A Typing Discipline for Statically Verified Crash Failure Handling in Distributed Systems *

Malte Viering[1(✉)], Tzu-Chun Chen[1], Patrick Eugster[1,2,3], Raymond Hu[4], and Lukasz Ziarek[5]

[1] Department of Computer Science, TU Darmstadt, Darmstadt, Germany
viering@dsp.tu-darmstadt.de
[2] Faculty of Informatics, Università della Svizzera italiana, Lugano, Switzerland
[3] Department of Computer Science, Purdue University, West Lafayette, USA
[4] Department of Computing, Imperial College London, London, UK
[5] Department of Computer Science and Engineering, SUNY Buffalo, Buffalo, USA

Abstract. A key requirement for many distributed systems is to be resilient toward partial failures, allowing a system to progress despite the failure of some components. This makes programming of such systems daunting, particularly in regards to avoiding inconsistencies due to failures and asynchrony. This work introduces a formal model for crash failure handling in asynchronous distributed systems featuring a lightweight coordinator, modeled in the image of widely used systems such as ZooKeeper and Chubby. We develop a typing discipline based on multiparty session types for this model that supports the specification and static verification of multiparty protocols with explicit failure handling. We show that our type system ensures subject reduction and progress in the presence of failures. In other words, in a well-typed system even if some participants crash during execution, the system is guaranteed to progress in a consistent manner with the remaining participants.

1 Introduction

Distributed programs, partial failures, and coordination. Developing programs that execute across a set of physically remote, networked processes is challenging. The correct operation of a *distributed program* requires correctly designed protocols by which concurrent processes interact asynchronously, and correctly implemented processes according to their roles in the protocols. This becomes particularly challenging when distributed programs have to be resilient to *partial failures*, where some processes crashes while others remain operational. Partial failures affect both *safety* and *liveness* of applications. Asynchrony is the key issue, resulting in the inability to distinguish slow processes from failed ones. In general, this makes it impossible for processes to reach agreement, even when only a single process can crash [19].

* Financially supported by ERC grant FP7-617805 "LiVeSoft - Lightweight Verification of Software" and NSF grants CNS-1405614 and IIS-1617586.

In practice, such impasses are overcome by making appropriate assumptions for the considered infrastructure and applications. One common approach is to assume the presence of a highly available *coordination service* [26] – realized using a set of replicated processes large enough to survive common rates of process failures (e.g., 1 out of 3, 2 out of 5) – and delegating critical decisions to this service. While this *coordinator model* has been in widespread use for many years (cf. *consensus service* [22]), the advent of cloud computing has recently brought it further into the mainstream, via instances like Chubby [4] and ZooKeeper [26]. Such systems are used not only by end applications but also by a variety of frameworks and middleware systems across the layers of the protocol stack [11,20,31,41].

Typing disciplines for distributed programs. Typing disciplines for distributed programs is a promising and active research area towards addressing the challenges in the correct development of distributed programs. See Hüttel et al. [27] for a broad survey. *Session types* are one of the established typing disciplines for message passing systems. Originally developed in the π-calculus [23], these have been later successfully applied to a range of practical languages, e.g., Java [25,42], Scala [40], Haskell [35,39], and OCaml [28,38]. *Multiparty* session types (MPSTs) [15,24] generalize session types beyond two participants. In a nutshell, a standard MPST framework takes (1) a specification of the whole multiparty message protocol as a *global type*; from which (2) *local types*, describing the protocol from the perspective of each participant, are derived; these are in turn used to (3) statically *type check* the I/O actions of endpoint programs implementing the session participants. A well-typed system of session endpoint programs enjoys important safety and liveness properties, such as *no reception errors* (only expected messages are received) and *session progress*. A basic intuition behind MPSTs is that the design (i.e., restrictions) of the type language constitutes a class of distributed protocols for which these properties can be statically guaranteed by the type system.

Unfortunately, *no* MPST work supports protocols for asynchronous distributed programs dealing with *partial failures due to process crashes*, so the aforementioned properties no longer hold in such an event. Several MPST works have treated communication patterns based on *exception messages* (or *interrupts*) [6,7,16]. In these works, such messages may convey exceptional states in an *application* sense; from a protocol compliance perspective, however, these messages are the same as any other message communicated during a *normal* execution of the session. This is in contrast to *process* failures, which may invalidate already in-transit (*orphan*) messages, and where the task of agreeing on the concerted handling of a crash failure is itself prone to such failures.

Outside of session types and other type-based approaches, there have been a number of advances on verifying fault tolerant distributed protocols and applications (e.g., based on model checking [29], proof assistants [44]); however, little work exists on providing direct compile-time support for *programming* such applications in the spirit of MPSTs.

Contributions and challenges. This paper puts forward a new typing discipline for safe specification and implementation of distributed programs prone to process crash failures based on MPSTs. The following summarizes the key challenges and contributions.

Multiparty session calculus with coordination service. We develop an extended multiparty session calculus as a formal model of processes prone to crash failures in asynchronous message passing systems. Unlike standard session calculi that reflect only "minimal" networking infrastructures, our model introduces a practically-motivated *coordinator* artifact and explicit, asynchronous messages for run-time crash notifications and failure handling.

MPSTs with explicit failure handling. We introduce new global and local type constructs for *explicit failure handling*, designed for specifying protocols tolerating partial failures. Our type system carefully reworks many of the key elements in standard MPSTs to manage the intricacies of handling crash failures. These include the well-formedness of failure-prone global types, and the crucial *coherence* invariant on MPST typing environments to reflect the notion of system consistency in the presence of crash failures and the resulting errors. We show safety and progress for a well-typed MPST session despite potential failures.

To fit our model to practice, we introduce programming constructs similar to well-known and intuitive exception handling mechanisms, for handling concurrent and asynchronous process crash failures in sessions. These constructs serve to integrate user-level session control flow in endpoint processes and the underlying communications with the coordination service, used by the target applications of our work to outsource critical failure management decisions (see Fig. 1). It is important to note that the coordinator does *not* magically solve all problems. Key design challenges are to ensure that communication with it is fully asynchronous as in real-life, and that it is involved only in a "minimal" fashion. Thus we treat the coordinator as a first-class, asynchronous network artifact, as opposed to a convenient but impractical global "oracle" (cf. [6]), and our operational semantics of multiparty sessions remains primarily *choreographic* in the original spirit of distributed MPSTs, unlike works that resort to a centralized *orchestrator* to conduct all actions [5,8]. As depicted in Fig. 1, application-specific communication does not involve the coordinator. Our model lends itself to common practical scenarios where processes monitor each other in a peer-based fashion to detect failures, and rely on a coordinator only to establish agreement on which processes have failed, and when.

We also developed a prototype implementation in Scala, that uses ZooKeeper for coordination. Due to space limitations we present it in the appendix.

Example. As a motivating example, Fig. 2 gives a global formal specification for a big data streaming task between a distributed file system (DFS) *dfs*, and two workers $w_{1,2}$. The DFS streams data to two workers, which process the data and

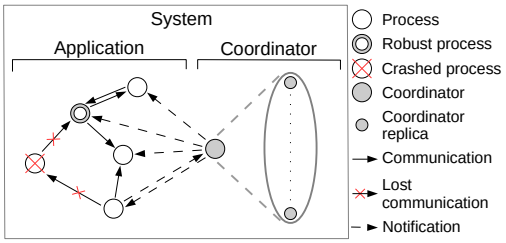

Fig. 1. Coordinator model for asynchronous distributed systems. The coordinator is implemented by replicated processes (internals omitted).

$$
\begin{aligned}
&[dfs]G = \mathsf{t}(\mu t.\\
&\quad dfs{\to}w_1\ l_{d_1}(S).dfs{\to}w_2\ l_{d_2}(S).\\
&\quad w_1{\to}dfs\ l_{r_1}(S').w_2{\to}dfs\ l_{r_2}(S').t\\
&)\mathsf{h}(\\
&\quad \{w_1\}:\mu t'.dfs{\to}w_2\ l'_{d_1}(S).\\
&\qquad\qquad w_2{\to}dfs\ l'_{r_1}(S').t',\\
&\quad \{w_2\}:\ldots,\{w_1,w_2\}:\mathsf{end})
\end{aligned}
$$

Fig. 2. Global type for a big data streaming task with failure handling capabilities.

write the result back. Most DFSs have built-in fault tolerance mechanisms [20], so we consider *dfs* to be *robust*, denoted by the annotation $[dfs]$; the workers, however, may individually fail. In the *try-handle* construct $\mathsf{t}(...)\mathsf{h}(...)$, the *try-block* $\mathsf{t}(...)$ gives the *normal* (i.e., failure-free) flow of the protocol, and $\mathsf{h}(...)$ contains the explicit *handlers* for potential crashes. In the try-block, the workers receive data from the DFS ($dfs{\to}w_i$), perform local computations, and send back the result ($w_i{\to}dfs$). If a worker crashes ($\{w_i\}:\ ...$), the other worker will also take over the computation of the crashed worker, allowing the system to still produce a valid result. If both workers crash (by any interleaving of their concurrent crash events), the global type specifies that the DFS should safely terminate its role in the session.

We shall refer to this basic example, that focuses on the new failure handling constructs, in explanations in later sections. See App. A.1 for examples of larger protocols featuring multiparty choices and recursion with explicit failure handling. We also give many further examples throughout the following sections to illustrate the potential session errors due to failures exposed by our model, and how our framework resolves them to recover MPST safety and progress.

Roadmap. Sec. 2 describes the adopted system and failure model. Sec. 3 introduces global types for guiding failure handling. Sec. 4 introduces our process calculus with failure handling capabilities and a coordinator. Sec. 5 introduces local types, derived from global types by projection. Sec. 6 describes typing rules, and defines *coherence* of session environments with respect to endpoint crashes. Sec. 7 states properties of our model. Sec. 8 discusses related work. Sec. 9 draws conclusions. App. B contains further formal definitions. App. A contains additional examples. App. C provides details of our prototype implementation in Scala using ZooKeeper. App. D gives full proofs of properties.

2 System and Failure Model

In distributed systems care is required to avoid partial failures affecting liveness (e.g., waiting on messages from crashed processes) or safety (e.g., when processes

manage to communicate with some peers but not others before crashing) properties of applications. Based on the nature of the infrastructure and application, appropriate *system and failure models* are chosen along with judiciously made assumptions to overcome such impasses in practice.

We pinpoint the key characteristics of our model, according to our practical motivations and standard distributed systems literature, that shape the design choices we make later for the process calculus and types. As it is common we augment our system with a *failure detector* (FD) to allow for distinguishing slow and failed processes. The advantage of the FD (1) in terms of reasoning is that it concentrates all assumptions to solve given problems and (2) implementation-wise it yields a single main module where time-outs are set and used.

Concretely we make the following assumptions on failures and the system:

(1) **Crash-stop failures**: Application processes fail by crashing (halting), and do not recover.
(2) **Asynchronous system**: Application processes and the network are asynchronous, meaning that there are no upper bounds on processes' relative speeds or message transmission delays.
(3) **Reliable communication**: Messages transmitted between correct (i.e., non-failed) participants are eventually received.
(4) **Robust coordinator**: The coordinator (coordination service) is permanently available.
(5) **Asynchronous reliable failure detection**: Application processes have access to local FDs which eventually detect all failed peers and do not falsely suspect peers.

(1)–(3) are standard in literature on fault-tolerant distributed systems [19].

Note that processes can still recover but will not do so *within* sessions (or will not be re-considered for those). Other failure models, e.g., network partitions [21] or Byzantine failures [33], are subject of future work. The former are not tolerated by ZooKeeper et al., and the latter have often been argued to be a too generic failure model (e.g., [3]).

The assumption on the coordinator (4) implicitly means that the number of concomitant failures among the coordinator replicas is assumed to remain within a minority, and that failed replicas are replaced in time (to tolerate further failures). Without loss of validity, the coordinator internals can be treated as a blackbox (e.g., ZooKeeper uses a variant of Paxos [32]). The final assumption (5) on failure detection is backed in practice by the concept of *program-controlled* crash [10], which consists in communicating decisions to disregard supposedly failed processes also to those processes, prompting them to reset themselves upon false suspicion. In practice systems can be configured to minimize the probability of such events, and by a "two-level" membership consisting in evicting processes from *individual* sessions (cf. recovery above) more quickly than from a system as a whole; several authors have also proposed network support to entirely avoid false suspicions (e.g., [34]).

These assumptions do not make handling of failures trivial, let alone mask them. For instance, the network can arbitrarily delay messages and thus reorder

(Basic type) $S ::= \mathsf{bool} \mid \mathsf{str} \mid \mathsf{int}$

(Global type) $G ::= p \to q\{l_i(S_i).G_i\}_{i\in I} \mid \mu t.G \mid t \mid \mathsf{end} \mid \mathsf{t}(G_1)\mathsf{h}(H)^{\kappa}.G_2$

(Handling env.) $H ::= F{:}G \mid H, H$ (Handler sig.) $F ::= \{p_i\}_{i\in I}$

Fig. 3. Syntax of global types with explicit handling of partial failures.

them with respect to their real sending times, and (so) different processes can detect failures at different points in time and in different orders.

3 Global Types for Explicit Handling of Partial Failures

Based on the foundations of MPSTs, we develop *global types* to formalize specifications of distributed protocols with explicit handling of *partial failures due to role crashes*, simply referred to as *failures*. We present global types before introducing the process calculus to provide a high-level intuition of how failure handling works in our model.

The syntax of *global types* is depicted in Fig. 3. We use the following base notations: $p, q, ...$ for *role* (i.e., participant) names; $l_1, l_2, ...$ for message *labels*; and $t, t', ...$ for type variables. *Base types* S may range over, $\mathsf{bool}, \mathsf{int}$, etc.

Global types are denoted by G. We first summarize the constructs from standard MPST [15,24]. A *branch* type $p \to q\{l_i(S_i).G_i\}_{i\in I}$ means that p can send to q *one* of the messages of type S_k with label l_k, where k is a member of the non-empty index set I. The protocol then proceeds according to the continuation G_k. When I is a singleton, we may simply write $p{\to}q\ l(S).G$. We use t for type variables and take an equi-recursive view, i.e., $\mu t.G$ and its unfolding $[\mu t.G/t]$ are equivalent. We assume type variable occurrences are bound and guarded (e.g., $\mu t.t$ is not permitted). end is for termination.

We now introduce our extensions for partial failure handling. A *try-handle* $\mathsf{t}(G_1)\mathsf{h}(H)^{\kappa}.G_2$ describes a "failure-atomic" protocol unit: all *live* (i.e., non-crashed) roles will eventually reach a consistent protocol state, despite any concurrent and asynchronous role crashes. The try-block G_1 defines the *default* protocol flow, and H is a *handling environment*. Each element of H maps a *handler signature* F, that specifies a set of *failed* roles $\{p_i\}_{i\in I}$, to a *handler body* specified by a G. The handler body G specifies how the live roles should proceed given the failure of roles F. The protocol then proceeds (for live roles) according to the continuation G_2 after the default block G_1 or failure handling defined in H has been completed as appropriate.

