrm{eval}(\cdot)$ that evaluates expressions to values.

Global types specify the desired communication structure from a global point of view. In local types, this global view is projected to the specification of a single role/participant. We start from standard MPST ([17, 18]) extended by unreliable communication and weakly reliable branching in [23, 24]. We then add an unreliable loop construct with weakly reliable global escapes (highlighted in blue) in Figure 1.

A new session s with n roles is initialised with $\overline{a}[\mathsf{n}](s).P$ and $a[\mathsf{r}](s).P$ via the shared channel a. We identify sessions with their unique session channel.

The type $\mathsf{r}_1 \to_{\mathrm{r}} \mathsf{r}_2{:}\langle \mathrm{S} \rangle.G$ specifies a strongly reliable communication from role r_1 to role r_2 to transmit a value of sort S and then continues with G. A system with this type will be guaranteed to perform a corresponding action. In a session s this communication is implemented by the sender $s[\mathsf{r}_1, \mathsf{r}_2]!_{\mathrm{r}}\langle e \rangle.P_1$ (specified as $[\mathsf{r}_2]!_{\mathrm{r}}\langle \mathrm{S} \rangle.T_1$) and the receiver $s[\mathsf{r}_2, \mathsf{r}_1]?_{\mathrm{r}}(x).P_2$ (specified as $[\mathsf{r}_1]?_{\mathrm{r}}\langle \mathrm{S} \rangle.T_2$). As a result, the receiver instantiates x in its continuation P_2 with the received value.

Global Types	Local Types	Processes
		$P ::= \overline{a}[\mathsf{n}](s).P \mid a[\mathsf{r}](s).P$
$G ::= \mathsf{r}_1 \to_{\mathrm{r}} \mathsf{r}_2{:}\langle \mathrm{S}\rangle.G$	$T ::= [\mathsf{r}_2]!_{\mathrm{r}}\langle \mathrm{S}\rangle.T$	$\mid s[\mathsf{r}_1,\mathsf{r}_2]!_{\mathrm{r}}\langle e\rangle.P$
	$\mid [\mathsf{r}_1]?_{\mathrm{r}}\langle \mathrm{S}\rangle.T$	$\mid s[\mathsf{r}_2,\mathsf{r}_1]?_{\mathrm{r}}(x).P$
$\mid \mathsf{r}_1 \to_{\mathrm{u}} \mathsf{r}_2{:}l\langle \mathrm{S}\rangle.G$	$\mid [\mathsf{r}_2]!_{\mathrm{u}}l\langle \mathrm{S}\rangle.T$	$\mid s[\mathsf{r}_1,\mathsf{r}_2]!_{\mathrm{u}}l\langle e\rangle.P$
	$\mid [\mathsf{r}_1]?_{\mathrm{u}}l\langle \mathrm{S}\rangle.T$	$\mid s[\mathsf{r}_2,\mathsf{r}_1]?_{\mathrm{u}}l\langle v\rangle(x).P$
$\mid \mathsf{r}_1 \to_{\mathrm{r}} \mathsf{r}_2{:}\{l_i.G_i\}_{i\in\mathrm{I}}$	$\mid [\mathsf{r}_2]!_{\mathrm{r}}\{l_i.T_i\}_{i\in\mathrm{I}}$	$\mid s[\mathsf{r}_1,\mathsf{r}_2]!_{\mathrm{r}}l.P$
	$\mid [\mathsf{r}_1]?_{\mathrm{r}}\{l_i.T_i\}_{i\in\mathrm{I}}$	$\mid s[\mathsf{r}_2,\mathsf{r}_1]?_{\mathrm{r}}\{l_i.P_i\}_{i\in\mathrm{I}}$
$\mid \mathsf{r} \to_{\mathrm{w}} \mathsf{R}{:}\{l_i.G_i\}_{i\in\mathrm{I},l_{\mathrm{d}}}$	$\mid [\mathsf{R}]!_{\mathrm{w}}\{l_i.T_i\}_{i\in\mathrm{I}}$	$\mid s[\mathsf{r},\mathsf{R}]!_{\mathrm{w}}l.P$
	$\mid [\mathsf{r}]?_{\mathrm{w}}\{l_i.T_i\}_{i\in\mathrm{I},l_{\mathrm{d}}}$	$\mid s[\mathsf{r}_j,\mathsf{r}]?_{\mathrm{w}}\{l_i.P_i\}_{i\in\mathrm{I},l_{\mathrm{d}}}$
$\mid G_1 \mid\mid G_2$		$\mid P_1 \mid P_2$
$\mid (\mu t,\mathsf{c})G \mid t$	$\mid (\mu t,\mathsf{c}=n)T \mid t$	$\mid (\mu X,\mathsf{c}=n)P \mid X$
$\mid \mathtt{end}$	$\mid \mathtt{end}$	$\mid \mathbf{0}$
$\mid [\mathsf{R}]\infty_e^{\mathsf{c}}\langle \mathrm{S}_0\rangle.G_0;\langle \mathrm{S}_2\rangle.G_2$	$\mid [\mathsf{R}]\infty_e^{\mathsf{c}=n}[\langle \mathrm{S}_0\rangle.T_0]T_1;\langle \mathrm{S}_2\rangle.T_2$	$\mid s[\mathsf{r},\mathsf{R}]\infty_e^{\mathsf{c}=n}[(x).P_0]P_1;(y).P_2$
$\mid \mathtt{call}\langle e\rangle$	$\mid \mathtt{call}\langle e\rangle$	$\mid \mathtt{call}\langle e,e'\rangle \mid \mathtt{exit}\langle e,e'\rangle$
		$\mid \mathtt{if}\ b\ \mathtt{then}\ P_1\ \mathtt{else}\ P_2$
		$\mid (\nu x)P \mid \perp$
$\mid \mathsf{r}_1 \to \mathsf{r}_2{:}\langle s'[\mathsf{r}]{:}T\rangle.G$	$\mid [\mathsf{r}_2]!\langle s'[\mathsf{r}]{:}T\rangle.T'$	$\mid s[\mathsf{r}_1,\mathsf{r}_2]!\langle\langle s'[\mathsf{r}]\rangle\rangle.P$
	$\mid [\mathsf{r}_1]?\langle s'[\mathsf{r}]{:}T\rangle.T'$	$\mid s[\mathsf{r}_2,\mathsf{r}_1]?((s'[\mathsf{r}])).P$
		$\mid s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{M}$

Message Types	Messages
$\mathsf{mt} ::= \langle \mathrm{S}\rangle^{\mathrm{r}} \mid l\langle \mathrm{S}\rangle^{\mathrm{u}} \mid l^{\mathrm{r}} \mid l^{\mathrm{w}}$	$\mathsf{m} ::= \langle v\rangle^{\mathrm{r}} \mid l\langle v\rangle^{\mathrm{u}} \mid l^{\mathrm{r}} \mid l^{\mathrm{w}}$
$\mid \mathtt{exit}\langle id,\mathrm{S}\rangle \mid s[\mathsf{r}]$	$\mid \mathtt{exit}\langle id,v\rangle \mid s[\mathsf{r}]$

Figure 1: Syntax of Fault-Tolerant MPST with Global Escape Loops.

The type $\mathsf{r}_1 \to_{\mathrm{u}} \mathsf{r}_2{:}l\langle \mathrm{S}\rangle.G$ specifies an unreliable communication from r_1 to r_2 transmitting (if successful) a label l and a value of sort S and then continues (regardless of the success of this communication) with G. The unreliable counterparts of senders and receivers are $s[\mathsf{r}_1,\mathsf{r}_2]!_{\mathrm{u}}l\langle e\rangle.P_1$ (specified as $[\mathsf{r}_2]!_{\mathrm{u}}l\langle \mathrm{S}\rangle.T_1$) and $s[\mathsf{r}_2,\mathsf{r}_1]?_{\mathrm{u}}l\langle v\rangle(x).P_2$ (specified as $[\mathsf{r}_1]?_{\mathrm{u}}l\langle \mathrm{S}\rangle.T_2$). The receiver $s[\mathsf{r}_2,\mathsf{r}_1]?_{\mathrm{u}}l\langle v\rangle(x).P_2$ declares a default value v that is used instead of a received value to instantiate x after a failure. Moreover, a label is communicated that helps us to ensure that a faulty unreliable communication does not influence later actions.

The strongly reliable branching $\mathsf{r}_1 \to_{\mathrm{r}} \mathsf{r}_2{:}\{l_i.G_i\}_{i\in\mathrm{I}}$ allows r_1 to pick one of the branches offered by r_2. We identify the branches with their respective label. Selection of a branch is by $s[\mathsf{r}_1,\mathsf{r}_2]!_{\mathrm{r}}l.P$ (specified as $[\mathsf{r}_2]!_{\mathrm{r}}\{l_i.T_i\}_{i\in\mathrm{I}}$). Upon receiving l_j, $s[\mathsf{r}_2,\mathsf{r}_1]?_{\mathrm{r}}\{l_i.P_i\}_{i\in\mathrm{I}}$ (specified as $[\mathsf{r}_1]?_{\mathrm{r}}\{l_i.T_i\}_{i\in\mathrm{I}}$) continues with P_j.

As discussed in [24], the counterpart of branching is weakly reliable and not unreliable. It is implemented by $\mathsf{r} \to_{\mathrm{w}} \mathsf{R}{:}\{l_i.G_i\}_{i\in\mathrm{I},l_{\mathrm{d}}}$, where $\mathsf{R} \subseteq \mathscr{R}$ and l_{d} with $\mathrm{d} \in \mathrm{I}$ is the default branch. We use a broadcast from r to all roles in R to ensure that the sender can influence several participants consistently (see [23, 24] for an explanation). The type system ensures that all processes that are not crashed will move to the same branch. We often abbreviate branching w.r.t. a small set of branches by omitting the set brackets and instead separating the branches by $\oplus$, where the last branch is always the default branch. In contrast to the strongly reliable cases, $s[\mathsf{r},\mathsf{R}]!_{\mathrm{w}}l.P$ (specified as $[\mathsf{R}]!_{\mathrm{w}}\{l_i.T_i\}_{i\in\mathrm{I}}$) allows to broadcast its decision to R and $s[\mathsf{r}_j,\mathsf{r}]?_{\mathrm{w}}\{l_i.P_i\}_{i\in\mathrm{I},l_{\mathrm{d}}}$ (specified as $[\mathsf{r}]?_{\mathrm{w}}\{l_i.T_i\}_{i\in\mathrm{I},l_{\mathrm{d}}}$) defines a default label l_{d}.

We extend the standard operators for recursion $(\mu t)G$, $(\mu X)P$ of [23, 24] by a counter $(\mu X,\mathsf{c}=n)P$ (specified as $(\mu t,\mathsf{c}=n)T$ with the global type $(\mu t,\mathsf{c})G$), where n is a natural number that is increased by unfolding recursion and c can be used as pointer to the current value of the counter within expressions. Note that we do that not only in processes but also in the corresponding types. These expressions allow us to construct unique identifiers for loops within a surrounding recursion.

A loop $s[\mathsf{r},\mathsf{R}]\infty_e^{\mathsf{c}=n}[(x).P_0]P_1;(y).P_2$ (specified as $[\mathsf{R}]\infty_e^{\mathsf{c}=n}[\langle \mathsf{S}_0\rangle.T_0]T_1;\langle \mathsf{S}_2\rangle.T_2$) creates a loop $\mathrm{eval}(e)$ in that role r of session s is currently running the *loop body* P_1 and may interact with the roles in R that are running their local versions of this loop. We identify a loop with its unique identifier $\mathrm{eval}(e)$ that is unique for the whole derivation of the system and the same for all roles $\mathsf{R} \cup \{\mathsf{r}\}$. Again, the loop has a counter $\mathsf{c} = n$ that is increased in unfolding loops and can be used to create the unique identifiers of loops nested within the current loop. Communication within a loop is unreliable. With $\mathtt{call}\langle e,v\rangle$ (specified as $\mathtt{call}\langle e\rangle$) role r invokes another iteration of the loop $\mathrm{eval}(e)$ given by the *loop program* $(x).P_0$, where x is instantiated with v. Role r can terminate its own loop and all loops of the other R by sending $\mathtt{exit}\langle e,v'\rangle$. In this case, or upon receiving $\mathtt{exit}\langle id,v'\rangle$, role r continues with the *loop continuation* $(y).P_2$ of loop $\mathrm{eval}(e) = id$, where y is instantiated by v'. The loop body P_1 contains whatever is left of the current iteration of the loop program P_0. We initialise, as expected by the type system, a loop as $s[\mathsf{r},\mathsf{R}]\infty_e^{\mathsf{c}=0}[(x).P_0]\mathtt{call}\langle e,v\rangle;(y).P_2$ such that its first step calls the first iteration of the loop.

The $\perp$ denotes a process that crashed. Similar to [18], we use message queues to implement asynchrony in sessions. Therefore, session initialisation introduces a directed and initially empty message queue $s_{\mathsf{r}_1\to\mathsf{r}_2}{:}[\,]$ for each pair of roles $\mathsf{r}_1 \neq \mathsf{r}_2$ of the session s. The separate message queues ensure that messages with different sources or destinations are not ordered, but each message queue is FIFO. Since the different forms of interaction might be implemented differently (e.g. by TCP or UDP), it makes sense to further split the message queues into three message queues for each pair $\mathsf{r}_1 \neq \mathsf{r}_2$ such that different kinds of messages do not need to be ordered. To simplify the presentation of examples in this paper and not to blow up the number of message queues, we stick to a single message queue for each pair $\mathsf{r}_1 \neq \mathsf{r}_2$. However, the correctness of our type system does not depend on this decision. We have six kinds of messages $\mathtt{m}$ and corresponding message types $\mathtt{mt}$ in Figure 1—one for each kind of interaction. In strongly reliable communication, a value v (of sort S) is transmitted in a message $\langle v\rangle^{\mathsf{r}}$ of type $\langle \mathsf{S}\rangle^{\mathsf{r}}$. In unreliable communication, the message $l\langle v\rangle^{\mathsf{u}}$ (of type $l\langle \mathsf{S}\rangle^{\mathsf{u}}$) additionally carries a label l. For branching, only the picked label l is transmitted and we add the kind of branching as superscript, i.e., message/type l^{r} is for strongly reliable branching and message/type l^{w} for weakly reliable branching. The message $\mathtt{exit}\langle id,v\rangle$ of type $\mathtt{exit}\langle id,\mathsf{S}\rangle$ signals that the loop id can be terminated. Finally, the message/type $s[\mathsf{r}]$ is for session delegation. A message queue M is a list of messages $\mathtt{m}$ and MT is a list of message types $\mathtt{mt}$.

The remaining operators for independence $G \mid\mid G'$; parallel composition $P \mid P'$; inaction $\mathtt{end}$, $\mathbf{0}$; conditionals $\mathtt{if}\ b\ \mathtt{then}\ P_1\ \mathtt{else}\ P_2$; session delegation $\mathsf{r}_1 \to \mathsf{r}_2{:}\langle s'[\mathsf{r}]{:}T\rangle.G$, $s[\mathsf{r}_1,\mathsf{r}_2]!\langle\langle s'[\mathsf{r}]\rangle\rangle.P$, $s[\mathsf{r}_2,\mathsf{r}_1]?((s'[\mathsf{r}])).P$; and restriction $(\nu x)P$ are all standard.

As usual, we assume that recursion variables are guarded and do not occur free in types or processes and, similarly, that recursive calls $\mathtt{call}\langle e,v\rangle, \mathtt{call}\langle e\rangle$ are guarded within loop programs and do not occur outside of the declaration of loop $\mathrm{eval}(e)$ in types or processes. To ensure that loops are uniquely identified, their identifiers are described as expressions that have to evaluate to a unique identifier in a type and all its unfoldings of recursion, i.e., within standard recursion or surrounding loops these identifiers have to be build by some mechanism (e.g. a counter) that ensures uniqueness. More precisely, all iterations of a loop have the same identifier, whereas a loop within a surrounding recursion or loop needs a fresh identifier for every iteration of the surrounding recursion or loop. Moreover, the type system ensures that neither loop bodies nor loop programs may contain free type variables.

In types $(\mu t,\mathsf{c})G$ and $(\mu t,\mathsf{c}=n)T$ the type variable t and the variable c are *bound* in G, T. In processes $(\mu X,\mathsf{c}=n)P$ the process variable X and the variable c are bound in P. Similarly, all names in round brackets are bound in the remainder of the respective process, e.g. s is bound in P by $\overline{a}[\mathsf{n}](s).P$ and x is bound in P by $s[\mathsf{r}_1, \mathsf{r}_2]?_{\mathsf{r}}(x).P$. A variable or name is *free* if it is not bound. Let $\mathrm{FN}(P)$ return the free names of P.

Let *subterm* denote a (type or process) expression that syntactically occurs within another (type or process) term. We use '.' (as e.g. in $\overline{a}[\mathsf{r}](s).P$) to denote sequential composition. In all operators the *prefix* before '.' guards the *continuation* after the '.'. Moreover, a loop is a guard for its loop continuation, but its loop body is unguarded. Let $\prod_{1 \leq i \leq n} P_i$ abbreviate $P_1 \mid \ldots \mid P_n$.

Let $\mathrm{R}(G)$ return all roles that occur in G. We write $\mathrm{nsr}(G)$, $\mathrm{nsr}(T)$, and $\mathrm{nsr}(P)$, if none of the prefixes in G, T, and P is strongly reliable or for delegation and if P, G, or T do not contain message queues. We write $\mathrm{unr}(A)$ if $\mathrm{nsr}(A)$ and none of the prefixes in A is a weakly reliable branching.

A session channel and a role together uniquely identify a participant of a session, called an *actor*. A process has an actor $s[\mathsf{r}]$ if it has an action prefix or a loop on s that mentions r as its first role. Let $\mathrm{A}(P)$ be the set of actors of P.

As discussed in [23, 24], labels may carry additional runtime information such as timestamps, in order to provide the technical means to implement the failure patterns introduced with the semantics below.

Allowing for runtime information in labels requires a subtle difference in the way labels are used. A timestamp may be added by the sender to capture the transmission time, but for the receiver it is hard to have this information already present in its label before or during reception. Similarly, types in our static type system should not depend on any runtime information. Hence, in contrast to standard MPST, we do not expect the labels of senders and receivers as well as the labels of processes and types to match exactly. Instead we assume a predicate $\doteq$ that compares two labels and is satisfied if the parts of the labels that do not refer to runtime information correspond. If labels do not contain runtime information, $\doteq$ can be instantiated with equality. We require that $\doteq$ is unambiguous on labels used in types, i.e., given two labels of processes l_P, l'_P and two labels of types l_T, l'_T then $l_P \doteq l'_P \wedge l_P \doteq l_T \Rightarrow l'_P \doteq l_T$ and $l_P \doteq l_T \wedge l_T \not\doteq l'_T \Rightarrow l_P \not\doteq l'_T$.

Of course, the presented type system remains valid if we use labels without additional runtime information. Interestingly, also the static information in labels, that have to coincide for senders and receivers and their types, can be exploited to guide communication. In contrast to standard MPST and to support unreliable communication, our MPST variant will ensure that all occurrences of the same label are associated with the same sort. This helps us in the case of failures to ensure the absence of communication mismatches, i.e., the type of a transmitted value has to be the type that the receiver expects. Similarly, labels are used in [5] to avoid communication errors.

Our type system verifies processes, i.e., implementations, against a specification that is a global type. Since processes implement local views, local types are used as a mediator between the global specification and the respective local end points. To ensure that the local types correspond to the global type, they are derived by *projection*.

Projection maps global types onto the respective local type for a given role p. Projection of recursion is standard except for the initialisation of the counter c with 0. Loops are projected as:

$$((\mu t, \mathsf{c})G)\upharpoonright_{\mathsf{p}} \triangleq \begin{cases} G\upharpoonright_{\mathsf{p}} & \text{if } t \text{ does not occur in } G \\ (\mu t, \mathsf{c} = 0)G\upharpoonright_{\mathsf{p}} & \text{else if } \mathsf{p} \in \mathrm{R}(G) \\ \mathtt{end} & \text{otherwise} \end{cases}$$

$$([\mathsf{R}]\infty_e^{\mathsf{c}}\langle \mathrm{S}_0\rangle.G_0; \langle \mathrm{S}_2\rangle.G_2)\upharpoonright_{\mathsf{p}} \triangleq \begin{cases} [\mathsf{R} \setminus \{\mathsf{p}\}]\infty_e^{\mathsf{c}=0}[\langle \mathrm{S}_0\rangle.G_0\upharpoonright_{\mathsf{p}}]\mathtt{call}\langle e\rangle; \langle \mathrm{S}_2\rangle.G_2\upharpoonright_{\mathsf{p}} & \text{if } \mathsf{p} \in \mathsf{R} \\ G_2\upharpoonright_{\mathsf{p}} & \text{otherwise} \end{cases}$$

Recursive types without their recursion variable are mapped to the projection of their recursion body (similar to [9]), else if p occurs in the recursion body we map to a recursive local type, or else to successful termination. If projected on one of its roles $\mathsf{p} \in \mathsf{R}$, the global specification of the loop program G_0 and the global specification of the loop continuation G_2 are projected on p. The counter is instantiated with 0 and the loop body is instantiated with $\mathtt{call}\langle e \rangle$ to call the first loop iteration. Else, the loop is skipped and we project the loop continuation G_2 on p.

Projection of the remaining operators is given in [24] (and Appendix A).

3 A Semantics with Failure Patterns for Global Escape Loops

Before we describe the semantics, we introduce substitution and structural congruence as auxiliary concepts. The application of a substitution $\{{}^{y}\!/\!{}_{x}\}$ on a term A, denoted as $A\{{}^{y}\!/\!{}_{x}\}$, is defined as the result of replacing all free occurrences of x in A by y, possibly applying alpha-conversion to avoid capture or name clashes. For all names $n \in \mathcal{N} \setminus \{x\}$ the substitution behaves as the identity mapping. We use substitution on types as well as processes and naturally extend substitution to the substitution of variables by terms (to unfold recursions) and names by expressions (to instantiate a bound name with a received value).

We use structural congruence to abstract from syntactically different processes with the same meaning, where $\equiv$ is the least congruence that satisfies alpha conversion and the rules:

$$\begin{array}{llll} P \mid \mathbf{0} \equiv P & P_1 \mid P_2 \equiv P_2 \mid P_1 & P_1 \mid (P_2 \mid P_3) \equiv (P_1 \mid P_2) \mid P_3 & (\mu X, \mathsf{c} = n)\mathbf{0} \equiv \mathbf{0} \\ (\nu x)\mathbf{0} \equiv \mathbf{0} & (\nu x)(\nu y)P \equiv (\nu y)(\nu x)P & (\nu x)(P_1 \mid P_2) \equiv P_1 \mid (\nu x)P_2 & \text{if } x \notin \mathrm{FN}(P_1) \end{array}$$

For the reduction semantics in Figure 2 we start with the rules of fault-tolerant processes from [24] that we extend with the rules for our new loops (in blue colour). Similar to [18], session initialisation is synchronous and communication within a session is asynchronous using message queues. The rules are standard except for the six failure pattern (five pattern from [24] and one new pattern $\mathtt{FP}_{\mathtt{drop}}$ for loops) and three rules for system failures: (Crash) for *crash failures*, (ML) for *message loss*, and the new rule (EDrop) that allows to drop exit-messages of loops. *Failure patterns* are predicates that we deliberately choose not to define here (see below). They allow us to provide information about the underlying communication medium and the reliability of processes.

Rule (Init) initialises a session with n roles. Session initialisation introduces a fresh session channel and unguards the participants of the session. Finally, the message queues of this session are initialised with the empty list under the restriction of the session channel.

Rule (RSend) implements an asynchronous strongly reliable message transmission. As a result, the value $\mathrm{eval}(y)$ is wrapped in a message and added to the end of the corresponding message queue and the continuation of the sender is unguarded. Rule (USend) is the counterpart of (RSend) for unreliable senders. (RGet) consumes a message that is marked as strongly reliable with the index r from the head of the respective message queue and replaces in the unguarded continuation of the receiver the bound variable x by the received value y.

There are two rules for the reception of a message in an unreliable communication that are guided by failure patterns. Rule (UGet) is similar to Rule (RGet), but specifies a failure pattern $\mathtt{FP}_{\mathtt{uget}}$ to decide whether this step is allowed. This failure pattern could, e.g., be used to reject messages that are too old. The condition $l \doteq l'$ ensures that the static information in the transmitted label matches the expectation specified in the label of the receiver to avoid communication mismatches. The Rule (USkip) allows to skip the reception of a message in an unreliable communication using a failure pattern $\mathtt{FP}_{\mathtt{uskip}}$ and instead substitutes the bound variable x in the continuation with the default value dv. The failure pattern $\mathtt{FP}_{\mathtt{uskip}}$ tells us whether a reception can be skipped (e.g. via failure detector).

$$\begin{array}{lll}
\text{(Init)} & \overline{a}[\mathsf{n}](s).P_{\mathsf{n}} \mid \prod_{1 \leq \mathsf{i} \leq \mathsf{n}-1} a[\mathsf{i}](s).P_{\mathsf{i}} \longmapsto (\nu s)\left(\prod_{1 \leq \mathsf{i} \leq \mathsf{n}} P_{\mathsf{i}} \mid \prod_{1 \leq \mathsf{i},\mathsf{j} \leq \mathsf{n}, \mathsf{i} \neq \mathsf{j}} s_{\mathsf{i} \to \mathsf{j}}{:}[\,]\right) & \text{if } a \neq s \\
\text{(RSend)} & s[\mathsf{r}_1,\mathsf{r}_2]!_{\mathrm{r}}\langle e \rangle.P \mid s_{\mathsf{r}_1 \to \mathsf{r}_2}{:}\mathrm{M} \longmapsto P \mid s_{\mathsf{r}_1 \to \mathsf{r}_2}{:}\mathrm{M}\#\langle v \rangle^{\mathrm{r}} & \text{if eval}(e) = v \\
\text{(RGet)} & s[\mathsf{r}_1,\mathsf{r}_2]?_{\mathrm{r}}(x).P \mid s_{\mathsf{r}_2 \to \mathsf{r}_1}{:}\langle v \rangle^{\mathrm{r}}\#\mathrm{M} \longmapsto P\{^{v}/_{x}\} \mid s_{\mathsf{r}_2 \to \mathsf{r}_1}{:}\mathrm{M} & \\
\text{(USend)} & s[\mathsf{r}_1,\mathsf{r}_2]!_{\mathrm{u}} l\langle e \rangle.P \mid s_{\mathsf{r}_1 \to \mathsf{r}_2}{:}\mathrm{M} \longmapsto P \mid s_{\mathsf{r}_1 \to \mathsf{r}_2}{:}\mathrm{M}\#l\langle v \rangle^{\mathrm{u}} & \text{if eval}(e) = v \\
\text{(UGet)} & s[\mathsf{r}_1,\mathsf{r}_2]?_{\mathrm{u}} l\langle dv \rangle(x).P \mid s_{\mathsf{r}_2 \to \mathsf{r}_1}{:}l'\langle v \rangle^{\mathrm{u}}\#\mathrm{M} \longmapsto P\{^{v}/_{x}\} \mid s_{\mathsf{r}_2 \to \mathsf{r}_1}{:}\mathrm{M} & \text{if } l \doteq l',\ \mathtt{FP}_{\mathtt{uget}}(s,\mathsf{r}_1,\mathsf{r}_2,l',\ldots) \\
\text{(USkip)} & s[\mathsf{r}_1,\mathsf{r}_2]?_{\mathrm{u}} l\langle dv \rangle(x).P \longmapsto P\{^{dv}/_{x}\} & \text{if } \mathtt{FP}_{\mathtt{uskip}}(s,\mathsf{r}_1,\mathsf{r}_2,l,\ldots) \\
\text{(ML)} & s_{\mathsf{r}_1 \to \mathsf{r}_2}{:}l\langle v \rangle^{\mathrm{u}}\#\mathrm{M} \longmapsto s_{\mathsf{r}_1 \to \mathsf{r}_2}{:}\mathrm{M} & \text{if } \mathtt{FP}_{\mathtt{ml}}(s,\mathsf{r}_1,\mathsf{r}_2,l,\ldots) \\
\text{(RSel)} & s[\mathsf{r}_1,\mathsf{r}_2]!_{\mathrm{r}} l.P \mid s_{\mathsf{r}_1 \to \mathsf{r}_2}{:}\mathrm{M} \longmapsto P \mid s_{\mathsf{r}_1 \to \mathsf{r}_2}{:}\mathrm{M}\#l^{\mathrm{r}} & \\
\text{(RBran)} & s[\mathsf{r}_1,\mathsf{r}_2]?_{\mathrm{r}}\{l_i.P_i\}_{i \in \mathrm{I}} \mid s_{\mathsf{r}_2 \to \mathsf{r}_1}{:}l^{\mathrm{r}}\#\mathrm{M} \longmapsto P_j \mid s_{\mathsf{r}_2 \to \mathsf{r}_1}{:}\mathrm{M} & \text{if } l \doteq l_j,\ j \in \mathrm{I} \\
\text{(WSel)} & s[\mathsf{r},\mathsf{R}]!_{\mathrm{w}} l.P \mid \prod_{\mathsf{r}_i \in \mathsf{R}} s_{\mathsf{r} \to \mathsf{r}_i}{:}\mathrm{M}_i \longmapsto P \mid \prod_{\mathsf{r}_i \in \mathsf{R}} s_{\mathsf{r} \to \mathsf{r}_i}{:}\mathrm{M}_i\#l^{\mathrm{w}} & \\
\text{(WBran)} & s[\mathsf{r}_1,\mathsf{r}_2]?_{\mathrm{w}}\{l_i.P_i\}_{i \in \mathrm{I},l_{\mathrm{d}}} \mid s_{\mathsf{r}_2 \to \mathsf{r}_1}{:}l^{\mathrm{w}}\#\mathrm{M} \longmapsto P_j \mid s_{\mathsf{r}_2 \to \mathsf{r}_1}{:}\mathrm{M} & \text{if } l \doteq l_j,\ j \in \mathrm{I} \\
\text{(WSkip)} & s[\mathsf{r}_1,\mathsf{r}_2]?_{\mathrm{w}}\{l_i.P_i\}_{i \in \mathrm{I},l_{\mathrm{d}}} \longmapsto P_{\mathrm{d}} & \text{if } \mathtt{FP}_{\mathtt{wskip}}(s,\mathsf{r}_1,\mathsf{r}_2,\ldots) \\
\text{(LStep)} & s[\mathsf{r},\mathsf{R}]\infty_{e}^{\mathsf{c}=n}[(x).P_0]P_1;(y).P_2 \mid Q \longmapsto s[\mathsf{r},\mathsf{R}]\infty_{e}^{\mathsf{c}=n}[(x).P_0]P_1';(y).P_2 \mid Q' & \\
& & \text{if } P_1 \mid Q \longmapsto P_1' \mid Q',\ \mathtt{onlyMQ}_{\mathsf{r} \leftrightarrow \mathsf{R}}(Q,Q') \\
\text{(LCall)} & s[\mathsf{r},\mathsf{R}]\infty_{e}^{\mathsf{c}=n}[(x).P_0]\mathtt{call}\langle e_l,e_v \rangle;(y).P_2 \longmapsto & \\
& s[\mathsf{r},\mathsf{R}]\infty_{e}^{\mathsf{c}=\text{eval}(n+1)}[(x).P_0]\left(P_0\{^{n}/_{\mathsf{c}}\}\right)\{^{v}/_{x}\};(y).P_2 & \text{if eval}(e) = \text{eval}(e_l),\ \text{eval}(e_v) = v \\
\text{(LExitS)} & s[\mathsf{r},\mathsf{R}]\infty_{e}^{\mathsf{c}=n}[(x).P_0]\mathtt{exit}\langle e_l,e_v \rangle;(y).P_2 \mid \prod_{\mathsf{r}_i \in \mathsf{R}} s_{\mathsf{r} \to \mathsf{r}_i}{:}\mathrm{M}_i \longmapsto & \\
& P_2\{^{v}/_{y}\} \mid \prod_{\mathsf{r}_i \in \mathsf{R}} s_{\mathsf{r} \to \mathsf{r}_i}{:}\mathrm{M}_i\#\mathtt{exit}\langle id,v \rangle & \text{if eval}(e) = \text{eval}(e_l) = id,\ \text{eval}(e_v) = v \\
\text{(LExitG)} & s[\mathsf{r},\mathsf{R}]\infty_{e}^{\mathsf{c}=n}[(x).P_0]P_1;(y).P_2 \mid s_{\mathsf{r}' \to \mathsf{r}}{:}\mathtt{exit}\langle id,v \rangle\#\mathrm{M} \longmapsto & \\
& P_2\{^{v}/_{y}\} \mid s_{\mathsf{r}' \to \mathsf{r}}{:}\mathrm{M} & \text{if eval}(e) = id,\ \mathsf{r}' \in \mathsf{R} \\
\text{(EDrop)} & s_{\mathsf{r}_2 \to \mathsf{r}_1}{:}\mathtt{exit}\langle id,v \rangle\#\mathrm{M} \longmapsto s_{\mathsf{r}_2 \to \mathsf{r}_1}{:}\mathrm{M} & \text{if } \mathtt{FP}_{\mathtt{drop}}(\mathsf{r},id) \\
\text{(Crash)} & P \longmapsto \bot & \text{if } \mathtt{FP}_{\mathtt{crash}}(P,\ldots) \\
\text{(If-T)} & \mathtt{if}\ e\ \mathtt{then}\ P\ \mathtt{else}\ P' \longmapsto P & \text{if eval}(e) \text{ is true} \\
\text{(If-F)} & \mathtt{if}\ e\ \mathtt{then}\ P\ \mathtt{else}\ P' \longmapsto P' & \text{if eval}(e) \text{ is false} \\
\text{(Deleg)} & s[\mathsf{r}_1,\mathsf{r}_2]!\langle\langle s'[\mathsf{r}] \rangle\rangle.P \mid s_{\mathsf{r}_1 \to \mathsf{r}_2}{:}\mathrm{M} \longmapsto P \mid s_{\mathsf{r}_1 \to \mathsf{r}_2}{:}\mathrm{M}\#s'[\mathsf{r}] & \\
\text{(SRecv)} & s[\mathsf{r}_1,\mathsf{r}_2]?((s'[\mathsf{r}])).P \mid s_{\mathsf{r}_2 \to \mathsf{r}_1}{:}s''[\mathsf{r}']\#\mathrm{M} \longmapsto P\{^{s''}/_{s'}\}\{^{\mathsf{r}'}/_{\mathsf{r}}\} \mid s_{\mathsf{r}_1 \to \mathsf{r}_2}{:}\mathrm{M} & \\
\text{(Par)} & P_1 \mid P_2 \longmapsto P_1' \mid P_2 & \text{if } P_1 \longmapsto P_1' \\
\text{(Res)} & (\nu x)P \longmapsto (\nu x)P' & \text{if } P \longmapsto P' \\
\text{(Rec)} & (\mu X, \mathsf{c} = n)P \longmapsto \left(P\{^{n}/_{\mathsf{c}}\}\right)\{^{(\mu X, \mathsf{c}=\text{eval}(n+1))P}/_{X}\} & \\
\text{(Struc)} & P_1 \longmapsto P_1' & \text{if } P_1 \equiv P_2,\ P_2 \longmapsto P_2',\ P_2' \equiv P_1'
\end{array}$$

Figure 2: Reduction Rules ($\longmapsto$) of Fault-Tolerant Processes with Global Escape Loops.