To simplify later technical developments, we annotate each try-handle term in a given G by a unique $\kappa \in \mathbb{N}$ that lexically identifies the term within G. These annotations may be assigned mechanically. As a short hand, we refer to the try-block and handling environment of a particular try-handle by its annotation; e.g., we use κ to stand for $\mathsf{t}(G_1)\mathsf{h}(H)^{\kappa}$. In the running examples (e.g., Fig. 2), if there exists only one try-handle, we omit κ for simplicity.

Top-level global types and robust roles. We use the term *top-level* global type to mean the source protocol specified by a user, following a typical top-down interpretation of MPST frameworks [15,24]. We allow top-level global types to be optionally annotated $[\![\tilde{p}]\!]G$, where $[\![\tilde{p}]\!]$ specifies a set of *robust* roles—i.e., roles that can be assumed to never fail. In practice, a participant may be robust if it is replicated or is made inherently fault tolerant by other means (e.g., the participant that represents the distributed file system in Fig. 2).

Well-formedness. The first stage of validation in standard MPSTs is to check that the top-level global type satisfies the supporting criteria used to ensure the desired properties of the type system. We first list basic syntactic conditions which we assume on any given G: (i) each F is non-empty; (ii) a role in a F cannot occur in the corresponding handler body (a failed role cannot be involved in the handling of its own failure); and (iii) every occurrence of a non-robust role p must be contained within a, possibly outer, try-handle that has a handler signature $\{p\}$ (the protocol must be able to handle its potential failure). Lastly, to simplify the presentation without loss of generality, we impose that separate branch types *not* defined in the same default block or handler body must have disjoint label sets. This can be implicitly achieved by combining label names with try-handle annotations.

Assuming the above, we define *well-formedness* for our extended global types. We write $G' \in G$ to mean that G' syntactically occurs in G ($\in$ is reflexive); similarly for the variations $\kappa \in G$ and $\kappa \in \kappa'$. Recall κ is shorthand for $\mathsf{t}(G_1)\mathsf{h}(H)^{\kappa}$. We use a lookup function $outer_G(\kappa)$ for the set of all try-handles in G that enclose a given κ, including κ itself, defined by $outer_G(\kappa) = \{\kappa' \mid \kappa \in \kappa' \wedge \kappa' \in G\}$. Full definitions for all the above are presented in App. B.1.

Definition 1 (Well-formedness). Let κ stand for $\mathsf{t}(G_1)\mathsf{h}(H)^{\kappa}$, and κ' for $\mathsf{t}(G'_1)\mathsf{h}(H')^{\kappa'}$. A global type G is *well-formed* if both of the following conditions hold. For all $\kappa \in G$:

1. $\forall F_1 \in dom(H).\forall F_2 \in dom(H).\exists \kappa' \in outer_G(\kappa)$ s.t. $F_1 \cup F_2 \in dom(H')$
2. $\nexists F \in dom(H).\exists \kappa' \in outer_G(\kappa).\exists F' \in dom(H')$ s.t. $\kappa' \neq \kappa \wedge F' \subseteq F$

The first condition asserts that for any two separate handler signatures of a handling environment of κ, there always exists a handler whose handler signature matches the union of their respective failure sets – this handler is either inside the handling environment of κ itself, or in the handling environment of an outer try-handle. This ensures that if roles are active in different handlers of the same try-handle then there is a handler whose signature corresponds to the union over the signatures of those different handlers. Example 2 together with Example 3 in Sec. 4 illustrate a case where this condition is needed. The second condition asserts that if the handling environment of a try-handle contains a handler for F, then there is no outer try-handle with a handler for F' such that $F' \subseteq F$. The reason for this condition is that in the case of *nested* try-handles, our communication model allows separate try-handles to start failure handling independently (the operational semantics will be detailed in the next section; see

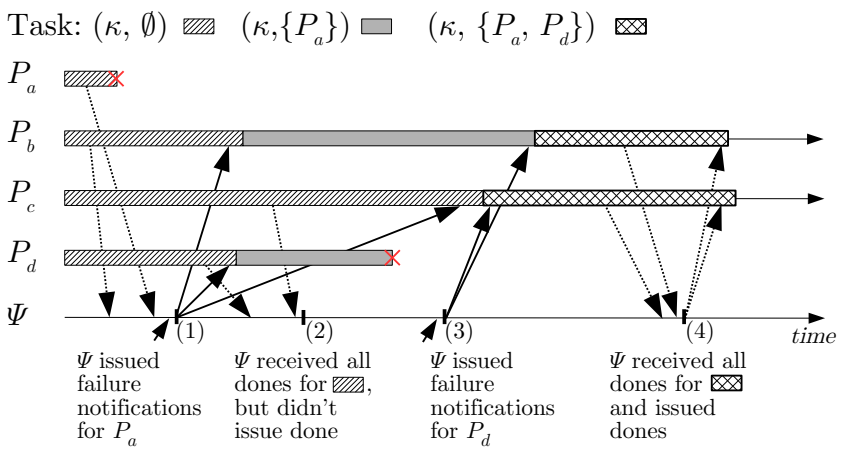

Fig. 4. Challenges under pure asynchronous interactions with a coordinator. Between time (1) and time (2), the task $\phi = (\kappa, \emptyset)$ is interrupted by the crash of P_a. Between time (3) and time (4), due to asynchrony and multiple crashes, P_c starts handling the crash of $\{P_a, P_d\}$ without handling the crash of $\{P_a\}$. Finally after (4) P_b and P_c finish their common task.

(TryHdl) in Fig. 6). The aim is to have the relevant roles eventually converge on performing the handling of the outermost try-handle, possibly by interrupting the handling of an inner try-handle. Consider the following example:

Example 1. $G = \mathsf{t}(\mathsf{t}(G')\mathsf{h}(\{p_1, p_2\} : G_1)^2)\mathsf{h}(\{p_1\} : G_1')^1$ violates condition 2 because, when p_1 and p_2 both failed, the handler signature $\{p_1\}$ will still be triggered (i.e., the outer try-handle will eventually take over). It is not sensible to run G_1' instead of G_1 (which is for the crashes of p_1 and p_2).

App. B.1 gives examples of well-formed and ill-formed global types, and further elaborates on issues with the latter.

4 A Process Calculus for Coordinator-based Failure Handling

Fig. 4 depicts a scenario that can occur in practical asynchronous systems with coordinator-based failure handling through frameworks such as ZooKeeper (Sec. 2). Using this scenario, we first illustrate challenges, formally define our model, and then develop a safe type system.

The scenario corresponds to a global type of the form $\mathsf{t}(G)\mathsf{h}(\{P_a\} : G_a, \{P_a, P_d\} : G_{ad}, ...)^\kappa$, with processes $P_{a..d}$ and a coordinator Ψ. We define a *task* to mean a unit of interactions, which includes failure handling behaviors. Initially all processes are collaborating on a task ϕ, which we label $(\kappa, \emptyset)$ (identifying the task context, and the set of failed processes). The shaded boxes signify which task each process is working on. Dotted arrows represent notifications between

(Expression)	$e ::= v \mid x \mid e + e \mid -e \mid ...$	(Channel)	$\mathsf{c} ::= \mathsf{s}[p] \mid y$
(Process)	$P ::= a[p](y).P \mid \mathsf{c} : \eta$	(Level)	$\phi ::= (\kappa, F)$
(Statement)	$\eta ::= \mathsf{t}(\eta)\mathsf{h}(\mathsf{H})^{\phi}.\eta \mid \underline{0} \mid 0 \mid p!\ l(e).\eta$	(Declaration)	$D ::= X(x) = \eta$
	$\mid p?\{l_i(x_i).\eta_i\}_{i \in I} \mid X\langle e \rangle$	(Handling)	$\mathsf{H} ::= F : \eta \mid \mathsf{H}, \mathsf{H}$
	$\mid \mathsf{def}\ D\ \mathsf{in}\ \eta \mid \mathsf{if}\ e\ \eta\ \mathsf{else}\ \eta$		
(Application)	$N ::= P \mid N \mid N \mid \mathsf{s} : h$	(Queue)	$h ::= \emptyset \mid h \cdot m$
(Message)	$m ::= \langle p, q, l(v) \rangle \mid \langle\!\langle p, \mathsf{crash}\ F \rangle\!\rangle \mid dn$	(Done)	$dn ::= \langle p, q \rangle^{\phi}$
(System)	$\mathcal{S} ::= \Psi \blacklozenge N \mid (\nu \mathsf{s})\mathcal{S} \mid \mathcal{S} \mid \mathcal{S}$	(Coordinator)	$\Psi ::= G : (F, d)$
(Context)	$E ::= \mathsf{t}(E)\mathsf{h}(\mathsf{H})^{\phi}.\eta \mid \mathsf{def}\ D\ \mathsf{in}\ E \mid [\,]$	(Done Queue)	$d ::= \emptyset \mid d \cdot dn$

Fig. 5. Grammar for processes, applications, systems, and evaluation contexts.

processes and Ψ related to task completion, and solid arrows for failure notifications from Ψ to processes. During the scenario, P_a first fails, then P_d fails: the execution proceeds through failure handling for $\{P_a\}$ and $\{P_a, P_d\}$.

(I) When P_b reaches the end of its part in ϕ, the application has P_b notify Ψ. P_b then remains in the context of ϕ (the continuation of the box after notifying) in consideration of other non-robust participants still working on ϕ—P_b may yet need to handle their potential failure(s).

(II) The processes of synchronizing on the completion of a task or performing failure handling *are themselves subject to failures* that may arise concurrently. In Fig. 4, all processes reach the end of ϕ (i.e., four dotted arrows from ϕ), but P_a fails. Ψ determines this failure and it initiates failure handling at time (1), while *done* notifications for ϕ continue to arrive asynchronously at time (2). The failure handling for crash of P_a is itself interrupted by the second failure at time (3).

(III) Ψ can receive notifications that are no longer relevant. For example, at time (2), Ψ has received all *done* notifications for ϕ, but the failure of P_a has already triggered failure handling from time (1).

(IV) Due to multiple concurrent failures, interacting participants may end up in different tasks: around time (2), P_b and P_d are in task $\phi' = (\kappa, \{P_a\})$, whereas P_c is still in ϕ (and asynchronously sending or receiving messages with the others). Moreover, P_c never executes ϕ' because of delayed notifications, so it goes from ϕ directly to $(\kappa, \{P_a, P_d\})$.

Processes. Fig. 5 defines the grammar of processes and (distributed) applications. Expressions $e, e_i, ..$ can be values $v, v_i, ...$, variables $x, x_i, ...$, and standard operations. (Application) processes are denoted by $P, P_i, ...$ An initialization $a[p](y).P$ agrees to play role p via shared name a and takes actions defined in P; actions are executed on a session channel $\mathsf{c} : \eta$, where c ranges over $\mathsf{s}[p]$ (session name and role name) and session variables y; η represents action statements.

A try-handle $\mathsf{t}(\eta)\mathsf{h}(\mathsf{H})^{\phi}$ attempts to execute the local action η, and can handle failures occurring therein as defined in the handling environment H, analogously to global types. H thus also maps a handler signature F to a handler body η defining how to handle F. Annotation $\phi = (\kappa, F)$ is composed of two elements:

an identity κ of a *global* try-handle, and an indication of the *current* handler signature which can be empty. $F = \emptyset$ means that the default try-block is executing, whereas $F \neq \emptyset$ means that the handler body for F is executing. Term $\underline{0}$ only occurs in a try-handle during runtime. It denotes a *yielding* for a *notification* from a *coordinator* (introduced shortly).

Other statements are similar to those defined in [15,24]. Term 0 represents an *idle* action. For convention, we omit 0 at the end of a statement. Action $p!\ l(e).\eta$ represents a sending action that sends p a label l with content e, then it continues as η. Branching $p?\{l_i(x_i).\eta_i\}_{i\in I}$ represents a receiving action from p with several possible branches. When label l_k is selected, the transmitted value v is saved in x_k, and $\eta_k\{v/x_k\}$ continues. For convenience, when there is only one branch, the curly brackets are omitted, e.g., $\mathsf{c} : p?l(x).P$ means there is only one branch $l(x)$. $X\langle e\rangle$ is for a statement variable with one parameter e, and $\mathsf{def}\ D\ \mathsf{in}\ \eta$ is for recursion, where declaration D defines the recursive body that can be called in η. The conditional statement is standard.

The structure of processes ensures that failure handling is not interleaved between different sessions. However, we note that in standard MPSTs [15,24], session interleaving must anyway be prohibited for the basic progress property. Since our aim will be to show progress, we disallow session interleaving within process bodies. Our model does allow parallel sessions at the top-level, whose actions may be concurrently interleaved during execution.

(Distributed) systems. A (distributed) *system* in our programming framework is a composition of an application, which contains more than one process, and a coordinator (cf. Fig. 1). A system can be running within a private session s, represented by $(\nu\mathsf{s})\mathcal{S}$, or $\mathcal{S} \mid \mathcal{S}'$ for systems running in different sessions independently and in parallel (i.e., no session interleaving). The job of the coordinator is to ensure that even in the presence of failures there is consensus on whether all participants in a given try-handle completed their local actions, or whether failures need to be handled, and which ones. We use $\Psi = G : (F, d)$ to denote a (robust) coordinator for the global type G, which stores in (F, d) the failures F that occurred in the application, and in d done notifications sent to the coordinator. The coordinator is denoted by ψ when viewed as a role.

A (distributed) *application*[6] is a process P, a parallel composition $N \mid N'$, or a global queue carrying messages $\mathsf{s} : h$. A global queue $\mathsf{s} : h$ carries a sequence of messages m, sent by participants in session s. A message is either a regular message $\langle p, q, l(v)\rangle$ with label l and content v sent from p to q or a *notification*. A notification may contain the role of a coordinator. There are *done* and *failure* notifications with two kinds of done notifications *dn* used for coordination: $\langle p, \psi\rangle^{\phi}$ notifies ψ that p has finished its local actions of the try-handle ϕ; $\langle \psi, p\rangle^{\phi}$ is sent from ψ to notify p that ψ has received all done notifications for the try-handle ϕ so that p shall end its current try-handle and

[6] Other works use the term *network* which is the reason why we use N instead of, e.g., A. We call it application to avoid confusion with the physical network which interconnects all processes as well as the coordinator.

$$a[p_1](y_1).P_1 \mid ... \mid a[p_n](y_n).P_n \to (\nu \mathsf{s})(G : (\emptyset,\emptyset)\blacklozenge P_1\{\mathsf{s}[p_1]/y_1\} \mid ... \mid P_n\{\mathsf{s}[p_n]/y_n\} \mid \mathsf{s} : \emptyset) \quad a : G \qquad \textbf{(Link)}$$

$$\mathsf{s}[p] : E[q!\ l(e).\eta] \mid \mathsf{s} : h \to \mathsf{s}[p] : E[\eta] \mid \mathsf{s} : h \cdot \langle p, q, l(v)\rangle \qquad e \Downarrow v \qquad \textbf{(Snd)}$$

$$\mathsf{s}[p] : E[q?\{l_i(x_i).\eta_i\}_{i\in I}] \mid \mathsf{s} : \langle q, p, l_k(v_k)\rangle \cdot h \to \mathsf{s}[p] : E[\eta_k\{v_k/x_k\}] \mid \mathsf{s} : h \quad k \in I \qquad \textbf{(Rcv)}$$

$$\mathsf{s}[p] : E[\mathsf{def}\ X(x) = \eta\ \mathsf{in}\ X\langle e\rangle] \to \mathsf{s}[p] : E[\mathsf{def}\ X(x) = \eta\ \mathsf{in}\ \eta\{v/x\}] \quad e \Downarrow v \qquad \textbf{(Rec)}$$

$$\frac{N_1 \equiv N_3 \to N_4 \equiv N_2}{N_1 \to N_2} \quad \frac{N_1 \to N_2}{N_1|N \to N_2|N} \qquad \textbf{(Str, Par)}$$

$$\frac{N_1 \to N_2}{\psi\blacklozenge N_1 \to \psi\blacklozenge N_2} \quad \frac{\mathcal{S} \to \mathcal{S}'}{(\nu \mathsf{s})\mathcal{S} \to (\nu \mathsf{s})\mathcal{S}'} \qquad \textbf{(Sys, New)}$$

$$N \mid \mathsf{s} : h \to N \setminus \mathsf{s}[p] : \eta \mid \mathsf{s} : \mathit{remove}(h, p) \cdot \langle\!\langle \psi, \mathsf{crash}\ \{p\}\rangle\!\rangle \quad \mathsf{s}[p] : \eta \text{ non-robust} \qquad \textbf{(Crash)}$$

Fig. 6. Operational semantics of distributed applications, for local actions.

move to its next task. For example, in Fig. 4 at time (4) the coordinator will inform P_b and P_c via $\langle \psi, P_b\rangle^{(\kappa,\{P_a,P_d\})}.\langle \psi, P_c\rangle^{(\kappa,\{P_a,P_d\})}$ that they can finish the try-handle $(\kappa, \{P_a, P_d\})$. Note that the appearance of $\langle \psi, p\rangle^{\phi}$ implies that the coordinator has been informed that all participants in ϕ have completed their local actions. We define two kinds of *failure* notifications: $\langle\!\langle \psi, \mathsf{crash}\ F\rangle\!\rangle$ notifies ψ that F occurred, e.g., $\{q\}$ means q has failed; $\langle\!\langle p, \mathsf{crash}\ F\rangle\!\rangle$ is sent from ψ to notify p about the failure F for possible handling. We write $\langle\!\langle \widetilde{p}, \mathsf{crash}\ F\rangle\!\rangle$, where $\widetilde{p} = p_1, ..., p_n$ short for $\langle\!\langle p_1, \mathsf{crash}\ F\rangle\!\rangle \cdot ... \cdot \langle\!\langle p_n, \mathsf{crash}\ F\rangle\!\rangle$; similarly for $\langle \psi, \widetilde{p}\rangle^{\phi}$. Following the tradition of other MPST works the global queue provides an abstraction for multiple FIFO queues, each queue being between two endpoints (cf. TCP) with no global ordering. Therefore $m_i \cdot m_j$ can be permuted to $m_j \cdot m_i$ in the global queue if the sender or the receiver differ. For example the following messages are permutable: $\langle p, q, l(v)\rangle \cdot \langle p, q', l(v)\rangle$ if $q \neq q'$ and $\langle p, q, l(v)\rangle \cdot \langle \psi, p\rangle^{\phi}$ and $\langle p, q, l(v)\rangle \cdot \langle\!\langle q, \mathsf{crash}\ F\rangle\!\rangle$. But $\langle \psi, p\rangle^{\phi} \cdot \langle\!\langle p, \mathsf{crash}\ F\rangle\!\rangle$ is not permutable, both have the same sender and receiver (ψ is the sender of $\langle\!\langle p, \mathsf{crash}\ F\rangle\!\rangle$). The formal definition of message permutation is in App. B Def. 13.

Basic dynamic semantics for applications. Fig. 6 shows the operational semantics of applications. We use evaluation contexts as defined in Fig. 5. Context E is either a hole $[\,]$, a default context $\mathsf{t}(E)\mathsf{h}(\mathsf{H})^{\phi}.\eta$, or a recursion context $\mathsf{def}\ D\ \mathsf{in}\ E$. We write $E[\eta]$ to denote the action statement obtained by filling the hole in $E[\cdot]$ with η.

Rule **(Link)** says that (local) processes who agree on shared name a, obeying to some protocol (global type), playing certain roles p_i represented by $a[p_i](y_i).P$, together will start a private session s; this will result in replacing every variable y_i in P_i and, at the same time, creating a new global queue $\mathsf{s} : \emptyset$, and appointing a coordinator $G : (\emptyset, \emptyset)$, which is novel in our work.

Rule **(Snd)** in Fig. 6 reduces a sending action $q!\ l(e)$ by emitting a message $\langle p, q, l(v)\rangle$ to the global queue $\mathsf{s} : h$. Rule **(Rcv)** reduces a receiving action if the

message arriving at its end is sent from the expected sender with an expected label. Rule **(Rec)** is for recursion. When the recursive body, defined inside η, is called by $X\langle e\rangle$ where e is evaluated to v, it reduces to the statement $\eta\{v/x\}$ which will again implement the recursive body. Rule **(Str)** says that processes which are structurally congruent have the same reduction. Processes, applications, and systems are considered modulo structural congruence, denoted by $\equiv$, along with α-renaming. Since the definition of structural congruence is mostly standard, we leave the full definition in App. B.2. Rule **(Par)** and **(Str)** together state that a parallel composition has a reduction if its sub-application can reduce. Rule **(Sys)** states that a system has a reduction if its application has a reduction, and **(New)** says a reduction can proceed under a session. Rule **(Crash)** states that a process on channel $\mathsf{s}[p]$ can fail at any point in time. **(Crash)** also adds a notification $\langle\!\langle \psi, \mathsf{crash}\ F \rangle\!\rangle$ which is sent to ψ (the coordinator). This is an abstraction for the failure detector described in Sec. 2 (5), the notification $\langle\!\langle \psi, \mathsf{crash}\ F \rangle\!\rangle$ is the first such notification issued by a participant based on its local failure detector. Adding the notification into the global queue instead of making the coordinator immediately aware of it models that failures are only detected eventually. Note that a failure is not annotated with a level because failures transcend all levels, and asynchrony makes it impossible to identify "where" exactly they occurred. As a failure is permanent it can affect multiple try-handles. The **(Crash)** rule does not apply to participants which are robust, i.e., that conceptually cannot fail (e.g., *dfs* in Fig. 2). Rule **(Crash)** removes channel $\mathsf{s}[p]$ (the failed process) from application N, and removes messages and notifications delivered from, or heading to, the failed p by function $\mathit{remove}(h, p)$. Function $\mathit{remove}(h, p)$ returns a new queue after removing all regular messages and notifications that contain p, e.g., let $h = \langle p_2, p_1, l(v)\rangle \cdot \langle p_3, p_2, l'(v')\rangle \cdot \langle p_3, p_4, l'(v')\rangle \cdot \langle p_2, \psi\rangle^{\phi} \cdot \langle\!\langle p_2, \mathsf{crash}\ \{p_3\}\rangle\!\rangle \cdot \langle \psi, p_2\rangle^{\phi}$ then $\mathit{remove}(h, p_2) = \langle p_3, p_4, l'(v')\rangle$. Messages are removed to model that in a real system send/receive does *not* constitute an atomic action.

Handling at processes. Failure handling, defined in Fig. 7, is based on the observations that (i) a process that fails stays down, and (ii) multiple processes can fail. As a consequence a failure can trigger multiple failure handlers either because these handlers are in different (subsequent) try-handles or because of additional failures. Therefore a process needs to retain the information of *who* failed. For simplicity we do not model state at processes, but instead processes read but do not remove failure notifications from the global queue. We define $\mathit{Fset}(h, p)$ to return the union of failures for which there are notifications heading to p, i.e., $\langle\!\langle p, \mathsf{crash}\ F \rangle\!\rangle$, issued by the coordinator in queue h *up to the first done notification heading to* p:

Definition 2 (Union of Existing Failures $\mathit{Fset}(h, p)$).

$$\mathit{Fset}(\emptyset, p) = \emptyset \quad \mathit{Fset}(h, p) = \begin{cases} F \cup \mathit{Fset}(h', p) & \text{if } h = \langle\!\langle p, \mathsf{crash}\ F \rangle\!\rangle \cdot h' \\ \emptyset & \text{if } h = \langle \psi, p\rangle^{\phi} \cdot h' \\ \mathit{Fset}(h', p) & \text{otherwise} \end{cases}$$

$$\frac{F' = \cup\{A \mid A \in dom(\mathsf{H}) \wedge F \subset A \subseteq Fset(h,p)\} \quad F' : \eta' \in \mathsf{H}}{\mathsf{s}[p] : E[\mathsf{t}(\eta)\mathsf{h}(\mathsf{H})^{(\kappa,F)}.\eta''] \mid \mathsf{s} : h \to \mathsf{s}[p] : E[\mathsf{t}(\eta')\mathsf{h}(\mathsf{H})^{(\kappa,F')}.\eta''] \mid \mathsf{s} : h} \quad \textbf{(TryHdl)}$$

$$\mathsf{s}[p] : E[\mathsf{t}(0)\mathsf{h}(\mathsf{H})^{\phi}.\eta] \mid \mathsf{s} : h \to \mathsf{s}[p] : E[\mathsf{t}(\underline{0})\mathsf{h}(\mathsf{H})^{\phi}.\eta] \mid \mathsf{s} : h \cdot \langle p, \psi\rangle^{\phi} \quad \textbf{(SndDone)}$$

$$\frac{\langle \psi, p\rangle^{\phi} \in h}{\mathsf{s}[p] : E[\mathsf{t}(\underline{0})\mathsf{h}(\mathsf{H})^{\phi}.\eta] \mid \mathsf{s} : h \to \mathsf{s}[p] : E[\eta] \mid \mathsf{s} : h \setminus \{\langle \psi, p\rangle^{\phi}\}} \quad \textbf{(RcvDone)}$$

$$\mathsf{s}[p] : E[\eta] \mid \mathsf{s} : \langle q, p, l(v)\rangle \cdot h \to \mathsf{s}[p] : E[\eta] \mid \mathsf{s} : h \quad l \notin labels(E[\eta]) \quad \textbf{(Cln)}$$

$$\frac{\langle \psi, p\rangle^{\phi} \in h \quad \phi \notin E[\eta]}{\mathsf{s}[p] : E[\eta] \mid \mathsf{s} : h \to \mathsf{s}[p] : E[\eta] \mid \mathsf{s} : h \setminus \langle \psi, p\rangle^{\phi}} \quad \textbf{(ClnDone)}$$

Fig. 7. Operational semantics of distributed applications, for endpoint handling.

In short, if the global queue is $\emptyset$, then naturally there are no failure notifications. If the global queue contains a failure notification sent from the coordinator, say $(\!|p, \mathsf{crash}\ F|\!)$, we collect the failure. If the global queue contains done notification $\langle \psi, p\rangle^{\phi}$ sent from the coordinator then *all* participants in ϕ have finished their local actions, which implies that the try-handle ϕ can be completed. Our failure handling semantics, **(TryHdl)**, allows a try-handle $\phi = (\kappa, F)$ to handle different failures or sets of failures by allowing a try-handle to switch between different handlers. F thus denotes the current set of handled failures. For simplicity we refer to this as the *current(ly handled) failure set*. This is a slight abuse of terminology, done for brevity, as obviously failures are only detected with a certain lag. The handling strategy for a process is to handle the — currently — largest set of failed processes that this process has been informed of and is able to handle. This largest set is calculated by $\cup\{A \mid A \in dom(\mathsf{H}) \wedge F \subset A \subseteq Fset(h,p)\}$, that selects all failure sets which are larger than the current one ($A \in dom(\mathsf{H}) \wedge F \subset A$) if they are also triggered by known failures ($A \subseteq Fset(h,p)$). Condition $F' : \eta' \in \mathsf{H}$ in **(TryHdl)** ensures that there exists a handler for F'. The following example shows how **(TryHdl)** is applied to switch handlers.

Example 2. Take h such that $Fset(h,p) = \{p_1\}$ and $\mathsf{H} = \{p_1\} : \eta_1, \{p_2\} : \eta_2, \{p_1, p_2\} : \eta_{12}$ in process $P = \mathsf{s}[p] : \mathsf{t}(\eta_1)\mathsf{h}(\mathsf{H})^{(\kappa,\{p_1\})}$, which indicates that P is handling failure $\{p_1\}$. Assume now one more failure occurs and results in a new queue h' such that $Fset(h',p) = \{p_1, p_2\}$. By **(TryHdl)**, the process acting at $\mathsf{s}[p]$ is handling the failure set $\{p_1, p_2\}$ such that $P = \mathsf{s}[p] : \mathsf{t}(\eta_{12})\mathsf{h}(\mathsf{H})^{(\kappa,\{p_1,p_2\})}$ (also notice the η_{12} inside the try-block). A switch to only handling $\{p_2\}$ does not make sense, since, e.g., η_2 can contain p_1. Fig. 2 shows a case where the handling strategy differs according to the number of failures.

In Sec. 3 we formally define well-formedness conditions, which guarantee that if there exist two handlers for two different handler signatures in a try-handle, then a handler exists for their union. The following example demonstrates why such a guarantee is needed.

Example 3. Assume a slightly different P compared to the previous examples (no handler for the union of failures): $P = \mathsf{s}[p] : E[\mathsf{t}(\eta)\mathsf{h}(\mathsf{H})^{(\kappa,\emptyset)}]$ with $\mathsf{H} =$

$\{p_1\} : \eta_1, \{p_2\} : \eta_2$. Assume also that $Fset(h, p) = \{p_1, p_2\}$. Here **(TryHdl)** will not apply since there is no failure handling for $\{p_1, p_2\}$ in P. If we would allow a handler for either $\{p_1\}$ or $\{p_2\}$ to be triggered we would have no guarantee that other participants involved in this try-handle will all select the same failure set. Even with a deterministic selection, i.e., all participants in that try-handle selecting the same handling activity, there needs to be a handler with handler signature $= \{p_1, p_2\}$ since it is possible that p_1 is involved in η_2. Therefore the type system will ensure that there is a handler for $\{p_1, p_2\}$ either at this level or at an outer level.

(I) explains that a process finishing its default action (P_b) cannot leave its current try-handle $(\kappa, \emptyset)$ immediately because other participants may fail (P_a failed). Below Eq. 1 also shows this issue from the perspective of semantics:

$$\mathsf{s}[p] : \mathsf{t}(0)\mathsf{h}(F : q!l(10).q?l'(x))^{(\kappa,\emptyset)}.\eta' \mid \mathsf{s}[q] : \mathsf{t}(p?l(x').p!l'(x'+10))\mathsf{h}(H)^{(\kappa,F)}.\eta'' \mid \mathsf{s} : \langle\!\langle q, \mathsf{crash}\ F \rangle\!\rangle \cdot \langle\!\langle p, \mathsf{crash}\ F \rangle\!\rangle \cdot h \quad (1)$$

In Eq. 1 the process acting on $\mathsf{s}[p]$ ended its try-handle (i.e., the action is 0 in the try-block), and if $\mathsf{s}[p]$ finishes its try-handle the participant acting on $\mathsf{s}[q]$ which started handling F would be stuck.