Rule (RSel) puts the label l selected by r_1 at the end of the message queue towards r_2. Its weakly reliable counterpart (WSel) is similar, but puts the label at the end of all relevant message queues. With (RBran) a label is consumed from the top of a message queue and the receiver moves to the indicated branch. There are again two weakly reliable counterparts of (RBran). Rule (WBran) is similar to (RBran), whereas (WSkip) allows r_1 to skip the message and to move to its default branch if the failure pattern $\mathtt{FP}_{\mathtt{wskip}}$ holds. The requirement $l \doteq l_j$ in RBran and WBran ensures as usual that indeed the branch specified by the message at the queue is picked by the receiver. Note that this branch has to be identified by the statically available information in the respective labels.

With (LStep) the body of a loop may (1) send a message to a message queue, (2) receive a message from a queue, or (3) skip an outer loop-construct of nested loops to perform an output, input, call another loop-iteration, or exit a loop. Therefore, the predicate $\mathtt{onlyMQ}_{\mathsf{r} \leftrightarrow \mathsf{R}}(Q,Q')$ checks that Q and Q' consist only of message queues from r into roles within R or the other way around. Rule (LCall) puts loop

$\mathrm{eval}(e)$ onto another iteration, where the counter is updated and x is instantiated with $\mathrm{eval}(e_v) = v$ in the loop program $(x)P_0$. Here, the side condition $\mathrm{eval}(e) = \mathrm{eval}(e_l)$ ensures that the correct loop is iterated. Role r can terminate its loop $\mathrm{eval}(e) = \mathrm{eval}(e_l) = id$ with (LExitS). This step reduces r to its loop continuation instantiated with $\mathrm{eval}(e_v) = v$ and adds to the message queues from r to all roles in R the message $\mathtt{exit}\langle id, v\rangle$. Upon receiving $\mathtt{exit}\langle id, v\rangle$ in rule (LExitG), role r is induced to also terminate its loop id and continue with its loop continuation instantiated with v.

The Rules (Crash) for *crash failures* and (ML) for *message loss*, describe failures of a system. With Rule (Crash), P can crash if $\mathtt{FP}_{\mathtt{crash}}$, where $\mathtt{FP}_{\mathtt{crash}}$ can e.g. model immortal processes or global bounds on the number of crashes. (ML) allows to drop an unreliable message if the failure pattern $\mathtt{FP}_{\mathtt{ml}}$ is valid. $\mathtt{FP}_{\mathtt{ml}}$ allows, e.g., to implement safe channels that never lose messages or a global bound on the number of lost messages. Rule (EDrop), similarly allows to *drop* a message from a queue, but it does not implement a failure. Instead it allows us to drop $\mathtt{exit}$-messages of already terminated loops, i.e., $\mathtt{FP}_{\mathtt{drop}}$ checks whether the loop mentioned by the $\mathtt{exit}$-message of the considered role is already terminated and only in this case allows to drop the message. Since a loop *id* is run concurrently by several roles of which each role runs its local version of the loop *id*, it cannot be avoided that several roles may actively terminate their loop concurrently, causing several $\mathtt{exit}$-messages for the same loop.

The remaining reduction rules for conditionals, delegation, parallel composition, restriction, recursion, and structural congruence are standard, except for the counter in unfolding recursion.

We deliberately do not specify failure pattern, although we usually assume that the failure patterns $\mathtt{FP}_{\mathtt{uget}}$, $\mathtt{FP}_{\mathtt{uskip}}$, $\mathtt{FP}_{\mathtt{wskip}}$, and $\mathtt{FP}_{\mathtt{drop}}$ use only local information, whereas $\mathtt{FP}_{\mathtt{ml}}$ and $\mathtt{FP}_{\mathtt{crash}}$ may use global information of the system in the current run. We provide these predicates to allow for the implementation of system requirements or abstractions like failure detectors that are typical for distributed algorithms. Directly including them in the semantics has the advantage that all traces satisfy the corresponding requirements, i.e., all traces are valid w.r.t. the assumed system requirements. An example for the instantiation of these patterns is given implicitly via the Conditions 1.1–1.7 in Section 4 and explicitly in Section 5. If we instantiate the patterns $\mathtt{FP}_{\mathtt{uget}}$ with true, the patterns $\mathtt{FP}_{\mathtt{uskip}}$, $\mathtt{FP}_{\mathtt{wskip}}$, $\mathtt{FP}_{\mathtt{crash}}$, $\mathtt{FP}_{\mathtt{ml}}$ with false, and the pattern $\mathtt{FP}_{\mathtt{drop}}$ such that it is true whenever the mentioned loop is terminated by the mentioned role, then we obtain a system without failures. In contrast, the instantiation of $\mathtt{FP}_{\mathtt{drop}}$ as above and the other five patterns with true results in a system, where failures can happen completely non-deterministically at any time.

Note that we keep the failure patterns abstract and do not model how to check them in producing runs. Indeed system requirements such as bounds on the number of processes that can crash usually cannot be checked, but result from observations, i.e., system designers ensure that a violation of this bound is very unlikely and algorithm designers are willing to ignore these unlikely events. In particular, $\mathtt{FP}_{\mathtt{ml}}$ and $\mathtt{FP}_{\mathtt{crash}}$ are thus often implemented as oracles for verification, whereas e.g. $\mathtt{FP}_{\mathtt{uskip}}$ and $\mathtt{FP}_{\mathtt{wskip}}$ are often implemented by system specific time-outs. Note that we are talking about implementing these failure patterns and not formalising them. Failure patterns are abstractions of real world system requirements or software. We implement them by conditions providing the necessary guarantees that we need in general (i.e., for subject reduction and progress) or for the verification of concrete algorithms. In practice, we expect that the systems on which the verified algorithms are running satisfy the respective conditions. Accordingly, the session channels, roles, labels, processes, and loop-identifiers mentioned in Figure 2 are not parameters of the failure patterns, but just a vehicle to more formally specify the conditions on failure patterns in Section 4. An implementation may or may not use these information to implement these patterns but may also use other information such as runtime information about time or the number of processes, as indicated by the ... in failure patterns in Figure 2 such as $\mathtt{FP}_{\mathtt{crash}}(P, \ldots)$.

Similarly, strongly reliable and weakly reliable interactions in potentially faulty systems are abstractions. They are usually implemented by handshakes and redundancy; replicated servers against crash failures and retransmission of late messages against message loss. Algorithm designers have to be aware of the additional costs of these interactions.

Consider an example of nested loops in types and its projection.

$$
\begin{aligned}
G &\triangleq (\mu t, \mathsf{c}_1)[\{1\}]\infty_{\mathsf{c}_1}^{\mathsf{c}_2}\langle\mathbb{N}\rangle.[\{1\}]\infty_{(\mathsf{c}_1,\mathsf{c}_2)}^{\mathsf{c}_3}\langle\mathbb{N}\rangle.\mathtt{end};\langle\mathbb{N}\rangle.\mathtt{call}\langle\mathsf{c}_1\rangle;\langle\mathbb{N}\rangle.t\\
G\!\upharpoonright_1 &= (\mu t, \mathsf{c}_1 = 0)[\emptyset]\infty_{\mathsf{c}_1}^{\mathsf{c}_2=0}\Big[\langle\mathbb{N}\rangle.[\emptyset]\infty_{(\mathsf{c}_1,\mathsf{c}_2)}^{\mathsf{c}_3=0}[\langle\mathbb{N}\rangle.\mathtt{end}]\mathtt{call}\langle(\mathsf{c}_1,\mathsf{c}_2)\rangle;\langle\mathbb{N}\rangle.\mathtt{call}\langle\mathsf{c}_1\rangle\Big]\mathtt{call}\langle\mathsf{c}_1\rangle;\langle\mathbb{N}\rangle.t
\end{aligned}
$$

To ensure that the loops are uniquely identified in all unfoldings of the surrounding recursion and the outer loop, their identifiers c_1 and $(\mathsf{c}_1, \mathsf{c}_2)$ are build from counters. This system can be implemented as:

$$
\begin{aligned}
P &\triangleq (\mu X, \mathsf{c}_1 = 0)P_{\mathsf{c}_1}\\
P_{\mathsf{c}_1} &\triangleq s[1,\emptyset]\infty_{\mathsf{c}_1}^{\mathsf{c}_2=0}[(x).P_{\mathsf{c}_1,\mathsf{c}_2}(x)]\mathtt{call}\langle\mathsf{c}_1, 0\rangle;(y).X\\
P_{\mathsf{c}_1,\mathsf{c}_2}(x) &\triangleq s[1,\emptyset]\infty_{(\mathsf{c}_1,\mathsf{c}_2)}^{\mathsf{c}_3=0}\big[(x').\mathtt{exit}\big\langle(\mathsf{c}_1,\mathsf{c}_2), x'+1\big\rangle\big]\mathtt{call}\langle(\mathsf{c}_1,\mathsf{c}_2), x+1\rangle;(y').P_{\mathsf{c}_1,\mathtt{cont}}(y')\\
P_{\mathsf{c}_1,\mathtt{cont}}(y') &\triangleq \mathtt{if}\ y' < 5\ \mathtt{then}\ \mathtt{call}\big\langle\mathsf{c}_1, y'+1\big\rangle\ \mathtt{else}\ \mathtt{exit}\big\langle\mathsf{c}_1, y'+1\big\rangle
\end{aligned}
$$

$$
\begin{aligned}
P &\longmapsto P_0\{{}^{(\mu X,\mathsf{c}_1=1)P_{\mathsf{c}_1}}/_X\}\\
&\longmapsto s[1,\emptyset]\infty_0^{\mathsf{c}_2=1}[(x).P_{0,\mathsf{c}_2}(x)]P_{0,0}(0);(y).(\mu X, \mathsf{c}_1 = 1)P_{\mathsf{c}_1}\\
&\longmapsto^* s[1,\emptyset]\infty_0^{\mathsf{c}_2=1}[(x).P_{0,\mathsf{c}_2}(x)]\mathtt{call}\langle 0, 2+1\rangle;(y).(\mu X, \mathsf{c}_1 = 1)P_{\mathsf{c}_1}\\
&\longmapsto s[1,\emptyset]\infty_0^{\mathsf{c}_2=2}[(x).P_{0,\mathsf{c}_2}(x)]P_{0,1}(3);(y).(\mu X, \mathsf{c}_1 = 1)P_{\mathsf{c}_1}\\
&\longmapsto^* s[1,\emptyset]\infty_0^{\mathsf{c}_2=2}[(x).P_{0,\mathsf{c}_2}(x)]\mathtt{exit}\langle 0, 5+1\rangle;(y).(\mu X, \mathsf{c}_1 = 1)P_{\mathsf{c}_1}\\
&\longmapsto (\mu X, \mathsf{c}_1 = 1)P_{\mathsf{c}_1}\{{}^{6}/_{y}\} \longmapsto P_1\{{}^{(\mu X,\mathsf{c}_1=2)P_{\mathsf{c}_1}}/_X\}
\end{aligned}
$$

4 Typing Fault-Tolerant Processes

The type of processes is checked using typing rules that define the derivation of type judgments. Within type judgements, the type information are stored in type environments.

Definition 1 (Type Environments)**.** The *global*, *loop* and *session environments* are given by

$$
\begin{aligned}
\Gamma &::= \emptyset \mid \Gamma\cdot x{:}\mathrm{S} \mid \Gamma\cdot a{:}G \mid \Gamma\cdot l{:}\mathrm{S}\\
\Theta &::= \emptyset \mid \Theta\cdot X{:}s[\mathsf{r}]t \mid e{:}s[\mathsf{r}]\langle\mathrm{S}_0, \mathrm{S}_2\rangle\\
\Delta &::= \emptyset \mid \Delta\cdot s[\mathsf{r}]{:}T \mid \Delta\cdot s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{MT}^*
\end{aligned}
$$

Global environments with assignments $x{:}\mathrm{S}$ of values to sorts, $a{:}G$ of shared channels a to global types (for session initialisation), and $l{:}\mathrm{S}$ of labels to sorts as well as session environments with assignments $s[\mathsf{r}]{:}T$ of actors to local types and $s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{MT}^*$ of message queues to a list of message types are inherited from [24]. We move assignments $X{:}s[\mathsf{r}]t$ of process variables to actors and type variables (to check standard recursion) to the new loop environments that also contains assignments $e{:}s[\mathsf{r}]\langle\mathrm{S}_0, \mathrm{S}_2\rangle$ of loop identifiers to actors and sorts (for the values used to call and exit a loop). Loop environments are used to list active recursion and loops inside their respective bodies.

We write $x\sharp\Gamma$, $x\sharp\Theta$, and $x\sharp\Delta$ if x does not occur in Γ, Θ, and Δ, respectively. We use $\cdot$ to add an assignment provided that the new assignment is not in conflict with the type environment. More precisely,

$$\text{(Req)}\ \frac{a{:}G \in \Gamma \quad |\mathrm{R}(G)| = \mathsf{n} \quad \Gamma, \Theta \vdash P \triangleright \Delta \cdot s[\mathsf{n}]{:}G{\upharpoonright}_{\mathsf{n}}}{\Gamma, \Theta \vdash \overline{a}[\mathsf{n}](s).P \triangleright \Delta} \qquad \text{(If)}\ \frac{\Gamma \Vdash e{:}\mathbb{B} \quad \Gamma, \Theta \vdash P \triangleright \Delta \quad \Gamma, \Theta \vdash P' \triangleright \Delta}{\Gamma, \Theta \vdash \mathtt{if}\ e\ \mathtt{then}\ P\ \mathtt{else}\ P' \triangleright \Delta}$$

$$\text{(Acc)}\ \frac{a{:}G \in \Gamma \quad 0 < \mathsf{r} < |\mathrm{R}(G)| \quad \Gamma, \Theta \vdash P \triangleright \Delta \cdot s[\mathsf{r}]{:}G{\upharpoonright}_{\mathsf{r}}}{\Gamma, \Theta \vdash a[\mathsf{r}](s).P \triangleright \Delta} \qquad \text{(End)}\ \frac{\mathsf{noLoop}(\Theta)}{\Gamma, \Theta \vdash \mathbf{0} \triangleright \emptyset}$$

$$\text{(RSend)}\ \frac{\Gamma \Vdash y{:}\mathrm{S} \quad \Gamma, \Theta \vdash P \triangleright \Delta \cdot s[\mathsf{r}_1]{:}T}{\Gamma, \Theta \vdash s[\mathsf{r}_1, \mathsf{r}_2]!_{\mathsf{r}}\langle y \rangle.P \triangleright \Delta \cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]!_{\mathsf{r}}\langle \mathrm{S} \rangle.T} \qquad \text{(Rec)}\ \frac{\Gamma \Vdash n{:}\mathbb{N} \quad \Gamma \cdot \mathsf{c}{:}\mathbb{N}, \Theta \cdot X{:}s[\mathsf{r}]t \vdash P \triangleright s[\mathsf{r}]{:}T}{\Gamma, \Theta \vdash (\mu X, \mathsf{c} = n)P \triangleright s[\mathsf{r}]{:}(\mu t, \mathsf{c} = n)T}$$

$$\text{(RGet)}\ \frac{x \sharp (\Gamma, \Delta, s) \quad \Gamma \cdot x{:}\mathrm{S}, \Theta \vdash P \triangleright \Delta \cdot s[\mathsf{r}_1]{:}T}{\Gamma, \Theta \vdash s[\mathsf{r}_1, \mathsf{r}_2]?_{\mathsf{r}}(x).P \triangleright \Delta \cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]?_{\mathsf{r}}\langle \mathrm{S} \rangle.T} \qquad \text{(Var)}\ \frac{}{\Gamma, \Theta \cdot X{:}s[\mathsf{r}]t \vdash X \triangleright s[\mathsf{r}]{:}t}$$

$$\text{(USend)}\ \frac{\Gamma \Vdash y{:}\mathrm{S} \quad l \doteq l' \quad l'{:}\mathrm{S} \in \Gamma \quad \Gamma, \Theta \vdash P \triangleright \Delta \cdot s[\mathsf{r}_1]{:}T}{\Gamma, \Theta \vdash s[\mathsf{r}_1, \mathsf{r}_2]!_{\mathsf{u}} l\langle y \rangle.P \triangleright \Delta \cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]!_{\mathsf{u}} l'\langle \mathrm{S} \rangle.T} \qquad \text{(Par)}\ \frac{\Gamma, \Theta \vdash P \triangleright \Delta \quad \Gamma, \Theta \vdash P' \triangleright \Delta'}{\Gamma, \Theta \vdash P \mid P' \triangleright \Delta \cdot \Delta'}$$

$$\text{(UGet)}\ \frac{x \sharp (\Gamma, \Delta, s) \quad \Gamma \Vdash v{:}\mathrm{S} \quad l \doteq l' \quad l'{:}\mathrm{S} \in \Gamma \quad \Gamma \cdot x{:}\mathrm{S}, \Theta \vdash P \triangleright \Delta \cdot s[\mathsf{r}_1]{:}T}{\Gamma, \Theta \vdash s[\mathsf{r}_1, \mathsf{r}_2]?_{\mathsf{u}} l\langle v \rangle(x).P \triangleright \Delta \cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]?_{\mathsf{u}} l'\langle \mathrm{S} \rangle.T} \qquad \text{(Crash)}\ \frac{\mathrm{nsr}(\Delta)}{\Gamma, \Theta \vdash \bot \triangleright \Delta}$$

$$\text{(RSel)}\ \frac{j \in \mathrm{I} \quad l \doteq l_j \quad \Gamma, \Theta \vdash P \triangleright \Delta \cdot s[\mathsf{r}_1]{:}T_j}{\Gamma, \Theta \vdash s[\mathsf{r}_1, \mathsf{r}_2]!_{\mathsf{r}} l.P \triangleright \Delta \cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]!_{\mathsf{r}}\{l_i.T_i\}_{i \in \mathrm{I}}} \qquad \text{(WSel)}\ \frac{j \in \mathrm{I} \quad l \doteq l_j \quad \Gamma, \Theta \vdash P \triangleright \Delta \cdot s[\mathsf{r}]{:}T_j}{\Gamma, \Theta \vdash s[\mathsf{r}, \mathrm{R}]!_{\mathsf{w}} l.P \triangleright \Delta \cdot s[\mathsf{r}]{:}[\mathrm{R}]!_{\mathsf{w}}\{l_i.T_i\}_{i \in \mathrm{I}}}$$

$$\text{(RBran)}\ \frac{\forall j \in \mathrm{I}_2.\ \exists i \in \mathrm{I}_1.\ l_i \doteq l'_j \wedge \Gamma, \Theta \vdash P_i \triangleright \Delta \cdot s[\mathsf{r}_1]{:}T_j}{\Gamma, \Theta \vdash s[\mathsf{r}_1, \mathsf{r}_2]?_{\mathsf{r}}\{l_i.P_i\}_{i \in \mathrm{I}_1} \triangleright \Delta \cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]?_{\mathsf{r}}\{l'_i.T_i\}_{i \in \mathrm{I}_2}} \qquad \text{(Res1)}\ \frac{x \sharp (\Gamma, \Delta) \quad \Gamma \cdot x{:}\mathrm{S}, \Theta \vdash P \triangleright \Delta}{\Gamma, \Theta \vdash (\nu x)P \triangleright \Delta}$$

$$\text{(WBran)}\ \frac{l_{\mathsf{d}} \doteq l'_{\mathsf{d}} \quad \forall j \in \mathrm{I}_2.\ \exists i \in \mathrm{I}_1.\ l_i \doteq l'_j \wedge \Gamma, \Theta \vdash P_i \triangleright \Delta \cdot s[\mathsf{r}_1]{:}T_j}{\Gamma, \Theta \vdash s[\mathsf{r}_1, \mathsf{r}_2]?_{\mathsf{w}}\{l_i.P_i\}_{i \in \mathrm{I}_1, l_{\mathsf{d}}} \triangleright \Delta \cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]?_{\mathsf{w}}\{l'_i.T_i\}_{i \in \mathrm{I}_2, l'_{\mathsf{d}}}}$$

$$\text{(Deleg)}\ \frac{\Gamma, \Theta \vdash P \triangleright \Delta \cdot s[\mathsf{r}_1]{:}T}{\begin{array}{c}\Gamma, \Theta \vdash s[\mathsf{r}_1, \mathsf{r}_2]!\langle\langle s'[\mathsf{r}] \rangle\rangle.P \triangleright \\ \Delta \cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]!\langle s'[\mathsf{r}]{:}T' \rangle.T \cdot s'[\mathsf{r}]{:}T'\end{array}} \qquad \text{(SRecv)}\ \frac{\Gamma, \Theta \vdash P \triangleright \Delta \cdot s[\mathsf{r}_1]{:}T \cdot s'[\mathsf{r}]{:}T'}{\begin{array}{c}\Gamma, \Theta \vdash s[\mathsf{r}_1, \mathsf{r}_2]?((s'[\mathsf{r}])).P \triangleright \\ \Delta \cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]?\langle s'[\mathsf{r}]{:}T' \rangle.T\end{array}}$$

$$\text{(Loop)}\ \frac{\begin{array}{c} x, y \sharp (\Gamma, \Delta, s) \quad \mathrm{unr}(T_0) \quad \mathrm{unr}(T_1) \quad \Gamma \cdot x{:}\mathrm{S}_0 \cdot \mathsf{c}{:}\mathbb{N}, e{:}s[\mathsf{r}]\langle \mathrm{S}_0, \mathrm{S}_2 \rangle \vdash P_0 \triangleright s[\mathsf{r}]{:}T_0 \\ \Gamma \Vdash n{:}\mathbb{N} \quad \Gamma \cdot \mathsf{c}{:}\mathbb{N}, e{:}s[\mathsf{r}]\langle \mathrm{S}_0, \mathrm{S}_2 \rangle \vdash P_1 \triangleright s[\mathsf{r}]{:}T_1 \quad \Gamma \cdot y{:}\mathrm{S}_2, \Theta \vdash P_2 \triangleright \Delta \cdot s[\mathsf{r}]{:}T_2 \end{array}}{\Gamma, \Theta \vdash s[\mathsf{r}, \mathrm{R}]\infty_e^{\mathsf{c}=n}[(x).P_0]P_1; (y).P_2 \triangleright \Delta \cdot s[\mathsf{r}]{:}[\mathrm{R}]\infty_e^{\mathsf{c}=n}[\langle \mathrm{S}_0 \rangle.T_0]T_1; \langle \mathrm{S}_2 \rangle.T_2}$$

$$\text{(Call)}\ \frac{\Gamma \Vdash e_v{:}\mathrm{S}_0}{\Gamma, \Theta \cdot e{:}s[\mathsf{r}]\langle \mathrm{S}_0, \mathrm{S}_2 \rangle \vdash \mathtt{call}\langle e, e_v \rangle \triangleright s[\mathsf{r}]{:}\mathtt{call}\langle e \rangle} \qquad \text{(Exit)}\ \frac{\Gamma \Vdash e_v{:}\mathrm{S}_2}{\Gamma, \Theta \cdot e{:}s[\mathsf{r}]\langle \mathrm{S}_0, \mathrm{S}_2 \rangle \vdash \mathtt{exit}\langle e, e_v \rangle \triangleright s[\mathsf{r}]{:}T_1}$$

Figure 3: Typing Rules for Fault-Tolerant Systems with Global Escape Loops.

$\Gamma \cdot x{:}\mathrm{S}$ implies $x \sharp \Gamma$, $\Gamma \cdot l{:}\mathrm{S}$ implies $l \sharp \Gamma$, $\Theta \cdot X{:}s[\mathsf{r}]t$ implies $X, t \sharp \Theta$, $\Theta \cdot e{:}s[\mathsf{r}]\langle \mathrm{S}_0, \mathrm{S}_2 \rangle$ implies $e \sharp \Theta$, $\Delta \cdot s[\mathsf{r}]{:}T$ implies $(\nexists T'.\ s[\mathsf{r}]{:}T' \in \Delta)$, and $\Delta \cdot s_{\mathsf{r}_1 \to \mathsf{r}_2}{:}\mathscr{M}$ implies $(\nexists \mathscr{M}'.\ s_{\mathsf{r}_1 \to \mathsf{r}_2}{:}\mathscr{M}' \in \Delta)$. We naturally extend this operator towards sets, i.e., $\Gamma \cdot \Gamma'$ implies $(\forall A \in \Gamma'.\ \Gamma \cdot A)$, $\Theta \cdot \Theta'$ implies $(\forall A \in \Theta'.\ \Theta \cdot A)$, and $\Delta \cdot \Delta'$ implies $(\forall A \in \Delta'.\ \Delta \cdot A)$. The conditions described for the operator $\cdot$ for global and session environments are referred to as *linearity*. Accordingly, we denote type environments that satisfy these properties as *linear* and restrict in the following our attention to linear environments. We abstract in session environments from assignments towards terminated local types, i.e., $\Delta \cdot s[\mathsf{r}]{:}\mathtt{end} = \Delta$.

A *type judgement* is of the form $\Gamma, \Theta \vdash P \triangleright \Delta$, where Γ is a global environment, Θ is a loop environment, $P \in \mathscr{P}$ is a process, and Δ is a session environment. A process P is *well-typed* w.r.t. Γ and Δ if $\Gamma \vdash P \triangleright \Delta$ can be derived from the rules in the Figures 3 and 4. We write $\mathrm{nsr}(\Delta)$ (or $\mathrm{unr}(\Delta)$) if for all types T in Δ we have $\mathrm{nsr}(T)$ (or $\mathrm{unr}(T)$) and if Δ does not contain message queues. With $\Gamma \Vdash y{:}\mathrm{S}$ we check that y is an expression of the sort S if all names x in y are replaced by arbitrary values of sort S_x for $x{:}\mathrm{S}_x \in \Gamma$.

$$(\mathsf{Res2})\ \frac{\begin{array}{c}\{s[\mathsf{r}]{:}G\upharpoonright_{\mathsf{r}} \mid \mathsf{r}\in \mathrm{R}(G)\}\cdot\{s_{\mathsf{r}\to\mathsf{r}'}{:}[\,] \mid \mathsf{r},\mathsf{r}'\in \mathrm{R}(G')\wedge \mathsf{r}\neq\mathsf{r}'\} \overset{s}{\mapsto\!\!\!\Rightarrow} \Delta' \\ s\sharp(\Gamma,\Delta) \quad a{:}G\in\Gamma,\Theta \quad \Gamma\vdash P\triangleright\Delta\cdot\Delta'\end{array}}{\Gamma\vdash(\nu s)P\triangleright\Delta}$$

$$(\mathsf{MQComR})\ \frac{\Gamma\Vdash v{:}\mathrm{S}\quad \Gamma,\Theta\vdash s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{M}\triangleright s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{MT}}{\Gamma,\Theta\vdash s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\langle v\rangle^{\mathrm{r}}\#\mathrm{M}\triangleright s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\langle \mathrm{S}\rangle^{\mathrm{r}}\#\mathrm{MT}}$$

$$(\mathsf{MQComU})\ \frac{\Gamma\Vdash v{:}\mathrm{S}\quad l\doteq l'\quad l'{:}\mathrm{S}\in\Gamma\quad \Gamma,\Theta\vdash s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{M}\triangleright s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{MT}}{\Gamma,\Theta\vdash s_{\mathsf{r}_1\to\mathsf{r}_2}{:}l\langle v\rangle^{\mathrm{u}}\#\mathrm{M}\triangleright s_{\mathsf{r}_1\to\mathsf{r}_2}{:}l'\langle \mathrm{S}\rangle^{\mathrm{u}}\#\mathrm{MT}}$$

$$(\mathsf{MQBranR})\ \frac{l\doteq l'\quad \Gamma,\Theta\vdash s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{M}\triangleright s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{MT}}{\Gamma,\Theta\vdash s_{\mathsf{r}_1\to\mathsf{r}_2}{:}l^{\mathrm{r}}\#\mathrm{M}\triangleright s_{\mathsf{r}_1\to\mathsf{r}_2}{:}l'^{\mathrm{r}}\#\mathrm{MT}} \qquad (\mathsf{MQBranW})\ \frac{l\doteq l'\quad \Gamma,\Theta\vdash s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{M}\triangleright s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{MT}}{\Gamma,\Theta\vdash s_{\mathsf{r}_1\to\mathsf{r}_2}{:}l^{\mathrm{w}}\#\mathrm{M}\triangleright s_{\mathsf{r}_1\to\mathsf{r}_2}{:}l'^{\mathrm{w}}\#\mathrm{MT}}$$

$$(\mathsf{MQDeleg})\ \frac{\Gamma,\Theta\vdash s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{M}\triangleright s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{MT}}{\Gamma,\Theta\vdash s_{\mathsf{r}_1\to\mathsf{r}_2}{:}s'[\mathsf{r}]\#\mathrm{M}\triangleright s_{\mathsf{r}_1\to\mathsf{r}_2}{:}s'[\mathsf{r}]\#\mathrm{MT}} \qquad (\mathsf{MQNil})\ \frac{}{\Gamma,\Theta\vdash s_{\mathsf{r}_1\to\mathsf{r}_2}{:}[\,]\triangleright s_{\mathsf{r}_1\to\mathsf{r}_2}{:}[\,]}$$

$$(\mathsf{MQExit})\ \frac{\mathrm{eval}(e)=id\quad \Gamma\Vdash v{:}\mathrm{S}\quad \Gamma,\Theta\vdash s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{M}\triangleright s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{MT}}{\Gamma,\Theta\vdash s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathtt{exit}\langle id,v\rangle\#\mathrm{M}\triangleright s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathtt{exit}\langle e,\mathrm{S}\rangle\#\mathrm{MT}}$$

Figure 4: Runtime Typing Rules for Fault-Tolerant Systems.