To solve the issue, we use **(SndDone)** and **(RcvDone)** for completing a local try-handle with the help of a coordinator. The rule **(SndDone)** sends out a done notification $\langle p, \psi \rangle^{\phi}$ if the current action in ϕ is 0 and sets the action to $\underline{0}$, indicating that a done notification from the coordinator is needed for ending the try-handle.

Assume process on channel $\mathsf{s}[p]$ finished its local actions in the try-block (i.e., as in Eq. 1 above), then by **(SndDone)**, we have

$$(1) \rightarrow \mathsf{s} : \langle\!\langle q, \mathsf{crash}\ F \rangle\!\rangle \cdot \langle\!\langle p, \mathsf{crash}\ F \rangle\!\rangle \cdot \langle p, \psi \rangle^{(\kappa,\emptyset)} \cdot h \mid \mathsf{s}[p] : \mathsf{t}(\underline{0})\mathsf{h}(F : q!l(10).q?l'(x))^{(\kappa,\emptyset)}.\eta' \mid \mathsf{s}[q] : \mathsf{t}(p?l(x').p!l'(x'+10))\mathsf{h}(H)^{(\kappa,F)}.\eta''$$

where notification $\langle p, \psi \rangle^{(\kappa,\emptyset)}$ is added to inform the coordinator. Now the process on channel $\mathsf{s}[p]$ can still handle failures defined in its handling environment. This is similar to the case described in (II).

Rule **(RcvDone)** is the counterpart of **(SndDone)**. Once a process receives a done notification for ϕ from the coordinator it can finish the try-handle ϕ and reduces to the continuation η. Consider Eq. 2 below, which is similar to Eq. 1 but we take a case where the try-handle can be reduced with **(RcvDone)**. In Eq. 2 **(SndDone)** is applied:

$$\begin{array}{l} \mathsf{s}[p] : \mathsf{t}(\underline{0})\mathsf{h}(F : q!l(10).q?l'(x))^{(\kappa,\emptyset)}.\eta' \mid \\ \mathsf{s}[q] : \mathsf{t}(\underline{0})\mathsf{h}(F : p?l(x').p!l'(x'+10))^{(\kappa,\emptyset)}.\eta'' \mid \mathsf{s} : h \end{array} \quad (2)$$

With $h = \langle \psi, q \rangle^{(\kappa,\emptyset)} \cdot \langle \psi, p \rangle^{(\kappa,\emptyset)} \cdot \langle\!\langle q, \mathsf{crash}\ F \rangle\!\rangle \cdot \langle\!\langle p, \mathsf{crash}\ F \rangle\!\rangle$ both processes can apply **(RcvDone)** and safely terminate the try-handle $(\kappa, \emptyset)$. Note that $Fset(h, p) =$

$$\frac{\tilde{p} = \mathit{roles}(G) \setminus F' \quad F' = F \cup \{p\} \quad m = \langle\!\langle \tilde{p}, \mathsf{crash}\ \{p\} \rangle\!\rangle}{G : (F, d)\blacklozenge N \ \mid\ \mathsf{s} : \langle\!\langle \psi, \mathsf{crash}\ \{p\} \rangle\!\rangle \cdot h \to G : (F', d)\blacklozenge N \ \mid\ \mathsf{s} : h \cdot m} \ (\mathbf{F})$$

$$\frac{d' = d \cdot \langle p, \psi \rangle^{\phi}}{G : (F, d)\blacklozenge \mathsf{s} : \langle p, \psi \rangle^{\phi} \cdot h \to G : (F, d')\blacklozenge \mathsf{s} : h} \ (\mathbf{CollectDone})$$

$$\frac{\mathit{roles}(d, \phi) \supseteq \mathit{roles}(G, \phi) \setminus F \quad \forall F' \in \mathit{hdl}(G, \phi).(F' \not\subseteq F)}{G : (F, d)\blacklozenge \mathsf{s} : h \to G : (F, \mathit{remove}(d, \phi))\blacklozenge \mathsf{s} : h \cdot \langle \psi, \mathit{roles}(G, \phi) \setminus F \rangle^{\phi}} \ (\mathbf{IssueDone})$$

Fig. 8. Operational semantics for the coordinator.

$\mathit{Fset}(h, q) = \emptyset$ (by Def. 2), i.e., rule **(TryHdl)** can not be applied since a done notification suppresses the failure notification. Thus Eq. 2 will reduce to:

$$(2) \to^* \mathsf{s}[p] : \eta' \ \mid\ \mathsf{s}[q] : \eta'' \ \mid\ \mathsf{s} : \langle\!\langle q, \mathsf{crash}\ F \rangle\!\rangle \cdot \langle\!\langle p, \mathsf{crash}\ F \rangle\!\rangle$$

It is possible that η' or η'' have handlers for F. Note that once a queue contains $\langle \psi, p \rangle^{(\kappa, \emptyset)}$, all non-failed process in the try-handle $(\kappa, \emptyset)$ have sent done notifications to ψ (i.e. applied rule **(SndDone)**). The coordinator which will be introduced shortly ensures this.

Rule **(Cln)** removes a normal message from the queue if the label in the message does not exist in the target process, which can happen when a failure handler was triggered. The function $\mathit{labels}(\eta)$ returns all labels of receiving actions in η which are able to receive messages now or possible later. (The function is formally defined in App. B. Def. 14). This removal based on the syntactic process is safe because in a global type separate branch types *not* defined in the same default block or handler body must have disjoint sets of labels (c.f., Sec. 3). Let $\phi \in P$ if try-handle ϕ appears inside P. Rule **(ClnDone)** removes a done notification of ϕ from the queue if no try-handle ϕ exists, which can happen in case of nesting when a handler of an outer try-handle is triggered.

Handling at coordinator. Fig. 8 defines the semantics of the coordinator. We firstly give the auxiliary definition of $\mathit{roles}(G)$ which gives the set of *all* roles appearing in G.

In rule **(F)**, F represents the failures that the coordinator is aware of. This rule states that the coordinator collects and removes a failure notification $\langle\!\langle \psi, \mathsf{crash}\ p \rangle\!\rangle$ heading to it, retains this notification by $G : (F', d)$, $F' = F \cup \{p\}$, and issues failure notifications to all non-failed participants.

Rules **(CollectDone, IssueDone)**, in short inform all participants in $\phi = (\kappa, F)$ to finish their try-handle ϕ if the coordinator has received sufficient done notifications of ϕ and did not send out failure notifications that interrupt the task (κ, F) (e.g. see (III)). Rule **(CollectDone)** collects done notifications, i.e., $\langle p, \psi \rangle^{\phi}$, from the queue and retains these notification; they are used in **(IssueDone)**. For introducing **(IssueDone)**, we first introduce $\mathit{hdl}(G, (\kappa, F))$ to return a set of handler signatures which can be triggered with respect to the current handler:

Definition 3. $\mathit{hdl}(G, (\kappa, F)) = \mathit{dom}(H) \setminus \mathcal{P}(F)$ if $\mathsf{t}(G_0)\mathsf{h}(H)^{\kappa} \in G$ where $\mathcal{P}(F)$ represents a powerset of F.

Also, we abuse the function *roles* to collect the non-coordinator roles of ϕ in d, written $\mathit{roles}(d, \phi)$; similarly, we write $\mathit{roles}(G, \phi)$ where $\phi = (\kappa, F)$ to collect the roles appearing in the handler body F in the try-handle of κ in G. Remember that d only contains done notifications sent by participants.

Rule (**IssueDone**) is applied for some ϕ when conditions $\forall F' \in \mathit{hdl}(G, \phi).(F' \not\subseteq F)$ and $\mathit{roles}(d, \phi) \supseteq \mathit{roles}(G, \phi) \setminus F$ are both satisfied, where F contains all failures the coordinator is aware of. Intuitively, these two conditions ensure that (1) the coordinator only issues done notifications to the participants in the try-handle ϕ if it did not send failure notifications which will trigger a handler of the try-handle ϕ; (2) the coordinator has received all done notifications from all non-failed participants of ϕ. We further explain both conditions in the following examples, starting from condition $\forall F' \in \mathit{hdl}(G, \phi).(F' \not\subseteq F)$, which ensures no handler in ϕ can be triggered based on the failure notifications F sent out by the coordinator.

Example 4. Assume a process playing role p_i is $P_i = \mathsf{s}[p_i] : \mathsf{t}(\eta_i)\mathsf{h}(\mathsf{H}_i)^{\phi_i}$. where $i \in \{1, 2, 3\}$ and $\mathsf{H}_i = \{p_2\} : \eta_{i2},\ \{p_3\} : \eta_{i3},\ \{p_2, p_3\} : \eta_{i23}$ and the coordinator is $G : (\{p_2, p_3\}, d)$ where $\mathsf{t}(...)\mathsf{h}(H)^{\kappa} \in G$ and $\mathit{dom}(H) = \mathit{dom}(\mathsf{H}_i)$ for any $i \in \{1, 2, 3\}$ and $d = \langle p_1, \psi\rangle^{(\kappa, \{p_2\})} \cdot \langle p_1, \psi\rangle^{(\kappa, \{p_2, p_3\})} \cdot d'$. For any ϕ in d, the coordinator checks if it has issued any failure notification that can possibly trigger a new handler of ϕ:

1. For $\phi = (\kappa, \{p_2\})$ the coordinator issued failure notifications that can interrupt a handler since

$$\mathit{hdl}(G, (\kappa, \{p_2\})) = \mathit{dom}(H) \setminus \mathcal{P}(\{p_2\}) = \{\{p_3\}, \{p_2, p_3\}\}$$

   and $\{p_2, p_3\} \subseteq \{p_2, p_3\}$. That means the failure notifications issued by the coordinator, i.e., $\{p_2, p_3\}$, can trigger the handler with signature $\{p_2, p_3\}$. Thus the coordinator will not issue done notifications for $\phi = (\kappa, \{p_2\})$. A similar case is visualized in Fig. 4 at time (2).
2. For $\phi = (\kappa, \{p_2, p_3\})$ the coordinator did not issue failure notifications that can interrupt a handler since

$$\mathit{hdl}(G, (\kappa, \{p_2, p_3\})) = \mathit{dom}(H) \setminus \mathcal{P}(\{p_2, p_3\}) = \emptyset$$

   so that $\forall F' \in \mathit{hdl}(G, (\kappa, \{p_2, p_3\})).(F' \not\subseteq \{p_2, p_3\})$ is true. The coordinator will issue done notifications for $\phi = (\kappa, \{p_2, p_3\})$.

Another condition $\mathit{roles}(d, \phi) \supseteq \mathit{roles}(G, \phi) \setminus F$ states that only when the coordinator sees sufficient done notifications (in d) for ϕ, it issues done notifications to *all* non-failed participants in ϕ, i.e., $\langle \psi, \mathit{roles}(G, \phi) \setminus F\rangle^{\phi}$. Recall that $\mathit{roles}(d, \phi)$ returns all roles which have sent a done notification for the handling of ϕ and $\mathit{roles}(G, \phi)$ returns all roles involving in the handling of ϕ. Intuitively one might expect the condition to be $\mathit{roles}(d, \phi) = \mathit{roles}(G, \phi)$; the following example shows why this would be wrong.

$$T ::= p!\{l_i(S_i).T_i\}_{i\in I} \mid p?\{l_i(S_i).T_i\}_{i\in I} \mid t \mid \mu t.T \mid \mathsf{end} \mid \underline{\mathsf{end}} \mid \mathsf{t}(T)\mathsf{h}(\mathcal{H})^{\phi}.T$$
$$\mathcal{H} ::= F : T \mid \mathcal{H}, \mathcal{H}$$

Fig. 9. The grammar of local types.

Example 5. Consider a process P acting on channel $\mathsf{s}[p]$ and $\{q\} \notin dom(\mathsf{H})$:

$$P = \mathsf{s}[p] : \mathsf{t}(...\mathsf{t}(...)\mathsf{h}(\{q\} : \eta, \mathsf{H}')^{\phi'}.\eta')\mathsf{h}(\mathsf{H})^{\phi}$$

Assume P has already reduced to:

$$P = \mathsf{s}[p] : \mathsf{t}(\underline{0})\mathsf{h}(\mathsf{H})^{\phi}$$

We show why $roles(d, \phi) \supseteq roles(G, \phi) \setminus F$ is necessary. We start with the simple cases and then move to the more involving ones.

(a) Assume q did not fail, the coordinator is $G : (\emptyset, d)$, and all roles in ϕ issued a done notification. Then $roles(d, \phi) = roles(G, \phi)$ and $F = \emptyset$.
(b) Assume q failed in the try-handle ϕ', the coordinator is $G : (\{q\}, d)$, and all roles except q in ϕ issued a done notification. $roles(d, \phi) \neq roles(G, \phi)$ however $roles(d, \phi) = roles(G, \phi) \setminus \{q\}$. Cases like this are the reason why **(IssueDone)** only requires done notifications from non-failed roles.
(c) Assume q failed after it has issued a done notification for ϕ (i.e., q finished try-handle ϕ') and the coordinator collected it (by **(CollectDone)**), so we have $G : (\{q\}, d)$ and $q \in roles(b, \phi)$. Then $roles(d, \phi) \supset roles(G, \phi) \setminus \{q\}$. i.e. **(IssueDone)** needs to consider done notifications from failed roles.

Thus rule **(IssueDone)** has the condition $roles(d, \phi) \supseteq roles(G, \phi) \setminus F$ because of cases like (b) and (c).

The interplay between issuing of done notification **(IssueDone)** and issuing of failure notifications **(F)** is non-trivial. The following proposition clarifies that the participants in the same try-handle ϕ will never get confused with handling failures or completing the try-handle ϕ.

Proposition 1. *Given $\mathsf{s} : h$ with $h = h' \cdot \langle \psi, p \rangle^{\phi} \cdot h''$ and $Fset(h, p) \neq \emptyset$, the rule* **(TryHdl)** *is not applicable for the try-handle ϕ at the process playing role p.*

Proof see App. D.1

5 Local Types

Fig. 9 defines local types for typing behaviors of endpoint processes with failure handling. Type $p!$ is the primitive for a sending type, and $p?$ is the primitive for a receiving type, derived from global type $p \to q\{l_i(S_i).G_i\}_{i\in I}$ by projection. Others correspond straightforwardly to process terms. Note that type $\underline{\mathsf{end}}$ only appears in *runtime* type checking. Below we define $G\upharpoonright p$ to project a global type G on p, thus generating p's local type.

Definition 4 (Projection). Consider a well-formed top-level global type $[\tilde{q}]G$. Then $G\upharpoonright p$ is defined as follows:

(1) $G\upharpoonright p$ where $G = \mathsf{t}(G_0)\mathsf{h}(F_1 : G_1, ..., F_n : G_n)^{\kappa}.G' =$

$$\begin{cases} \mathsf{t}(G_0\upharpoonright p)\mathsf{h}(F_1 : G_1\upharpoonright p, ..., F_n : G_n\upharpoonright p)^{(\kappa,\emptyset)}.G'\upharpoonright p & \text{if } p \in \mathit{roles}(G) \\ G'\upharpoonright p & \text{otherwise} \end{cases}$$

(2) $p_1 \to p_2\{l_i(S_i).G_i\}_{i\in I}\upharpoonright p = \begin{cases} p_2!\{l_i(S_i).G_i\upharpoonright p\}_{i\in I} & \text{if } p = p_1 \\ p_1?\{l_i(S_i).G_i\upharpoonright p\}_{i\in I} & \text{if } p = p_2 \\ G_1\upharpoonright p & \text{if } \forall i, j \in I.G_i\upharpoonright p = G_j\upharpoonright p \end{cases}$

(3) $(\mu t.G)\upharpoonright p = \mu t.(G\upharpoonright p)$ if $\nexists \mathsf{t}(G')\mathsf{h}(H) \in G$ and $G\upharpoonright p \neq t'$ for any t'

(4) $t\upharpoonright p = t$ (5) $\mathsf{end}\upharpoonright p = \mathsf{end}$

Otherwise it is undefined.