For the rules in Figure 3 we adapted the rules of [24] and extended them by rules for loops. We added the loop environment to all rules that is only relevant for typing recursion and loops. In (End) we add the condition $\mathsf{noLoop}(\Theta)$ that checks that Θ does not contain loop identifiers, to ensure that no branch of a loop program or loop body terminates with $\mathbf{0}$.

(Loop) requires the types of a loop program T_0 and a loop body T_1 to be unreliable ($\mathrm{unr}(T_0)$ and $\mathrm{unr}(T_1)$). It checks the loop program P_0 and the loop body P_1 against their types, but reduces in this check the loop environment to the information for the current loop. This ensures that P_0 and P_1 do not contain free process variables and no calls or exists of surrounding loops. We do not forbid complete recursions or nested loops inside a loop program/body, where the type system ensures their completion before the end of the loop program/body. As in recursion via (Rec), we also reduce the session environment to the actor that initiates this loop. Finally, (Loop) checks the loop continuation P_2 against its type T_2, where Θ and Δ are not reduced. Note that to apply this rule, the expression e used to create the identifier of the loop in the process and the type have to match exactly, i.e., are not evaluated.

(Call) is similar to (Var) and checks that the considered recursion or loop is considered active by the loop environment. Additionally it verifies the sort of the transmitted value. Also (Exit) checks the sort of the transmitted value, requires that the current session environment contains only the actor that invoked the considered loop, and that this loop is considered active by the loop environment. Since (Exit) does not implement any requirement on the type T_1, it does intuitively allow to ignore whatever is left of the loop body.

Figure 4 presents the runtime typing rules, i.e., the typing rules for processes that may result from steps of a system that implements a global type. Since it covers only operators that are not part of initial systems, a type checking tool might ignore them. We need these rules however for the proofs of progress and subject reduction. Under the assumption that initial systems cannot contain crashed processes, Rule (Crash) may be moved to the set of runtime typing rules.

Rule (Res2) types sessions that are already initialised and that may have performed already some of the steps described by their global type. The relation $\overset{s}{\mapsto}$ is given in Figure 5 in the Appendix and describes how a session environment evolves alongside reductions of the system, i.e., it emulates the reduction steps of processes. As an example consider the rule $\Delta \cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]!_{\mathsf{r}}\langle \mathrm{S}\rangle.T \cdot s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{MT} \overset{s}{\mapsto} \Delta \cdot s[\mathsf{r}_1]{:}T \cdot s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{MT}\#\langle \mathrm{S}\rangle^{\mathsf{r}}$ that emulates (RSend). Let $\overset{s}{\Mapsto}$ denote the reflexive and transitive closure of $\overset{s}{\mapsto}$.

(Res2) and the remaining rules of Figure 4 except for (MQExit) are from [24] extended by the loop environment Θ. (MQExit) checks exit-messages on a message queue.

We have to prove that our extended type system satisfies the standard properties of MPST, i.e., subject reduction and progress. Because of the failure pattern in the reduction semantics in Figure 2, subject reduction and progress do not hold in general. Instead we have to fix conditions on failure patterns that ensure these properties. Subject reduction needs one condition on crashed processes and progress requires that no part of the system is blocked. In fact, different instantiations of these failure patterns may allow for progress. As in [23, 24], we leave it for future work to determine what kind of conditions on failure patterns or requirements on their interactions are necessary. Here, we extend the conditions given in [23, 24] by a condition for $\mathtt{FP}_{\mathtt{drop}}$.

Condition 1 (Failure Pattern)**.**

1. *If* $\mathtt{FP}_{\mathtt{crash}}(P,\ldots)$ *then* $\mathrm{nsr}(P)$.
2. *The failure pattern* $\mathtt{FP}_{\mathtt{uget}}(s,\mathsf{r}_1,\mathsf{r}_2,l,\ldots)$ *is always valid.*
3. *The pattern* $\mathtt{FP}_{\mathtt{ml}}(s,\mathsf{r}_1,\mathsf{r}_2,l,\ldots)$ *is valid iff* $\mathtt{FP}_{\mathtt{uskip}}(s,\mathsf{r}_2,\mathsf{r}_1,l,\ldots)$ *is valid.*
4. *If* $\mathtt{FP}_{\mathtt{crash}}(P,\ldots)$ *and* $s[\mathsf{r}] \in \mathrm{A}(P)$ *is an actor then eventually the pattern* $\mathtt{FP}_{\mathtt{uskip}}(s,\mathsf{r}_2,\mathsf{r},l,\ldots)$ *and* $\mathtt{FP}_{\mathtt{wskip}}(s,\mathsf{r}_2,\mathsf{r},l,\ldots)$ *hold for all* r_2, l.
5. *If* $\mathtt{FP}_{\mathtt{crash}}(P,\ldots)$ *and* $s[\mathsf{r}] \in \mathrm{A}(P)$ *then eventually* $\mathtt{FP}_{\mathtt{ml}}(s,\mathsf{r}_1,\mathsf{r},l,\ldots)$ *for all* r_1, l *and* $\mathtt{FP}_{\mathtt{drop}}(\mathsf{r},id)$.
6. *If* $\mathtt{FP}_{\mathtt{wskip}}(s,\mathsf{r}_1,\mathsf{r}_2,\ldots)$ *then* $s[\mathsf{r}_2]$ *is crashed, i.e., the system does no longer contain an actor* $s[\mathsf{r}_2]$ *and the message queue* $s_{\mathsf{r}_2\to\mathsf{r}_1}$ *is empty.*
7. *If* r *terminated the loop* id *then eventually* $\mathtt{FP}_{\mathtt{drop}}(\mathsf{r},id)$ *and if* $\mathtt{FP}_{\mathtt{drop}}(\mathsf{r},id)$ *then* r *terminated* id*.*

The crash of a process should not block strongly reliable actions, i.e., only processes with $\mathrm{nsr}(P)$ can crash (Condition 1.1). Condition 1.2 requires that no process can refuse to consume a message on its queue to prevent deadlocks that may arise from refusing a message that is never dropped. Condition 1.3 requires that if a message can be dropped from a message queue then the corresponding receiver has to be able to skip this message and vice versa. Similarly, processes that wait for messages from a crashed process have to be able to skip (Condition 1.4) and all messages of a queue towards a crashed receiver can be dropped (Condition 1.5). A weakly reliable branching request should not be lost. To ensure that the receiver of such a branching request can proceed if the sender is crashed but is not allowed to skip the reception of the branching request before the sender crashed, we require that $\mathtt{FP}_{\mathtt{wskip}}(s,\mathsf{r}_1,\mathsf{r}_2,\ldots)$ is false as long as $s[\mathsf{r}_2]$ is alive or messages on the respective queue are still in transit (Condition 1.6). Condition 1.7 ensures that exit-messages can be dropped after the corresponding loop was terminated but not before.

It is important to remember that these conditions are minimal assumptions on the system requirements and that system requirements are abstractions. Parts of them may be realised by actual software-code (which then allows to check them), whereas other parts of the system requirements may not be realised at all but rather observed (which then does not allow to verify them). Because of that, it is an established method to verify the correctness of algorithms w.r.t. given system requirements (e.g. in [10, 20, 26]), even if these system requirements are not verified and often do not hold in all (but only nearly all) cases.

Subject reduction tells us that derivatives of well-typed systems are again well-typed. This ensures that our formalism can be used to analyse processes by static type checking. For subject reduction we consider only types that were generated from a set of global types, one for each session, using coherence. *Coherence* intuitively describes that a session environment captures all local endpoints of a collection of global types. Since we capture all relevant global types in the global environment, we define coherence on pairs of global and session environments.

Definition 2 (Coherence)**.** The type environments Γ, Δ are *coherent* if, for all session channels s in Δ, there exists a global type G in Γ such that the restriction of Δ on assignments with s is the set Δ' such that:

$$\{s[\mathsf{r}]{:}G{\restriction}_{\mathsf{r}} \mid \mathsf{r} \in \mathrm{R}(G)\} \cdot \{s_{\mathsf{r}\to\mathsf{r}'}{:}[\,] \mid \mathsf{r}, \mathsf{r}' \in \mathrm{R}(G)\} \overset{s}{\Mapsto} \Delta'$$

We use $\overset{s}{\Mapsto}$ in the above definition to define coherence for systems that already performed some steps.

Theorem 2 (Subject Reduction)**.** *If* $\Gamma, \Theta \vdash P \triangleright \Delta$, Γ, Δ *are coherent, and* $P \longmapsto P'$*, then there is some* Δ' *such that* $\Gamma, \Theta \vdash P' \triangleright \Delta'$.

The proof is by induction on the derivation of $P \longmapsto P'$. In every case, we use the information about the structure of the processes to generate partial proof trees for the respective typing judgement. Additionally, we use Condition 1.1 to ensure that the type environment of a crashed process cannot contain the types of reliable communication prefixes.

Progress states that no part of a well-typed and coherent system can block other parts, that eventually all matching communication partners are unguarded, that interactions specified by the global type can happen, and that there are no communication mismatches. Subject reduction and progress together then imply *session fidelity*, i.e., that processes behave as specified in their global types.

To ensure that the interleaving of sessions and session delegation cannot introduce deadlocks, we assume an interaction type system as introduced in [3, 18]. For this type system it does not matter whether the considered actions are strongly reliable, weakly reliable, or unreliable. More precisely, we can adapt the interaction type system of [3] in a straightforward way to the above session calculus, where unreliable communication and weakly reliable branching is treated in exactly the same way as strongly reliable communication/branching and loops are treated in the same way as standard recursion. We say that P *is free of cyclic dependencies between sessions* if this interaction type system does not detect any cyclic dependencies. In this sense fault-tolerance is more flexible than explicit failure handling, which often requires a more substantial revision of the interaction type system to cover the additional dependencies that are introduced e.g. by the propagation of faults.

Theorem 3 (Progress/Session Fidelity)**.** *Let* $\Gamma, \Theta \vdash P \triangleright \Delta$, Γ, Δ *be coherent, and let P be free of cyclic dependencies between sessions. Assume that in the derivation of* $\Gamma, \Theta \vdash P \triangleright \Delta$*, whenever* $\overline{a}[\mathsf{n}](s).Q$ *or* $a[\mathsf{r}](s).Q$ *in P, then* $a{:}G \in \Gamma$, $|\mathrm{R}(G)| = \mathsf{n}$*, and there are* $\overline{a}[\mathsf{n}](s).Q_n$ *as well as* $a[\mathsf{r}_i](s).Q_i$ *in P for all* $1 \leq \mathsf{r}_i < \mathsf{n}$.

1. *Then either P does not contain any action prefixes or* $P \longmapsto P'$.
2. *If P does not contain recursion or loops, then there exists P' such that* $P \longmapsto^{*} P'$ *and P' does not contain any action prefixes.*

The proof of progress relies on the Conditions 1.2–1.7 to ensure that failures cannot block the system: in the failure-free case unreliable messages are eventually received (1.2), the receiver of a lost message can skip (1.3), no receiver is blocked by a crashed sender (1.4), messages towards receivers that crashed or skipped can be dropped (1.5 + 1.3), branching requests cannot be ignored (1.6), and exit-messages can be dropped eventually if and only if the corresponding loop was already terminated.

5 The Rotating Coordinator Algorithm

To illustrate the benefits of our global escape loops, we present an implementation of the rotating coordinator algorithm [10, 15], which is superior to the version without loops presented in [23, 24].

The rotating coordinator algorithm is a small but not trivial consensus algorithm. It was designed for systems with crash failures, but the majority of the algorithm can be implemented with unreliable communication. The goal is that every agent i eventually decides on a proposed belief value, where no two agents decide on different values. It is a round based algorithm, where each round consists of four phases. In each round, one process acts as a coordinator decided by round robin, denoted by c.

In Phase 1 every agent i sends its current belief to the coordinator c.

In Phase 2 the coordinator waits until it has received at least half of the messages of the current round and then sends the best belief to all other agents.

In Phase 3 the agents either receive the message of the coordinator or suspect the coordinator to have crashed and reply with ack or nack accordingly. Suspicion can yield false positives.

In Phase 4 the coordinator waits, as in Phase 2, until it has received at least half of the messages of the current round. Then, if at least half of the messages were ack, it sends a weakly reliable global escape containing the decision.

It is possible for agents to skip rounds by suspecting the coordinator of the current round and by proceeding to the next round. There are also no synchronisation fences thus it is possible for the agents to be in different rounds and have messages of different rounds in the system. Having agents in different rounds makes proving correctness much more difficult.

We use the labels p_i and (p_i, r), where $i \in \{1,2,3\}$ specifies the number of the current phase and r is a natural number that specifies the current round. We use p_i as static information and r as runtime information in the labels. Therefore, $p_i \doteq (p_i, r) \doteq (p_i, r')$ holds for all i, r, and r'. The additional runtime information can be used in the failure patterns, e.g. to drop outdated messages. We assume the sorts $\mathrm{S}_{\text{belief}} = \{0,1\}$ and $\mathrm{S}_{\text{ack}} = \{\mathtt{t}, \mathtt{f}\}$. Let n be the number of agents.

We start with the specification of the algorithm as a global type. Let $\left(\bigodot_{1 \leq i \leq n} \pi_i\right).G$ abbreviate $\pi_1.\ldots.\pi_n.G$ to simplify the presentation, where G is a global type and $\pi_1, \ldots, \pi_n$ are sequences of prefixes. More precisely, each π_i is of the form $\pi_{i,1}.\ldots.\pi_{i,m}$ and each $\pi_{i,j}$ is a type prefix of the form $\mathsf{r}_1 \to_{\mathrm{u}} \mathsf{r}_2{:}l\langle \mathrm{S}\rangle$ or $\mathsf{r} \to_{\mathrm{w}} \mathsf{R}{:}l_1.T_1 \oplus \ldots \oplus l_n.T_n \oplus l_{\mathrm{d}}$, where the latter case represents a weakly reliable branching prefix (as used in [24]) with the branches $l_1, \ldots, l_n, l_{\mathrm{d}}$, the default branch l_{d}, and where the next global type provides the missing specification for the default case.

$$G_{rc}(\mathsf{n}) \triangleq [\{1,\ldots,\mathsf{n}\}]\infty_1^{\mathsf{k}}\langle \mathrm{S}_{\text{belief}}\rangle.\left(\bigodot_{1\leq \mathsf{c}\leq \mathsf{n}} \left(\bigodot_{1\leq \mathsf{i}\leq \mathsf{n}, \mathsf{i}\neq \mathsf{c}} \mathsf{i} \to_{\mathrm{u}} \mathsf{c}{:}p_1\langle \mathrm{S}_{\text{belief}}\rangle\right).\left(\bigodot_{1\leq \mathsf{i}\leq \mathsf{n}, \mathsf{i}\neq \mathsf{c}} \mathsf{c} \to_{\mathrm{u}} \mathsf{i}{:}p_2\langle \mathrm{S}_{\text{belief}}\rangle\right). \left(\bigodot_{1\leq \mathsf{i}\leq \mathsf{n}, \mathsf{i}\neq \mathsf{c}} \mathsf{i} \to_{\mathrm{u}} \mathsf{c}{:}p_3\langle \mathrm{S}_{\text{ack}}\rangle\right)\right).\mathtt{call}\langle 1\rangle;\langle \mathrm{S}_{\text{belief}}\rangle.\mathtt{end}$$

$G_{rc}(\mathsf{n})$ specifies a loop containing a collection of n rounds, where each process functions as a coordinator once. This collection of n rounds is specified with the first $\bigodot$. By unfolding the loop and increasing the counter k, $G_{rc}(\mathsf{n})$ starts the next n rounds. The following three $\bigodot$ specify the Phases 1–3 of the algorithm within one round. Since Phase 4 only involves exiting the loop, it is not directly visible in the global type.

In Phase 1 all processes except the coordinator c transmit a belief to c using label p_1. In Phase 2 c transmits a belief to all other processes using label p_2. Then all processes transmit a value of type S_{ack} to the coordinator using label p_3 in Phase 3. Finally, in Phase 4 the coordinator can terminate the protocol by sending a global escape message containing the decision. All interactions in the specification are unreliable.

In the following, we implement the algorithm as a process. Let $\left(\bigodot_{1\leq i\leq n}\pi_i\right).P$ abbreviate the sequence $\pi_1.\ldots.\pi_n.P$, where P is a process and $\pi_1,\ldots,\pi_n$ are sequences of prefixes.

$$\begin{aligned}
Sys\left(\mathsf{n},\vec{V}\right) &\triangleq \overline{a}[\mathsf{n}](s).P(\mathsf{n},\mathsf{n},v_{\mathsf{n}}) \mid \prod_{1\leq \mathsf{i}<\mathsf{n}} a[\mathsf{i}](s).P(\mathsf{i},\mathsf{n},v_{\mathsf{i}})\\
P(\mathsf{i},\mathsf{n},v_{\mathsf{i}}) &\triangleq s[\mathsf{i},\{1,\ldots,\mathsf{n}\}\setminus\{\mathsf{i}\}]\infty_1^{\mathsf{k}=0}[(v_{\mathsf{i}}).P_0(\mathsf{i},\mathsf{n},k,v_{\mathsf{i}})]\mathtt{call}\langle 1,v_{\mathsf{i}}\rangle;(v).\mathbf{0}\\
P_0(\mathsf{i},\mathsf{n},k,v_{\mathsf{i}}) &\triangleq \left(\bigodot_{1\leq \mathsf{c}\leq \mathsf{n}} \mathtt{if}\ \mathsf{i}=\mathsf{c}\ \mathtt{then}\ P_1^{\mathrm{C}}(\mathsf{i},\mathsf{n},k\cdot\mathsf{n}+\mathsf{c},v_{\mathsf{i}})\ \mathtt{else}\ P_1^{\mathrm{NC}}(\mathsf{i},\mathsf{n},\mathsf{c},k\cdot\mathsf{n}+\mathsf{c},v_{\mathsf{i}})\right).\mathtt{call}\langle 1,v_{\mathsf{i}}\rangle
\end{aligned}$$

$Sys\left(\mathsf{n},\vec{V}\right)$ describes the session initialisation of a system with n participants and the (initial) knowledge $\vec{V}=\{v_{\mathsf{i}}\mid 1\leq\mathsf{i}\leq\mathsf{n}\}$, where v_{i} is the initial belief of role i. Let $\left|\vec{V}\right| \triangleq |\{\mathsf{i}\mid v_{\mathsf{i}}\neq\perp\}|$ return the number of non-empty entries. $P(\mathsf{i},\mathsf{n},v_{\mathsf{i}})$ describes a process i in a set of n processes. Each process is described as a loop, where the loop program runs n rounds before calling another loop-iteration. Once a decision is reached and the loop ends, the loop continuation is instantiated with the decision value.

$$\begin{aligned}
P_1^{\mathrm{C}}(\mathsf{c},\mathsf{n},r,v_{\mathsf{c}}) &\triangleq \left(\bigodot_{1\leq\mathsf{i}\leq\mathsf{n},\mathsf{i}\neq\mathsf{c}} s[\mathsf{c},\mathsf{i}]?_{\mathsf{u}}(p_1,r)\langle\perp\rangle(v_{\mathsf{i}})\right).\\
&\qquad \mathtt{if}\ \left|\vec{V}\right|\geq\left\lceil\frac{\mathsf{n}-1}{2}\right\rceil\ \mathtt{then}\ P_2^{\mathrm{C}}\left(\mathsf{c},\mathsf{n},r,\mathrm{best}(\vec{V}),\mathrm{best}(\vec{V})\right)\ \mathtt{else}\ P_2^{\mathrm{C}}(\mathsf{c},\mathsf{n},r,v_{\mathsf{c}},\perp)\\
P_1^{\mathrm{NC}}(\mathsf{i},\mathsf{n},\mathsf{c},r,v_{\mathsf{i}}) &\triangleq s[\mathsf{i},\mathsf{c}]!_{\mathsf{u}}(p_1,r)\langle v_{\mathsf{i}}\rangle.P_2^{\mathrm{NC}}(\mathsf{i},\mathsf{n},\mathsf{c},r,v_{\mathsf{i}})
\end{aligned}$$

Every non-coordinator $P_1^{\mathrm{NC}}(\mathsf{i},\mathsf{n},\mathsf{c},r,v_{\mathsf{i}})$ sends its own belief via unreliable communication to the coordinator and proceeds to Phase 2. The coordinator receives (some of) these messages and writes each one into its knowledge vector before proceeding to Phase 2. If the reception of at least half of the messages was successful, it is updating its belief using the function best() that returns the best belief value. Otherwise, it continues to use its own belief. We are using $\left\lceil\frac{\mathsf{n}-1}{2}\right\rceil$ to check for a majority, since in our implementation processes do not transmit to themselves.

$$\begin{aligned}
P_2^{\mathrm{C}}(\mathsf{c},\mathsf{n},r,v_{\mathsf{c}},x) &\triangleq \left(\bigodot_{1\leq\mathsf{i}\leq\mathsf{n},\mathsf{i}\neq\mathsf{c}} s[\mathsf{c},\mathsf{i}]!_{\mathsf{u}}(p_2,r)\langle x\rangle\right).P_3^{\mathrm{C}}(\mathsf{c},\mathsf{n},r,v_{\mathsf{c}})\\
P_2^{\mathrm{NC}}(\mathsf{i},\mathsf{n},\mathsf{c},r,v_{\mathsf{i}}) &\triangleq s[\mathsf{i},\mathsf{c}]?_{\mathsf{u}}(p_2,r)\langle\perp\rangle(x).\\
&\qquad \mathtt{if}\ x=\perp\ \mathtt{then}\ P_3^{\mathrm{NC}}(\mathsf{i},\mathsf{n},\mathsf{c},r,v_{\mathsf{i}},\mathtt{f})\ \mathtt{else}\ P_3^{\mathrm{NC}}(\mathsf{i},\mathsf{n},\mathsf{c},r,x,\mathtt{t})
\end{aligned}$$

In Phase 2, the coordinator sends its updated belief to all other processes via unreliable communication and proceeds. Note that x is either $\perp$ or the best belief identified in Phase 1. If a non-coordinator process successfully receives a belief other than $\perp$, it updates its own belief with the received value and proceeds to Phase 3, where we use the Boolean value `t` for the acknowledgement. If the coordinator is suspected to have crashed or $\perp$ was received, the process proceeds to Phase 3 with the Boolean value `f`, signalling nack.

$$\begin{aligned}
P_3^{\mathrm{C}}(\mathsf{c},\mathsf{n},r,v_{\mathsf{c}}) &\triangleq \left(\bigodot_{1\leq\mathsf{i}\leq\mathsf{n},\mathsf{i}\neq\mathsf{c}} s[\mathsf{c},\mathsf{i}]?_{\mathsf{u}}(p_3,r)\langle\perp\rangle(v_{\mathsf{i}})\right).P_4^{\mathrm{C}}\left(\mathsf{c},\mathsf{n},\vec{V}\right)\\
P_3^{\mathrm{NC}}(\mathsf{i},\mathsf{n},\mathsf{c},r,v_{\mathsf{i}},b) &\triangleq s[\mathsf{i},\mathsf{c}]!_{\mathsf{u}}(p_3,r)\langle b\rangle
\end{aligned}$$

In Phase 3, every non-coordinator sends either ack or nack to the coordinator. If the coordinator successfully receives the message, it writes the Boolean value at the index of the sender into its knowledge vector. In case of failure, $\perp$ is used as default. After that the processes continue with Phase 4. The missing continuation for non-coordinators is implemented by the next round.

$$P_4^{\mathrm{C}}\left(\mathsf{c},\mathsf{n},\vec{V}\right) \triangleq \texttt{if ack}(\vec{V}) \geq \left\lceil \frac{\mathsf{n}-1}{2} \right\rceil \texttt{then exit}\langle l, v_{\mathsf{c}}\rangle \texttt{ else}$$

In Phase 4, all non-coordinators move on to the next round. The coordinator checks if at least half of the non-coordinator roles signalled acknowledgement, utilising the function ack() to count. If it received enough acknowledgments, it sends a global escape message containing the decision value, which causes all participants to eventually terminate. Otherwise, the coordinator continues with the next round. The missing continuation after `else` is implemented by the next round.

The main difference between this implementation and the previous version without loops of [23, 24] lies in Phase 4. In the previous version, the coordinator transmitted the decision via broadcasting one of the labels *Zero*, *One*, or l_{d}. The first two labels represented a decision and terminated the protocol, whereas the default label l_{d} specified the need for another round:

$$\begin{aligned}
P_4^{\mathrm{C}}\left(\mathsf{c},\mathsf{n},\vec{V}\right) &\triangleq \texttt{if ack}(\vec{V}) \geq \left\lceil \frac{\mathsf{n}-1}{2} \right\rceil \texttt{then (if } v_{\mathsf{c}} = 0 \texttt{ then } s[\mathsf{c},\mathscr{I}]!_{\mathrm{w}}\mathit{Zero}.\mathbf{0} \\
&\qquad \texttt{else } s[\mathsf{c},\mathscr{I}]!_{\mathrm{w}}\mathit{One}.\mathbf{0}) \texttt{ else } s[\mathsf{c},\mathscr{I}]!_{\mathrm{w}} l_{\mathrm{d}} \\
P_4^{\mathrm{NC}}\left(\mathsf{i},\mathsf{n},\mathsf{c},\vec{V}\right) &\triangleq s[\mathsf{i},\mathsf{n}]?_{\mathrm{w}}\mathit{Zero}.\mathbf{0} \oplus \mathit{One}.\mathbf{0} \oplus l_{\mathrm{d}}
\end{aligned}$$

where $\mathscr{I} = \{1,\ldots,\mathsf{n}\} \setminus \{\mathsf{c}\}$. This caused all non-coordinators to wait for the coordinator's decision before proceeding to the next round.

In our new implementation, presented above, non-coordinators can proceed to the next round immediately after Phase 3. They can also skip entire rounds by suspecting the coordinator. Thus, processes can diverge as freely in their rounds as in the original rotating coordinator algorithm [10]. Exiting the loop mimics the so-called *reliable broadcast* of the original algorithm.

Chandra and Toueg [10] introduce the failure detector $\Diamond\mathscr{S}$ that is called *eventually strong*, meaning that (1) eventually every process that crashes is permanently suspected by every correct process and (2) there is a time after which some correct process is never suspected by any other process. We observe that the suspicion of senders is only possible in Phase 3, where processes may suspect the coordinator of the round. Accordingly, the failure pattern $\mathtt{FP}_{\mathtt{uskip}}$ implements this failure detector to allow processes to suspect unreliable coordinators in Phase 2, i.e., with label p_2. In Phase 1 and Phase 3 $\mathtt{FP}_{\mathtt{uskip}}$ may allow to suspect processes that are not crashed after the coordinator received enough messages. In all other cases, this pattern eventually returns true iff the respective sender is crashed. Moreover, $\mathtt{FP}_{\mathtt{uskip}}$ is true for outdated messages, i.e., messages with a round number smaller than the current round of the process.

$\mathtt{FP}_{\mathtt{uget}}$ returns true. To prevent the system from becoming blocked, $\mathtt{FP}_{\mathtt{ml}}$ and $\mathtt{FP}_{\mathtt{drop}}$ eventually return true for messages that cannot be consumed, i.e., for messages with label p_2 that were suspected using $\Diamond\mathscr{S}$, skipped p_1/p_3-messages, messages from old rounds, and messages after the termination of the loop. Otherwise, $\mathtt{FP}_{\mathtt{ml}}$ and $\mathtt{FP}_{\mathtt{drop}}$ returns false. By the system requirements in [10], no messages get lost, but it is realistic to assume that receivers can drop messages of skipped receptions on their incoming message queues. As there are at least half of the processes required to be correct for this algorithm, we implement $\mathtt{FP}_{\mathtt{crash}}$ by false if only half of the processes are alive and true otherwise. These failure patterns satisfy the Conditions 1.1–1.7.

The proof of termination, agreement, and validity of the algorithm is discussed in [24]. The main difference is, that there might be several exit-messages, but the requirement on the majority in Phase 4 ensures that all such messages carry the same decision value.

6 Conclusions

We present an unreliable loop construct with weakly reliable global escape for fault-tolerant multiparty session types (FTMPST) for systems that may suffer from message loss or crash failures. We prove subject reduction and progress and present a small but relevant case study.

Currently we require all actions within loop programs/bodies to be unreliable. This ensures that a communication partner is not blocked if a loop is terminated. An interesting question for further work is how to relax this requirement. For instance, we may allow for a variant of weakly reliable branching within loop programs/bodies, where moving to the default branch is not only allowed if the sender is suspected to be crashed but also if the receiver suspects that the sender already terminated its loop or at least already moved to another loop iteration.

Moreover, there are a couple of open problems from [23, 24]. A really difficult challenge is to extend branching to at least some kind of message loss, while maintaining the strong properties of the type system and ensuring that no two alive processes move to different branches.

We also want to study whether and in how far we can introduce weakly reliable or unreliable session delegation. Similarly, we want to study unreliable variants of session initialisation including process crashes and lost messages during session initialisation. Unreliable variants of session initialisation open a new perspective on MPST-frameworks such as [11] with dynamically changing network topologies and sessions for that the number of roles is determined at run-time.

In [24] one set of conditions on failure patterns was fixed to prove subject reduction and progress. We can also think of other sets of conditions. The failure pattern $\mathtt{FP}_{\mathtt{uget}}$ can be used to reject the reception of outdated messages. Therefore, we drop Condition 1.2 and instead require for each message m whose reception is refused that $\mathtt{FP}_{\mathtt{ml}}$ ensures that m is eventually dropped from the respective queue and that $\mathtt{FP}_{\mathtt{uskip}}$ allows to skip the reception of these messages. An interesting question is to find minimal requirements and minimal sets of conditions that allow to prove correctness in general.

It would be nice to also fully automate the remaining proofs for the distributed algorithm in Section 5, namely for validity, agreement, and termination. The approach in [25] sequentialises well-typed systems and gives the much simpler remaining verification problem to a model checker. Interestingly, the main challenges to adopt this approach are not the unreliable or weakly reliable prefixes but the failure patterns.

References

[1] Manuel Adameit, Kirstin Peters & Uwe Nestmann (2017): *Session Types for Link Failures*. In: *Proc. of FORTE*, *LNCS* 10321, pp. 1–16, doi:10.1007/978-3-319-60225-7_1.

[2] Marcos Kawazoe Aguilera, Wei Chen & Sam Toueg (1997): *Heartbeat: A Timeout-Free Failure Detector for Quiescent Reliable Communication*. In: *Proc. of WDAG*, *LNCS* 1320, Springer, pp. 126–140, doi:10.1007/BFb0030680.

[3] Lorenzo Bettini, Mario Coppo, Loris D'Antoni, Marco De Luca, Mariangiola Dezani-Ciancaglini, & Nobuko Yoshida (2008): *Global Progress in Dynamically Interleaved Multiparty Sessions*. In: *Proc. of CONCUR*, *LNCS* 5201, Springer, pp. 418–433, doi:10.1007/978-3-540-85361-9_33.

[4] Laura Bocchi, Kohei Honda, Emilio Tuosto & Nobuko Yoshida (2010): *A Theory of Design-by-Contract for Distributed Multiparty Interactions*. In: *Proc. of CONCUR*, *LNCS* 6269, Springer, pp. 162–176, doi:10.1007/978-3-642-15375-4_12.

[5] Luís Caires & Hugo Torres Vieira (2010): *Conversation types*. *Theoretical Computer Science* 411(51–52), pp. 4399–4440, doi:10.1016/j.tcs.2010.09.010.

[6] Sara Capecchi, Elena Giachino & Nobuko Yoshida (2016): *Global escape in multiparty sessions*. Mathematical Structures in Computer Science 26(2), pp. 156–205, doi:10.1017/S0960129514000164.

[7] Marco Carbone, Kohei Honda & Nobuko Yoshida (2008): *Structured Interactional Exceptions in Session Types*. In: Proc. of CONCUR, LNCS 5201, Springer, pp. 402–417, doi:10.1007/978-3-540-85361-9_32.