The main rule is (1): if p appears somewhere in the target try-handle global type then the endpoint type has a try-handle annotated with κ and the default logic (i.e., $F = \emptyset$). Note that even if $G_0\upharpoonright p = \mathsf{end}$ the endpoint still gets such a try-handle because it needs to be ready for (possible) failure handling; if p does not appear anywhere in the target try-handle global type, then the projection skips to the continuation.

Rule (2) produces local types for interaction endpoints. If the endpoint is a sender (i.e., $p = p_1$), then its local type abstracts that it will send something from one of the possible internal choices defined in $\{l_i(S_i)\}_{i\in I}$ to p_2, then continue as $G_k\upharpoonright p$, gained from the projection, if $k \in I$ is chosen. If the endpoint is a receiver (i.e., $p = p_2$), then its local type abstracts that it will receive something from one of the possible external choices defined in $\{l_i(S_i)\}_{i\in I}$ sent by p_1; the rest is similarly as for the sender. However, if p is not in this interaction, then its local type starts from the next interaction which p is in; moreover, because p does not know what choice that p_1 has made, every path $G_i\upharpoonright p$ lead by branch l_i shall be the same for p to ensure that interactions are consistent. For example, in $G = p_1 \to p_2\{l_1(S_1).p_3 \to p_1\ l_3(S),\ l_2(S_2).p_3 \to p_1\ l_4(S)\}$, interaction $p_3 \to p_1$ continues after $p_1 \to p_2$ takes place. If $l_3 \neq l_4$, then G is not projectable for p_3 because p_3 does not know which branch that p_1 has chosen; if p_1 chose branch l_1, but p_3 (blindly) sends out label l_4 to p_1, for p_1 it is a mistake (but it is not a mistake for p_3) because p_1 is expecting to receive label l_3. To prevent such inconsistencies, we adopt the projection algorithm proposed in [24]. Other session type works [17,40] provide ways to weaken the classical restriction on projection of branching which we use.

Rule (3) forbids a try-handle to appear in a recursive body, e.g., $\mu t.\mathsf{t}(G)\mathsf{h}(F : t)^{\kappa}.G$ is not allowed, but $\mathsf{t}(\mu t.G)\mathsf{h}(H)^{\kappa}$ and $\mathsf{t}(G)\mathsf{h}(F : \mu t.G', H)^{\kappa}$ are allowed. This is because κ is used to avoid confusion of messages from different try-handles. If a recursive body contains a try-handle, we have to dynamically generate different levels to maintain interaction consistency, so static type checking does not suffice.

We are investigating alternative runtime checking mechanisms, but this is beyond the scope of this paper. Other rules are straightforward.

Example 6. Recall the global type G from Fig. 2 in Sec. 1. Applying projection rules defined in Def. 4 to G on every role in G we obtain the following:

$$\begin{aligned}
T_{dfs} = G\upharpoonright dfs &= \mathsf{t}(\mu t.w_1!l_{d_1}(S).w_2!l_{d_2}(S).w_1?l_{r_1}(S').w_2?l_{r_2}(S').t)\mathsf{h}(\mathcal{H}_{dfs})^{(1,\emptyset)}\\
\mathcal{H}_{dfs} &= \{w_1\}:\mu t'.w_2!l'_{d_1}(S).w_2?l'_{r_1}(S').t',\\
&\quad \{w_2\}:\mu t''.w_1!l'_{d_2}(S).w_1?l'_{r_2}(S').t'', \{w_1, w_2\}:\mathsf{end}\\
T_{w_1} = G\upharpoonright w_1 &= \mathsf{t}(\mu t.dfs?l_{d_1}(S).dfs!l_{r_1}(S').t)\mathsf{h}(\mathcal{H}_{w_1})^{(1,\emptyset)}\\
\mathcal{H}_{w_1} &= \{w_1\}:\mathsf{end}, \{w_2\}:\mu t'.dfs?l'_{d_2}(S).dfs!l'_{r_2}(S').t', \{w_1, w_2\}:\mathsf{end}\\
T_{w_2} = G\upharpoonright w_2 &= \mathsf{t}(\mu t.dfs?l_{d_2}(S).dfs!l_{r_2}(S').t)\mathsf{h}(\mathcal{H}_{w_2})^{(1,\emptyset)}\\
\mathcal{H}_{w_2} &= \{w_1\}:\mu t''.dfs?l'_{d_1}(S).dfs!l'_{r_1}(S').t'', \{w_2\}:\mathsf{end}, \{w_1, w_2\}:\mathsf{end}
\end{aligned}$$

6 Type System

Next we introduce our type system for typing processes. Fig. 10 and Fig. 11 present typing rules for endpoints processes, and typing judgments for applications and systems respectively.

We define shared environments Γ to keep information on variables and the coordinator, and session environments Δ to keep information on endpoint types:

$$\begin{aligned}
&\Gamma ::= \emptyset \mid \Gamma, X : S\ T \mid \Gamma, x : S \mid \Gamma, a : G \mid \Gamma, \Psi \qquad \Delta ::= \emptyset \mid \Delta, \mathsf{c} : T \mid \Delta, \mathsf{s} : \mathtt{h}\\
&\mathtt{m} ::= \langle p, q, l(S)\rangle \mid \langle\!\langle p, \mathsf{crash}\ F\rangle\!\rangle \mid \langle p, q\rangle^{\phi} \qquad \mathtt{h} ::= \emptyset \mid \mathtt{h}\cdot\mathtt{m}
\end{aligned}$$

Γ maps process variables X and content variables x to their types, shared names a to global types G, and a coordinator $\Psi = G : (F, d)$ to failures and done notifications it has observed. Δ maps session channels c to local types and session queues to queue types. We write $\Gamma, \Gamma' = \Gamma \cup \Gamma'$ when $dom(\Gamma) \cap dom(\Gamma') = \emptyset$; same for Δ, Δ'. Queue types $\mathtt{h}$ are composed of message types $\mathtt{m}$. Their permutation is defined analogously to the permutation for messages. The typing judgment for local processes $\Gamma \vdash P \triangleright \Delta$ states that process P is well-typed by Δ under Γ.

Since we do not define sequential composition for processes, our type system implicitly forbids session interleaving by $\lfloor$`T-ini`$\rfloor$. This is different from other session type works [15,24], where session interleaving is prohibited for the progress property; here the restriction is inherent to the type system.

Fig. 10 lists our typing rules for endpoint processes. Rule $\lfloor$`T-ini`$\rfloor$ says that if a process's set of actions is well-typed by $G\upharpoonright p$ on some c, this process can play role p in a, which claims to have interactions obeying behaviors defined in G. $\langle G\rangle$ means that G is closed, i.e., devoid of type variables. This rule forbids $a[p].b[q].P$ because a process can only use one session channel. Rule $\lfloor$`T-snd`$\rfloor$ states that an action for sending is well-typed to a sending type if the label and the type of the content are expected; $\lfloor$`T-rcv`$\rfloor$ states that an action for branching (i.e., for receiving) is well-typed to a branching type if all labels and the types of contents are as expected. Their follow-up actions shall also be well-typed. Rule $\lfloor$`T-0`$\rfloor$ types an idle process. Predicate end-only Δ is defined as stating whether all endpoints in Δ have type end:

$$\frac{\Gamma \vdash\ a : \langle G \rangle \quad \Gamma \vdash\ P \rhd\ \{\mathsf{c} : G\lfloor p \rfloor\}}{\Gamma \vdash\ a[p].P \rhd\ \emptyset} \qquad \frac{k \in I \quad \Gamma \vdash\ e : S_k \quad \Gamma\ \vdash\ \mathsf{c} : \eta_k \rhd\ \{\mathsf{c} : T_k\}}{\Gamma \vdash \mathsf{c} : p!\ l_k(e).\eta_k \rhd\ \{\mathsf{c} : p!\ \{l_i(S_i).T_i\}_{i \in I}\}} \quad \lfloor \texttt{T-ini/T-snd} \rfloor$$

$$\frac{\forall i \in I.\ \Gamma, x_i : S_i\ \vdash\ \mathsf{c} : \eta_i \rhd\ \{\mathsf{c} : T_i\}}{\Gamma\ \vdash\ \mathsf{c} : p?\ \{l_i(x_i).\eta_i\}_{i \in I} \rhd\ \{\mathsf{c} : p?\ \{l_i(S_i).T_i\}_{i \in I}\}} \quad \lfloor \texttt{T-rcv} \rfloor$$

$$\frac{\Delta \text{ end-only}}{\Gamma \vdash\ \mathsf{c} : 0 \rhd\ \Delta} \quad \frac{\Gamma\ \vdash\ \mathsf{c} : \eta \rhd\ \{\mathsf{c} : \mathsf{end}\}}{\Gamma\ \vdash\ \mathsf{c} : \underline{0}.\eta \rhd\ \{\mathsf{c} : \underline{\mathsf{end}}.\mathsf{end}\}} \quad \lfloor \texttt{T-0/T-yd} \rfloor$$

$$\frac{\Gamma \vdash\ e : \mathsf{bool} \quad \forall i \in \{1,2\}.\ \Gamma\ \vdash\ \mathsf{c} : \eta_i \rhd\ \Delta}{\Gamma\ \vdash\ \mathsf{c} : \mathsf{if}\ e\ \eta_1\ \mathsf{else}\ \eta_2 \rhd\ \Delta} \qquad \frac{\Gamma \vdash\ e : S}{\Gamma, X : S\ T\ \vdash\ \mathsf{c} : X\langle e \rangle \rhd\ \{\mathsf{c} : T\}} \quad \lfloor \texttt{T-if/T-var} \rfloor$$

$$\frac{\Gamma, X : S\ t, x : S\ \vdash\ \mathsf{c} : \eta_1 \rhd\ \{\mathsf{c} : T'\} \quad \Gamma, X : S\ \mu t.T'\ \vdash\ \mathsf{c} : \eta_2 \rhd\ \{\mathsf{c} : T\}}{\Gamma\ \vdash\ \mathsf{c} : \mathsf{def}\ X(x) = \eta_1\ \mathsf{in}\ \eta_2 \rhd\ \{\mathsf{c} : T\}} \quad \lfloor \texttt{T-def} \rfloor$$

$$\frac{\Gamma\ \vdash\ \mathsf{c} : \eta \rhd\ \{\mathsf{c} : T\} \quad \Gamma\ \vdash\ \mathsf{c} : \eta' \rhd\ \{\mathsf{c} : T'\} \quad dom(\mathsf{H}) = dom(\mathcal{H}) \quad \forall F \in dom(\mathsf{H}).\ \Gamma\ \vdash\ \mathsf{c} : \mathsf{H}(F) \rhd\ \{\mathsf{c} : \mathcal{H}(F)\}}{\Gamma\ \vdash\ \mathsf{c} : \mathsf{t}(\eta)\mathsf{h}(\mathsf{H})^{\phi}.\eta' \rhd\ \{\mathsf{c} : \mathsf{t}(T)\mathsf{h}(\mathcal{H})^{\phi}.T'\}} \quad \lfloor \texttt{T-th} \rfloor$$

Fig. 10. Typing rules for processes

Definition 5 (End-only Δ). We say Δ is end-only if and only if $\forall \mathsf{s}[p] \in dom(\Delta)$, $\Delta(\mathsf{s}[p]) = \mathsf{end}$.

Rule $\lfloor$T-yd$\rfloor$ types yielding actions, which only appear at runtime. Rule $\lfloor$T-if$\rfloor$ is standard in the sense that the process is well-typed by Δ if e has boolean type and its sub-processes (i.e., η_1 and η_2) are well-typed by Δ. Rules $\lfloor$T-var,T-def$\rfloor$ are based on a recent summary of MPSTs [14]. Note that $\lfloor$T-def$\rfloor$ forbids the type $\mu t.t$. Rule $\lfloor$T-th$\rfloor$ states that a try-handle is well-typed if it is annotated with the expected level ϕ, its default statement is well-typed, $\mathcal{H}$ and H have the same handler signatures, and all handling actions are well-typed.

Fig. 11 shows typing rules for applications and systems. Rule $\lfloor$T-$\emptyset\rfloor$ types an empty queue. Rules $\lfloor$T-m,T-D,T-$F\rfloor$ simply type messages based on their shapes. Rule $\lfloor$T-pa$\rfloor$ says two applications composed in parallel are well-typed if they do not share any session channel. Rule $\lfloor$T-s$\rfloor$ says a part of a system $\mathcal{S}$ can start a private session, say s, if $\mathcal{S}$ is well-typed according to a $\Gamma \vdash\ \Delta_{\mathsf{s}}$ that is *coherent* (defined shortly). The system $(\nu \mathsf{s})\mathcal{S}$ with a part becoming private in s is well-typed to $\Delta \setminus \Delta_{\mathsf{s}}$, that is, Δ after removing Δ_{s}.

Definition 6 (A Session Environment Having s Only: Δ_{s}).

$$\Delta_{\mathsf{s}} = \{\mathsf{s}[p] : T \mid \mathsf{s}[p] \in dom(\Delta)\} \cup \{\mathsf{s} : \mathtt{h}\ \mid\ \mathsf{s} \in dom(\Delta)\}$$

Rule $\lfloor$T-sys$\rfloor$ says that a system $\Psi \blacklozenge N$ is well-typed if application N is well-typed and there exists a coordinator Ψ for handling this application. We say $\Gamma \vdash\ \Delta$ is coherent under Γ if the local types of all endpoints are dual to each other after their local types are updated because of messages or notifications in $s : \mathtt{h}$.

$$\Gamma \vdash \mathtt{s} : \emptyset \triangleright \{\mathtt{s} : \emptyset\} \quad \frac{\Gamma \vdash \mathtt{s} : h \triangleright \{\mathtt{s} : \mathtt{h}\} \quad \Gamma \vdash e : S}{\Gamma \vdash \mathtt{s} : h \cdot \langle p, q, l(e)\rangle \triangleright \{\mathtt{s} : \mathtt{h} \cdot \langle p, q, l(S)\rangle\}} \quad \lfloor \texttt{T-}\emptyset\texttt{/T-m} \rfloor$$

$$\frac{(p_1, p_2) \in \{(p, \psi), (\psi, p)\} \quad \Gamma \vdash \mathtt{s} : h \triangleright \{\mathtt{s} : \mathtt{h}\}}{\Gamma \vdash \mathtt{s} : h \cdot \langle p_1, p_2\rangle^{\phi} \triangleright \{\mathtt{s} : \mathtt{h} \cdot \langle p_1, p_2\rangle^{\phi}\}} \quad \lfloor \texttt{T-D} \rfloor$$

$$\frac{p \in \{q, \psi\} \quad \mathtt{m} = \langle\!\langle p, \mathsf{crash}\ F\rangle\!\rangle \quad \Gamma \vdash \mathtt{s} : h \triangleright \{\mathtt{s} : \mathtt{h}\}}{\Gamma \vdash \mathtt{s} : h \cdot \langle\!\langle p, \mathsf{crash}\ F\rangle\!\rangle \triangleright \{\mathtt{s} : \mathtt{h} \cdot \mathtt{m}\}} \quad \frac{\Gamma \vdash N_1 \triangleright \Delta_1 \quad \Gamma \vdash N_2 \triangleright \Delta_2 \quad dom(\Delta_1) \cap dom(\Delta_2) = \emptyset}{\Gamma \vdash N_1 \mid N_2 \triangleright \Delta_1, \Delta_2} \quad \lfloor \texttt{T-}F\texttt{/T-pa} \rfloor$$

$$\frac{\Gamma \vdash \mathcal{S} \triangleright \Delta \quad \Gamma \vdash \Delta_{\mathsf{s}} \text{ coherent}}{\Gamma \vdash (\nu \mathsf{s})\mathcal{S} \triangleright \Delta \setminus \Delta_{\mathsf{s}}} \quad \frac{\Gamma' = \Gamma, \Psi \quad \Gamma \vdash N \triangleright \Delta}{\Gamma' \vdash \Psi \blacklozenge N \triangleright \Delta} \quad \lfloor \texttt{T-s/T-sys} \rfloor$$

Fig. 11. Typing rules for applications and systems.