[8] Ilaria Castellani, Mariangiola Dezani-Ciancaglini & Paola Giannini (2017): *Concurrent Reversible Sessions*. In: Proc. of CONCUR, LIPIcs 85, pp. 30:1–30:17, doi:10.4230/LIPIcs.CONCUR.2017.30.

[9] Ilaria Castellani, Mariangiola Dezani-Ciancaglini, Paola Giannini & Ross Horne (2020): *Global types with internal delegation*. Theoretical Computer Science 807, pp. 128–153, doi:10.1016/j.tcs.2019.09.027.

[10] Tushar Deepak Chandra & Sam Toueg (1996): *Unreliable Failure Detectors for Reliable Distributed Systems*. Journal of the ACM 43(2), pp. 225–267, doi:10.1145/226643.226647.

[11] Minas Charalambides, Peter Dinges & Gul Agha (2016): *Parameterized, concurrent session types for asynchronous multi-actor interactions*. Science of Computer Programming 115–116, pp. 100–126, doi:10.1016/j.scico.2015.10.006.

[12] Tzu-Chun Chen, Malte Viering, Andi Bejleri, Lukasz Ziarek & Patrick Eugster (2016): *A Type Theory for Robust Failure Handling in Distributed Systems*. In: Proc. of FORTE, LNCS 9688, Springer, pp. 96–113, doi:10.1007/978-3-319-39570-8_7.

[13] Mario Coppo, Mariangiola Dezani-Ciancaglini, Luca Padovani & Nobuko Yoshida (2015): *A Gentle Introduction to Multiparty Asynchronous Session Types*. In: Proc. of SFM, LNCS 9104, pp. 146–178, doi:10.1007/978-3-319-18941-3_4.

[14] Romain Demangeon, Kohei Honda, Raymond Hu, Rumyana Neykova & Nobuko Yoshida (2015): *Practical interruptible conversations: distributed dynamic verification with multiparty session types and Python*. Formal Methods in System Design 46(3), pp. 197–225, doi:10.1007/s10703-014-0218-8.

[15] Rachele Fuzzati, Massimo Merro & Uwe Nestmann (2007): *Distributed Consensus, revisited*. Acta Informatica, pp. 377–425, doi:10.1007/s00236-007-0052-1.

[16] Felix C. Gärtner (1999): *Fundamentals of Fault-Tolerant Distributed Computing in Asynchronous Environments*. ACM Computing Surveys 31(1), pp. 1–26, doi:10.1145/311531.311532.

[17] Kohei Honda, Nobuko Yoshida & Marco Carbone (2008): *Multiparty Asynchronous Session Types*. In: Proc. of POPL, 43, ACM, pp. 273–284, doi:10.1145/1328438.1328472.

[18] Kohei Honda, Nobuko Yoshida & Marco Carbone (2016): *Multiparty Asynchronous Session Types*. Journal of the ACM 63(1), doi:10.1145/2827695.

[19] Dimitrios Kouzapas, Ramūnas Gutkovas & Simon J. Gay (2014): *Session Types for Broadcasting*. In: Proc. of PLACES, EPTCS 155, pp. 25–31, doi:10.4204/EPTCS.155.4.

[20] Leslie Lamport (2001): *Paxos Made Simple*. ACM Sigact News 32(4), pp. 18–25.

[21] Nancy A. Lynch (1996): *Distributed Algorithms*. Morgan Kaufmann.

[22] Rumyana Neykova & Nobuko Yoshida (2017): *Let it recover: multiparty protocol-induced recovery*. In: Proc. of CC, ACM, pp. 98–108, doi:10.1145/3033019.3033031.

[23] Kirstin Peters, Uwe Nestmann & Christoph Wagner (2022): *Fault-Tolerant Multiparty Session Types*. In: Proc. of FORTE, LNCS 13273, Springer, pp. 93–113, doi:10.1007/978-3-031-08679-3_7.

[24] Kirstin Peters, Uwe Nestmann & Christoph Wagner (2023): *FTMPST: Fault-Tolerant Multiparty Session Types*. Logical Methods in Computer Science 19(4), doi:10.46298/LMCS-19(4:14)2023.

[25] Kirstin Peters, Christoph Wagner & Uwe Nestmann (2019): *Taming Concurrency for Verification Using Multiparty Session Types*. In: Proc. of ICTAC, LNCS 11884, pp. 196–215, doi:10.1007/978-3-030-32505-3_-12.

[26] Maarten van Steen & Andrew S. Tanenbaum (2017): *Distributed Systems*. Maarten van Steen.

[27] Gerard Tel (1994): *Introduction to Distributed Algorithms*. Cambridge University Press.

[28] Malte Viering, Tzu-Chun Chen, Patrick Eugster, Raymond Hu & Lukasz Ziarek (2018): *A Typing Discipline for Statically Verified Crash Failure Handling in Distributed Systems*. In: *Proc. of ESOP*, *LNCS* 10801, Springer, pp. 799–826, doi:10.1007/978-3-319-89884-1_28.

[29] Nobuko Yoshida, Pierre-Malo Deniélou, Andi Bejleri & Raymond Hu (2010): *Parameterised Multiparty Session Types*. In: *Proc. of FoSSaCS*, *LNCS* 6014, pp. 128–145, doi:10.1007/978-3-642-12032-9_10.

This Appendix contains additional material and the missing proofs of the above paper. In case of acceptance, we will publish the Appendix as technical report on arXiv.

A Fault-Tolerant Types and Processes

Definition 3 (Well-Formedness, Global Type)**.** A global type G is *well-formed* if
(1) it neither contains free nor unguarded type variables,
(2) all loops $\mathrm{eval}(e)$ in G and all iterations of recursion or loops in G are pairwise distinct,
(3) $\mathrm{R}(G) = \{1, \ldots, |\mathrm{R}(G)|\}$,
(4) for all its subterms of the form $\mathsf{r}_1 \to_{\mathrm{r}} \mathsf{r}_2{:}\langle \mathrm{S} \rangle.G'$ or $\mathsf{r}_1 \to_{\mathrm{u}} \mathsf{r}_2{:}l\langle \mathrm{S} \rangle.G'$, we have $\mathsf{r}_1 \neq \mathsf{r}_2$,
(5) for all its subterms of the form $\mathsf{r}_1 \to_{\mathrm{r}} \mathsf{r}_2{:}\{l_i.G_i\}_{i\in\mathrm{I}}$ or $\mathsf{r} \to_{\mathrm{w}} \mathsf{R}{:}\{l_i.G_i\}_{i\in\mathrm{I},l_{\mathrm{d}}}$, we have $\mathsf{r}_1 \neq \mathsf{r}_2$, $\mathsf{r} \notin \mathsf{R}$, $\mathrm{d} \in \mathrm{I}$, and the labels l_i are pairwise distinct, and
(6) for all its subterms of the form $G_1 \mid\mid G_2$, we have $\mathrm{R}(G_1) \cap \mathrm{R}(G_2) = \emptyset$.

We restrict our attention to well-formed global types.

Definition 4 (Well-Formedness, Local Type)**.** A local type T is *well-formed* if
(1) it neither contains free nor unguarded type variables,
(2) all loops $\mathrm{eval}(e)$ in T and all iterations of recursion or loops in T are pairwise distinct, and
(3) for all its subterms of the form $[\mathsf{r}]!_{\mathrm{r}}\{l_i.T_i\}_{i\in\mathrm{I}}$, $[\mathsf{r}]?_{\mathrm{r}}\{l_i.T_i\}_{i\in\mathrm{I}}$, $[\mathsf{R}]!_{\mathrm{w}}\{l_i.T_i\}_{i\in\mathrm{I},l_{\mathrm{d}}}$, or $[\mathsf{R}]?_{\mathrm{w}}\{l_i.T_i\}_{i\in\mathrm{I},l_{\mathrm{d}}}$, we have $\mathrm{d} \in \mathrm{I}$ and the labels l_i are pairwise distinct.

We restrict our attention to well-formed local types.

The projections of the global types for communication in [23, 24] are obtained straightforwardly from the projection of their respective strongly reliable counterparts:

$$(\mathsf{r}_1 \to_{\diamond} \mathsf{r}_2{:}\mathfrak{S}.G)\upharpoonright_{\mathsf{p}} \triangleq \begin{cases} [\mathsf{r}_2]!_{\diamond}\mathfrak{S}.G\upharpoonright_{\mathsf{p}} & \text{if } \mathsf{p} = \mathsf{r}_1 \\ [\mathsf{r}_1]?_{\diamond}\mathfrak{S}.G\upharpoonright_{\mathsf{p}} & \text{if } \mathsf{p} = \mathsf{r}_2 \\ G\upharpoonright_{\mathsf{p}} & \text{otherwise} \end{cases}$$

where either $\diamond = \mathrm{r}$, $\mathfrak{S} = \langle \mathrm{S} \rangle$ or $\diamond = \mathrm{u}$, $\mathfrak{S} = l\langle \mathrm{S} \rangle$ and

$$\left(\mathsf{r}_1 \to_{\diamond} \mathfrak{R}{:}\{l_i.G_i\}_{i\in\mathrm{I}\mathfrak{D}}\right)\upharpoonright_{\mathsf{p}} \triangleq \begin{cases} [\mathfrak{R}]!_{\diamond}\{l_i.G_i\upharpoonright_{\mathsf{p}}\}_{i\in\mathrm{I}} & \text{if } \mathsf{p} = \mathsf{r}_1 \\ [\mathsf{r}_1]?_{\diamond}\{l_i.G_i\upharpoonright_{\mathsf{p}}\}_{i\in\mathrm{I}\mathfrak{D}} & \text{if } \mathfrak{B} \\ \bigsqcup_{i\in\mathrm{I}}(G_i\upharpoonright_{\mathsf{p}}) & \text{otherwise} \end{cases}$$

where either $\diamond = \mathrm{r}$, $\mathfrak{R} = \mathsf{r}_2$, $\mathfrak{B}$ is $\mathsf{p} = \mathsf{r}_2$, $\mathfrak{D}$ is empty or $\diamond = \mathrm{w}$, $\mathfrak{R} = \mathsf{R}$, $\mathfrak{B}$ is $\mathsf{p} \in \mathsf{R}$, $\mathfrak{D}$ is $,l_{\mathrm{d}}$. In the last case of strongly reliable or weakly reliable branching—when projecting onto a role that does not participate in this branching—we map to $\bigsqcup_{i\in\{1,\ldots,n\}}(G_i\upharpoonright_{\mathsf{p}}) = (G_1\upharpoonright_{\mathsf{p}}) \sqcup \ldots \sqcup (G_n\upharpoonright_{\mathsf{p}})$. The $\sqcup$ allows to unify the projections $G_i\upharpoonright_{\mathsf{p}}$ if all of them return the same kind of branching input $[\mathsf{p}]?_{\diamond}\ldots$ were the respective sets of branches my differ as long as the same label is always followed by the same local type. The operation $\sqcup$ is (similar to [29]) inductively defined as:

$$\begin{aligned} T \sqcup T &= T \\ ([\mathsf{r}]?_{\mathrm{r}}\mathrm{I}_1) \sqcup ([\mathsf{r}]?_{\mathrm{r}}\mathrm{I}_2) &= [\mathsf{r}]?_{\mathrm{r}}(\mathrm{I}_1 \sqcup \mathrm{I}_2) \\ ([\mathsf{r}]?_{\mathrm{w}}\mathrm{I}_1) \sqcup ([\mathsf{r}]?_{\mathrm{w}}\mathrm{I}_2) &= [\mathsf{r}]?_{\mathrm{w}}(\mathrm{I}_1 \sqcup \mathrm{I}_2) \quad \text{if } \mathrm{I}_1 \text{ and } \mathrm{I}_2 \text{ have the same default branch} \\ \mathrm{I} \sqcup \emptyset &= \mathrm{I} \\ \mathrm{I} \sqcup (\{l.T\} \cup \mathrm{J}) &= \begin{cases} \{l.(T' \sqcup T)\} \cup ((\mathrm{I} \setminus \{l.T'\}) \sqcup \mathrm{J}) & \text{if } l.T' \in \mathrm{I} \\ \{l.T\} \cup (\mathrm{I} \sqcup \mathrm{J}) & \text{if } l \notin \mathrm{I} \end{cases} \end{aligned}$$

where $T, T' \in \mathscr{T}$ are local types, $\mathrm{I}, \mathrm{I}_1, \mathrm{I}_2, \mathrm{J}$ are sets of branches of local types of the form $l.T$, $l \notin \mathrm{I}$ is short hand for $\nexists T'.\ l.T' \in \mathrm{I}$, and is undefined in all other cases. By the first line, identical types can be merged. By the second and third line, local types for the reception of a branching request can be merged if they have the same prefix and the respective sets of branches can be merged. The third line, for the weakly reliable case, additionally requires that the two sets of branches have the same default branch. The sets of branches, that need to be merged according to the second and third line, contain elements of the form $l.T$, where l is a label and T a local type. The last two lines above inductively define how to merge such sets, i.e., here we overload the operator $\sqcup$ on local types to an operator on sets of branches of local types. The case distinction in the last line ensures that elements $l.T$ with a label that occurs in only one of the two sets can be kept, but if both sets contain an element with the same label then the respective local types have to be merged for the resulting set.

The mergeability relation $\sqcup$ states that two types are identical up to their branching types, where only branches with distinct labels are allowed to be different. This ensures that if the sender r_1 in $\mathsf{r}_1 \to_{\mathsf{r}} \mathsf{r}_2{:}\{l_i.G_i\}_{i \in \mathrm{I}}$ decides to branch then only processes that are informed about this decision can adapt their behaviour accordingly; else projection is **not** defined.

The remaining global types are projected as follows:

$$(\mathsf{r}_1 \to \mathsf{r}_2{:}\langle s[\mathsf{r}]{:}T\rangle.G)\!\upharpoonright_{\mathsf{p}} \triangleq \begin{cases} [\mathsf{r}_2]!\langle s[\mathsf{r}]{:}T\rangle.G\!\upharpoonright_{\mathsf{p}} & \text{if } \mathsf{p} = \mathsf{r}_1 \\ [\mathsf{r}_1]?\langle s[\mathsf{r}]{:}T\rangle.G\!\upharpoonright_{\mathsf{p}} & \text{if } \mathsf{p} = \mathsf{r}_2 \\ G\!\upharpoonright_{\mathsf{p}} & \text{otherwise} \end{cases} \qquad (G_1 \mid\mid G_2)\!\upharpoonright_{\mathsf{p}} \triangleq \begin{cases} G_1\!\upharpoonright_{\mathsf{p}} & \text{if } \mathsf{p} \notin \mathrm{R}(G_2) \\ G_2\!\upharpoonright_{\mathsf{p}} & \text{if } \mathsf{p} \notin \mathrm{R}(G_1) \end{cases}$$

$$t\!\upharpoonright_{\mathsf{p}} \triangleq t \qquad \mathtt{end}\!\upharpoonright_{\mathsf{p}} \triangleq \mathtt{end}$$

The projection of delegation is similar to communication. The projection of $G_1 \mid\mid G_2$ on p is **not** defined if p occurs on both sides of this parallel composition; it is $G_i\!\upharpoonright_{\mathsf{p}}$ if p occurs in exactly one side $i \in \{1,2\}$; or it is $(G_1 \mid\mid G_2)\!\upharpoonright_{\mathsf{p}} = G_1\!\upharpoonright_{\mathsf{p}} = G_2\!\upharpoonright_{\mathsf{p}} = \mathtt{end}$ if p does not occur at all. Type variables and successful termination are mapped onto themselves. We denote a global type G as *projectable* if for all $\mathsf{r} \in \mathrm{R}(G)$ the projection $G\!\upharpoonright_{\mathsf{r}}$ is defined. We restrict our attention to projectable global types. Projection maps well-formed global types onto the respective local type for a given role p, where the results of projection—if defined—are again well-formed.

B Typing Fault-Tolerant Processes

We use *typing rules* to derive type judgements, where we assume that all mentioned global types are well-formed and projectable, all local types are well-formed, and all environments are linear.We observe that all new cases are quite similar to their strongly reliable counterparts.

Rule (RSend) in Figure 3 checks strongly reliable senders, i.e., requires a matching strongly reliable sending in the local type of the actor and compares the actor with this type. With $\Gamma \Vdash y{:}\mathrm{S}$ we check that y is an expression of the sort S if all names x in y are replaced by arbitrary values of sort S_x for $x{:}\mathrm{S}_x \in \Gamma$. Then the continuation of the process is checked against the continuation of the type. The unreliable case is very similar, but additionally checks that the label is assigned to the sort of the expression in Γ. Rule (RGet) type strongly reliable receivers, where again the prefix is checked against a corresponding type prefix and the assumption $x{:}\mathrm{S}$ is added for the continuation. Again the unreliable case is very similar, but apart from the label also checks the sort of the default value.

Rule (RSel) checks the strongly reliable selection prefix, that the selected label matches one of the specified labels, and that the process continuation is well-typed w.r.t. the type continuation following the selected label. The only difference in the weakly reliable case is the set of roles for the receivers. For strongly reliable branching in (RBran) we check the prefix and that for each branch in the type there is a matching branch in the process that is well-typed w.r.t. the respective branch in the type. For the weakly reliable case we have to additionally check that the default labels of the process and the type coincide.

Rule (Crash) for crashed processes checks that $\mathrm{nsr}(\Delta)$, i.e., that for every type G or T in Δ the predicate $\mathrm{nsr}(G)$ or $\mathrm{nsr}(T)$ holds.

The combination of the 7 conditions in Conditions 1 in Section 4 might appear quite restrictive as e.g. the combination of the Condition 1.4 and 1.6 ensures the correct behaviour of weakly reliable branching such that branching messages can be skipped if and only if the respective sender has crashed. An implementation of such a weakly reliable interaction in an asynchronous system that is subject to message losses and process crashes, might require something like a perfect failure detector or actually solving consensus[1]. It is important to remember that these conditions are minimal assumptions on the system requirements and that system requirements are abstractions. Parts of them may be realised by actual software-code (which then allows to check them), whereas other parts of the system requirements may not be realised at all but rather observed (which then does not allow to verify them). Weakly reliable branching is a good example of this case. The easiest way to obtain a weakly reliable interaction, is by using a handshake communication and time-outs. If the sender time-outs while waiting for an acknowledgement, it resends the message. If the sender does not hear from its receiver for a long enough period of time, it assumes that the receiver has crashed and proceeds. With carefully chosen time-frames for the time-outs, this approach is a compromise between correctness and efficiency. In a theoretical sense, it is clearly not correct. There is no time-frame such that the sender can be really sure that the receiver has crashed. From a practical point of view, this is not so problematic, since in many systems failures are very unlikely. If failures that are so severe that they are not captured by the time-outs are extremely unlikely, then it is often much more efficient to just accept that the algorithm is not correct in these cases. Trying to obtain an algorithm that is always correct might be impossible or at least usually results into very inefficient algorithms. Moreover, verifying this requires to also verify the underlying communication infrastructure and the way in that failures may occur, which is impossible or at least impracticable. Because of that, it is an established method to verify the correctness of algorithms w.r.t. given system requirements (e.g. in [10, 20, 26]), even if these system requirements are not verified and often do not hold in all (but only nearly all) cases.

Let us have a closer look at the typing rules in the Figures 3 and 4. We observe that all typing rules are clearly distinguished by the outermost operator of the process in the conclusion except that there are two typing rules for restriction. With that, given a type judgement $\Gamma, \Theta \vdash P \triangleright \Delta$, we can use the structure of P—with a case split for restriction—to reason about the structure of the proof tree that was necessary to obtain $\Gamma, \Theta \vdash P \triangleright \Delta$ and from that derive conditions about the nature of the involved type environments. If P is e.g. a parallel composition $P_1 \mid P_2$ then, since there is only one rule to type parallel compositions (the Rule (Par)), $\Gamma, \Theta \vdash P_1 \mid P_2 \triangleright \Delta$ implies that there are Δ_1, Δ_2 such that $\Delta = \Delta_1 \cdot \Delta_2$, $\Gamma, \Theta \vdash P_1 \triangleright \Delta_1$, and $\Gamma, \Theta \vdash P_2 \triangleright \Delta_2$. In the following, we write 'by Rule (Par)' as short hand for 'by the clear distinction of the typing rules by the process in the conclusion and Rule (Par) in particular' and similar for the other rules.

In the following we prove some properties of our MPST variant. We start with an auxiliary result, proving that structural congruence preserves the validity of type judgements. The proof is by induction on $P \equiv P'$. In each case we can use the information about the structure of the process that is provided by the considered rule of structural congruence to conclude on the last few typing rules that had to be applied to

[1]Note that we consider in Section 5 a consensus algorithm. So, if the Condition 1 requires a solution of consensus, an example on top of that solving consensus would be pointless.

$$(\text{T-RSend})\ \frac{}{\Delta \cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]!_{\mathrm{r}}\langle \mathrm{S}\rangle.T \cdot s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{MT} \overset{s}{\mapsto} \Delta \cdot s[\mathsf{r}_1]{:}T \cdot s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{MT}\#\langle \mathrm{S}\rangle^{\mathrm{r}}}$$

$$(\text{T-RGet})\ \frac{}{\Delta \cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]?_{\mathrm{r}}\langle \mathrm{S}\rangle.T \cdot s_{\mathsf{r}_2\to\mathsf{r}_1}{:}\langle \mathrm{S}\rangle^{\mathrm{r}}\#\mathrm{MT} \overset{s}{\mapsto} \Delta \cdot s[\mathsf{r}_1]{:}T \cdot s_{\mathsf{r}_2\to\mathsf{r}_1}{:}\mathrm{MT}}$$

$$(\text{T-USend})\ \frac{}{\Delta \cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]!_{\mathrm{u}}l\langle \mathrm{S}\rangle.T \cdot s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{MT} \overset{s}{\mapsto} \Delta \cdot s[\mathsf{r}_1]{:}T \cdot s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{MT}\#l\langle \mathrm{S}\rangle^{\mathrm{u}}}$$

$$(\text{T-UGet})\ \frac{}{\Delta \cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]?_{\mathrm{u}}l\langle \mathrm{S}\rangle.T \cdot s_{\mathsf{r}_2\to\mathsf{r}_1}{:}l\langle \mathrm{S}\rangle^{\mathrm{u}}\#\mathrm{MT} \overset{s}{\mapsto} \Delta \cdot s[\mathsf{r}_1]{:}T \cdot s_{\mathsf{r}_2\to\mathsf{r}_1}{:}\mathrm{MT}}$$

$$(\text{T-USkip})\ \frac{}{\Delta \cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]?_{\mathrm{u}}l\langle \mathrm{S}\rangle.T \overset{s}{\mapsto} \Delta \cdot s[\mathsf{r}_1]{:}T}$$

$$(\text{T-ML})\ \frac{}{\Delta \cdot s_{\mathsf{r}_1\to\mathsf{r}_2}{:}l\langle \mathrm{S}\rangle^{\mathrm{u}}\#\mathrm{MT} \overset{s}{\mapsto} \Delta \cdot s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{MT}}$$

$$(\text{T-RSel})\ \frac{j \in \mathrm{I}}{\Delta \cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]!_{\mathrm{r}}\{l_i.T_i\}_{i\in \mathrm{I}} \cdot s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{MT} \overset{s}{\mapsto} \Delta \cdot s[\mathsf{r}_1]{:}T_j \cdot s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{MT}\#l_j^{\mathrm{r}}}$$

$$(\text{T-RBran})\ \frac{j \in \mathrm{I}}{\Delta \cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]?_{\mathrm{r}}\{l_i.T_i\}_{i\in \mathrm{I}} \cdot s_{\mathsf{r}_2\to\mathsf{r}_1}{:}l_j^{\mathrm{r}}\#\mathrm{MT} \overset{s}{\mapsto} \Delta \cdot s[\mathsf{r}_1]{:}T_j \cdot s_{\mathsf{r}_2\to\mathsf{r}_1}{:}\mathrm{MT}}$$

$$(\text{T-WSel})\ \frac{j \in \mathrm{I} \quad \mathsf{R} = \{\mathsf{r}_1,\ldots,\mathsf{r}_n\}}{\begin{array}{c}\Delta \cdot s[\mathsf{r}]{:}[\mathsf{R}]!_{\mathrm{w}}\{l_i.T_i\}_{i\in \mathrm{I},l_{\mathrm{d}}} \cdot s_{\mathsf{r}\to\mathsf{r}_1}{:}\mathrm{MT}_1 \cdot \ldots \cdot s_{\mathsf{r}\to\mathsf{r}_n}{:}\mathrm{MT}_n \overset{s}{\mapsto} \\ \Delta \cdot s[\mathsf{r}]{:}T_j \cdot s_{\mathsf{r}\to\mathsf{r}_1}{:}\mathrm{MT}_1\#l_j^{\mathrm{w}} \cdot \ldots \cdot s_{\mathsf{r}\to\mathsf{r}_n}{:}\mathrm{MT}_n\#l_j^{\mathrm{w}}\end{array}}$$

$$(\text{T-WBran})\ \frac{j \in \mathrm{I}}{\Delta \cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]?_{\mathrm{w}}\{l_i.T_i\}_{i\in \mathrm{I},l_{\mathrm{d}}} \cdot s_{\mathsf{r}_2\to\mathsf{r}_1}{:}l_j^{\mathrm{w}}\#\mathrm{MT} \overset{s}{\mapsto} \Delta \cdot s[\mathsf{r}_1]{:}T_j \cdot s_{\mathsf{r}_2\to\mathsf{r}_1}{:}\mathrm{MT}}$$

$$(\text{T-WSkip})\ \frac{}{\Delta \cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]?_{\mathrm{w}}\{l_i.T_i\}_{i\in \mathrm{I},l_{\mathrm{d}}} \overset{s}{\mapsto} \Delta \cdot s[\mathsf{r}_1]{:}T_{\mathrm{d}}}$$

$$(\text{T-Rec})\ \frac{}{\Delta \cdot s[\mathsf{r}]{:}(\mu t, \mathsf{c} = n)T \overset{s}{\mapsto} \Delta \cdot s[\mathsf{r}]{:}\left(T\{{}^{n}/_{\mathsf{c}}\}\right)\{{}^{(\mu t,\mathsf{c}=\mathrm{eval}(n+1))T}/_{t}\}}$$

$$(\text{T-Deleg})\ \frac{}{\Delta \cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]!\langle s'[\mathsf{r}]{:}T'\rangle.T \cdot s'[\mathsf{r}]{:}T' \cdot s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{MT} \overset{s}{\mapsto} \Delta \cdot s[\mathsf{r}_1]{:}T \cdot s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{MT}\#s'[\mathsf{r}]}$$

$$(\text{T-SRecv})\ \frac{}{\Delta \cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]?\langle s'[\mathsf{r}]{:}T'\rangle.T \cdot s_{\mathsf{r}_2\to\mathsf{r}_1}{:}s'[\mathsf{r}]\#\mathrm{MT} \overset{s}{\mapsto} \Delta \cdot s[\mathsf{r}_1]{:}T \cdot s'[\mathsf{r}]{:}T' \cdot s_{\mathsf{r}_2\to\mathsf{r}_1}{:}\mathrm{MT}}$$

$$(\text{T-LStep})\ \frac{\Delta \cdot s[\mathsf{r}]{:}T_1 \overset{s}{\mapsto} \Delta' \cdot s[\mathsf{r}]{:}T_1'}{\Delta \cdot s[\mathsf{r}]{:}[\mathsf{R}]\infty_e^{\mathsf{c}=n}[\langle \mathrm{S}_0\rangle.T_0]T_1;\langle \mathrm{S}_2\rangle.T_2 \overset{s}{\mapsto} \Delta' \cdot s[\mathsf{r}]{:}[\mathsf{R}]\infty_e^{\mathsf{c}=n}[\langle \mathrm{S}_0\rangle.T_0]T_1';\langle \mathrm{S}_2\rangle.T_2}$$

$$(\text{T-LCall})\ \frac{}{\Delta \cdot s[\mathsf{r}]{:}[\mathsf{R}]\infty_e^{\mathsf{c}=n}[\langle \mathrm{S}_0\rangle.T_0]\mathtt{call}\langle e\rangle;\langle \mathrm{S}_2\rangle.T_2 \overset{s}{\mapsto} \Delta \cdot s[\mathsf{r}]{:}[\mathsf{R}]\infty_e^{\mathsf{c}=\mathrm{eval}(n+1)}[\langle \mathrm{S}_0\rangle.T_0]T_0\{{}^{n}/_{\mathsf{c}}\};\langle \mathrm{S}_2\rangle.T_2}$$

$$(\text{T-LExitS})\ \frac{\mathsf{R} = \{\mathsf{r}_1,\ldots,\mathsf{r}_n\}}{\begin{array}{c}\Delta \cdot s[\mathsf{r}]{:}[\mathsf{R}]\infty_e^{\mathsf{c}=n}[\langle \mathrm{S}_0\rangle.T_0]T_1;\langle \mathrm{S}_2\rangle.T_2 \cdot s_{\mathsf{r}\to\mathsf{r}_1}{:}\mathrm{MT}_1 \cdot \ldots \cdot s_{\mathsf{r}\to\mathsf{r}_n}{:}\mathrm{MT}_n \overset{s}{\mapsto} \\ \Delta \cdot s[\mathsf{r}]{:}T_2 \cdot s_{\mathsf{r}\to\mathsf{r}_1}{:}\mathrm{MT}_1\#\mathtt{exit}\langle e,\mathrm{S}_2\rangle \cdot \ldots \cdot s_{\mathsf{r}\to\mathsf{r}_n}{:}\mathrm{MT}_n\#\mathtt{exit}\langle e,\mathrm{S}_2\rangle\end{array}}$$

$$(\text{T-LExitG})\ \frac{\mathsf{r}_e \in \mathsf{R}}{\Delta \cdot s[\mathsf{r}]{:}[\mathsf{R}]\infty_e^{\mathsf{c}=n}[\langle \mathrm{S}_0\rangle.T_0]T_1;\langle \mathrm{S}_2\rangle.T_2 \cdot s_{\mathsf{r}_e\to\mathsf{r}}{:}\mathtt{exit}\langle e,\mathrm{S}_2\rangle\#\mathrm{MT} \overset{s}{\mapsto} \Delta \cdot s[\mathsf{r}]{:}T_2 \cdot s_{\mathsf{r}_e\to\mathsf{r}}{:}\mathrm{MT}}$$

$$(\text{T-EDrop})\ \frac{}{\Delta \cdot s_{\mathsf{r}_2\to\mathsf{r}_1}{:}\mathtt{exit}\langle e,\mathrm{S}\rangle\#\mathrm{MT} \overset{s}{\mapsto} \Delta \cdot s_{\mathsf{r}_2\to\mathsf{r}_1}{:}\mathrm{MT}}$$

Figure 5: Reduction Rules for Session Environments.

derive the type judgement in the assumption. From these partial proof trees we obtain enough information to construct the proof tree for the conclusion. The proof is very similar to the respective proof in [24]. We highlight the differences in blue color.

Lemma 4 (Subject Congruence)**.** *If* $\Gamma, \Theta \vdash P \triangleright \Delta$ *and* $P \equiv P'$ *then* $\Gamma, \Theta \vdash P' \triangleright \Delta$.

Proof. The proof is by induction on $P \equiv P'$.

Case $P \mid \mathbf{0} \equiv P$**:** Assume $\Gamma, \Theta \vdash P \mid \mathbf{0} \triangleright \Delta$. By the Rule (Par), then there are $\Delta_P, \Delta_{\mathbf{0}}$ such that $\Delta = \Delta_P \cdot \Delta_{\mathbf{0}}$, $\Gamma, \Theta \vdash P \triangleright \Delta_P$, and $\Gamma, \Theta \vdash \mathbf{0} \triangleright \Delta_{\mathbf{0}}$. Moreover, by Rule (End), $\Gamma, \Theta \vdash \mathbf{0} \triangleright \Delta_{\mathbf{0}}$ implies that $\Delta_{\mathbf{0}} = \emptyset$ and, thus, $\Delta = \Delta_P$. Then also $\Gamma, \Theta \vdash P \triangleright \Delta$.
For the opposite direction assume $\Gamma, \Theta \vdash P \triangleright \Delta$. By Rule (End), $\Gamma, \Theta \vdash \mathbf{0} \triangleright \emptyset$. With Rule (Par) and because $\Delta = \Delta \cdot \emptyset$, then $\Gamma, \Theta \vdash P \mid \mathbf{0} \triangleright \Delta$.

Case $P_1 \mid P_2 \equiv P_2 \mid P_1$**:** Assume $\Gamma, \Theta \vdash P_1 \mid P_2 \triangleright \Delta$. By Rule (Par), then there are $\Delta_{P_1}, \Delta_{P_2}$ such that $\Delta = \Delta_{P_1} \cdot \Delta_{P_2}$ and $\Gamma, \Theta \vdash P_i \triangleright \Delta_{P_i}$. By Rule (Par) and since $\Delta = \Delta_{P_1} \cdot \Delta_{P_2}$ implies $\Delta = \Delta_{P_2} \cdot \Delta_{P_1}$, then $\Gamma, \Theta \vdash P_2 \mid P_1 \triangleright \Delta$.
The opposite direction is similar.

Case $P_1 \mid (P_2 \mid P_3) \equiv (P_1 \mid P_2) \mid P_3$**:** Assume $\Gamma, \Theta \vdash P_1 \mid (P_2 \mid P_3) \triangleright \Delta$.
By Rule (Par), then there are $\Delta_{P_1}, \Delta_{P_2}, \Delta_{P_3}$ such that $\Delta = \Delta_{P_1} \cdot (\Delta_{P_2} \cdot \Delta_{P_3})$ and $\Gamma, \Theta \vdash P_i \triangleright \Delta_{P_i}$. By Rule (Par) and because $\Delta = (\Delta_{P_1} \cdot \Delta_{P_2}) \cdot \Delta_{P_3}$, then $\Gamma, \Theta \vdash (P_1 \mid P_2) \mid P_3 \triangleright \Delta$.
The opposite direction is similar.

Case $(\mu X, \mathsf{c} = n)\mathbf{0} \equiv \mathbf{0}$**:** Assume $\Gamma, \Theta \vdash (\mu X, \mathsf{c} = n)\mathbf{0} \triangleright \Delta$. By Rule (Rec), then $\Delta = s[\mathsf{r}]{:}(\mu t, \mathsf{c} = n)T$, $\Gamma \Vdash n{:}\mathbb{N}$, $\Gamma \cdot \mathsf{c}{:}\mathbb{N}, \Theta \cdot X{:}s[\mathsf{r}]t \vdash \mathbf{0} \triangleright s[\mathsf{r}]{:}T$, and, by Rule (End), then $\Delta' = \emptyset$, $T = \mathtt{end}$, and $\mathsf{noLoop}(\Theta)$. Since $(\mu t, \mathsf{c} = n)\mathtt{end} = \mathtt{end}$ and $\Delta \cdot s[\mathsf{r}]{:}\mathtt{end} = \Delta$, then $\Delta = \emptyset$. By Rule (End), then $\Gamma, \Theta \vdash \mathbf{0} \triangleright \Delta$.
For the opposite direction assume $\Gamma, \Theta \vdash \mathbf{0} \triangleright \Delta$. By Rule (End), then $\Delta = \emptyset$ and $\mathsf{noLoop}(\Theta)$. Since n is a number, $\Gamma \Vdash n{:}\mathbb{N}$. By Rule (End), then also $\Gamma \cdot \mathsf{c}{:}\mathbb{N}, \Theta \cdot X{:}s[\mathsf{r}]t \vdash \mathbf{0} \triangleright \Delta$. By Rule (Rec) and since $(\mu t)\mathtt{end} = \mathtt{end}$ and $\Delta \cdot s[\mathsf{r}]{:}\mathtt{end} = \Delta$, then $\Gamma, \Theta \vdash (\mu X, \mathsf{c} = n)\mathbf{0} \triangleright \Delta$.

Case $(\nu x)\mathbf{0} \equiv \mathbf{0}$**:** Assume $\Gamma, \Theta \vdash (\nu x)\mathbf{0} \triangleright \Delta$. By one of the Rules (Res1) or (Res2), then there are Γ', Δ' such that $\Gamma \cdot \Gamma', \Theta \vdash \mathbf{0} \triangleright \Delta$ or $\Gamma \cdot \Gamma', \Theta \vdash \mathbf{0} \triangleright \Delta \cdot \Delta'$. In both cases we can conclude with Rule (End) that the session environment is empty, i.e., $\Delta = \emptyset$ and $\Delta \cdot \Delta' = \emptyset$, and that $\mathsf{noLoop}(\Theta)$. By Rule (End), then $\Gamma, \Theta \vdash \mathbf{0} \triangleright \Delta$.
For the opposite direction assume $\Gamma, \Theta \vdash \mathbf{0} \triangleright \Delta$. By Rule (End), then $\Delta = \emptyset$ and $\mathsf{noLoop}(\Theta)$. By Rule (End), then $\Gamma \cdot x{:}\mathrm{S}, \Theta \vdash \mathbf{0} \triangleright \Delta$ for some sort S—regardless of whether x is a value or a session channel. By Rule (Res1), then $\Gamma, \Theta \vdash (\nu x)\mathbf{0} \triangleright \Delta$.

Case $(\nu x)(\nu y)P \equiv (\nu y)(\nu x)P$**:** Assume $\Gamma, \Theta \vdash (\nu x)(\nu y)P \triangleright \Delta$. By alpha-conversion and the Rules (Res1) and (Res2), then there is Γ' and some (possibly empty) Δ' such that—for all combinations of the Rules (Res1) and (Res2) for the restrictions of x and y—we have $\Gamma \cdot \Gamma', \Theta \vdash P \triangleright \Delta \cdot \Delta'$. By the commutativity and associativity of $\cdot$ and two corresponding applications of the Rules (Res1) and (Res2), then also $\Gamma, \Theta \vdash (\nu y)(\nu x)P \triangleright \Delta$.
The opposite direction is similar.

Case $(\nu x)(P_1 \mid P_2) \equiv P_1 \mid (\nu x)P_2$ **if** $x \notin \mathrm{FN}(P_1)$**:** Assume $\Gamma, \Theta \vdash (\nu x)(P_1 \mid P_2) \triangleright \Delta$. By one of the restriction rules, (Res1) or (Res2), then $x \sharp (\Gamma, \Delta)$ and there are Γ', Δ' such that $\Gamma \cdot \Gamma', \Theta \vdash P_1 \mid P_2 \triangleright \Delta \cdot \Delta'$, where Γ' assigns to x either a sort or a global type and Δ' is either empty or contains only actors and message queues. Since $x \notin \mathrm{FN}(P_1)$ and by Rule (Par), then there are $\Delta_{P_1}, \Delta_{P_2}$ such that $\Delta = \Delta_{P_1} \cdot \Delta_{P_2}$, $\Gamma, \Theta \vdash P_1 \triangleright \Delta_{P_1}$, and $\Gamma \cdot \Gamma', \Theta \vdash P_2 \triangleright \Delta_{P_2} \cdot \Delta'$. With one of the Rules (Res1) or (Res2),

then $\Gamma, \Theta \vdash (\nu x)P_2 \triangleright \Delta_{P_2}$. With Rule (Par), then $\Gamma, \Theta \vdash P_1 \mid (\nu x)P_2 \triangleright \Delta$.
The opposite direction is similar. □

Moreover, types are preserved modulo substitution of names by values of the same sort. The proof is by induction on the typing rules. Again the proof is similar to the proof in [24]. Since we use expression to construct unique identifiers for loops in types, we have to apply the substitution to all parts of the type judgement. In the type environments substitution only affects names and expressions.

Lemma 5 (Substitution). *If* $\Gamma \cdot c{:}\mathrm{S}_c, \Theta \vdash Q \triangleright \Delta$ *and* $\Gamma \Vdash d{:}\mathrm{S}_c$*, then* $\Gamma\{^d/_c\}, \Theta\{^d/_c\} \vdash Q\{^d/_c\} \triangleright \Delta\{^d/_c\}$.

Proof. The proof is by induction on the derivation of $\Gamma \cdot c{:}\mathrm{S}_c, \Theta \vdash Q \triangleright \Delta$.

(Req) Then, $Q = \overline{a}[\mathsf{n}](s).P$, $a{:}G \in \Gamma$, $|\mathrm{R}(G)| = \mathsf{n}$, and $\Gamma \cdot c{:}\mathrm{S}_c, \Theta \vdash P \triangleright \Delta \cdot s[\mathsf{n}]{:}G\upharpoonright_{\mathsf{n}}$. Without loss of generality, assume $s \neq c$. Because of linearity, $a{:}G \in \Gamma$ implies $a \neq c$. By the induction hypothesis, $\Gamma \cdot c{:}\mathrm{S}_c, \Theta \vdash P \triangleright \Delta \cdot s[\mathsf{n}]{:}G\upharpoonright_{\mathsf{n}}$ and $\Gamma \Vdash d{:}\mathrm{S}_c$ imply $\Gamma\{^d/_c\}, \Theta\{^d/_c\} \vdash P\{^d/_c\} \triangleright \Delta\{^d/_c\} \cdot s[\mathsf{n}]{:}(G\{^d/_c\})\upharpoonright_{\mathsf{n}}$. By (RReq), then $\Gamma\{^d/_c\}, \Theta\{^d/_c\} \vdash Q\{^d/_c\} \triangleright \Delta\{^d/_c\}$.

(Acc) Then, $Q = a[\mathsf{r}](s).P$, $a{:}G \in \Gamma$, $0 < \mathsf{r} < |\mathrm{R}(G)|$, and $\Gamma \cdot c{:}\mathrm{S}_c, \Theta \vdash P \triangleright \Delta \cdot s[\mathsf{r}]{:}G\upharpoonright_{\mathsf{r}}$. Without loss of generality, assume $s \neq c$. Because of linearity, $a{:}G \in \Gamma$ implies $a \neq c$. By the induction hypothesis, $\Gamma\{^d/_c\}, \Theta\{^d/_c\} \vdash P\{^d/_c\} \triangleright \Delta\{^d/_c\} \cdot s[\mathsf{r}]{:}G\upharpoonright_{\mathsf{r}}$. By (Acc), then $\Gamma\{^d/_c\}, \Theta\{^d/_c\} \vdash Q\{^d/_c\} \triangleright \Delta\{^d/_c\}$.

(RSend) Then, $Q = s[\mathsf{r}_1, \mathsf{r}_2]!_{\mathrm{r}}\langle y\rangle.P$, $\Delta = \Delta' \cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]!_{\mathrm{r}}\langle \mathrm{S}\rangle.T$, $\Gamma \cdot c{:}\mathrm{S}_c, \Theta \Vdash y{:}\mathrm{S}$, and $\Gamma \cdot c{:}\mathrm{S}_c, \Theta \vdash P \triangleright \Delta' \cdot s[\mathsf{r}_1]{:}T$. Because $\Gamma \cdot c{:}\mathrm{S}_c \Vdash y{:}\mathrm{S}$, $s \neq c$. With $\Gamma \Vdash d{:}\mathrm{S}_c$, then $\Gamma \Vdash y\{^d/_c\}{:}\mathrm{S}$ and $\Gamma\{^d/_c\} \Vdash y\{^d/_c\}{:}\mathrm{S}$. By the induction hypothesis, $\Gamma\{^d/_c\}, \Theta\{^d/_c\} \vdash P\{^d/_c\} \triangleright \Delta'\{^d/_c\} \cdot s[\mathsf{r}_1]{:}T\{^d/_c\}$. By (RSend), then $\Gamma\{^d/_c\}, \Theta\{^d/_c\} \vdash Q\{^d/_c\} \triangleright \Delta\{^d/_c\}$.

(RGet) Then, $Q = s[\mathsf{r}_1, \mathsf{r}_2]?_{\mathrm{r}}(x).P$, $\Delta = \Delta' \cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]?_{\mathrm{r}}\langle \mathrm{S}\rangle.T$, $x\sharp(\Gamma, c, \Delta', s)$, and $\Gamma \cdot c{:}\mathrm{S}_c \cdot x{:}\mathrm{S}, \Theta \vdash P_i \triangleright \Delta' \cdot s[\mathsf{r}_1]{:}T$. Because $\Gamma \cdot c{:}\mathrm{S}_c \cdot x{:}\mathrm{S}, \Theta \vdash P_i \triangleright \Delta' \cdot s[\mathsf{r}_1]{:}T$, $s \neq c$. By the induction hypothesis, $\Gamma\{^d/_c\} \cdot x{:}\mathrm{S}, \Theta\{^d/_c\} \vdash P\{^d/_c\} \triangleright \Delta'\{^d/_c\} \cdot s[\mathsf{r}_1]{:}T\{^d/_c\}$. By (RGet), then $\Gamma\{^d/_c\}, \Theta\{^d/_c\} \vdash Q\{^d/_c\} \triangleright \Delta\{^d/_c\}$.

(USend) Then, $Q = s[\mathsf{r}_1, \mathsf{r}_2]!_{\mathrm{u}} l\langle y\rangle.P$, $\Delta = \Delta' \cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]!_{\mathrm{u}} l'\langle \mathrm{S}\rangle.T$, $l \doteq l'$, $\Gamma \cdot c{:}\mathrm{S}_c \Vdash y{:}\mathrm{S}$, and $\Gamma \cdot c{:}\mathrm{S}_c, \Theta \vdash P \triangleright \Delta' \cdot s[\mathsf{r}_1]{:}T$. Because of $\Gamma \cdot c{:}\mathrm{S}_c \Vdash y{:}\mathrm{S}$, $s \neq c$. With $\Gamma \Vdash d{:}\mathrm{S}_c$, then $\Gamma \Vdash y\{^d/_c\}{:}\mathrm{S}$ and $\Gamma\{^d/_c\} \Vdash y\{^d/_c\}{:}\mathrm{S}$. By the induction hypothesis, $\Gamma\{^d/_c\}, \Theta\{^d/_c\} \vdash P\{^d/_c\} \triangleright \Delta'\{^d/_c\} \cdot s[\mathsf{r}_1]{:}T\{^d/_c\}$. By (USend), then $\Gamma\{^d/_c\}, \Theta\{^d/_c\} \vdash Q\{^d/_c\} \triangleright \Delta\{^d/_c\}$.

(UGet) Then, $Q = s[\mathsf{r}_1, \mathsf{r}_2]?_{\mathrm{u}} l\langle v\rangle(x).P$, $\Delta = \Delta' \cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]?_{\mathrm{u}} l'\langle \mathrm{S}\rangle.T$, $l \doteq l'$, $\Gamma \cdot c{:}\mathrm{S}_c \Vdash v{:}\mathrm{S}$, $x\sharp(\Gamma, c, \Delta', s)$, and $\Gamma \cdot c{:}\mathrm{S}_c \cdot x{:}\mathrm{S}, \Theta \vdash P \triangleright \Delta' \cdot s[\mathsf{r}_1]{:}T$. Because of $\Gamma \cdot c{:}\mathrm{S}_c \Vdash v{:}\mathrm{S}$, $s \neq c$. With $\Gamma \Vdash d{:}\mathrm{S}_c$, then $\Gamma \Vdash v\{^d/_c\}{:}\mathrm{S}$ and $\Gamma\{^d/_c\} \Vdash v\{^d/_c\}{:}\mathrm{S}$. By the induction hypothesis, $\Gamma\{^d/_c\}, \Theta\{^d/_c\} \vdash P\{^d/_c\} \triangleright \Delta'\{^d/_c\} \cdot s[\mathsf{r}_1]{:}T\{^d/_c\}$. By (UGet), then $\Gamma\{^d/_c\}, \Theta\{^d/_c\} \vdash Q\{^d/_c\} \triangleright \Delta\{^d/_c\}$.

(RSel) Then, $Q = s[\mathsf{r}_1, \mathsf{r}_2]!_{\mathrm{r}} l.P$, $\Delta = \Delta' \cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]!_{\mathrm{r}}\{l_i.T_i\}_{i \in \mathrm{I}}$, $j \in \mathrm{I}$, $l \doteq l_j$, and $\Gamma \cdot c{:}\mathrm{S}_c, \Theta \vdash P \triangleright \Delta' \cdot s[\mathsf{r}_1]{:}T_j$. Because $\Gamma \cdot c{:}\mathrm{S}_c, \Theta \vdash P \triangleright \Delta' \cdot s[\mathsf{r}_1]{:}T_j$, $s \neq c$. By the induction hypothesis, $\Gamma\{^d/_c\}, \Theta\{^d/_c\} \vdash P\{^d/_c\} \triangleright \Delta'\{^d/_c\} \cdot s[\mathsf{r}_1]{:}T_j\{^d/_c\}$. By (RSel), then $\Gamma\{^d/_c\}, \Theta\{^d/_c\} \vdash Q\{^d/_c\} \triangleright \Delta\{^d/_c\}$.

(RBran) Then, $Q = s[\mathsf{r}_1, \mathsf{r}_2]?_{\mathrm{r}}\{l_i.P_i\}_{i \in \mathrm{I}_1}$, $\Delta = \Delta' \cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]?_{\mathrm{r}}\{l_i.T_i\}_{i \in \mathrm{I}_2}$, and, for all $j \in \mathrm{I}_2$ exists some $i \in \mathrm{I}_1$ such that $l_i \doteq l_j$ and $\Gamma \cdot c{:}\mathrm{S}_c, \Theta \vdash P_i \triangleright \Delta' \cdot s[\mathsf{r}_1]{:}T_j$. Fix j and i. Because $\Gamma \cdot c{:}\mathrm{S}_c, \Theta \vdash P_i \triangleright \Delta' \cdot s[\mathsf{r}_1]{:}T_j$, $s \neq c$. By the induction hypothesis, $\Gamma\{^d/_c\}, \Theta\{^d/_c\} \vdash P_i\{^d/_c\} \triangleright \Delta'\{^d/_c\} \cdot s[\mathsf{r}_1]{:}T_j\{^d/_c\}$. By (RBran), then $\Gamma\{^d/_c\}, \Theta\{^d/_c\} \vdash Q\{^d/_c\} \triangleright \Delta\{^d/_c\}$.

(WSel) Then, $Q = s[\mathsf{r}, \mathsf{R}]!_{\mathrm{w}} l.P$, $\Delta = \Delta' \cdot s[\mathsf{r}]{:}[\mathsf{R}]!_{\mathrm{w}}\{l_i.T_i\}_{i \in \mathrm{I}, l_{\mathrm{d}}}$, $j \in \mathrm{I}$, $l \doteq l_j$, and $\Gamma \cdot c{:}\mathrm{S}_c, \Theta \vdash P \triangleright \Delta' \cdot s[\mathsf{r}]{:}T_j$. Because $\Gamma \cdot c{:}\mathrm{S}_c, \Theta \vdash P \triangleright \Delta' \cdot s[\mathsf{r}]{:}T_j$, $s \neq c$. By the induction hypothesis, $\Gamma\{^d/_c\}, \Theta\{^d/_c\} \vdash P\{^d/_c\} \triangleright \Delta'\{^d/_c\} \cdot s[\mathsf{r}]{:}T_j\{^d/_c\}$. By (WSel), then $\Gamma\{^d/_c\}, \Theta\{^d/_c\} \vdash Q\{^d/_c\} \triangleright \Delta\{^d/_c\}$.

(WBran) Then, $Q = s[\mathsf{r}_1,\mathsf{r}_2]?_{\mathrm{w}}\{l_i.P_i\}_{i\in \mathrm{I}_1,l_{\mathrm{d}}}$, $\Delta = \Delta' \cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]?_{\mathrm{w}}\{l_i.T_i\}_{i\in \mathrm{I}_2,l'_{\mathrm{d}}}$, $l_{\mathrm{d}} \doteq l'_{\mathrm{d}}$, and, for all $j \in \mathrm{I}_2$ exists some $i \in \mathrm{I}_1$ such that $l_i \doteq l_j$ and $\Gamma\cdot c{:}\mathrm{S}_c,\Theta \vdash P_i \triangleright \Delta'\cdot s[\mathsf{r}_1]{:}T_j$. Fix j and i. Because $\Gamma\cdot c{:}\mathrm{S}_c,\Theta \vdash P_i \triangleright \Delta'\cdot s[\mathsf{r}_1]{:}T_j$, $s \neq c$. By the induction hypothesis, $\Gamma\{^d/_c\},\Theta\{^d/_c\} \vdash P_i\{^d/_c\} \triangleright \Delta'\{^d/_c\}\cdot s[\mathsf{r}_1]{:}T_j\{^d/_c\}$. By (WBran), then $\Gamma\{^d/_c\},\Theta\{^d/_c\} \vdash Q\{^d/_c\} \triangleright \Delta\{^d/_c\}$.

(If) Then, $Q = \texttt{if}\ e\ \texttt{then}\ P\ \texttt{else}\ P'$, $\Gamma\cdot c{:}\mathrm{S}_c \Vdash e{:}\mathbb{B}$, $\Gamma\cdot c{:}\mathrm{S}_c,\Theta \vdash P \triangleright \Delta$, and $\Gamma\cdot c{:}\mathrm{S}_c,\Theta \vdash P' \triangleright \Delta$. With $\Gamma \Vdash d{:}\mathrm{S}_c$, then $\Gamma \Vdash e\{^d/_c\}{:}\mathbb{B}$ and $\Gamma\{^d/_c\} \Vdash e\{^d/_c\}{:}\mathbb{B}$. By the induction hypothesis, $\Gamma\{^d/_c\},\Theta\{^d/_c\} \vdash P\{^d/_c\} \triangleright \Delta\{^d/_c\}$ and $\Gamma\{^d/_c\},\Theta\{^d/_c\} \vdash P'\{^d/_c\} \triangleright \Delta\{^d/_c\}$. By (If), then $\Gamma\{^d/_c\},\Theta\{^d/_c\} \vdash Q\{^d/_c\} \triangleright \Delta\{^d/_c\}$.

(Deleg) Then $Q = s[\mathsf{r}_1,\mathsf{r}_2]!\langle\langle s'[\mathsf{r}]\rangle\rangle.P$, $\Delta = \Delta'\cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]!\langle s'[\mathsf{r}]{:}T'\rangle.T \cdot s'[\mathsf{r}]{:}T'$, and $\Gamma\cdot c{:}\mathrm{S}_c,\Theta \vdash P \triangleright \Delta'\cdot s[\mathsf{r}_1]{:}T$. Because $\Gamma\cdot c{:}\mathrm{S}_c,\Theta \vdash Q \triangleright \Delta$, $c \neq s$ and $c \neq s'$. By the induction hypothesis, $\Gamma\{^d/_c\},\Theta\{^d/_c\} \vdash P\{^d/_c\} \triangleright \Delta'\{^d/_c\}\cdot s[\mathsf{r}_1]{:}T\{^d/_c\}$. By (Deleg), then $\Gamma\{^d/_c\},\Theta\{^d/_c\} \vdash Q\{^d/_c\} \triangleright \Delta\{^d/_c\}$.

(SRecv) Then $Q = s[\mathsf{r}_1,\mathsf{r}_2]?((s'[\mathsf{r}])).P$, $\Delta = \Delta'\cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]?\langle s'[\mathsf{r}]{:}T'\rangle.T$, and $\Gamma\cdot c{:}\mathrm{S}_c,\Theta \vdash P \triangleright \Delta'\cdot s[\mathsf{r}_1]{:}T \cdot s'[\mathsf{r}]{:}T'$. Because $\Gamma\cdot c{:}\mathrm{S}_c,\Theta \vdash P \triangleright \Delta'\cdot s[\mathsf{r}_1]{:}T\cdot s'[\mathsf{r}]{:}T'$, $c \neq s$ and $c \neq s'$. By the induction hypothesis, $\Gamma\{^d/_c\},\Theta\{^d/_c\} \vdash P\{^d/_c\} \triangleright \Delta'\{^d/_c\}\cdot s[\mathsf{r}_1]{:}T\{^d/_c\}\cdot s'[\mathsf{r}]{:}T'\{^d/_c\}$. By (SRecv), then $\Gamma\{^d/_c\},\Theta\{^d/_c\} \vdash Q\{^d/_c\} \triangleright \Delta\{^d/_c\}$.

(Par) Then, $Q = P \mid P'$, $\Delta = \Delta_1\cdot\Delta_2$, $\Gamma\cdot c{:}\mathrm{S}_c,\Theta \vdash P \triangleright \Delta_1$, and $\Gamma\cdot c{:}\mathrm{S}_c,\Theta \vdash P' \triangleright \Delta_2$. By the induction hypothesis, $\Gamma\{^d/_c\},\Theta\{^d/_c\} \vdash P\{^d/_c\} \triangleright \Delta_1\{^d/_c\}$ and $\Gamma\{^d/_c\},\Theta\{^d/_c\} \vdash P'\{^d/_c\} \triangleright \Delta_2\{^d/_c\}$. By (Par), then $\Gamma\{^d/_c\},\Theta\{^d/_c\} \vdash Q\{^d/_c\} \triangleright \Delta\{^d/_c\}$.

(Res1) Then, $Q = (\nu x)P$, $x\sharp(\Gamma,c,\Delta)$, and $\Gamma\cdot c{:}\mathrm{S}_c\cdot x{:}\mathrm{S},\Theta \vdash P \triangleright \Delta$. By the induction hypothesis, $\Gamma\{^d/_c\}\cdot x{:}\mathrm{S},\Theta\{^d/_c\} \vdash P\{^d/_c\} \triangleright \Delta\{^d/_c\}$. By (Res1), then $\Gamma\{^d/_c\},\Theta\{^d/_c\} \vdash Q\{^d/_c\} \triangleright \Delta\{^d/_c\}$.

(Rec) Then, $Q = (\mu X,\mathsf{cnt}=n)P$, $\Delta = s[\mathsf{r}]{:}(\mu t,\mathsf{cnt}=n)T$, $\Gamma \Vdash n{:}\mathbb{N}$, and $\Gamma\cdot c{:}\mathrm{S}_c\cdot\mathsf{cnt}{:}\mathbb{N},\Theta\cdot X{:}s[\mathsf{r}]t \vdash P \triangleright s[\mathsf{r}]{:}T$. Without loss of generality, assume $\mathsf{cnt} \neq c$. By the induction hypothesis, $\Gamma\{^d/_c\}\cdot\mathsf{cnt}{:}\mathbb{N},\Theta\{^d/_c\}\cdot X{:}s[\mathsf{r}]t \vdash P\{^d/_c\} \triangleright s[\mathsf{r}]{:}T\{^d/_c\}$. By (Rec), then $\Gamma\{^d/_c\},\Theta\{^d/_c\} \vdash Q\{^d/_c\} \triangleright \Delta\{^d/_c\}$.

(Var) Then, $Q = X$, $\Theta = \Theta'\cdot X{:}s[\mathsf{r}]{:}t$, and $\Delta = s[\mathsf{r}]{:}t$. By (Var), then $\Gamma\{^d/_c\},\Theta\{^d/_c\} \vdash Q\{^d/_c\} \triangleright \Delta\{^d/_c\}$.

(End) Then, $Q = \mathbf{0}$, $\Delta = \emptyset$, and $\mathsf{noLoop}(\Theta)$. Then $\mathsf{noLoop}(\Theta\{^d/_c\})$. By (End), then $\Gamma,\Theta\{^d/_c\} \vdash Q\{^d/_c\} \triangleright \Delta$.

(Crash) Then, $Q = \bot$ and $\mathrm{nsr}(\Delta)$. By (UCrash), then $\Gamma\{^d/_c\},\Theta\{^d/_c\} \vdash Q\{^d/_c\} \triangleright \Delta\{^d/_c\}$.

(Loop) Then, $Q = s[\mathsf{r},\mathrm{R}]\infty_e^{\mathsf{cnt}=n}[(x).P_0]P_1;(y).P_2$, $\Delta = \Delta'\cdot s[\mathsf{r}]{:}[\mathrm{R}]\infty_e^{\mathsf{cnt}=n}[\langle \mathrm{S}_0\rangle.T_0]T_1;\langle \mathrm{S}_2\rangle.T_2$, we have $x,y\sharp(\Gamma,c,\Delta,s)$, $\Gamma \Vdash n{:}\mathbb{N}$, $\mathrm{unr}(T_0)$, $\mathrm{unr}(T_1)$, $\Gamma\cdot c{:}\mathrm{S}_c\cdot x{:}\mathrm{S}_0\cdot \mathsf{c}{:}\mathbb{N}, e{:}s[\mathsf{r}]\langle \mathrm{S}_0,\mathrm{S}_2\rangle \vdash P_0 \triangleright s[\mathsf{r}]{:}T_0$, $\Gamma\cdot c{:}\mathrm{S}_c\cdot \mathsf{c}{:}\mathbb{N}, e{:}s[\mathsf{r}]\langle \mathrm{S}_0,\mathrm{S}_2\rangle \vdash P_1 \triangleright s[\mathsf{r}]{:}T_1$, and $\Gamma\cdot c{:}\mathrm{S}_c\cdot y{:}\mathrm{S}_2,\Theta \vdash P_2 \triangleright \Delta\cdot s[\mathsf{r}]{:}T_2$. Because of the judgement $\Gamma\cdot c{:}\mathrm{S}_c\cdot x{:}\mathrm{S}_0, e{:}s[\mathsf{r}]\langle \mathrm{S}_0,\mathrm{S}_2\rangle \vdash P_0 \triangleright s[\mathsf{r}]{:}T_0$, $s \neq c$. By the induction hypothesis, we have $\Gamma\{^d/_c\}\cdot x{:}\mathrm{S}_0\cdot \mathsf{cnt}{:}\mathbb{N}, e\{^d/_c\}{:}s[\mathsf{r}]\langle \mathrm{S}_0,\mathrm{S}_2\rangle \vdash P_0\{^d/_c\} \triangleright s[\mathsf{r}]{:}T_0\{^d/_c\}$ (program), $\Gamma\{^d/_c\}\cdot \mathsf{cnt}{:}\mathbb{N}, e\{^d/_c\}{:}s[\mathsf{r}]\langle \mathrm{S}_0,\mathrm{S}_2\rangle \vdash P_1\{^d/_c\} \triangleright s[\mathsf{r}]{:}T_1\{^d/_c\}$ (loop body), and $\Gamma\{^d/_c\}\cdot y{:}\mathrm{S}_2,\Theta\{^d/_c\} \vdash P_2\{^d/_c\} \triangleright \Delta\{^d/_c\}\cdot s[\mathsf{r}]{:}T_2\{^d/_c\}$ (loop continuation). By (Loop), then $\Gamma\{^d/_c\},\Theta\{^d/_c\} \vdash Q\{^d/_c\} \triangleright \Delta\{^d/_c\}$.

(Call) Then, $Q = \texttt{call}\langle e,e_v\rangle$, $\Theta = \Theta'\cdot e{:}s[\mathsf{r}]\langle \mathrm{S}_0,\mathrm{S}_2\rangle$, $\Delta = s[\mathsf{r}]{:}\texttt{call}\langle e\rangle$, and $\Gamma\cdot c{:}\mathrm{S}_c \Vdash e_v{:}\mathrm{S}_0$. With $\Gamma \Vdash d{:}\mathrm{S}_c$, then $\Gamma \Vdash e_v\{^d/_c\}{:}\mathrm{S}_0$ and $\Gamma\{^d/_c\} \Vdash e_v\{^d/_c\}{:}\mathrm{S}_0$. By (Call), then $\Gamma\{^d/_c\},\Theta\{^d/_c\} \vdash Q\{^d/_c\} \triangleright \Delta\{^d/_c\}$.

(Exit) Then, $Q = \texttt{exit}\langle e,e_v\rangle$, $\Theta = \Theta'\cdot e{:}s[\mathsf{r}]\langle \mathrm{S}_0,\mathrm{S}_2\rangle$, $\Delta = s[\mathsf{r}]{:}T_1$, and $\Gamma\cdot c{:}\mathrm{S}_c \Vdash e_v{:}\mathrm{S}_2$. With $\Gamma \Vdash d{:}\mathrm{S}_c$, then $\Gamma \Vdash e_v\{^d/_c\}{:}\mathrm{S}_2$ and $\Gamma\{^d/_c\} \Vdash e_v\{^d/_c\}{:}\mathrm{S}_2$. By (Exit), then $\Gamma\{^d/_c\},\Theta\{^d/_c\} \vdash Q\{^d/_c\} \triangleright \Delta\{^d/_c\}$.