Coherence. We say that a session environment is *c*oherent if, at any time, given a session with its latest messages and notifications, every endpoint participating in it is able to find someone to interact with (i.e., its dual party exists) right now or afterwards.

Example 7. Continuing with Example 6 – the session environment $\Gamma \vdash \ \Delta$ is coherent even if w_2 will not receive any message from *dfs* at this point. The only possible action to take in Δ is that *dfs* sends out a message to w_1. When this action fires, Δ is reduced to Δ' under a coordinator. (The reduction relation $\Gamma \vdash \ \Delta \to_T \Gamma' \vdash \ \Delta'$, where $\Gamma = \Gamma_0, \Psi$ and $\Gamma' = \Gamma_0, \Psi'$, is defined based on the rules of operational semantics of applications in Sec. 4, Fig. 6 and Fig. 7. Due to space limitations we put the complete rules into App. D.2). In Δ', which abstracts the environment when *dfs* sends a message to w_1, w_2 will be able to receive this message.

$$\begin{aligned}\Delta &= \mathsf{s}[dfs] : T_{dfs},\ \mathsf{s}[w_1] : T_{w_1},\ \mathsf{s}[w_2] : T_{w_2}, \mathsf{s} : \emptyset \\ \Delta' &= \mathsf{s}[dfs] : \mathsf{t}(w_2!l_{d_2}(S).w_1?l_{r_1}(S').w_2?l_{r_2}(S').T)\mathsf{h}(\mathcal{H})^{(1,\emptyset)}, \\ &\quad\ \mathsf{s}[w_1] : T_{w_1},\ \mathsf{s}[w_2] : T_{w_2}, \mathsf{s} : \langle dfs, w_1, l_{d_1}(S)\rangle \\ &\qquad \text{where } T = \mu t.w_1!l_{d_1}(S).w_2!l_{d_2}(S).w_1?l_{r_1}(S').w_2?l_{r_2}(S').t\end{aligned}$$

We write $\mathsf{s}[p] : T \bowtie \mathsf{s}[q] : T'$ to state that actions of the two types are *dual*:

Definition 7 (Duality). We define $\mathsf{s}[p] : T \bowtie \mathsf{s}[q] : T'$ as follows:

$$\mathsf{s}[p] : \mathsf{end} \bowtie \mathsf{s}[q] : \mathsf{end} \quad \mathsf{s}[p] : \underline{\mathsf{end}} \bowtie \mathsf{s}[q] : \underline{\mathsf{end}} \quad \mathsf{s}[p] : \mathsf{end} \bowtie \mathsf{s}[q] : \underline{\mathsf{end}}$$

$$\mathsf{s}[p] : \underline{\mathsf{end}} \bowtie \mathsf{s}[q] : \mathsf{end} \quad \mathsf{s}[p] : t \bowtie \mathsf{s}[q] : t \quad \frac{\mathsf{s}[p] : T \bowtie \mathsf{s}[q] : T'}{\mathsf{s}[p] : \mu t.T \bowtie \mathsf{s}[q] : \mu t.T'}$$

$$\frac{\forall i \in I.\ \mathsf{s}[p] : T_i \bowtie \mathsf{s}[q] : T_i'}{\mathsf{s}[p] : q!\ \{l_i(S_i).T_i\}_{i\in I} \bowtie \mathsf{s}[q] : p?\ \{l_i(S_i).T_i'\}_{i\in I}}$$

$$\frac{\mathsf{s}[p] : T_1 \bowtie \mathsf{s}[q] : T_2 \quad \mathsf{s}[p] : T_1' \bowtie \mathsf{s}[q] : T_2' \quad dom(\mathcal{H}_1) = dom(\mathcal{H}_2) \quad \forall F \in dom(\mathcal{H}_1).\ \mathsf{s}[p] : \mathcal{H}_1(F) \bowtie \mathsf{s}[q] : \mathcal{H}_2(F)}{\mathsf{s}[p] : \mathsf{t}(T_1)\mathsf{h}(\mathcal{H}_1)^{\phi}.T_1' \bowtie \mathsf{s}[q] : \mathsf{t}(T_2)\mathsf{h}(\mathcal{H}_2)^{\phi}.T_2'}$$

Due to space limitations, we provide the following formal definitions in App. B.3. Operation $T \lfloor p$ is to filter T to get the partial type which only contains actions of p. For example, $p_1!l'(S').p_2!l(S) \lfloor p_2 = p_2!l(S)$ and $p_1!\{T_1, T_2\} \lfloor p_2 = p_2?l(S)$ where $T_1 = l_1(S_1).p_2?l(S)$ and $T_2 = l_2(S_2).p_2?l(S)$. Next we define $(\mathtt{h})_{p \to q}$ to filter $\mathtt{h}$ to generate (1) the normal message types sent from p heading to q, and (2) the notifications heading to q. For example $(\langle p, q, l(S)\rangle \cdot \langle\!\langle q, \mathsf{crash}\ F\rangle\!\rangle \cdot \langle \psi, q\rangle^{\phi} \cdot \langle\!\langle p, \mathsf{crash}\ F\rangle\!\rangle)_{p \to q} = p?l(S) \cdot \langle\!\langle F\ \rangle\!\rangle \cdot \langle\psi\rangle^{\phi}$. The message types are abbreviated to contain only necessary information.

We define T–$\mathtt{ht}$ to mean the effect of $\mathtt{ht}$ on T. Its concept is similar to the *session remainder* defined in [36], which returns new local types of participants after participants consume messages from the global queue. Since failure notifications will not be consumed in our system, and we only have to observe the change of a participant's type after receiving or being triggered by some message types in $\mathtt{ht}$, we say that T–$\mathtt{ht}$ represents the effect of $\mathtt{ht}$ on T. The behaviors follows our operational semantics of applications and systems defined in Fig. 6, Fig. 7, and Fig. 8. For example $\mathtt{t}(q?\{l_i(S_i).T_i\}_{i \in I})\mathsf{h}(\mathcal{H})^{\phi}.T'$–$q?l_k(S_k) \cdot \mathtt{ht} = \mathtt{t}(T_k)\mathsf{h}(\mathcal{H})^{\phi}.T'$–$\mathtt{ht}$ where $k \in I$.

Now we define what it means for Δ to be coherent under Γ:

Definition 8 (Coherence). $\Gamma \vdash \Delta$ coherent if the following conditions hold:

1. If $\mathtt{s} : \mathtt{h} \in \Delta$, then $\exists G : (F, d) \in \Gamma$ and $\{p \mid \mathtt{s}[p] \in \mathit{dom}(\Delta)\} \subseteq \mathit{roles}(G)$ and G is well-formed and $\forall p \in \mathit{roles}(G), G \upharpoonright p$ is defined.
2. $\forall \mathtt{s}[p] : T, \mathtt{s}[q] : T' \in \Delta$ we have $\mathtt{s}[p] : T \lfloor q$–$(\mathtt{h})_{q \to p} \bowtie \mathtt{s}[q] : T' \lfloor p$–$(\mathtt{h})_{p \to q}$.

In condition 1, we require a coordinator for every session so that when a failure occurs, the coordinator can announce failure notifications to ask participants to handle the failure. Condition 2 requires that, for any two endpoints, say $\mathtt{s}[p]$ and $\mathtt{s}[q]$, in Δ, equation $\mathtt{s}[p] : T \lfloor q$–$(\mathtt{h})_{q \to p} \bowtie \mathtt{s}[q] : T' \lfloor p$–$(\mathtt{h})_{p \to q}$, must hold. This condition asserts that interactions of non-failed endpoints are dual to each other after the effect of $\mathtt{h}$; while failed endpoints are removed from Δ, thus the condition is satisfied immediately.

7 Properties

We show that our type system ensures properties of subject congruence, subject reduction, and progress. All auxiliary definitions and proof details are in App. D.

The property of subject congruence states that if $\mathcal{S}$ (a system containing an application and a coordinator) is well-typed by some session environment, then a $\mathcal{S}'$ that is structurally congruent to it is also well-typed by the same session environment:

Theorem 1 (Subject Congruence). $\Gamma \vdash \mathcal{S} \triangleright \Delta$ and $\mathcal{S} \equiv \mathcal{S}'$ imply $\Gamma \vdash \mathcal{S}' \triangleright \Delta$.

Subject reduction states that a well-typed $\mathcal{S}$ (coherent session environment respectively) is always well-typed (coherent respectively) after reduction:

Theorem 2 (Subject Reduction).

- $\Gamma \vdash \mathcal{S} \rhd \Delta$ with $\Gamma \vdash \Delta$ coherent and $\mathcal{S} \to^* \mathcal{S}'$ imply that $\exists \Delta', \Gamma'$ such that $\Gamma' \vdash \mathcal{S}' \rhd \Delta'$ and $\Gamma \vdash \Delta \to_T^* \Gamma' \vdash \Delta'$ or $\Delta \equiv \Delta'$ and $\Gamma' \vdash \Delta'$ coherent.
- $\Gamma \vdash \mathcal{S} \rhd \emptyset$ and $\mathcal{S} \to^* \mathcal{S}'$ imply that $\Gamma' \vdash \mathcal{S}' \rhd \emptyset$ for some Γ'.

We allow sessions to run in parallel at the top level, e.g., $\mathcal{S} = (\nu \mathsf{s}_1)(\Psi_1 \blacklozenge N_1) \mid ... \mid (\nu \mathsf{s}_n)(\Psi_n \blacklozenge N_n)$. Assume we have $\mathcal{S}$ with $a[p].P \in \mathcal{S}$. If we cannot apply rule **(Link)**, $\mathcal{S}$ cannot reduce. To prevent this kind of situation, we require $\mathcal{S}$ to be *initializable* such that, $\forall a[p].P \in \mathcal{S}$, **(Link)** is applicable.

The following property states that $\mathcal{S}$ never gets stuck (property of progress):

Theorem 3 (Progress). If $\Gamma \vdash \mathcal{S} \rhd \emptyset$ and $\mathcal{S}$ is initializable, then either $\mathcal{S} \to^* \mathcal{S}'$ and $\mathcal{S}'$ is initializable or $\mathcal{S}' = \Psi \blacklozenge \mathsf{s} : h \mid ... \mid \Psi' \blacklozenge \mathsf{s}' : h'$ and $h, ..., h'$ only contain failure notifications sent by coordinators and messages heading to failed participants.

After all processes in $\mathcal{S}$ terminate, failure notifications sent by coordinators are left; thus the final system can be of the form $\Psi \blacklozenge \mathsf{s} : h \mid ... \mid \Psi' \blacklozenge \mathsf{s}' : h'$, where $h, ..., h'$ only have failure notifications sent by coordinators and thus reduction rules **(CollectDone)**, **(IssueDone)**, and **(F)** will not be applied.

Minimality. The following proposition points out that, when all roles defined in a global type, say G, are robust, then the application obeying to G will never have interaction with a coordinator (i.e., interactions of the application are equivalent to those without a coordinator). This is an important property, as it states that our model does not incur coordination overhead when all participants are robust, or in failure-agnostic contexts as considered in previous MPST works.

Proposition 2. Assume $\forall p \in \mathit{roles}(G) = \{p_1, ..., p_n\}$, p is robust and $P_i = \mathsf{s}[p_i] : \eta_i$ for $i \in \{1..n\}$ and $\mathcal{S} = (\nu\ \mathsf{s})(\Psi \blacklozenge P_1 | ... | P_n | \mathsf{s} : h)$ where $P_i, i \in \{1..n\}$ contains no try-handle. Then we have $\Gamma \vdash \mathcal{S} \rhd \emptyset$ and whenever $\mathcal{S} \to^* \mathcal{S}'$ we have $\Psi \in \mathcal{S}'$, $\Psi = G : (\emptyset, \emptyset)$.

Proof. Immediately by typing rules $\lfloor$`T-ini,T-s,T-sys`$\rfloor$, Def. 4 (Projection), and the operational semantics defined in Fig. 6, Fig. 7, and Fig. 8.

8 Related Work

Several session type works study exception handling [7,9,16,30]. However, to the best of our knowledge this is the first theoretical work to develop a formalism and typing discipline for the coordinator-based model of *crash failure* handling in practical asynchronous distributed systems.

Structured interactional exceptions [7] study exception handling for binary sessions. The work extends session types with a *try-catch* construct and a *throw* instruction, allowing participants to raise runtime exceptions. Global escape [6]

extends previous works on exception handling in binary session types to MPSTs. It supports nesting and sequencing of try-catch blocks with restrictions. Reduction rules for exception handling are of the form $\Sigma \vdash P \to \Sigma' \vdash P'$, where Σ is the *exception environment*. This central environment at the core of the semantics is updated synchronously and atomically. Furthermore, the reduction of a try-catch block to its continuation is done in a synchronous reduction step involving all participants in a block. Lastly this work can only handle exceptions, i.e., explicitly raised application-level failures. These do not affect communication channels [6], unlike participant crashes.

Similarly, our previous work [13] only deals with exceptions. An interaction $p \to q : S \vee F$ defines that p can send a message of type S to q. If F is not empty then instead of sending a message p can throw F. If a failure is thrown only participants that have casual dependencies to that failure are involved in the failure handling. No concurrent failures are allowed therefore all interaction which can raise failures is executed in a lock step fashion. As a consequence, the model can not be used to deal with crash failures.

Adameit et al. [1] propose session types for link failures, which extend session types with an optional block which surrounds a process and contains default values. The default values are used if a link failure occurs. In contrast to our work, the communication model is overall synchronous whereas our model is asynchronous; the optional block returns default values in case of a failure but it is still the task of the developer to do something useful with it.