(Res2) Then, $\{s[\mathsf{r}]{:}G\lceil_{\mathsf{r}} \mid \mathsf{r}\in \mathrm{R}(G)\}\cdot\{s_{\mathsf{r}\to\mathsf{r}'}{:}[\,] \mid \mathsf{r},\mathsf{r}' \in \mathrm{R}(G') \wedge \mathsf{r}\neq\mathsf{r}'\} \overset{s}{\Rrightarrow} \Delta'$, $Q = (\nu s)P$, $s\sharp(\Gamma,c,\Delta)$, $a{:}G \in \Gamma$, and $\Gamma\cdot c{:}\mathrm{S}_c,\Theta \vdash P \triangleright \Delta\cdot\Delta'$. Since Figure 5 does not pose any requirements on expressions except that they have to coincide on loops and matching exit-messages, $\{s[\mathsf{r}]{:}G\{{}^{d}/_{c}\}\upharpoonright_{\mathsf{r}} \mid \mathsf{r} \in \mathsf{R}(G\{{}^{d}/_{c}\})\} \cdot \{s_{\mathsf{r}\to\mathsf{r}'}{:}[\,] \mid \mathsf{r},\mathsf{r}' \in \mathsf{R}(G'\{{}^{d}/_{c}\}) \wedge \mathsf{r} \neq \mathsf{r}'\} \overset{s}{\Mapsto} \Delta'\{{}^{d}/_{c}\}$. By the induction hypothesis, $\Gamma\{{}^{d}/_{c}\}, \Theta\{{}^{d}/_{c}\} \vdash P\{{}^{d}/_{c}\} \triangleright \Delta\{{}^{d}/_{c}\} \cdot \Delta'\{{}^{d}/_{c}\}$. By (ResS), then $\Gamma\{{}^{d}/_{c}\}, \Theta\{{}^{d}/_{c}\} \vdash Q\{{}^{d}/_{c}\} \triangleright \Delta\{{}^{d}/_{c}\}$.

(MQComR) Then, $Q = s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\langle v\rangle^{\mathsf{r}}\#\mathrm{M}$, $\Delta = s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\langle \mathrm{S}\rangle^{\mathsf{r}}\#\mathrm{MT}$, $\Gamma \cdot c{:}\mathrm{S}_c \Vdash v{:}\mathrm{S}$, and $\Gamma \cdot c{:}\mathrm{S}_c, \Theta \vdash s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{M} \triangleright s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{MT}$. Because $\Gamma \cdot c{:}\mathrm{S}_c \Vdash v{:}\mathrm{S}$, $s \neq c$. With $\Gamma \Vdash d{:}\mathrm{S}_c$, then $\Gamma \Vdash v\{{}^{d}/_{c}\}{:}\mathrm{S}$ and $\Gamma\{{}^{d}/_{c}\} \Vdash v\{{}^{d}/_{c}\}{:}\mathrm{S}$. By the induction hypothesis, $\Gamma\{{}^{d}/_{c}\}, \Theta\{{}^{d}/_{c}\} \vdash s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{M}\{{}^{d}/_{c}\} \triangleright s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{MT}\{{}^{d}/_{c}\}$. By (MQComR), then $\Gamma\{{}^{d}/_{c}\}, \Theta\{{}^{d}/_{c}\} \vdash Q\{{}^{d}/_{c}\} \triangleright \Delta\{{}^{d}/_{c}\}$.

(MQComU) Then, $Q = s_{\mathsf{r}_1\to\mathsf{r}_2}{:}l\langle v\rangle^{\mathsf{u}}\#\mathrm{M}$, $\Delta = s_{\mathsf{r}_1\to\mathsf{r}_2}{:}l'\langle \mathrm{S}\rangle^{\mathsf{u}}\#\mathrm{MT}$, $l \doteq l'$, $\Gamma \cdot c{:}\mathrm{S}_c \Vdash v{:}\mathrm{S}$, and $\Gamma \cdot c{:}\mathrm{S}_c, \Theta \vdash s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{M} \triangleright s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{MT}$. Because $\Gamma \cdot c{:}\mathrm{S}_c \Vdash v{:}\mathrm{S}$, $s \neq c$. With $\Gamma \Vdash d{:}\mathrm{S}_c$, then $\Gamma \Vdash v\{{}^{d}/_{c}\}{:}\mathrm{S}$ and $\Gamma\{{}^{d}/_{c}\} \Vdash v\{{}^{d}/_{c}\}{:}\mathrm{S}$. By the induction hypothesis, $\Gamma\{{}^{d}/_{c}\}, \Theta\{{}^{d}/_{c}\} \vdash s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{M}\{{}^{d}/_{c}\} \triangleright s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{MT}\{{}^{d}/_{c}\}$. By (MQComU), then $\Gamma\{{}^{d}/_{c}\}, \Theta\{{}^{d}/_{c}\} \vdash Q\{{}^{d}/_{c}\} \triangleright \Delta\{{}^{d}/_{c}\}$.

(MQBranR) Then, $Q = s_{\mathsf{r}_1\to\mathsf{r}_2}{:}l^{\mathsf{r}}\#\mathrm{M}$, $\Delta = s_{\mathsf{r}_1\to\mathsf{r}_2}{:}l'^{\mathsf{r}}\#\mathrm{MT}$, $l \doteq l'$, and $\Gamma \cdot c{:}\mathrm{S}_c, \Theta \vdash s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{M} \triangleright s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{MT}$. Because $\Gamma \cdot c{:}\mathrm{S}_c, \Theta \vdash s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{M} \triangleright s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{MT}$, $s \neq c$. By the induction hypothesis, $\Gamma\{{}^{d}/_{c}\}, \Theta\{{}^{d}/_{c}\} \vdash s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{M}\{{}^{d}/_{c}\} \triangleright s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{MT}\{{}^{d}/_{c}\}$. By (MQBranR), then $\Gamma\{{}^{d}/_{c}\}, \Theta\{{}^{d}/_{c}\} \vdash Q\{{}^{d}/_{c}\} \triangleright \Delta\{{}^{d}/_{c}\}$.

(MQBranW) Then, $Q = s_{\mathsf{r}_1\to\mathsf{r}_2}{:}l^{\mathsf{w}}\#\mathrm{M}$, $\Delta = s_{\mathsf{r}_1\to\mathsf{r}_2}{:}l'^{\mathsf{w}}\#\mathrm{MT}$, $l \doteq l'$, and $\Gamma \cdot c{:}\mathrm{S}_c, \Theta \vdash s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{M} \triangleright s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{MT}$. Because $\Gamma \cdot c{:}\mathrm{S}_c \vdash s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{M} \triangleright s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{MT}$, $s \neq c$. By the induction hypothesis, $\Gamma\{{}^{d}/_{c}\}, \Theta\{{}^{d}/_{c}\} \vdash s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{M}\{{}^{d}/_{c}\} \triangleright s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{MT}\{{}^{d}/_{c}\}$. By (MQBranW), then $\Gamma\{{}^{d}/_{c}\}, \Theta\{{}^{d}/_{c}\} \vdash Q\{{}^{d}/_{c}\} \triangleright \Delta\{{}^{d}/_{c}\}$.

(MQDeleg) Then $Q = s_{\mathsf{r}_1\to\mathsf{r}_2}{:}s'[\mathsf{r}]\#\mathrm{M}$, $\Delta = s_{\mathsf{r}_1\to\mathsf{r}_2}{:}s'[\mathsf{r}]\#\mathrm{MT}$, and $\Gamma \cdot c{:}\mathrm{S}_c, \Theta \vdash s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{M} \triangleright s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{MT}$. Because $\Gamma \cdot c{:}\mathrm{S}_c, \Theta \vdash s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{M} \triangleright s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{MT}$, $s \neq c$. By the induction hypothesis, $\Gamma\{{}^{d}/_{c}\}, \Theta\{{}^{d}/_{c}\} \vdash s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{M}\{{}^{d}/_{c}\} \triangleright s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{MT}\{{}^{d}/_{c}\}$. By (MQDeleg), then $\Gamma\{{}^{d}/_{c}\}, \Theta\{{}^{d}/_{c}\} \vdash Q\{{}^{d}/_{c}\} \triangleright \Delta\{{}^{d}/_{c}\}$.

(MQNil) Then, $Q = s_{\mathsf{r}_1\to\mathsf{r}_2}{:}[\,]$ and $\Delta = s_{\mathsf{r}_1\to\mathsf{r}_2}{:}[\,]$. By (MQNil), then we have $\Gamma\{{}^{d}/_{c}\}, \Theta\{{}^{d}/_{c}\} \vdash Q\{{}^{d}/_{c}\} \triangleright \Delta\{{}^{d}/_{c}\}$.

(MQExit) Then, $Q = s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathtt{exit}\langle e, v\rangle\#\mathrm{M}$, $\Delta = s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathtt{exit}\langle e, \mathrm{S}\rangle\#\mathrm{MT}$, $\Gamma \cdot c{:}\mathrm{S}_c \Vdash v{:}\mathrm{S}$, and $\Gamma \cdot c{:}\mathrm{S}_c, \Theta \vdash s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{M} \triangleright s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{MT}$. Because $\Gamma \cdot c{:}\mathrm{S}_c \Vdash v{:}\mathrm{S}$, $s \neq c$. With $\Gamma \Vdash d{:}\mathrm{S}_c$, then $\Gamma \Vdash v\{{}^{d}/_{c}\}{:}\mathrm{S}$ and $\Gamma\{{}^{d}/_{c}\} \Vdash v\{{}^{d}/_{c}\}{:}\mathrm{S}$. By the induction hypothesis, $\Gamma\{{}^{d}/_{c}\}, \Theta\{{}^{d}/_{c}\} \vdash s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{M}\{{}^{d}/_{c}\} \triangleright s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{MT}\{{}^{d}/_{c}\}$. By (MQExit), then $\Gamma\{{}^{d}/_{c}\}, \Theta\{{}^{d}/_{c}\} \vdash Q\{{}^{d}/_{c}\} \triangleright \Delta\{{}^{d}/_{c}\}$. □

Subject reduction tells us that derivatives of well-typed systems are again well-typed. This ensures that our formalism can be used to analyse processes by static type checking. We extend subject reduction such that it provides some information on how the session environment evolves alongside reductions of the system using $\overset{s}{\mapsto}$. In Figure 5 we define the relation $\overset{s}{\mapsto}$ between session environments that emulates the reduction semantics.

For the proof of subject reduction (Theorem 2) we further strengthen its goal and show additionally that there is some s such that Γ, Δ' is coherent and $\Delta \overset{s}{\Mapsto} \Delta'$, i.e., that the session environment evolves by mimicking the respective reduction step and that this emulation reduces the session environment modulo $\overset{s}{\Mapsto}$ w.r.t. a single session s. Moreover we use an additional goal—with weak coherence instead of coherence—to obtain a stronger induction hypothesis for the case of Rule (Par).

Definition 5 (Weak Coherence)**.** The type environments Γ, Δ are *weakly coherent* if there exists some Δ' such that $\Gamma, \Delta \cdot \Delta'$ are coherent.

Ultimately, we are however interested into coherence. Note that obviously the coherent case implies the respective weakly coherent case. Our strengthened goal for subject reduction thus becomes:

$$\Gamma, \Theta \vdash P \rhd \Delta \wedge \Gamma, \Delta \text{ are coherent} \wedge P \longmapsto P' \longrightarrow$$
$$\exists \Delta'.\ \Gamma, \Theta \vdash P' \rhd \Delta' \wedge \Gamma, \Delta' \text{ are coherent} \wedge \Delta \overset{s}{\Mapsto} \Delta'$$
and
$$\Gamma, \Theta \vdash P \rhd \Delta \wedge \Gamma, \Delta \text{ are weakly coherent} \wedge P \longmapsto P' \longrightarrow$$
$$\exists \Delta'.\ \Gamma, \Theta \vdash P' \rhd \Delta' \wedge \Gamma, \Delta' \text{ are weakly coherent} \wedge \Delta \overset{s}{\Mapsto} \Delta'$$

The proof is again similar to the proof in [24] and we highlight the differences in blue colour.

Proof of Theorem 2. The proof is by induction on the reduction $P \longmapsto P'$ that is derived from the rules of Figure 2.

Case of Rule (Init) In this case

$$P = \overline{a}[\mathsf{n}](s).Q_{\mathsf{n}} \mid \prod_{1 \leq \mathsf{i} \leq \mathsf{n}-1} a[\mathsf{i}](s).Q_{\mathsf{i}}$$
$$P' = (\nu s)P''$$
$$P'' = \prod_{1 \leq \mathsf{i} \leq \mathsf{n}} Q_{\mathsf{i}} \mid \prod_{1 \leq \mathsf{i},\mathsf{j} \leq \mathsf{n}, \mathsf{i} \neq \mathsf{j}} s_{\mathsf{i} \to \mathsf{j}}:[]$$

$a \neq s$, and we use alpha conversion to ensure that $s\sharp(\Gamma, \Delta)$. By the typing Rules (Par), (Req), and (Acc), $\Gamma, \Theta \vdash P \rhd \Delta$ implies that there are $G, \Delta_{Q_1}, \ldots, \Delta_{Q_{\mathsf{n}}}$ such that $\Delta = \Delta_{Q_1} \cdot \ldots \cdot \Delta_{Q_{\mathsf{n}}}$, $a{:}G \in \Gamma$, $\Gamma \vdash Q_{\mathsf{i}} \rhd \Delta_{Q_{\mathsf{i}}} \cdot s[\mathsf{i}]{:}G\upharpoonright_{\mathsf{i}}$ for all $1 \leq \mathsf{i} \leq \mathsf{n}$, and $|\mathrm{R}(G)| = \mathsf{n}$. By Rule (MQNil), $\Gamma, \Theta \vdash s_{\mathsf{i}\to\mathsf{j}}{:}[] \rhd s_{\mathsf{i}\to\mathsf{j}}{:}[]$ for all $\mathsf{i},\mathsf{j} \in \mathrm{R}(G)$ with $\mathsf{i} \neq \mathsf{j}$. Since $s\sharp\Delta$, the composition $\Delta \cdot \Delta_s$ for $\Delta_s = \{s[\mathsf{i}]{:}G\upharpoonright_{\mathsf{i}}, s_{\mathsf{i}\to\mathsf{j}}{:}[] \mid \mathsf{i},\mathsf{j} \in \mathrm{R}(G) \wedge \mathsf{i} \neq \mathsf{j}\}$ is defined. By Rule (Par), then $\Gamma, \Theta \vdash P'' \rhd \Delta \cdot \Delta_s$. By Rule (T-Res2), where we use reflexivity to obtain $\Delta_s \overset{s}{\Mapsto} \Delta_s$, then $\Gamma, \Theta \vdash P' \rhd \Delta$.
Since $\Delta' = \Delta$, Γ, Δ' is coherent and, by reflexivity, $\Delta \overset{s}{\Mapsto} \Delta'$.

Case of Rule (RSend) In this case $P = s[\mathsf{r}_1, \mathsf{r}_2]!_{\mathsf{r}}\langle y\rangle.Q \mid s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{M}$, $\mathrm{eval}(y) = v$, and $P' = Q \mid s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{M}\#\langle v\rangle^{\mathsf{r}}$. By the Rules (Par), (RSend), and the typing rules for message queues, $\Gamma, \Theta \vdash P \rhd \Delta$ implies that there are $\Delta_Q, \mathrm{S}, T, \mathrm{MT}$ such that $\Delta = \Delta_Q \cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]!_{\mathsf{r}}\langle \mathrm{S}\rangle.T \cdot s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{MT}$, $\Gamma \Vdash y{:}\mathrm{S}$, $\Gamma, \Theta \vdash Q \rhd \Delta_Q \cdot s[\mathsf{r}_1]{:}T$, and $\Gamma, \Theta \vdash s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{M} \rhd s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{MT}$. By Rule (MQComR), then $\Gamma, \Theta \vdash s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{M}\#\langle y\rangle^{\mathsf{r}} \rhd s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{MT}\#\langle \mathrm{S}\rangle^{\mathsf{r}}$. Since $\Delta_Q \cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]!_{\mathsf{r}}\langle \mathrm{S}\rangle.T \cdot s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{MT}$ is defined, so is $\Delta' = \Delta_Q \cdot s[\mathsf{r}_1]{:}T \cdot s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{MT}\#\langle \mathrm{S}\rangle^{\mathsf{r}}$. By Rule (Par), then $\Gamma, \Theta \vdash P' \rhd \Delta'$.
By Rule (T-RSend) of Figure 5, then $\Delta \overset{s}{\Mapsto} \Delta'$. Since Γ, Δ are coherent and by transitivity of $\overset{s}{\Mapsto}$, then Γ, Δ' are coherent.

Case of Rule (RGet) In this case $P = s[\mathsf{r}_1, \mathsf{r}_2]?_{\mathsf{r}}(x).Q \mid s_{\mathsf{r}_2\to\mathsf{r}_1}{:}\langle v\rangle^{\mathsf{r}}\#\mathrm{M}$, $P' = Q\{v/x\} \mid s_{\mathsf{r}_2\to\mathsf{r}_1}{:}\mathrm{M}$, and we use alpha conversion to ensure that $x\sharp(\Gamma, \Theta, \Delta, s)$. By (Par), (RGet), and the typing rules for message queues, $\Gamma, \Theta \vdash P \rhd \Delta$ implies that there are $\Delta_Q, \mathrm{S}_1, \mathrm{S}_2, T, \mathrm{MT}$ such that $\Delta = \Delta_Q \cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]?_{\mathsf{r}}\langle \mathrm{S}_1\rangle.T \cdot s_{\mathsf{r}_2\to\mathsf{r}_1}{:}\langle \mathrm{S}_2\rangle^{\mathsf{r}}\#\mathrm{MT}$, $\Gamma \cdot x{:}\mathrm{S}_1, \Theta \vdash Q \rhd \Delta_Q \cdot s[\mathsf{r}_1]{:}T$, $\Gamma \Vdash v{:}\mathrm{S}_2$, and $\Gamma, \Theta \vdash s_{\mathsf{r}_2\to\mathsf{r}_1}{:}\mathrm{M} \rhd s_{\mathsf{r}_2\to\mathsf{r}_1}{:}\mathrm{MT}$. Since Γ, Δ are coherent, $\mathrm{S}_1 = \mathrm{S}_2$. By Lemma 5 and because $x\sharp(\Gamma, \Theta, \Delta, s)$, then $\Gamma, \Theta \vdash Q\{v/x\} \rhd \Delta_Q \cdot s[\mathsf{r}_1]{:}T$. Since $\Delta_{Q_j} \cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]?_{\mathsf{r}}\langle \mathrm{S}\rangle.T \cdot s_{\mathsf{r}_2\to\mathsf{r}_1}{:}\langle \mathrm{S}_2\rangle^{\mathsf{r}}\#\mathrm{MT}$ is defined, so is $\Delta' = \Delta_Q \cdot s[\mathsf{r}_1]{:}T \cdot s_{\mathsf{r}_2\to\mathsf{r}_1}{:}\mathrm{MT}$. By Rule (Par), then $\Gamma, \Theta \vdash P' \rhd \Delta'$.
By Rule (T-RGet) of Figure 5, then $\Delta \overset{s}{\Mapsto} \Delta'$. Since Γ, Δ are coherent and by transitivity of $\overset{s}{\Mapsto}$, then Γ, Δ' are coherent.

Case of Rule (USend) Here $P = s[\mathsf{r}_1, \mathsf{r}_2]!_{\mathsf{u}}l\langle y\rangle.Q \mid s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{M}$, $\mathrm{eval}(y) = v$, and $P' = Q \mid s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{M}\#l\langle v\rangle^{\mathsf{u}}$. By the Rules (Par), (USend), and the typing rules for message queues, $\Gamma, \Theta \vdash P \rhd \Delta$ implies that there are $\Delta_Q, l', \mathrm{S}, T, \mathrm{MT}$ such that $\Delta = \Delta_Q \cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]!_{\mathsf{u}}l'\langle \mathrm{S}\rangle.T \cdot s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathrm{MT}$, $l \doteq l'$, $l'{:}\mathrm{S} \in \Gamma$, $\Gamma \Vdash$

y:S, $\Gamma, \Theta \vdash Q \triangleright \Delta_Q \cdot s[\mathsf{r}_1]{:}T$, and $\Gamma, \Theta \vdash s_{\mathsf{r}_1 \to \mathsf{r}_2}{:}\mathrm{M} \triangleright s_{\mathsf{r}_1 \to \mathsf{r}_2}{:}\mathrm{MT}$. By Rule (MQComU), then $\Gamma, \Theta \vdash s_{\mathsf{r}_1 \to \mathsf{r}_2}{:}\mathrm{M}\#l\langle y\rangle^{\mathrm{u}} \triangleright s_{\mathsf{r}_1 \to \mathsf{r}_2}{:}\mathrm{MT}\#l'\langle \mathrm{S}\rangle^{\mathrm{u}}$. Since $\Delta_Q \cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]!_{\mathrm{u}}l'\langle \mathrm{S}\rangle.T \cdot s_{\mathsf{r}_1 \to \mathsf{r}_2}{:}\mathrm{MT}$ is defined, so is $\Delta' = \Delta_Q \cdot s[\mathsf{r}_1]{:}T \cdot s_{\mathsf{r}_1 \to \mathsf{r}_2}{:}\mathrm{MT}\#l'\langle \mathrm{S}\rangle^{\mathrm{u}}$. By Rule (Par), then $\Gamma, \Theta \vdash P' \triangleright \Delta'$.

By Rule (T-USend) of Figure 5, then $\Delta \overset{s}{\mapsto} \Delta'$. Since Γ, Δ are coherent and by transitivity of $\overset{s}{\mapsto}$, then Γ, Δ' are coherent.

Case of Rule (UGet) Here $P = s[\mathsf{r}_1, \mathsf{r}_2]?_{\mathrm{u}}l\langle dv\rangle(x).P \mid s_{\mathsf{r}_2 \to \mathsf{r}_1}{:}l'\langle v\rangle^{\mathrm{u}}\#\mathrm{M}$, $P' = Q\{{}^{v}/_{x}\} \mid s_{\mathsf{r}_2 \to \mathsf{r}_1}{:}\mathrm{M}$, $l \doteq l'$, and (using alpha conversion) $x \sharp (\Gamma, \Theta, \Delta, s)$. By (Par), (UGet), and the typing rules for message queues, $\Gamma, \Theta \vdash P \triangleright \Delta$ implies that there are $\Delta_Q, \mathrm{S}_1, \mathrm{S}_2, T, \mathrm{MT}$ such that $\Delta = \Delta_Q \cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]?_{\mathrm{u}}l''\langle \mathrm{S}_1\rangle.T \cdot s_{\mathsf{r}_2 \to \mathsf{r}_1}{:}l'''\langle \mathrm{S}_2\rangle^{\mathrm{u}}\#\mathrm{MT}$, $l \doteq l''$, $l' \doteq l''$, $l''{:}\mathrm{S}_1 \in \Gamma$, $l'''{:}\mathrm{S}_2 \in \Gamma$, $\Gamma \cdot x{:}\mathrm{S}_1, \Theta \vdash Q \triangleright \Delta_Q \cdot s[\mathsf{r}_1]{:}T$, $\Gamma \Vdash v{:}\mathrm{S}_2$, and $\Gamma, \Theta \vdash s_{\mathsf{r}_2 \to \mathsf{r}_1}{:}\mathrm{M} \triangleright s_{\mathsf{r}_2 \to \mathsf{r}_1}{:}\mathrm{MT}$. Since $l''{:}\mathrm{S}_1 \in \Gamma$, $l'''{:}\mathrm{S}_2 \in \Gamma$, and $l'' \doteq l \doteq l' \doteq l'''$, we have $l'' = l'''$ and $\mathrm{S}_1 = \mathrm{S}_2$. By Lemma 5 and because $x \sharp (\Gamma, \Theta, \Delta, s)$, then $\Gamma, \Theta \vdash Q\{{}^{v}/_{x}\} \triangleright \Delta_Q \cdot s[\mathsf{r}_1]{:}T$. By Rule (Par), then $\Gamma, \Theta \vdash P' \triangleright \Delta'$.

By Rule (T-UGet) of Figure 5, then $\Delta \overset{s}{\mapsto} \Delta'$. Since Γ, Δ are coherent and by transitivity of $\overset{s}{\mapsto}$, then Γ, Δ' are coherent.

Case of Rule (USkip) In this case $P = s[\mathsf{r}_1, \mathsf{r}_2]?_{\mathrm{u}}l\langle dv\rangle(x).P$, $P' = Q\{{}^{dv}/_{x}\}$, and we use alpha conversion to ensure that $x \sharp (\Gamma, \Theta, \Delta, s)$. By (UGet), $\Gamma \vdash P \triangleright \Delta$ implies that there are Δ_Q, S, T such that $\Delta = \Delta_Q \cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]?_{\mathrm{u}}l'\langle \mathrm{S}\rangle.T$, $l \doteq l'$, $l'{:}\mathrm{S} \in \Gamma$, $\Gamma \Vdash dv{:}\mathrm{S}$, and $\Gamma \cdot x{:}\mathrm{S}, \Theta \vdash Q \triangleright \Delta_Q \cdot s[\mathsf{r}_1]{:}T$. By Lemma 5 and because $x \sharp (\Gamma, \Theta, \Delta, s)$, then $\Gamma, \Theta \vdash P' \triangleright \Delta'$ with $\Delta' = \Delta_Q \cdot s[\mathsf{r}_1]{:}T$.

By Rule (T-USkip) of Figure 5, then $\Delta \overset{s}{\mapsto} \Delta'$. Since Γ, Δ are coherent and by transitivity of $\overset{s}{\mapsto}$, then Γ, Δ' are coherent.

Case of Rule (ML) In this case $P = s_{\mathsf{r}_1 \to \mathsf{r}_2}{:}l\langle v\rangle^{\mathrm{u}}\#\mathrm{M}$, $P' = s_{\mathsf{r}_1 \to \mathsf{r}_2}{:}\mathrm{M}$. By the typing rules for message queues, $\Gamma, \Theta \vdash P \triangleright \Delta$ implies that there are S, MT such that $\Delta = s_{\mathsf{r}_1 \to \mathsf{r}_2}{:}l'\langle \mathrm{S}\rangle^{\mathrm{u}}\#\mathrm{MT}$, $l \doteq l'$, $l'{:}\mathrm{S} \in \Gamma$, and $\Gamma, \Theta \vdash P' \triangleright \Delta'$ with $\Delta' = s_{\mathsf{r}_1 \to \mathsf{r}_2}{:}\mathrm{MT}$.

By Rule (T-ML) of Figure 5, then $\Delta \overset{s}{\mapsto} \Delta'$. Since Γ, Δ are coherent and by transitivity of $\overset{s}{\mapsto}$, then Γ, Δ' are coherent.

Case of Rule (RSel) In this case $P = s[\mathsf{r}_1, \mathsf{r}_2]!_{\mathrm{r}}l.Q \mid s_{\mathsf{r}_1 \to \mathsf{r}_2}{:}\mathrm{M}$, and $P' = Q \mid s_{\mathsf{r}_1 \to \mathsf{r}_2}{:}\mathrm{M}\#l^{\mathrm{r}}$. By (Par), (RSel), and the typing rules for message queues, $\Gamma, \Theta \vdash P \triangleright \Delta$ implies that there are $\Delta_Q, \mathrm{I}, j, \mathrm{MT}$ and for all $i \in \mathrm{I}$ there are l_i, T_i such that $\Delta = \Delta_Q \cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]!_{\mathrm{r}}\{l_i.T_i\}_{i \in \mathrm{I}} \cdot s_{\mathsf{r}_1 \to \mathsf{r}_2}{:}\mathrm{MT}$, $j \in \mathrm{I}$, $l \doteq l_j$, $\Gamma, \Theta \vdash Q \triangleright \Delta_Q \cdot s[\mathsf{r}_1]{:}T_j$, and $\Gamma, \Theta \vdash s_{\mathsf{r}_1 \to \mathsf{r}_2}{:}\mathrm{M} \triangleright s_{\mathsf{r}_1 \to \mathsf{r}_2}{:}\mathrm{MT}$. By Rule (MQBranR), then $\Gamma, \Theta \vdash s_{\mathsf{r}_1 \to \mathsf{r}_2}{:}\mathrm{M}\#l^{\mathrm{r}} \triangleright s_{\mathsf{r}_1 \to \mathsf{r}_2}{:}\mathrm{MT}\#l_j^{\mathrm{r}}$. Since $\Delta_Q \cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]!_{\mathrm{r}}\{l_i.T_i\}_{i \in \mathrm{I}} \cdot s_{\mathsf{r}_1 \to \mathsf{r}_2}{:}\mathrm{MT}$ is defined, so is $\Delta' = \Delta_Q \cdot s[\mathsf{r}_1]{:}T_j \cdot s_{\mathsf{r}_1 \to \mathsf{r}_2}{:}\mathrm{MT}\#l_j^{\mathrm{r}}$. By Rule (Par), then $\Gamma, \Theta \vdash P' \triangleright \Delta'$.

By Rule (T-RSel) of Figure 5, then $\Delta \overset{s}{\mapsto} \Delta'$. Since Γ, Δ are coherent and by transitivity of $\overset{s}{\mapsto}$, then Γ, Δ' are coherent.

Case of Rule (RBran) In this case $P = s[\mathsf{r}_1, \mathsf{r}_2]?_{\mathrm{r}}\{l_i.Q_i\}_{i \in \mathrm{I}_1} \mid s_{\mathsf{r}_2 \to \mathsf{r}_1}{:}l^{\mathrm{r}}\#\mathrm{M}$, $j \in \mathrm{I}_1$, $l \doteq l_j$, and $P' = Q_j \mid s_{\mathsf{r}_2 \to \mathsf{r}_1}{:}\mathrm{M}$. By the Rules (Par), (RBran), and the typing rules for message queues, $\Gamma, \Theta \vdash P \triangleright \Delta$ implies that there are $\Delta_Q, \mathrm{I}_2, \mathrm{MT}, l''$ and for all $i \in \mathrm{I}_2$ there are l_i', T_i such that $\Delta = \Delta_Q \cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]?_{\mathrm{r}}\{l_i', T_i\}_{i \in \mathrm{I}_2} \cdot s_{\mathsf{r}_2 \to \mathsf{r}_1}{:}l''^{\mathrm{r}}\#\mathrm{MT}$, $\Gamma, \Theta \vdash s_{\mathsf{r}_2 \to \mathsf{r}_1}{:}\mathrm{M} \triangleright s_{\mathsf{r}_2 \to \mathsf{r}_1}{:}\mathrm{MT}$, $l_j \doteq l''$, and for all $k \in \mathrm{I}_2$ exists some $m \in \mathrm{I}_1$ such that $l_m \doteq l_k'$, $\Gamma, \Theta \vdash Q_m \triangleright \Delta_Q \cdot s[\mathsf{r}_1]{:}T_k$. Since $l \doteq l_j \doteq l''$ and because Γ, Δ are coherent, there is some $n \in \mathrm{I}_2$ such that $l_j \doteq l'' = l_n$ and $\Gamma, \Theta \vdash Q_j \triangleright \Delta_Q \cdot s[\mathsf{r}_1]{:}T_n$. Since $\Delta_Q \cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]?_{\mathrm{r}}\{l_i, T_i\}_{i \in \mathrm{I}} \cdot s_{\mathsf{r}_2 \to \mathsf{r}_1}{:}l''^{\mathrm{r}}\#\mathrm{MT}$ is defined, so is $\Delta' = \Delta_Q \cdot s[\mathsf{r}_1]{:}T_n \cdot s_{\mathsf{r}_2 \to \mathsf{r}_1}{:}\mathrm{MT}$. By Rule (Par), then $\Gamma, \Theta \vdash P' \triangleright \Delta'$.

By Rule (T-RBran) of Figure 5, then $\Delta \overset{s}{\mapsto} \Delta'$. Since Γ, Δ are coherent and by transitivity of $\overset{s}{\mapsto}$, then Γ, Δ' are coherent.

Case of Rule (WSel) In this case $P = s[\mathsf{r}, \mathsf{R}]!_{\mathrm{w}}l.Q \mid \prod_{\mathsf{r}_i \in \mathsf{R}} s_{\mathsf{r} \to \mathsf{r}_i}{:}\mathrm{M}$ and $P' = Q \mid \prod_{\mathsf{r}_i \in \mathsf{R}} s_{\mathsf{r} \to \mathsf{r}_i}{:}\mathrm{M}_i\#l^{\mathrm{w}}$. Let $\mathsf{R} = \{\mathsf{r}_1, \ldots, \mathsf{n}\}$. By the Rules (Par), (WSel), and the typing rules for message queues, $\Gamma, \Theta \vdash$

$P \triangleright \Delta$ implies that there are $\Delta_Q, \mathrm{I}, j, \mathrm{MT}_1, \ldots, \mathrm{MT}_n$ and for all $i \in \mathrm{I}$ there are l_i, T_i such that $\Delta = \Delta_Q \cdot s[\mathsf{r}]{:}[\mathsf{R}]!_{\mathrm{w}}\{l_i, T_i\}_{i\in\mathrm{I}} \cdot s_{\mathsf{r}\to\mathsf{r}_1}{:}\mathrm{MT} \cdot \ldots \cdot s_{\mathsf{r}\to\mathsf{r}_n}{:}\mathrm{MT}$, $j \in \mathrm{I}$, $l \doteq l_j$, $\Gamma, \Theta \vdash Q \triangleright \Delta_Q \cdot s[\mathsf{r}]{:}T_j$, and $\Gamma, \Theta \vdash s_{\mathsf{r}\to\mathsf{r}_1}{:}\mathrm{M}_1 \triangleright s_{\mathsf{r}\to\mathsf{r}_1}{:}\mathrm{MT}_1$, $\ldots$, $\Gamma, \Theta \vdash s_{\mathsf{r}\to\mathsf{r}_n}{:}\mathrm{M}_n \triangleright s_{\mathsf{r}\to\mathsf{r}_n}{:}\mathrm{MT}_n$. By Rule (MQBranW), then $\Gamma, \Theta \vdash s_{\mathsf{r}\to\mathsf{r}_1}{:}\mathrm{M}_1\#l^{\mathrm{w}} \triangleright s_{\mathsf{r}\to\mathsf{r}_1}{:}\mathrm{MT}_1\#l_j^{\mathrm{w}}$, $\ldots$, $\Gamma, \Theta \vdash s_{\mathsf{r}\to\mathsf{r}_n}{:}\mathrm{M}_n\#l^{\mathrm{w}} \triangleright s_{\mathsf{r}\to\mathsf{r}_n}{:}\mathrm{MT}_n\#l_j^{\mathrm{w}}$. Since the session environment $\Delta_Q \cdot s[\mathsf{r}]{:}[\mathsf{R}]!_{\mathrm{w}}\{l_i, T_i\}_{i\in\mathrm{I}} \cdot s_{\mathsf{r}\to\mathsf{r}_1}{:}\mathrm{MT}_1 \cdot \ldots \cdot s_{\mathsf{r}\to\mathsf{r}_n}{:}\mathrm{MT}_n$ is defined, so is $\Delta' = \Delta_Q \cdot s[\mathsf{r}]{:}T_j \cdot s_{\mathsf{r}\to\mathsf{r}_1}{:}\mathrm{MT}_1\#l_j^{\mathrm{w}} \cdot \ldots \cdot s_{\mathsf{r}\to\mathsf{r}_n}{:}\mathrm{MT}_n\#l_j^{\mathrm{w}}$.

By Rule (T-WSel) of Figure 5, then $\Delta \overset{s}{\Mapsto} \Delta'$. Since Γ, Δ are coherent and by transitivity of $\overset{s}{\Mapsto}$, then Γ, Δ' are coherent.

Case of Rule (WBran) Here $P = s[\mathsf{r}_1, \mathsf{r}_2]?_{\mathrm{w}}\{l_i.Q_i\}_{i\in\mathrm{I}_1, l_{\mathrm{d}}} \mid s_{\mathsf{r}_2\to\mathsf{r}_1}{:}l^{\mathrm{w}}\#\mathrm{M}$, $j \in \mathrm{I}_1$, $l \doteq l_j$, and $P' = Q_j \mid s_{\mathsf{r}_2\to\mathsf{r}_1}{:}\mathrm{M}$. By the Rules (Par), (WBran), and the typing rules for message queues, $\Gamma, \Theta \vdash P \triangleright \Delta$ implies that there are $\Delta_Q, l'_{\mathrm{d}}, \mathrm{I}_2, \mathrm{MT}, l''$ and for all $i \in \mathrm{I}_2$ there are l'_i, T_i such that $\Delta = \Delta_Q \cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]?_{\mathrm{w}}\{l'_i, T_i\}_{i\in\mathrm{I}_2, l'_{\mathrm{d}}} \cdot s_{\mathsf{r}_2\to\mathsf{r}_1}{:}l''^{\mathrm{w}}\#\mathrm{MT}$, $l_{\mathrm{d}} \doteq l'_{\mathrm{d}}$, $\Gamma, \Theta \vdash s_{\mathsf{r}_2\to\mathsf{r}_1}{:}\mathrm{M} \triangleright s_{\mathsf{r}_2\to\mathsf{r}_1}{:}\mathrm{MT}$, $l_j \doteq l''$, and for all $k \in \mathrm{I}_2$ exists some $m \in \mathrm{I}_1$ such that $l_m \doteq l'_k$, $\Gamma, \Theta \vdash Q_m \triangleright \Delta_Q \cdot s[\mathsf{r}_1]{:}T_k$. Since $l \doteq l_j \doteq l''$ and because Γ, Δ are coherent, there is some $n \in \mathrm{I}_2$ such that $l_j \doteq l'' = l_n$ and $\Gamma, \Theta \vdash Q_j \triangleright \Delta_Q \cdot s[\mathsf{r}_1]{:}T_n$. Since $\Delta_Q \cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]?_{\mathrm{w}}\{l_i, T_i\}_{i\in\mathrm{I}, l'_{\mathrm{d}}} \cdot s_{\mathsf{r}_2\to\mathsf{r}_1}{:}l''^{\mathrm{w}}\#\mathrm{MT}$ is defined, so is $\Delta' = \Delta_Q \cdot s[\mathsf{r}_1]{:}T_n \cdot s_{\mathsf{r}_2\to\mathsf{r}_1}{:}\mathrm{MT}$. By Rule (Par), then $\Gamma, \Theta \vdash P' \triangleright \Delta'$.

By Rule (T-WBran) of Figure 5, then $\Delta \overset{s}{\Mapsto} \Delta'$. Since Γ, Δ are coherent and by transitivity of $\overset{s}{\Mapsto}$, then Γ, Δ' are coherent.

Case of Rule (WSkip) In this case $P = s[\mathsf{r}_1, \mathsf{r}_2]?_{\mathrm{w}}\{l_i.Q_i\}_{i\in\mathrm{I}_1, l_{\mathrm{d}}}$ and $P' = Q_{\mathrm{d}}$. By the Rule (WSkip), $\Gamma, \Theta \vdash P \triangleright \Delta$ implies that there are $\Delta_Q, l'_{\mathrm{d}}, \mathrm{I}_2$ and for all $i \in \mathrm{I}_2$ there are l'_i, T_i such that $\Delta = \Delta_Q \cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]?_{\mathrm{w}}\{l'_i, T_i\}_{i\in\mathrm{I}_2, l'_{\mathrm{d}}}$, $l_{\mathrm{d}} \doteq l'_{\mathrm{d}}$, and for all $k \in \mathrm{I}_2$ exists some $m \in \mathrm{I}_1$ such that $l_m \doteq l'_k$, $\Gamma, \Theta \vdash Q_m \triangleright \Delta_Q \cdot s[\mathsf{r}_1]{:}T_k$. Since $l_{\mathrm{d}} \doteq l'_{\mathrm{d}}$, there is some $n \in \mathrm{I}_2$ such that $l_{\mathrm{d}} \doteq l'_{\mathrm{d}} = l_n$ and $\Gamma, \Theta \vdash P' \triangleright \Delta'$ with $\Delta' = \Delta_Q \cdot s[\mathsf{r}_1]{:}T_{\mathrm{d}}$.

By Rule (T-WSkip) of Figure 5, then $\Delta \overset{s}{\Mapsto} \Delta'$. Since Γ, Δ are coherent and by transitivity of $\overset{s}{\Mapsto}$, then Γ, Δ' are coherent.

Case of Rule (LStep) In this case

$$P = s[\mathsf{r}, \mathsf{R}]\infty_e^{\mathsf{c}=n}[(x).Q_0]Q_1; (y).Q_2 \mid Q,$$
$$P' = s[\mathsf{r}, \mathsf{R}]\infty_e^{\mathsf{c}=n}[(x).Q_0]Q'_1; (y).Q_2 \mid Q',$$

$Q_1 \mid Q \longmapsto Q'_1 \mid Q'$, and $\mathtt{onlyMQ}_{\mathsf{r}\leftrightarrow\mathsf{R}}(Q, Q')$. By Rules (Par), (Loop), and the typing rules for message queues, $\Gamma, \Theta \vdash P \triangleright \Delta$ implies that there are $\Delta_Q, \Delta_{Q_2}, \mathrm{S}_0, \mathrm{S}_2, T_0, T_1, T_2$ such that $\Delta = \Delta_{Q_2} \cdot s[\mathsf{r}]{:}[\mathsf{R}]\infty_e^{\mathsf{c}=n}[\langle \mathrm{S}_0\rangle.T_0]T_1; \langle \mathrm{S}_2\rangle.T_2 \cdot \Delta_Q$, $x, y \sharp (\Gamma, \Delta_{Q_2}, s, e, \mathsf{c}, T_0, T_1, T_2)$, $\Gamma \Vdash n{:}\mathbb{N}$, $\mathrm{unr}(T_0)$, $\mathrm{unr}(T_1)$,

$$\Gamma \cdot x{:}\mathrm{S}_0 \cdot \mathsf{c}{:}\mathbb{N}, e{:}s[\mathsf{r}]\langle \mathrm{S}_0, \mathrm{S}_2\rangle \vdash Q_0 \triangleright s[\mathsf{r}]{:}T_0,$$
$$\Gamma \cdot \mathsf{c}{:}\mathbb{N}, e{:}s[\mathsf{r}]\langle \mathrm{S}_0, \mathrm{S}_2\rangle \vdash Q_1 \triangleright s[\mathsf{r}]{:}T_1,$$
$$\Gamma \cdot y{:}\mathrm{S}, \Theta \vdash Q_2 \triangleright \Delta_{Q_2} \cdot s[\mathsf{r}]{:}T_2, \text{ and } \qquad \Gamma, \Theta \vdash Q \triangleright \Delta_Q.$$

By Rule (Par), then $\Gamma \cdot \mathsf{c}{:}\mathbb{N}, \Theta \cdot e{:}s[\mathsf{r}]\langle \mathrm{S}_0, \mathrm{S}_2\rangle \vdash Q_1 \mid Q \triangleright s[\mathsf{r}]{:}T_1 \cdot \Delta_Q$. Since Γ, Δ are coherent, $\Gamma \cdot \mathsf{c}{:}\mathbb{N}, \Theta \cdot e{:}s[\mathsf{r}]\langle \mathrm{S}_0, \mathrm{S}_2\rangle$ are weakly coherent. By the induction hypothesis (for the weakly coherent case), $\Gamma \cdot \mathsf{c}{:}\mathbb{N}, \Theta \cdot e{:}s[\mathsf{r}]\langle \mathrm{S}_0, \mathrm{S}_2\rangle \vdash Q'_1 \mid Q' \triangleright \Delta'_1$ for some Δ'_1 with $s[\mathsf{r}]{:}T_1 \cdot \Delta_Q \overset{s}{\Mapsto} \Delta'_1$. By Rule (Par) and global weakening, then there are T'_1 and Δ'_Q such that $\Delta'_1 = s[\mathsf{r}]{:}T'_1 \cdot \Delta'_Q$, $\Gamma \cdot \mathsf{c}{:}\mathbb{N}, e{:}s[\mathsf{r}]\langle \mathrm{S}_0, \mathrm{S}_2\rangle \vdash Q'_1 \triangleright s[\mathsf{r}]{:}T'_1$ and $\Gamma, \Theta \vdash Q' \triangleright \Delta'_Q$. By Rule (Loop), then $\Gamma, \Theta \vdash s[\mathsf{r}, \mathsf{R}]\infty_e^{\mathsf{c}=n}[(x).Q_0]Q'_1; (y).Q_2 \triangleright \Delta_{Q_2} \cdot s[\mathsf{r}]{:}[\mathsf{R}]\infty_e^{\mathsf{c}=n}[\langle \mathrm{S}_0\rangle.T_0]T'_1; \langle \mathrm{S}_2\rangle.T_2$. By Rule (Par), then $\Gamma, \Theta \vdash P' \triangleright \Delta'$ with the resulting session

environment $\Delta' = \Delta_{Q_2} \cdot s[\mathsf{r}]{:}[\mathsf{R}]\infty_e^{\mathsf{c}=n}[\langle \mathrm{S}_0\rangle.T_0]T_1'; \langle \mathrm{S}_2\rangle.T_2 \cdot \Delta_Q'$.
Since $s[\mathsf{r}]{:}T_1 \cdot \Delta_Q \overset{s}{\Mapsto} s[\mathsf{r}]{:}T_1' \cdot \Delta_Q'$ and by (T-LStep), then $\Delta \overset{s}{\Mapsto} \Delta'$. Since Γ, Δ are coherent and by transitivity of $\overset{s}{\Mapsto}$, then Γ, Δ' are coherent.

Case of Rule (LCall) In this case

$$P = s[\mathsf{r},\mathsf{R}]\infty_e^{\mathsf{c}=n}[(x).Q_0]\mathtt{call}\langle e_l, e_v\rangle; (y).Q_2,$$
$$P' = s[\mathsf{r},\mathsf{R}]\infty_e^{\mathsf{c}=\mathrm{eval}(n+1)}[(x).Q_0]\,(Q_0\{{}^{n}/_{\mathsf{c}}\})\,\{{}^{v}/_{x}\}; (y).Q_2,$$

$\mathrm{eval}(e) = \mathrm{eval}(e_l)$, and $\mathrm{eval}(e_v) = v$. By the Rules (Loop) and (Call), $\Gamma, \Theta \vdash P \triangleright \Delta$ implies that there are some $\Delta_{Q_2}, \mathrm{S}_0, \mathrm{S}_2, T_0, T_1, T_2$ such that $\Delta = \Delta_{Q_2} \cdot s[\mathsf{r}]{:}[\mathsf{R}]\infty_e^{\mathsf{c}=n}[\langle \mathrm{S}_0\rangle.T_0]\mathtt{call}\langle e\rangle; \langle \mathrm{S}_2\rangle.T_2$, the variables x, y, c are fresh, i.e., $x, y \sharp (\Gamma, \Delta_{Q_2}, s, e, \mathsf{c}, T_0, T_2)$, $\Gamma \Vdash n{:}\mathbb{N}$, $\mathrm{unr}(T_0)$,

$$\Gamma \cdot x{:}\mathrm{S}_0 \cdot \mathsf{c}{:}\mathbb{N}, e{:}s[\mathsf{r}]\langle \mathrm{S}_0, \mathrm{S}_2\rangle \vdash Q_0 \triangleright s[\mathsf{r}]{:}T_0,$$
$$\Gamma \cdot \mathsf{c}{:}\mathbb{N}, e{:}s[\mathsf{r}]\langle \mathrm{S}_0, \mathrm{S}_2\rangle \vdash \mathtt{call}\langle e_l, e_v\rangle \triangleright s[\mathsf{r}]{:}\mathtt{call}\langle e\rangle,$$
$$\Gamma \cdot y{:}\mathrm{S}, \Theta \vdash Q_2 \triangleright \Delta_{Q_2} \cdot s[\mathsf{r}]{:}T_2,$$

and $\Gamma \Vdash e_v{:}\mathrm{S}_0$. Since $\mathrm{eval}(e_v) = v$, then $\Gamma \Vdash v{:}\mathrm{S}_0$. By Lemma 5 and $x \sharp (\Gamma, \Delta_{Q_2}, s, e, \mathsf{c}, T_0, T_2)$, then $\Gamma, e{:}s[\mathsf{r}]\langle \mathrm{S}_0, \mathrm{S}_2\rangle \vdash (Q_0\{{}^{n}/_{\mathsf{c}}\})\,\{{}^{v}/_{x}\} \triangleright s[\mathsf{r}]{:}T_0\{{}^{n}/_{\mathsf{c}}\}$. By Rule (Loop), then $\Gamma, \Theta \vdash P' \triangleright \Delta'$ with $\Delta' = \Delta_{Q_2} \cdot s[\mathsf{r}]{:}[\mathsf{R}]\infty_e^{\mathsf{c}=\mathrm{eval}(n+1)}[\langle \mathrm{S}_0\rangle.T_0]T_0\{{}^{n}/_{\mathsf{c}}\}; \langle \mathrm{S}_2\rangle.T_2$.
By Rule (T-Call) of Figure 5, then $\Delta \overset{s}{\Mapsto} \Delta'$. Since Γ, Δ are coherent and by transitivity of $\overset{s}{\Mapsto}$, then Γ, Δ' are coherent.

Case of Rule (LExitS) In this case $P = s[\mathsf{r},\mathsf{R}]\infty_e^{\mathsf{c}=n}[(x).Q_0]\mathtt{exit}\langle e_l, e_v\rangle; (y).Q_2 \mid \prod_{\mathsf{r}_i \in \mathsf{R}} s_{\mathsf{r} \to \mathsf{r}_i}{:}\mathrm{M}_i$, $P' = Q_2\{{}^{v}/_{y}\} \mid \prod_{\mathsf{r}_i \in \mathsf{R}} s_{\mathsf{r} \to \mathsf{r}_i}{:}\mathrm{M}_i\#\mathtt{exit}\langle id, v\rangle$, $\mathrm{eval}(e) = \mathrm{eval}(e_l) = id$, and $\mathrm{eval}(e_v) = v$. Let $\mathsf{R} = \{\mathsf{r}_1, \ldots, \mathsf{r}_n\}$. By the Rules (Par), (Loop), (Exit), and the typing rules for message queues, $\Gamma, \Theta \vdash P \triangleright \Delta$ implies that there are $\Delta_{Q_2}, \mathrm{S}, \mathrm{S}_0, T_0, T_1, T_2, \mathrm{MT}_1, \ldots \mathrm{MT}_n$ such that $\Delta = \Delta_{Q_2} \cdot s[\mathsf{r}]{:}[\mathsf{R}]\infty_e^{\mathsf{c}=n}[\langle \mathrm{S}_0\rangle.T_0]T_1; \langle \mathrm{S}_2\rangle.T_2 \cdot \prod_{\mathsf{r}_i \in \mathsf{R}} s_{\mathsf{r} \to \mathsf{r}_i}{:}\mathrm{MT}_i$, we have $x, y \sharp (\Gamma, \Delta_{Q_2}, s, e, \mathsf{c}, T_0, T_1, T_2)$, $\Gamma \Vdash n{:}\mathbb{N}$, $\mathrm{unr}(T_0)$, $\mathrm{unr}(T_1)$,

$$\Gamma \cdot x{:}\mathrm{S}_0 \cdot \mathsf{c}{:}\mathbb{N}, e{:}s[\mathsf{r}]\langle \mathrm{S}_0, \mathrm{S}_2\rangle \vdash Q_0 \triangleright s[\mathsf{r}]{:}T_0,$$
$$\Gamma \cdot \mathsf{c}{:}\mathbb{N}, e{:}s[\mathsf{r}]\langle \mathrm{S}_0, \mathrm{S}_2\rangle \vdash \mathtt{exit}\langle e_l, e_v\rangle \triangleright s[\mathsf{r}]{:}T_1,$$
$$\Gamma \cdot y{:}\mathrm{S}, \Theta \vdash Q_2 \triangleright \Delta_{Q_2} \cdot s[\mathsf{r}]{:}T_2,$$

$\Gamma \Vdash e_v{:}\mathrm{S}_2$, and $\Gamma \vdash s_{\mathsf{r} \to \mathsf{r}_i}{:}\mathrm{M}_i \triangleright s_{\mathsf{r} \to \mathsf{r}_i}{:}\mathrm{MT}_i$ for all $\mathsf{r}_i \in \mathsf{R}$. Since $\mathrm{eval}(e_v) = v$, then $\Gamma \Vdash v{:}\mathrm{S}_0$. By Lemma 5 and because $y \sharp (\Gamma, \Delta_{Q_2}, s, e, \mathsf{c}, T_0, T_1, T_2)$, then $\Gamma, \Theta \vdash Q_2\{{}^{v}/_{y}\} \triangleright \Delta_{Q_2} \cdot s[\mathsf{r}]{:}T_2$. By Rule (MQExit), $\Gamma, \Theta \vdash s_{\mathsf{r} \to \mathsf{r}_i}{:}\mathtt{exit}\langle l, v\rangle\#\mathrm{M}_i \triangleright s_{\mathsf{r} \to \mathsf{r}_i}{:}\mathtt{exit}\langle l, \mathrm{S}\rangle\#\mathrm{MT}_i$ for all $\mathsf{r}_i \in \mathsf{R}$. Since the session environment $\Delta = \Delta_{Q_2} \cdot s[\mathsf{r}]{:}[\mathsf{R}]\infty_e^{\mathsf{c}=n}[\langle \mathrm{S}_0\rangle.T_0]T_1; \langle \mathrm{S}_2\rangle.T_2 \cdot \prod_{\mathsf{r}_i \in \mathsf{R}} s_{\mathsf{r} \to \mathsf{r}_i}{:}\mathrm{MT}_i$ is defined, so is the session environment $\Delta' = \Delta_{Q_2} \cdot s[\mathsf{r}]{:}T_2 \cdot \prod_{\mathsf{r}_i \in \mathsf{R}} s_{\mathsf{r} \to \mathsf{r}_i}{:}\mathtt{exit}\langle l, \mathrm{S}\rangle\#\mathrm{MT}_i$. By Rule (Par), then $\Gamma, \Theta \vdash P' \triangleright \Delta'$.
By Rule (T-LExitS), then $\Delta \overset{s}{\Mapsto} \Delta'$. Since Γ, Δ are coherent and by transitivity of $\overset{s}{\Mapsto}$, then Γ, Δ' are coherent.

Case of Rule (LExitG) In this case

$$P = s[\mathsf{r},\mathsf{R}]\infty_e^{\mathsf{c}=n}[(x).Q_0]Q_1; (y).Q_2 \mid s_{\mathsf{r}_e \to \mathsf{r}}{:}\mathtt{exit}\langle id, v\rangle\#\mathrm{M},$$
$$P' = Q_2\{{}^{v}/_{y}\} \mid s_{\mathsf{r}_e \to \mathsf{r}}{:}\mathrm{M},$$

$\mathrm{eval}(e) = id$, and $\mathsf{r}_e \in \mathsf{R}$. By Rules (Par), (Loop), (MQExit), and the typing rules for message queues, the judgement $\Gamma, \Theta \vdash P \triangleright \Delta$ implies that there are $\Delta_Q, \Delta_{Q_2}, \mathrm{S}_0, \mathrm{S}_2, T_0, T_1, T_2, \mathrm{MT}$ such that $\Delta =$

$\Delta_{Q_2} \cdot s[\mathsf{r}]{:}[\mathsf{R}]\infty_e^{\mathsf{c}=n}[\langle \mathsf{S}_0\rangle.T_0]T_1; \langle \mathsf{S}_2\rangle.T_2 \cdot s_{\mathsf{r}_e \to \mathsf{r}}{:}\mathtt{exit}\langle e, \mathsf{S}_2\rangle\#\mathrm{MT}$, we have $x, y \sharp (\Gamma, \Delta_{Q_2}, s, e, \mathsf{c}, T_0, T_1, T_2)$, $\Gamma \Vdash n{:}\mathbb{N}$, $\mathrm{unr}(T_0)$, $\mathrm{unr}(T_1)$,

$$\Gamma \cdot x{:}\mathsf{S}_0 \cdot \mathsf{c}{:}\mathbb{N}, e{:}s[\mathsf{r}]\langle \mathsf{S}_0, \mathsf{S}_2\rangle \vdash Q_0 \rhd s[\mathsf{r}]{:}T_0,$$
$$\Gamma \cdot \mathsf{c}{:}\mathbb{N}, e{:}s[\mathsf{r}]\langle \mathsf{S}_0, \mathsf{S}_2\rangle \vdash Q_1 \rhd s[\mathsf{r}]{:}T_1,$$
$$\Gamma \cdot y{:}\mathsf{S}, \Theta \vdash Q_2 \rhd \Delta_{Q_2} \cdot s[\mathsf{r}]{:}T_2, \qquad \Gamma, \Theta \vdash s_{\mathsf{r}_e \to \mathsf{r}}{:}\mathrm{M} \rhd s_{\mathsf{r}_e \to \mathsf{r}}{:}\mathrm{MT},$$

and $\Gamma \Vdash v{:}\mathsf{S}_2$. By Lemma 5 and because $y \sharp (\Gamma, \Delta_{Q_2}, s, e, \mathsf{c}, T_0, T_1, T_2)$, then $\Gamma, \Theta \vdash Q_2\{^{v}/_{y}\} \rhd \Delta_{Q_2} \cdot s[\mathsf{r}]{:}T_2$. Since $\Delta_{Q_2} \cdot s[\mathsf{r}]{:}[\mathsf{R}]\infty_e^{\mathsf{c}=n}[\langle \mathsf{S}_0\rangle.T_0]T_1; \langle \mathsf{S}_2\rangle.T_2 \cdot s_{\mathsf{r}_e \to \mathsf{r}}{:}\mathtt{exit}\langle e, \mathsf{S}_2\rangle\#\mathrm{MT}$ is defined, so is $\Delta' = \Delta_{Q_2} \cdot s[\mathsf{r}]{:}T_2 \cdot s_{\mathsf{r}_e \to \mathsf{r}}{:}\mathrm{MT}$. By Rule (Par), then $\Gamma, \Theta \vdash P' \rhd \Delta'$.
By Rule (T-LExitG), then $\Delta \overset{s}{\Mapsto} \Delta'$. Since Γ, Δ are coherent and by transitivity of $\overset{s}{\Mapsto}$, then Γ, Δ' are coherent.

Case of Rule (EDrop) In this case $P = s_{\mathsf{r}_2 \to \mathsf{r}_1}{:}\mathtt{exit}\langle id, v\rangle\#\mathrm{M}$ and $P' = s_{\mathsf{r}_2 \to \mathsf{r}_1}{:}\mathrm{M}$. By the typing rules for message queues, $\Gamma, \Theta \vdash P \rhd \Delta$ implies that there are S, MT such that $\Delta = s_{\mathsf{r}_2 \to \mathsf{r}_1}{:}\mathtt{exit}\langle e, \mathsf{S}\rangle\#\mathrm{MT}$, $\mathrm{eval}(e) = id$, and $\Gamma, \Theta \vdash P' \rhd \Delta'$ with $\Delta' = s_{\mathsf{r}_2 \to \mathsf{r}_1}{:}\mathrm{MT}$.
By Rule (T-EDrop) of Figure 5, then $\Delta \overset{s}{\Mapsto} \Delta'$. Since Γ, Δ are coherent and by transitivity of $\overset{s}{\Mapsto}$, then Γ, Δ' are coherent.

Case of Rule (Crash) In this case $\mathtt{FP}_{\mathtt{crash}}$ and $P' = \bot$. By the typing rules and Condition 1.1, $\Gamma, \Theta \vdash P \rhd \Delta$ and $\mathtt{FP}_{\mathtt{crash}}$ imply $\mathrm{nsr}(\Delta)$. By Rule (Crash), then $\Gamma, \Theta \vdash \bot \rhd \Delta$.
Since $\Delta' = \Delta$, Γ, Δ' is coherent and, by reflexivity, $\Delta \overset{s}{\Mapsto} \Delta'$.

Case of Rule (If-T) In this case $P = \mathtt{if}\ e\ \mathtt{then}\ Q\ \mathtt{else}\ Q'$ and $P' = Q$. By Rule (If), $\Gamma, \Theta \vdash P \rhd \Delta$ implies $\Gamma, \Theta \vdash P' \rhd \Delta$.
Since $\Delta' = \Delta$, Γ, Δ' is coherent and, by reflexivity, $\Delta \overset{s}{\Mapsto} \Delta'$.

Case of Rule (If-F) In this case $P = \mathtt{if}\ e\ \mathtt{then}\ Q\ \mathtt{else}\ Q'$ and $P' = Q'$. By Rule (If), $\Gamma, \Theta \vdash P \rhd \Delta$ implies $\Gamma, \Theta \vdash P' \rhd \Delta$.
Since $\Delta' = \Delta$, Γ, Δ' is coherent and, by reflexivity, $\Delta \overset{s}{\Mapsto} \Delta'$.

Case of Rule (Deleg) In this case $P = s[\mathsf{r}_1, \mathsf{r}_2]!\langle\langle s'[\mathsf{r}]\rangle\rangle.Q \mid s_{\mathsf{r}_1 \to \mathsf{r}_2}{:}\mathrm{M}$ and $P' = Q \mid s_{\mathsf{r}_1 \to \mathsf{r}_2}{:}\mathrm{M}\#s'[\mathsf{r}]$. By the Rules (Par), (Deleg), and the typing rules for message queues, $\Gamma, \Theta \vdash P \rhd \Delta$ implies that there are $\Delta_Q, T, T', \mathrm{MT}$ such that $\Delta = \Delta_Q \cdot s[\mathsf{r}_1]{:}[\mathsf{r}2]!\langle s'[\mathsf{r}]{:}T'\rangle.T \cdot s'[\mathsf{r}]{:}T' \cdot s_{\mathsf{r}_1 \to \mathsf{r}_2}{:}\mathrm{MT}$, $\Gamma, \Theta \vdash Q \rhd \Delta_Q \cdot s[\mathsf{r}_1]{:}T$, and $\Gamma, \Theta \vdash s_{\mathsf{r}_1 \to \mathsf{r}_s}{:}\mathrm{MT} \rhd s_{\mathsf{r}_1 \to \mathsf{r}_2}{:}\mathrm{MT}$. By Rule (MQDeleg), then $\Gamma, \Theta \vdash s_{\mathsf{r}_1 \to \mathsf{r}_s}{:}\mathrm{MT}\#s'[\mathsf{r}] \rhd s_{\mathsf{r}_1 \to \mathsf{r}_2}{:}\mathrm{MT}\#s'[\mathsf{r}]$. Since $\Delta = \Delta_Q \cdot s[\mathsf{r}_1]{:}[\mathsf{r}2]!\langle s'[\mathsf{r}]{:}T'\rangle.T \cdot s'[\mathsf{r}]{:}T' \cdot s_{\mathsf{r}_1 \to \mathsf{r}_2}{:}\mathrm{MT}$ is defined, so is $\Delta' = \Delta_Q \cdot s[\mathsf{r}_1]{:}T \cdot s_{\mathsf{r}_1 \to \mathsf{r}_2}{:}\mathrm{MT}\#s'[\mathsf{r}]$. By Rule (Par), then $\Gamma, \Theta \vdash P' \rhd \Delta'$.
By Rule (T-Deleg) of Figure 5, then $\Delta \overset{s}{\Mapsto} \Delta'$. Since Γ, Δ are coherent and by transitivity of $\overset{s}{\Mapsto}$, then Γ, Δ' are coherent.

Case of Rule (SRecv) In this case $P = s[\mathsf{r}_1, \mathsf{r}_2]?((s'[\mathsf{r}])).Q \mid s_{\mathsf{r}_2 \to \mathsf{r}_1}{:}s''[\mathsf{r}']\#\mathrm{M}$ and $P' = Q\{^{s''}/_{s'}\}\{^{\mathsf{r}'}/_{\mathsf{r}}\} \mid \mathrm{M}$. We use alpha conversion to ensure that $s' = s''$ and $\mathsf{r} = \mathsf{r}''$. By the Rules (Par), (SRecv), and the typing rules for message queues, $\Gamma, \Theta \vdash P \rhd \Delta$ implies that there are $\Delta_Q, T, T', \mathrm{MT}$ such that $\Delta = \Delta_Q \cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]?\langle s'[\mathsf{r}]{:}T'\rangle.T \cdot s_{\mathsf{r}_2 \to \mathsf{r}_1}{:}s'[\mathsf{r}]\#\mathrm{MT}$, $\Gamma, \Theta \vdash Q \rhd \Delta_Q \cdot s[\mathsf{r}_1]{:}T \cdot s'[\mathsf{r}]{:}T'$, and $\Gamma, \Theta \vdash s_{\mathsf{r}_2 \to \mathsf{r}_1}{:}\mathrm{M} \rhd s_{\mathsf{r}_2 \to \mathsf{r}_1}{:}\mathrm{MT}$. Since $\Delta = \Delta_Q \cdot s[\mathsf{r}_1]{:}[\mathsf{r}_2]?\langle s'[\mathsf{r}]{:}T'\rangle.T \cdot s_{\mathsf{r}_2 \to \mathsf{r}_1}{:}s''[\mathsf{r}']\#\mathrm{MT}$ is defined, so is $\Delta' = \Delta_Q \cdot s[\mathsf{r}_1]{:}T \cdot s'[\mathsf{r}]{:}T' \cdot s_{\mathsf{r}_2 \to \mathsf{r}_1}{:}\mathrm{MT}$. By Rule (Par), then $\Gamma, \Theta \vdash P' \rhd \Delta'$.
By Rule (T-SRecv) of Figure 5, then $\Delta \overset{s}{\Mapsto} \Delta'$. Since Γ, Δ are coherent and by transitivity of $\overset{s}{\Mapsto}$, then Γ, Δ' are coherent.