Demangeon et al. study interrupts in MPSTs [16]. This work introduces an interruptible block $\{|G|\}^c\langle l \text{ by } \mathtt{r}\rangle; G'$ identified by c; here the protocol G can be interrupted by a message l from $\mathtt{r}$ and is continued by G' after either a normal or an interrupted completion of G. Interrupts are more a control flow instruction like exceptions than an actual failure handling construct, and the semantics can not model participant crashes.

Neykova and Yoshida [37] show that MPSTs can be used to calculate safe global states for a safe recovery in Erlang's *let it crash* model [2]. That work is well suited for recovery of lightweight processes in an actor setting. However, while it allows for elaborate failure handling by connecting (endpoint) processes with runtime monitors, the model does not address the fault tolerance of runtime monitors themselves. As monitors can be interacting in complex manners replication does not seem straightforwardly applicable, at least not without potentially hampering performance (just as with *strainghtforward* replication of entire applications).

Failure handling is studied in several process calculi and communication-centered programming languages without typing discipline. The conversation calculus [43] models exception behavior in abstract service-based systems with message-passing based communication. The work does not use channel types but studies the behavioral theory of bisimilarity. Error recovery is also studied in a concurrent object setting [45]; interacting objects are grouped into coordinated atomic actions (CAs) which enable safe error recovery. CAs can however not be nested. PSYNC [18] is a domain specific language based on the *heard-of*

model of distributed computing [12]. Programs written in PSYNC are structured into rounds which are executed in a lock step manner. PSYNC comes with a state-based verification engine which enables checking of safety and liveness properties; for that programmers have to define non-trivial inductive invariants and ranking functions. In contrast to the coordinator model, the heard-of model is not widely deployed in practice. Verdi [44] is a framework for implementing and verifying distributed systems in Coq. It provides the possibility to verify the system against different network models. Verdi enables the verification of properties in an idealized fault model and then transfers the guarantees to more realistic fault models by applying transformation functions. Verdi supports safety properties but no liveness properties.

9 Final Remarks

Implementation. Based on our presented calculus we developed a domain-specific language and corresponding runtime system in Scala, using ZooKeeper as the coordinator. Specifically our implementation provides mechanisms for (1) interacting with ZooKeeper as coordinator, (2) done and failure notification delivery and routing, (3) practical failure detection and dealing with false suspicions and (4) automatically inferring try-handle levels(see more in App. C).

Conclusions. This work introduces a formal model of verified crash failure handling featuring a lightweight coordinator as common in many real-life systems. The model carefully exposes potential problems that may arise in distributed applications due to partial failures, such as inconsistent endpoint behaviors and orphan messages. Our typing discipline addresses these challenges by building on the mechanisms of MPSTs, e.g., global type well-formedness for sound failure handling specifications, modeling asynchronous permutations between regular messages and failure notifications in sessions, and the type-directed mechanisms for determining correct and orphaned messages in the event of failure. We adapt coherence of session typing environments (i.e., endpoint consistency) to consider failed roles and orphan messages, and show that our type system statically ensures subject reduction and progress in the presence of failures.

Future work. We plan to expand our implementation and develop further applications. We believe dynamic role participation and role parameterization would be valuable for failure handling. Also, we are investigating options to enable addressing the coordinator as part of the protocol so that pertinent runtime information can be persisted by the coordinator. We plan to add support to our language and calculus for solving various explicit agreement tasks (e.g., consensus, atomic commit) via the coordinator.

References

1. Adameit, M., Peters, K., Nestmann, U.: Session Types for Link Failures. In: FORTE '17. vol. 10321, pp. 1–16. Springer (2017)

2. Armstrong, J.: Making reliable distributed systems in the presence of software errors. Ph.D. thesis, Royal Institute of Technology, Stockholm, Sweden (2003)
3. Birman, K.P.: Byzantine clients (2017), `https://goo.gl/1Qbc4r`
4. Burrows, M.: The Chubby Lock Service for Loosely-Coupled Distributed Systems. In: OSDI '06. pp. 335–350. USENIX Association (2006)
5. Caires, L., Pérez, J.A.: Multiparty Session Types Within a Canonical Binary Theory, and Beyond. In: FORTE '16. LNCS, vol. 9688, pp. 74–95. Springer (2016)
6. Capecchi, S., Giachino, E., Yoshida, N.: Global escape in multiparty sessions. MSCS 26(2), 156–205 (2016)
7. Carbone, M., Honda, K., Yoshida, N.: Structured Interactional Exceptions in Session Types. In: CONCUR '08. LNCS, vol. 5201, pp. 402–417. Springer (2008)
8. Carbone, M., Lindley, S., Montesi, F., Schürmann, C., Wadler, P.: Coherence Generalises Duality: A Logical Explanation of Multiparty Session Types. In: CONCUR '16. LIPIcs, vol. 59, pp. 33:1–33:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2016)
9. Carbone, M., Yoshida, N., Honda, K.: Asynchronous Session Types: Exceptions and Multiparty Interactions. In: SFM 2009. LNCS, vol. 5569, pp. 187–212. Springer (2009)
10. Chandra, T.D., Hadzilacos, V., Toueg, S., Charron-Bost, B.: On the Impossibility of Group Membership. In: PODC '96. pp. 322–330. ACM (1996)
11. Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., Gruber, R.: Bigtable: A Distributed Storage System for Structured Data. In: OSDI '06. pp. 205–218. USENIX Association (2006)
12. Charron-Bost, B., Schiper, A.: The Heard-Of model: computing in distributed systems with benign faults. Distributed Computing 22(1), 49–71 (2009)
13. Chen, T., Viering, M., Bejleri, A., Ziarek, L., Eugster, P.: A Type Theory for Robust Failure Handling in Distributed Systems. In: FORTE '16. vol. 9688, pp. 96–113. Springer (2016)
14. Coppo, M., Dezani-Ciancaglini, M., Padovani, L., Yoshida, N.: A Gentle Introduction to Multiparty Asynchronous Session Types. In: SFM '15. LNCS, vol. 9104, pp. 146–178. Springer (2015)
15. Coppo, M., Dezani-Ciancaglini, M., Yoshida, N., Padovani, L.: Global progress for dynamically interleaved multiparty sessions. MSCS 26(2), 238–302 (2016)
16. Demangeon, R., Honda, K., Hu, R., Neykova, R., Yoshida, N.: Practical Interruptible Conversations. Formal Methods in System Design 46(3), 197–225 (2015)
17. Deniélou, P.M., Yoshida, N.: Dynamic Multirole Session Types. In: POPL '11. pp. 435–446. ACM (2011)
18. Dragoi, C., Henzinger, T., Zufferey, D.: PSync: A Partially Synchronous Language for Fault-tolerant Distributed Algorithms. In: POPL '16. pp. 400–415. ACM (2016)
19. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of Distributed Consensus with One Faulty Process. In: PODS '85. pp. 374–382. ACM (1983)
20. Ghemawat, S., Gobioff, H., Leung, S.T.: The Google File System. In: SOSP '03. pp. 29–43. ACM (2003)
21. Gilbert, S., Lynch, N.: Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-tolerant Web Services. SIGACT News 33(2), 51–59 (2002)
22. Guerraoui, R., Schiper, A.: The Generic Consensus Service. IEEE Trans. Software Eng. 27(1), 29–41 (2001)
23. Honda, K., Vasconcelos, V.T., Kubo, M.: Language Primitives and Type Discipline for Structured Communication-Based Programming. In: ESOP '98. LNCS, vol. 1381, pp. 122–138. Springer (1998)

24. Honda, K., Yoshida, N., Carbone, M.: Multiparty Asynchronous Session Types. J. ACM 63(1), 9:1–9:67 (2016)
25. Hu, R., Yoshida, N.: Hybrid Session Verification Through Endpoint API Generation . In: FASE '16. LNCS, vol. 9633, pp. 401–418. Springer (2016)
26. Hunt, P.: ZooKeeper: Wait-free Coordination for Internet-scale Systems. In: USENIX '10. USENIX Association (2010)
27. Hüttel, H., et al.: Foundations of Session Types and Behavioural Contracts. ACM Comput. Surv. 49(1), 3:1–3:36 (2016)
28. Imai, K., Yoshida, N., Yuen, S.: Session-ocaml: A Session-Based Library with Polarities and Lenses. In: COORDINATION '17. LNCS, vol. 10319, pp. 99–118. Springer (2017)
29. Killian, C.E., Anderson, J.W., Braud, R., Jhala, R., Vahdat, A.M.: Mace: Language Support for Building Distributed Systems. In: PLDI '07. vol. 42, pp. 179–188. ACM (2007)
30. Kouzapas, D., Yoshida, N.: Globally Governed Session Semantics. LMCS 10(4) (2014)
31. Kreps, J., Narkhede, N., Rao, J.: Kafka: A Distributed Messaging System for Log Processing. In: NetDB '11 (2011)
32. Lamport, L.: The Part-time Parliament. ACM Trans. Comput. Syst. 16(2), 133–169 (1998)
33. Lamport, L., Shostak, R., Pease, M.: The Byzantine Generals Problem. ACM Trans. Program. Lang. Syst. 4(3), 382–401 (1982)
34. Leners, J.B., Wu, H., Hung, W.L., Aguilera, M.K., Walfish, M.: Detecting failures in distributed systems with the FALCON spy network. In: SOSP '11. pp. 279–294. ACM (2011)
35. Lindley, S., Morris, J.G.: Embedding Session Types in Haskell. In: Haskell '16. pp. 133–145. ACM (2016)
36. Mostrous, D., Yoshida, N.: Session typing and asynchronous subtyping for the higher-order π-calculus. Inf. Comput. 241, 227–263 (2015)
37. Neykova, R., Yoshida, N.: Let It Recover: Multiparty Protocol-Induced Recovery. In: CC '17. pp. 98–108. ACM (2017)
38. Padovani, L.: A simple library implementation of binary sessions. J. Funct. Program. 27, e4 (2017)
39. Pucella, R., Tov, J.A.: Haskell Session Types with (Almost) No Class. In: Haskell '08. pp. 25–36. ACM (2008)
40. Scalas, A., Dardha, O., Hu, R., Yoshida, N.: A Linear Decomposition of Multiparty Sessions for Safe Distributed Programming. In: ECOOP '17. LIPIcs, vol. 74, pp. 24:1–24:31. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2017)
41. Shvachko, K., Kuang, H., Radia, S., Chansler, R.: The Hadoop Distributed File System. In: MSST '10. pp. 1–10. IEEE Computer Society (2010)
42. Sivaramakrishnan, K.C., Qudeisat, M., Ziarek, L., Nagaraj, K., Eugster, P.: Efficient sessions. Sci. Comput. Program. 78(2), 147–167 (2013)
43. Vieira, H.T., Caires, L., Seco, J.C.: The Conversation Calculus: A Model of Service-Oriented Computation. In: ESOP '08. LNCS, vol. 4960, pp. 269–283. Springer (2008)
44. Wilcox, J.R., Woos, D., Panchekha, P., Tatlock, Z., Wang, X., Ernst, M.D., Anderson, T.E.: Verdi: A Framework for Implementing and Formally Verifying Distributed Systems. In: PLDI '15. pp. 357–368. ACM (2015)
45. Xu, J., Randell, B., Romanovsky, A.B., Rubira, C.M.F., Stroud, R.J., Wu, Z.: Fault Tolerance in Concurrent Object-Oriented Software Through Coordinated Error Recovery. In: FTCS '95. pp. 499–508. IEEE Computer Society (1995)

A Examples

This section provides interesting examples which are not included in the main content. We first give examples which are based on MPST and are extended with our explicit failure handling structure. Then we provide more examples of coordinator-based failure handling to show several non-trivial cases.

A.1 Examples of MPST Types and Processes with Coordinator-based Failure Handling

We give versions of the main motivating examples in standard MPST literature [24] extended to support partial failures.

Two-buyers protocol. The Two-Buyers Protocol [24, § 2.3,3.4] derives from a Web services use case. In the original protocol specification, the roles Buyer1 (B1) and Buyer2 (B2) carry out a joint transaction to buy a book from an online Seller (S). The role of B1 is to offer to pay some share of the total price to B2. B2 makes the final *choice* whether to proceed with the purchase (by paying the remaining amount) or not.

$$
\begin{array}{l}
G_{\mathsf{TB}} = \mathsf{t}(\\
\quad \mathsf{t}(\\
\quad\quad \mathtt{B2}\to\mathtt{S}\ l_1(\mathtt{string}). \\
\quad\quad \mathtt{S}\to\mathtt{B1}\ l_2(\mathtt{int}). \\
\quad\quad \mathtt{S}\to\mathtt{B2}\ l_3(\mathtt{int}). \\
\quad\quad \mathtt{B1}\to\mathtt{B2}\ l_4(\mathtt{int}). \\
\quad\quad \mathtt{B2} \to \mathtt{S} \begin{cases} \mathtt{ok}_1().\mathtt{B2}\to\mathtt{S}\ l_5(\mathtt{string}).\mathtt{S}\to\mathtt{B2}\ l_6(\mathtt{date}).\mathsf{end} \\ \mathtt{quit}_1().\mathsf{end} \end{cases} \\
\quad)\mathsf{h}(\\
\quad\quad \{\mathtt{B1}\}\mathtt{:S}\to\mathtt{B2}\ l_7(\mathtt{int}). \\
\quad\quad\quad \mathtt{B2} \to \mathtt{S} \begin{cases} \mathtt{ok}_2().\mathtt{B2}\to\mathtt{S}\ l_8(\mathtt{string}).\mathtt{S}\to\mathtt{B2}\ l_9(\mathtt{date}).\mathsf{end} \\ \mathtt{quit}_2().\mathsf{end} \end{cases} \\
\quad)^2.\mathsf{end} \\
)\mathsf{h}(\\
\quad \{\mathtt{B2}\}\mathtt{:end}, \\
\quad \{\mathtt{B1},\mathtt{B2}\}\mathtt{:end} \\
)^1. \\
\mathsf{end}
\end{array}
$$

Fig. 12. The Two-Buyers Protocol [24, § 3.4] extended with partial failure handling.

Fig. 12 extends the original global type to a fault tolerant version of this application protocol, bearing in mind the asymmetry of the B1 and B2 roles. We shall assume S is robust.

In the initial exchanges, given by the inner try-block, B2 sends S the name of the book (l_1), S sends B1 and B2 the price (l_2), and B1 sends B2 the amount it is willing to contribute (l_3) (as in the original protocol). If B1 crashes during this part, then S acknowledges this event by resending the price to B2 (l_7) in the inner handling enviroment—asynchrony and inherent concurrency of the interactions between S and the Buyers and this potential failure means that S cannot be certain about the order of the relevant messages arriving at B2.

Whether or not B1 crashes, B2 and S proceed to the choice of B2 to buy (ok) or not buy (quit) the book (this segment is also as in the original protocol). However, if B2 crashes at any point of the protocol, then the protocol must simply end for S, and also for B1 if it is live, as given by the outer handling environment.

Regarding the unique κ-annotations condition of well-formedness (Def. 1), as noted in Sec. 3, and implemented in App. C, it is straightforward to mechanically infer identifiers for an "unannotated" global type to satisfy this condition. As for many protocols, it is also straightforward to add labels to non-explicitly labelled interactions to satisfy the labelling condition. In this example, it would have been sufficient to use just three labels in total to distinguish the interactions in the try-block, the inner handler, and the choice construct; however, we used unique labels throughout to clarify the explanations given above.