Case of Rule (Par) In this case $P = Q_1 \mid Q_2$, $Q_1 \longmapsto Q_1'$, and $P' = Q_1' \mid Q_2$. By Rule (Par), $\Gamma, \Theta \vdash P \triangleright \Delta$ implies that there are $\Delta_{Q_1}, \Delta_{Q_2}$ such that $\Delta = \Delta_{Q_1} \cdot \Delta_{Q_2}$, $\Gamma, \Theta \vdash Q_1 \triangleright \Delta_{Q_1}$, and $\Gamma, \Theta \vdash Q_2 \triangleright \Delta_{Q_2}$. Since Γ, Δ are coherent, Γ, Δ_{Q_1} is weakly coherent. By the induction hypothesis (for the weakly coherent case), $\Gamma, \Theta \vdash Q_1' \triangleright \Delta_{Q_1}'$ with $\Delta_{Q_1} \overset{s}{\mapsto} \Delta_{Q_1}'$. Since $\Delta_{Q_1} \cdot \Delta_{Q_2}$ is defined, so is $\Delta' = \Delta_{Q_1} \cdot \Delta_{Q_2}$. By Rule (Par), then $\Gamma, \Theta \vdash P' \triangleright \Delta'$.
By $\Delta_{Q_1} \overset{s}{\mapsto} \Delta_{Q_1}'$, then $\Delta \overset{s}{\mapsto} \Delta'$. Since Γ, Δ are coherent and by transitivity of $\overset{s}{\mapsto}$, then Γ, Δ' are coherent.

Case of Rule (Res) In this case $P = (\nu x)Q$, $Q \longmapsto Q'$, and $P' = (\nu x)Q'$. Then, one of the Rules (Res1) or (Res2) was used to type the restriction on x in P.

Case of (Res1) Then there is some S such that $x \sharp (\Gamma, \Delta)$ and $\Gamma \cdot x{:}\mathsf{S}, \Theta \vdash Q \triangleright \Delta$. Since Γ, Δ are coherent, so are $\Gamma \cdot x{:}\mathsf{S}, \Delta$. By the induction hypothesis, $\Gamma \cdot x{:}\mathsf{S}, \Theta \vdash Q' \triangleright \Delta'$ for some Δ' such that Γ, Δ' is coherent and $\Delta \overset{s}{\mapsto} \Delta'$. With Rule (Res1), then $\Gamma, \Theta \vdash P' \triangleright \Delta'$.

Case of (Res2) In this case $x = s$ and there are G, a, Δ'' such that $s \sharp (\Gamma, \Delta)$, $\{ s[\mathsf{r}]{:}G{\restriction_{\mathsf{r}}} \mid \mathsf{r} \in \mathrm{R}(G) \} \cdot \{ s_{\mathsf{r} \to \mathsf{r}'}{:}[\,] \mid \mathsf{r}, \mathsf{r}' \in \mathrm{R}(G') \wedge \mathsf{r} \neq \mathsf{r}' \} \overset{s}{\mapsto} \Delta''$, $a{:}G \in \Gamma$, and $\Gamma, \Theta \vdash Q \triangleright \Delta \cdot \Delta''$. Then $\Gamma, \Delta \cdot \Delta''$ are coherent. By the induction hypothesis, $\Gamma, \Theta \vdash Q' \triangleright \Delta'''$ for some Δ''' such that $\Delta \cdot \Delta'' \overset{s}{\mapsto} \Delta'''$. By Rule (Res2), then $\Gamma, \Theta \vdash P' \triangleright \Delta$. Since $\Delta' = \Delta$, Γ, Δ' is coherent and, by reflexivity, $\Delta \overset{s}{\mapsto} \Delta'$.

Case of (Rec) In this case $P = (\mu X, \mathsf{c} = n)Q$ and $P' = (Q\{^{n}/_{\mathsf{c}}\})\{^{(\mu X, \mathsf{c}=\mathrm{eval}(n+1))Q}/_{X}\}$. We use alpha conversion to ensure that $\mathsf{c} \sharp (\Gamma, \Theta, \Delta)$. By Rule (Rec), then there are $\Delta_Q, s, \mathsf{r}, t, T$ such that $\Delta = \Delta_Q \cdot s[\mathsf{r}]{:}(\mu t, \mathsf{c} = n)T$, $\Gamma \Vdash n{:}\mathbb{N}$, and $\Gamma \cdot \mathsf{c}{:}\mathbb{N}, \Theta \cdot X{:}s[\mathsf{r}]{:}t \vdash Q \triangleright \Delta_Q \cdot s[\mathsf{r}]{:}T$. Since $\mathsf{c} \sharp (\Gamma, \Theta, \Delta)$ and because of Lemma 5, then $\Gamma, \Theta \cdot X{:}s[\mathsf{r}]{:}t \vdash Q\{^{n}/_{\mathsf{c}}\} \triangleright \Delta_Q \cdot s[\mathsf{r}]{:}T\{^{n}/_{\mathsf{c}}\}$. By replacing in the proof tree of $\Gamma, \Theta \cdot X{:}s[\mathsf{r}]{:}t \vdash Q\{^{n}/_{\mathsf{c}}\} \triangleright \Delta_Q \cdot s[\mathsf{r}]{:}T\{^{n}/_{\mathsf{c}}\}$ all occurrences of Rule (Var) by the proof tree, we obtain $\Gamma, \Theta \vdash P' \triangleright \Delta'$ with $\Delta' = \Delta_Q \cdot s[\mathsf{r}]{:}(T\{^{n}/_{\mathsf{c}}\})\{^{(\mu t, \mathsf{c}=\mathrm{eval}(n+1))T}/_{t}\}$.
By Rule (T-Rec) of Figure 5, then $\Delta \overset{s}{\mapsto} \Delta'$. Since Γ, Δ are coherent and by transitivity of $\overset{s}{\mapsto}$, then Γ, Δ' are coherent.

Case of Rule (Struc) In this case $P \equiv Q$, $Q \longmapsto Q'$, $Q' \equiv P'$. By Lemma 4, $\Gamma, \Theta \vdash P \triangleright \Delta$ and $P \equiv Q$ imply $\Gamma, \Theta \vdash Q \triangleright \Delta$. By the induction hypothesis, $\Gamma, \Theta \vdash Q' \triangleright \Delta'$ for some Δ' such that $\Delta \overset{s}{\mapsto} \Delta'$ and Γ, Δ' are coherent. By Lemma 4, then $\Gamma, \Theta \vdash P' \triangleright \Delta'$. □

Since we restrict our attention to linear environments, type judgements ensure linearity of session channels. With subject reduction, this holds for all derivatives of well-typed processes.

Lemma 6 (Linearity)**.** *Let $\Gamma \vdash P \triangleright \Delta$, Γ, Δ be coherent, and there are no name clashes on session channels. Then all session channels of P are linear, i.e., for all $P \longmapsto^* P'$ and all $s, \mathsf{r}, \mathsf{r}_1, \mathsf{r}_2$ there is at most one unguarded actor $s[\mathsf{r}]$ and at most one queue $s_{\mathsf{r}_1 \to \mathsf{r}_2}$ in P'.*

Proof. By Theorem 2, there is some Δ' such that $\Gamma \vdash P' \triangleright \Delta'$ and Γ, Δ' are coherent. By the Definition 2 of coherence and projection, Δ' contains at most one actor $s[\mathsf{r}]$ and at most one queue $s_{\mathsf{r}_1 \to \mathsf{r}_2}$ for each $a{:}G \in \Gamma$ and $\mathsf{r} \in \mathrm{R}(G)$. By the Figures 3 and 4, only the Rules (Req), (Acc), and (Res2) can introduce new actors or queues. The linearity of global environments ensures, that all new actors and queues introduced by the rules are on fresh channel names and are pairwise distinct. The Rules (Req) and (Acc) introduce exactly one actor each on a fresh session channel s that is bound by a prefix for session initialisation. Rule (Res2) introduces assignments for actors and queues for pairwise different roles on a fresh session channel s that is bound by restriction. Since there are no name clashes, the session channels in binders are pairwise different and distinct from free session channels. By the typing rules and because $\Gamma, \Theta \vdash P' \triangleright \Delta'$, all actors and queues in P' have to satisfy their specification as described by an assignment of this actor or queue towards a local type. By the linearity of session environments and since new assignments for actors result from bound session channels, all unguarded actors and queues in P' are pairwise different. $\square$

For strongly reliable systems coherence ensures that for each actor there is a matching communication partner. In the case of asynchronous communication, this means that for each sender (or message on a queue) there is a receiver and for each receiver there is a sender or a message on a queue, where the receiver as well as the sender or the message queue appear under the same binder of the session channel or both are free. In the case of unreliable communication, messages get lost, senders can crash, and receivers can crash themselves or suspect the sender. In the case of weakly reliable branching for each sender (or message on a queue) there are all specified receivers that are not crashed and vice versa.

We summarise these properties of strongly reliable and weakly reliable interactions in *error-freedom* that we inherit verbatim from [24]: for each strongly reliable sender or message there is a matching receiver and vice versa, for each weakly reliable sender or message there is a possibly crashed receiver and vice versa. We obtain similar requirements for session delegation.

Lemma 7 (Error-Freedom). *If $\Gamma \vdash P \triangleright \Delta$ and Γ, Δ is coherent then:*

- *for each unguarded $s[\mathsf{r}_1, \mathsf{r}_2]!_{\mathsf{r}}\langle y\rangle.Q_1$ and each message $\langle y\rangle^{\mathsf{r}}$ on a message queue $s_{\mathsf{r}_1 \to \mathsf{r}_2}$ in P there is some $s[\mathsf{r}_2, \mathsf{r}_1]?_{\mathsf{r}}(x).Q_2$ in P,*
- *for each unguarded $s[\mathsf{r}_2, \mathsf{r}_1]?_{\mathsf{r}}(x).Q_2$ in P there is some $s[\mathsf{r}_1, \mathsf{r}_2]!_{\mathsf{r}}\langle y\rangle.Q_1$ or a message $\langle y\rangle^{\mathsf{r}}$ on a message queue $s_{\mathsf{r}_1 \to \mathsf{r}_2}$ in P,*
- *for each unguarded $s[\mathsf{r}_1, \mathsf{r}_2]!_{\mathsf{r}}l.Q$ and each message l^{r} on a message queue $s_{\mathsf{r}_1 \to \mathsf{r}_2}$ in P there is some $j \in \mathrm{I}$ and $s[\mathsf{r}_2, \mathsf{r}_1]?_{\mathsf{r}}\{l_i.Q_i\}_{i \in \mathrm{I}}$ in P with $l_j \doteq l$,*
- *for each unguarded $s[\mathsf{r}_2, \mathsf{r}_1]?_{\mathsf{r}}\{l_i.Q_i\}_{i \in \mathrm{I}}$ in P there is $j \in \mathrm{I}$ and $s[\mathsf{r}_1, \mathsf{r}_2]!_{\mathsf{r}}l.Q$ or a message l^{r} on a message queue $s_{\mathsf{r}_1 \to \mathsf{r}_2}$ in P with $l_j \doteq l$,*
- *for each unguarded $s[\mathsf{r}, \mathsf{R}]!_{\mathsf{w}}l.Q$ and each message l^{w} on a message queue $s_{\mathsf{r} \to \mathsf{r}'}$ in P and each $\mathsf{r}' \in \mathsf{R}$ there is some $s[\mathsf{r}', \mathsf{r}]?_{\mathsf{w}}\{l_i.P_i\}_{i \in \mathrm{I}, l_{\mathrm{d}}}$ and $j \in \mathrm{I}$ in P with $l_j \doteq l$ or P does not contain an actor $s[\mathsf{r}']$,*
- *for each unguarded $s[\mathsf{r}', \mathsf{r}]?_{\mathsf{w}}\{l_i.P_i\}_{i \in \mathrm{I}, l_{\mathrm{d}}}$ in P there is $j \in \mathrm{I}$ and $s[\mathsf{r}, \mathsf{R}]!_{\mathsf{w}}l.Q$ or a message l^{w} on a message queue $s_{\mathsf{r} \to \mathsf{r}'}$ in P with $l_j \doteq l$ and $\mathsf{r}' \in \mathsf{R}$ or P does not contain an actor $s[\mathsf{r}]$,*
- *for each unguarded $s[\mathsf{r}_1, \mathsf{r}_2]!\langle\langle s'[\mathsf{r}]\rangle\rangle.Q_1$ and each message $s'[\mathsf{r}]$ on a message queue $s_{\mathsf{r}_1 \to \mathsf{r}_2}$ in P there is some $s[\mathsf{r}_2, \mathsf{r}_1]?((s''[\mathsf{r}'])).Q_2$ in P, and*
- *for each unguarded $s[\mathsf{r}_2, \mathsf{r}_1]?((s''[\mathsf{r}'])).Q_2$ in P there is some $s[\mathsf{r}_1, \mathsf{r}_2]!\langle\langle s'[\mathsf{r}]\rangle\rangle.Q_1$ or a message $s'[\mathsf{r}]$ on a message queue $s_{\mathsf{r}_1 \to \mathsf{r}_2}$ in P.*

Proof. By coherence and projection for each strongly reliable and each weakly reliable sender there is initially a matching receiver for each free session channel in the session environment. By the typing rules and Rule (Res2) in particular, this holds also for restricted session channels. Session environments may evolve using $\overset{s}{\mapsto}$ but all such steps preserve the above defined requirements, i.e., strongly reliable or weakly reliable send prefixes can be mapped onto the type of the respective message in a queue but no such message can be dropped. The typing rules ensure that the processes follow their specification in the local types of session environments. Then, the first four and the last two cases follow from the typing rules and coherence, and the fact that only unreliable processes can crash. The remaining two cases follow from the typing rules and coherence. $\square$

Session fidelity claims that the interactions of a well-typed process follow exactly the specification described by its global types, i.e., if a system is well-typed w.r.t. to coherent type environments then the system follows its specification in the global type. One direction of this property already follows from the above variant of subject reduction. The steps of well-typed systems are reflected by corresponding steps of the session environment and, thus, respect their specification in global types. What remains to show is that the specified interactions can indeed be performed. The above formulation of error-freedom alone is not strong enough to show this, because it ensures only the existence of matching communication partners and not that they can be unguarded.

To obtain session fidelity we prove *progress*. *Progress* states that no part of a well-typed and coherent system can block other parts, that eventually all matching communication partners as described by error-freedom are unguarded, that interactions specified by the global type can happen, and that there are no communication mismatches. Subject reduction and progress together then imply *session fidelity*, i.e., that processes behave as specified in their global types.

In the literature there are different formulations of progress. We are interested in a rather strict definition of progress that ensures that well-typed systems cannot block. Therefore, we need an additional assumption on session requests and acceptances. Coherence ensures the existence of communication partners within sessions only. If we want to avoid blocking, we need to be sure, that no participant of a session is missing during its initialisation. Note that without action prefixes all participants either terminated or crashed.

Proof of Theorem 3. 1.) If P contains an unguarded conditional, then it can perform a step $P \longmapsto P'$ such that $\Gamma, \Theta \vdash P' \triangleright \Delta$ using one of the Rules (If-T) or (If-F) as described in the corresponding cases of the proof of Theorem 2. Similarly, if P contains an unguarded recursion, then it can perform a step $P \longmapsto P'$ such that $\Gamma, \Theta \vdash P' \triangleright \Delta'$ with $\Delta \overset{s}{\mapsto} \Delta'$ using Rule (Rec) as described in the corresponding case of the proof of Theorem 2. Assume that P is not structural congruent to $\mathbf{0}$ and does not contain unguarded conditionals or recursions.

Since P is not structural congruent to $\mathbf{0}$ it contains session channels. All session channels of P that are not contained in Δ are bound in P, i.e., the names of Δ are exactly the free session channels of P. Since there are no cyclic dependencies between sessions, we can pick a minimal session in P, i.e., a session such that its next action is not blocked by any action of another session or session delegation. Let s denote this session channel. By the typing rules in the Figures 3 and 4, there are some G, b such that $b{:}G \in \Gamma$ and G specifies the session s. Since the next action of this session is not blocked, P contains at least one unguarded prefix or at least one unguarded not empty queue on s. Among all such unguarded prefixes and messages that are head of a queue we pick a minimal, i.e., one that is typed by the projection of a part of G such that no part of G that is guarding it is used to type another unguarded prefix or message in P:

$\overline{a}[\mathsf{n}](s).Q$ Then $s \sharp \Delta$ and $a{:}G \in \Gamma$. By the assumption on session initialisation, then for all $1 \leq \mathsf{r}_i < \mathsf{n}$ we have $a[\mathsf{r}_i](s).$ in P. Since s is minimal, all these session acceptances are unguarded. By Rule (Init), then $P \longmapsto P'$.

$a[\mathsf{r}](s).Q$ Then $s \sharp \Delta$ and $a{:}G \in \Gamma$. By the assumption on session initialisation, then $\overline{a}[\mathsf{n}](s).Q$ and for all $1 \leq \mathsf{r}_i < \mathsf{n}$ with $\mathsf{r}_i \neq \mathsf{r}$ we have $a[\mathsf{r}_i](s).$ in P. Since s is minimal, all this session request and these session acceptances are unguarded. By Rule (Init), then $P \longmapsto P'$.

$s[\mathsf{r}_1, \mathsf{r}_2]!_{\mathsf{r}}\langle y \rangle.Q$ If $s \sharp \Delta$ then the typing rules in the Figures 3 and 4 and (Req), (Acc), and (Res2) in particular ensure that $s_{\mathsf{r}_1 \to \mathsf{r}_2}{:}\mathrm{MT}$ in P. If s is free in Δ, then $s_{\mathsf{r}_1 \to \mathsf{r}_2}{:}\mathrm{MT}$ in P because of coherence. Since s and the action are minimal, $s_{\mathsf{r}_1 \to \mathsf{r}_2}{:}\mathrm{MT}$ is unguarded. By Rule (RSend), then $P \longmapsto P'$.

$s_{\mathsf{r}_1 \to \mathsf{r}_2}{:}\langle v \rangle^{\mathsf{r}}\#\mathrm{M}$ By Lemma 7, then $s[\mathsf{r}_2, \mathsf{r}_1]?_{\mathsf{r}}(x).Q$ in P. Since s and the action are minimal, $s[\mathsf{r}_2, \mathsf{r}_1]?_{\mathsf{r}}(x).Q$ is unguarded. By Rule (RGet), then $P \longmapsto P'$.

$s[\mathsf{r}_2, \mathsf{r}_1]?_{\mathsf{r}}(x).Q$ By Lemma 7, then $s_{\mathsf{r}_1 \to \mathsf{r}_2}{:}\langle v \rangle^{\mathsf{r}}\#\mathrm{M}$ in P. Since s and the action are minimal, $s_{\mathsf{r}_1 \to \mathsf{r}_2}{:}\langle v \rangle^{\mathsf{r}}\#\mathrm{M}$ is unguarded. By Rule (RGet), then $P \longmapsto P'$.

$s[\mathsf{r}_1,\mathsf{r}_2]!_{\mathrm{u}}l\langle y\rangle.Q$ If $s\sharp\Delta$ then the typing rules in the Figures 3 and 4 and (Req), (Acc), and (Res2) in particular ensure that $s_{\mathsf{r}_1\to\mathsf{r}_2}$:MT in P. If s is free in Δ, then $s_{\mathsf{r}_1\to\mathsf{r}_2}$:MT in P, because of coherence. Since s and the action are minimal, $s_{\mathsf{r}_1\to\mathsf{r}_2}$:MT is unguarded. By Rule (USend), then $P\longmapsto P'$.

$s_{\mathsf{r}_1\to\mathsf{r}_2}:l\langle v\rangle^{\mathrm{u}}\#\mathrm{M}$ Then the typing rules and coherence ensure that either there is no actor $s[\mathsf{r}_2]$, the actor $s[\mathsf{r}_2]$ had skipped, or $s[\mathsf{r}_2,\mathsf{r}_1]?_{\mathrm{u}}l\langle dv\rangle(x).Q$ in P. If there is no actor $s[\mathsf{r}_2]$, then it has crashed. Then the pattern $\mathtt{FP}_{\mathtt{crash}}(Q,\ldots)$ was satisfied with $s[\mathsf{r}_2]\in\mathrm{A}(Q)$. By Condition 1.5, then eventually $\mathtt{FP}_{\mathtt{ml}}(s,\mathsf{r}_1,\mathsf{r}_2,l,\ldots)$. By Rule (ML), then $P\longmapsto P'$. If the actor $s[\mathsf{r}_2]$ skipped this reception, then $\mathtt{FP}_{\mathtt{uskip}}(s,\mathsf{r}_2,\mathsf{r}_1,l,\ldots)$. By Condition 1.3, then $\mathtt{FP}_{\mathtt{ml}}(s,\mathsf{r}_1,\mathsf{r}_2,l,\ldots)$. By Rule (ML), then $P\longmapsto P'$. If $s[\mathsf{r}_2,\mathsf{r}_1]?_{\mathrm{u}}l\langle dv\rangle(x).Q$ in P, since s and the action are minimal, then this term is unguarded. By Condition 1.2 and Rule (UGet), then $P\longmapsto P'$.

$s[\mathsf{r}_2,\mathsf{r}_1]?_{\mathrm{u}}l\langle dv\rangle(x).Q$ Then the typing rules and coherence ensure that there is the queue $s_{\mathsf{r}_1\to\mathsf{r}_2}$ and either $l\langle v\rangle^{\mathrm{u}}$ is on top of it, or this message was dropped, or the sender did not yet send this message, or the sender crashed before transmitting this message. If $l\langle v\rangle^{\mathrm{u}}$ is on top of the queue, then $P\longmapsto P'$ by Rule (UGet) and Condition 1.2. If the message was dropped, then $\mathtt{FP}_{\mathtt{ml}}(s,\mathsf{r}_1,\mathsf{r}_2,l,\ldots)$. By Condition 1.3, then $\mathtt{FP}_{\mathtt{uskip}}(s,\mathsf{r}_2,\mathsf{r}_1,l,\ldots)$. By Rule (USkip), then $P\longmapsto P'$. If the sender did not yet send the message, we proceed as in the Case $s[\mathsf{r}_1,\mathsf{r}_2]!_{\mathrm{u}}l\langle y\rangle.Q$ above. If the sender crashed, then $\mathtt{FP}_{\mathtt{crash}}(Q,\ldots)$ with $s[\mathsf{r}_1]\in\mathrm{A}(Q)$. By Condition 1.4, then eventually $\mathtt{FP}_{\mathtt{uskip}}(s,\mathsf{r}_2,\mathsf{r}_1,l,\ldots)$. By Rule (USkip), then $P\longmapsto P'$.

$s[\mathsf{r}_1,\mathsf{r}_2]!_{\mathrm{r}}l.Q$ If $s\sharp\Delta$ then the typing rules in the Figures 3 and 4 and (Req), (Acc), and (Res2) in particular ensure that $s_{\mathsf{r}_1\to\mathsf{r}_2}$:MT in P. If s is free in Δ, then $s_{\mathsf{r}_1\to\mathsf{r}_2}$:MT in P, because of coherence. Since s and the action are minimal, $s_{\mathsf{r}_1\to\mathsf{r}_2}$:MT is unguarded. By Rule (RSel), then $P\longmapsto P'$.

$s_{\mathsf{r}_1\to\mathsf{r}_2}:l^{\mathrm{r}}\#\mathrm{M}$ By Lemma 7, then $s[\mathsf{r}_2,\mathsf{r}_1]?_{\mathrm{r}}\{l_i.Q_i\}_{i\in\mathrm{I}}$ in P. Since s and the action are minimal, the action $s[\mathsf{r}_2,\mathsf{r}_1]?_{\mathrm{r}}\{l_i.Q_i\}_{i\in\mathrm{I}}$ is unguarded. By (RBran), then $P\longmapsto P'$.

$s[\mathsf{r}_2,\mathsf{r}_1]?_{\mathrm{r}}\{l_i.Q_i\}_{i\in\mathrm{I}}$ By Lemma 7, then $s_{\mathsf{r}_1\to\mathsf{r}_2}:l^{\mathrm{r}}\#\mathrm{M}$ in P with $j\in\mathrm{I}$ and $l\doteq l_j$. Since s and the action are minimal, $s_{\mathsf{r}_1\to\mathsf{r}_2}:l^{\mathrm{r}}\#\mathrm{M}$ is unguarded. By (RBran), then $P\longmapsto P'$.

$s[\mathsf{r},\mathsf{R}]!_{\mathrm{w}}l.Q$ If $s\sharp\Delta$ then the typing rules and (Req), (Acc), and (Res2) in particular ensure that $s_{\mathsf{r}\to\mathsf{r}_i}$:MT$_i$ in P for all $\mathsf{r}_i\in\mathsf{R}$. If s is free in Δ, then $s_{\mathsf{r}\to\mathsf{r}_i}$:MT$_i$ in P for all $\mathsf{r}_i\in\mathsf{R}$, because of coherence. Since s and the action are minimal, all $s_{\mathsf{r}\to\mathsf{r}_i}$:MT$_i$ are unguarded. By Rule (WSel), then $P\longmapsto P'$.

$s_{\mathsf{r}_1\to\mathsf{r}_2}:l^{\mathrm{w}}\#\mathrm{M}$ Then the typing rules, coherence, and Condition 1.6 ensure that either there is no actor $s[\mathsf{r}_2]$ or $s[\mathsf{r}_2,\mathsf{r}_1]?_{\mathrm{w}}\{l_i.Q_i\}_{i\in\mathrm{I},l_{\mathrm{d}}}$ in P. In the former case the actor $s[\mathsf{r}_2]$ has crashed. Then, since the message queue will not be needed any more, proceed with the next session and action that are minimal if you ignore the queue $s_{\mathsf{r}_1\to\mathsf{r}_2}$.

$s[\mathsf{r}_2,\mathsf{r}_1]?_{\mathrm{w}}\{l_i.Q_i\}_{i\in\mathrm{I},l_{\mathrm{d}}}$ Then the typing rules and coherence ensure that there is the queue $s_{\mathsf{r}_1\to\mathsf{r}_2}$ and either l^{w} is on top of it, or the sender did not yet send this message, or the sender crashed before transmitting this message. If l^{w} is on top of the queue, then $P\longmapsto P'$ by Rule (WBran). If the sender did not yet send the message, we proceed as in the Case $s[\mathsf{r},\mathsf{R}]!_{\mathrm{w}}l.Q$ above. If the sender crashed, then $\mathtt{FP}_{\mathtt{crash}}(Q',\ldots)$ with $s[\mathsf{r}_1]\in Q'$. By Condition 1.4, then eventually $\mathtt{FP}_{\mathtt{wskip}}(s,\mathsf{r}_2,\mathsf{r}_1,l,\ldots)$ for $s[\mathsf{r}_2]$. By Rule (WSkip), then $P\longmapsto P'$.

$s[\mathsf{r}_1,\mathsf{r}_2]!\langle\langle s'[\mathsf{r}]\rangle\rangle.Q$ If $s\sharp\Delta$ then the typing rules in the Figures 3 and 4 and (Req), (Acc), and (Res2) in particular ensure that $s_{\mathsf{r}_1\to\mathsf{r}_2}$:MT in P. If s is free in Δ, then $s_{\mathsf{r}_1\to\mathsf{r}_2}$:MT in P because of coherence. Since s and the action are minimal, $s_{\mathsf{r}_1\to\mathsf{r}_2}$:MT is unguarded. By Rule (Deleg), then $P\longmapsto P'$.

$s_{\mathsf{r}_1\to\mathsf{r}_2}:s'[\mathsf{r}]\#\mathrm{M}$ By Lemma 7, then $s[\mathsf{r}_2,\mathsf{r}_1]?((s''[\mathsf{r}'])).Q$ in P. Since s and the action are minimal, the action $s[\mathsf{r}_2,\mathsf{r}_1]?((s''[\mathsf{r}'])).Q$ is unguarded. By Rule (SRecv), then $P\longmapsto P'$.

$s[\mathsf{r}_2,\mathsf{r}_1]?((s''[\mathsf{r}'])).Q$ By Lemma 7, then $s_{\mathsf{r}_1\to\mathsf{r}_2}{:}s'[\mathsf{r}]\#\mathrm{M}$ in P. Since s and the action are minimal, the action $s_{\mathsf{r}_1\to\mathsf{r}_2}{:}s'[\mathsf{r}]\#\mathrm{M}$ is unguarded. By Rule (SRecv), then $P \longmapsto P'$.

$\mathtt{call}\langle e_l, e_v\rangle$ By the typing rules in the Figures 3 and 4 and (Req), (Acc), and (Res2) in particular or by coherence, $\mathtt{call}\langle e_l, e_v\rangle$ is in the loop body of an unguarded loop $\mathrm{eval}(e_l)$ in P. By Rule (LCall), then $P \longmapsto P'$.

$\mathtt{exit}\langle e_l, e_v\rangle$ By the typing rules in the Figures 3 and 4 and (Req), (Acc), and (Res2) in particular or by coherence, $\mathtt{exit}\langle e_l, e_v\rangle$ is in the loop body of an unguarded loop $\mathrm{eval}(e_l)$ in P. Let $s[\mathsf{r}]$ be the channel of this loop and R its remaining roles. If $s\sharp\Delta$ then the typing rules in the Figures 3 and 4 and (Req), (Acc), and (Res2) in particular ensure that $s_{\mathsf{r}\to\mathsf{r}_i}{:}\mathrm{MT}$ in P for all $\mathsf{r}_i \in \mathsf{R}$. If s is free in Δ, then $s_{\mathsf{r}\to\mathsf{r}_i}{:}\mathrm{MT}$ in P for all $\mathsf{r}_i \in \mathsf{R}$, because of coherence. Since s and the action are minimal, these $s_{\mathsf{r}\to\mathsf{r}_i}{:}\mathrm{MT}$ are unguarded for all $\mathsf{r}_i \in \mathsf{R}$. By Rule (LExitS), then $P \longmapsto P'$.

$s_{\mathsf{r}_1\to\mathsf{r}_2}{:}\mathtt{exit}\langle id, v\rangle\#\mathrm{M}$ Then the typing rules and coherence ensure that either there is no actor $s[\mathsf{r}_2]$, the actor $s[\mathsf{r}_2]$ already terminated the loop id, or the actor $s[\mathsf{r}_2]$ still has the loop id in P. If there is no actor $s[\mathsf{r}_2]$, then it has crashed. Then the pattern $\mathtt{FP}_{\mathtt{crash}}(Q,\ldots)$ was satisfied with $s[\mathsf{r}_2] \in \mathrm{A}(Q)$. By Condition 1.5, then eventually $\mathtt{FP}_{\mathtt{drop}}(\mathsf{r}_2, id)$. By Rule (EDrop), then $P \longmapsto P'$. If the actor $s[\mathsf{r}_2]$ already terminated the loop, then eventually $\mathtt{FP}_{\mathtt{drop}}(\mathsf{r}_2, id)$, by Condition 1.7. By Rule (EDrop), then $P \longmapsto P'$. If the loop is still in P, since s and the action are minimal, then this loop is unguarded. By Rule (LExitG), then $P \longmapsto P'$.

2.) By Theorem 2, P' is well-typed w.r.t. coherent Γ, Δ' with $\Delta \overset{s}{\Mapsto} \Delta'$. If P does not contain recursion or loops, then the session environment strictly reduces with every reduction of the process that does not reduce a conditional, though it may grow in cases of session initialisation. Since P is finite and loop-free there are only finitely many possible session initialisations. We can repeat the above proof for 1.) to show that P cannot get stuck as long as it contains action prefixes. $\square$