Projection to local types. The local type projection (Def. 4) of G_{TB} onto B1, $G_{\mathsf{TB}}\upharpoonright\mathtt{B1}$, is:

$$\begin{array}{l}\mathsf{t}(\\ \quad \mathsf{t}(\mathtt{S}?l_2(\mathtt{int}).\mathtt{B2}!l_4(\mathtt{int}).\mathsf{end})\mathsf{h}(\{\mathtt{B1}:\mathsf{end}\})^{(2,\emptyset)}.\mathsf{end}\\ )\mathsf{h}(\{\mathtt{B2}:\mathsf{end},\{\mathtt{B1},\mathtt{B2}\}:\mathsf{end}\})^{(1,\emptyset)}.\mathsf{end}\end{array}$$

The projection onto B2, $G_{\mathsf{TB}}\upharpoonright\mathtt{B2}$, is:

$$\begin{array}{l}\mathsf{t}(\\ \quad \mathsf{t}(\\ \quad\quad \mathtt{S}!l_1(\mathtt{string}).\mathtt{S}?l_3(\mathtt{int}).\mathtt{B1}?l_4(\mathtt{int}).\\ \quad\quad \mathtt{S}!\begin{cases}\mathtt{ok}_1().\mathtt{S}!l_5(\mathtt{string}).\mathtt{S}?l_6(\mathtt{date}).\mathsf{end}\\ \mathtt{quit}_1().\mathsf{end}\end{cases}\\ \quad)\mathsf{h}(\\ \quad\quad \{\mathtt{B1}\}:\mathtt{S}?l_7(\mathtt{int}).\\ \quad\quad\quad \mathtt{S}!\begin{cases}\mathtt{ok}_2().\mathtt{S}!8(\mathtt{string}).\mathtt{S}?l_9(\mathtt{date}).\mathsf{end}\\ \mathtt{quit}_2().\mathsf{end}\end{cases}\\ \quad)^{(2,\emptyset)}\\ )\mathsf{h}(\{\mathtt{B2}:\mathsf{end},\{\mathtt{B1},\mathtt{B2}\}:\mathsf{end}\})^{(1,\emptyset)}.\mathsf{end}\end{array}$$

The projection onto S, $G_{\mathsf{TB}}\upharpoonright\mathtt{S}$, is:

$$
\begin{array}{l}
\mathsf{t}(\\
\quad \mathsf{t}(\\
\quad\quad \mathtt{B2}?l_1(\mathtt{string}).\mathtt{B1}!l_2(\mathtt{int}).\mathtt{B2}!l_3(\mathtt{int}). \\
\quad\quad \mathtt{B2}?\begin{cases} \mathtt{ok}().\mathtt{B2}?l_5(\mathtt{string}).\mathtt{B2}!l_6(\mathtt{date}).\mathsf{end} \\ \mathtt{quit}().\mathsf{end} \end{cases} \\
\quad)\mathsf{h}(\\
\quad\quad \{\mathtt{B1}\}:\mathtt{B2}!l_7(\mathtt{int}).\mathsf{end} \\
\quad\quad\quad \mathtt{B2}?\begin{cases} \mathtt{ok}().\mathtt{B2}?l_8(\mathtt{string}).\mathtt{B2}!l_9(\mathtt{date}).\mathsf{end} \\ \mathtt{quit}().\mathsf{end} \end{cases} \\
\quad)^{(2,\emptyset)} \\
)\mathsf{h}(\{\mathtt{B2}\}:\mathsf{end}, \{\mathtt{B1},\mathtt{B2}\}:\mathsf{end})^{(1,\emptyset)}.\mathsf{end}
\end{array}
$$

Processes. We give example of well-typed processes implementing each of roles.

For `B1`:

$$
\begin{array}{l}
a[\mathtt{B1}](y).y : \mathsf{t}(\\
\quad \mathsf{t}(\mathtt{S}?l_2(x_2).\mathtt{B2}!l_4(x_2 \div 2).0)\mathsf{h}(\{\mathtt{B1}\}:0)^{(2,\emptyset)}.0 \\
)\mathsf{h}(\{\mathtt{B2}\}:0, \{\mathtt{B1},\mathtt{B2}\}:0)^{(1,\emptyset)}.0
\end{array}
$$

For `B2`:

$$
\begin{array}{l}
a[\mathtt{B2}](y).y : \mathsf{def}\ X(z) = \mathsf{if}\ \ z < 100\ \ \mathtt{S}!\mathtt{ok}().\mathtt{S}!l_5(\text{“addr”}).\mathtt{S}?l_6(x_6).0 \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{else}\ \ \ \mathtt{S}!\mathtt{quit}().0 \\
\mathsf{in}\ \mathsf{t}(\\
\quad \mathsf{t}(\mathtt{S}!l_1(\text{“title”}).\mathtt{S}?l_2(x_2).\mathtt{B1}?l_4(x_4).X\langle x_2 - x_4\rangle) \\
\quad \mathsf{h}(\{\mathtt{B1}\}:\mathtt{S}?l_7(x_7).X\langle x_7\rangle)^{(2,\emptyset)}.0) \\
)\mathsf{h}(\{\mathtt{B2}\}:0, \{\mathtt{B1},\mathtt{B2}\}:0)^{(1,\emptyset)}.0
\end{array}
$$

For `S`, assuming some helper functions $\mathsf{getPrice}$ and $\mathsf{getData}$ on data:

$$
\begin{array}{l}
a[\mathtt{S}](y).y : \mathsf{def}\ X() = \mathtt{B2}?\{\mathtt{ok}().\mathtt{B2}?l_5(x_5).\mathtt{S}?l_6(\mathsf{getDate}(x_5)).0, \mathtt{quit}().0\} \\
\mathsf{in}\ \mathsf{t}(\\
\quad \mathsf{t}(\mathtt{B2}?l_1(x_1).\mathtt{B1}!l_2(\mathsf{getPrice}(x_1)).\mathtt{B2}!l_4(\mathsf{getPrice}(x_1)).X\langle\rangle) \\
\quad \mathsf{h}(\{\mathtt{B1}\}:\mathtt{B1}!l_7(\mathsf{getPrice}(x_1)).X\langle\rangle)^{(2,\emptyset)}.0 \\
)\mathsf{h}(\{\mathtt{B2}\}:0, \{\mathtt{B1},\mathtt{B2}\}:0)^{(1,\emptyset)}.0
\end{array}
$$

Streaming protocol. The original Streaming Protocol [24, § 3.4] demonstrated a *recursive* global type for a continuous stream of messages, where two producer roles (`DP`, `KP`) independently send to a middleman role (`K`) in a join pattern, followed by `K` forwarding a message to a consumer role (`C`). (The role names are taken from the original protocol.)

Fig. 13 extends the protocol to handle the potential failures of the `DP` and `KP`; we assume the other two roles are robust. The idea is if just one of the producers crashes, the now fault tolerant protocol should attempt to continue

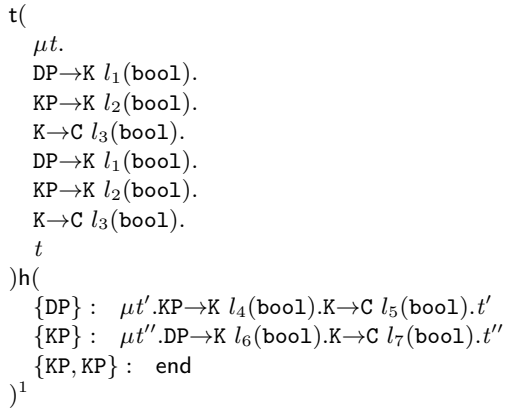

Fig. 13. The Stream Protocol [24, § 3.4] extended with partial failure handling.

with the other producer. We keep the "once-unfolded" specification from the original protocol definition in the try-block, but use the shorter folded versions in the handling activities.

This example also gives some intuition for the complexity in the design of our failure handling constructs: *without* any explicit choice construct, adding failure handling naturally introduces a *safe* notion of choice into the protocol between distinct paths, and also allows the normally recursive protocol to end (if both producers crash).

A.2 Examples of Coordinator-based Failure Handling

In this part, we give more handling examples for readers who are interested in the coordinator-based failure handling.

Handling failure notifications. The following example shows handling mechanism of **(TryHdl)** and **(F)**:

Example 8. Let P_i play role p_i in session s and h does not contain any notifications. In this example both P_2 and P_3 fail and it shows a reduction for P_1 with a focus on failure handling. Assume we start with:

$$
\begin{array}{rl}
 & \ldots\ G : (F, \emptyset)\blacklozenge\ P_1 \mid P_2 \mid P_3 \mid \mathsf{s} : h \cdot \langle p_2, p_1, l(v) \rangle \\
\textbf{(Crash)} & \to G : (F, \emptyset)\blacklozenge\ P_1 \mid P_3 \mid \mathsf{s} : h_1 \cdot \langle\!\langle \psi, \mathsf{crash}\ \{p_2\} \rangle\!\rangle \\
 & \qquad\qquad \text{where } h_1 = \mathit{remove}(h \cdot \langle p_2, p_1, l(v) \rangle, p_2) \\
 & \equiv G : (F, \emptyset)\blacklozenge\ P_1 \mid P_3 \mid \mathsf{s} : \langle\!\langle \psi, \mathsf{crash}\ \{p_2\} \rangle\!\rangle \cdot h_1 \\
\textbf{(F)} & \to G : (F \cup \{p_2\}, \emptyset)\blacklozenge\ P_1 \mid P_3 \mid \mathsf{s} : h_1 \cdot \langle\!\langle \{p_1, p_3\}, \mathsf{crash}\ \{p_2\} \rangle\!\rangle
\end{array}
$$

Note that, by Definition 13 (Permutable Messages), because h (therefore also h_1) does not contain any notification message the notification $\langle\!\langle \psi, \mathsf{crash}\ \{p_2\} \rangle\!\rangle$

can be permuted to the top of queue. Next the the coordinator receives notification $\langle\!\langle \psi, \mathsf{crash}\ \{p_2\}\rangle\!\rangle$ and issues the failure notification $\langle\!\langle \{p_1, p_3\}, \mathsf{crash}\ \{p_2\}\rangle\!\rangle$ to P_1 and P_3 for handling the failure of P_2 by applying **(F)**.

Assume (possible concurrently) that P_3 fails, i.e. **(Crash)** is applied to P_3

$$\begin{array}{rl}\textbf{(Crash)} & \to G : (F \cup \{p_2\}, \emptyset)\blacklozenge\ P_1 \ \mid\ \mathsf{s} : h_2 \cdot \langle\!\langle p_1, \mathsf{crash}\ \{p_2\}\rangle\!\rangle \cdot \langle\!\langle \psi, \mathsf{crash}\ \{p_3\}\rangle\!\rangle \\ & \quad \text{where } h_2 \cdot \langle\!\langle \{p_1\}, \mathsf{crash}\ \{p_2\}\rangle\!\rangle = \mathit{remove}(h_1 \cdot \langle\!\langle \{p_1, p_3\}, \mathsf{crash}\ \{p_2\}\rangle\!\rangle, p_3)\end{array}$$

Assume we have $P_1 = \mathsf{s}[p_1] : \mathsf{t}(\eta)\mathsf{h}(\mathsf{H})^{(\phi,\emptyset))}$ and $\mathsf{H} = \{p_2\} : \eta_2, \{p_3\} : \eta_3\ \{p_1, p_3\} : \eta_{13}$. The reduction shows P_1 starts handling the failure $\{p_2\}$ (P_1 is not aware of the failure of P_3), then the coordinator will pick up the notification that P_3 has failed and forward it. Lastly P_1 will perform the handling for both failures:

$$\begin{array}{rl}\textbf{(TryHdl)} & \to G : (F \cup \{p_2\}, \emptyset)\blacklozenge\ \mathsf{s}[p_1] : \mathsf{t}(\eta_2)\mathsf{h}(\mathsf{H})^{(\phi,\{p_2\})} \ \mid \\ & \quad \mathsf{s} : h_2 \cdot \langle\!\langle p_1, \mathsf{crash}\ \{p_2\}\rangle\!\rangle \cdot \langle\!\langle \psi, \mathsf{crash}\ \{p_3\}\rangle\!\rangle \\ \equiv, \textbf{(F)} & \to G : (F \cup \{p_2, p_3\}, \emptyset)\blacklozenge\ \mathsf{s}[p_1] : \mathsf{t}(\eta_2)\mathsf{h}(\mathsf{H})^{(\phi,\{p_2\})} \ \mid \\ & \quad \mathsf{s} : h_2 \cdot \langle\!\langle p_1, \mathsf{crash}\ \{p_2\}\rangle\!\rangle \cdot \langle\!\langle p_1, \mathsf{crash}\ \{p_3\}\rangle\!\rangle \\ \textbf{(TryHdl)} & \to G : (F \cup \{p_2, p_3\}\}, \emptyset)\blacklozenge\ \mathsf{s}[p_1] : \mathsf{t}(\eta_{23})\mathsf{h}(\mathsf{H})^{(\phi,\{p_2,p_3\})} \ \mid \\ & \quad \mathsf{s} : h_2 \cdot \langle\!\langle p_1, \mathsf{crash}\ \{p_2\}\rangle\!\rangle \cdot \langle\!\langle p_1, \mathsf{crash}\ \{p_3\}\rangle\!\rangle \\ & \ldots\end{array}$$

The rest of the reduction is straight forward by using **(SndDone)**, **(CollectDone)**, **(IssueDone)** and **(RcvDone)**.

Handling mixed notifications. We illustrate the hardness of consistent handling of mixed notifications (i.e., done and failure notifications), and the need for a coordinator.

Example 9. Assume we have processes P_1, P_2, P_3 acting respectively on channels $\mathsf{s}[p_1], \mathsf{s}[p_2]$ and $\mathsf{s}[p_3]$. Given Eq. (1), in which they have finished their respective actions in the try-handle of $\phi = (\kappa, \emptyset)$, and the queue contains the done notifications sent to the coordinator

$$\begin{array}{l}(1)\ \mathsf{s}[p_1] : \mathsf{t}(\underline{0})\mathsf{h}(\mathsf{H}_{p_1})^{\phi}.\eta_1 \mid \mathsf{s}[p_2] : \mathsf{t}(\underline{0})\mathsf{h}(\mathsf{H}_{p_2})^{\phi}.\eta_2 \mid \mathsf{s}[p_3] : \mathsf{t}(\underline{0})\mathsf{h}(\mathsf{H}_{p_3})^{\phi}.\eta_3 \mid \\ \quad \mathsf{s} : \langle p_2, \psi\rangle^{\phi} \cdot \langle p_1, \psi\rangle^{\phi} \cdot \langle p_3, \psi\rangle^{\phi}\end{array}$$

where $\mathsf{H}_{p_1} = \{p_2\} : \eta_1, \mathsf{H}'_{p_1}$ and $\mathsf{H}_{p_3} = \{p_2\} : \eta_3, \mathsf{H}'_{p_3}$. The coordinator is the key in ensuring consistency in the presence of concurrent done and failure notifications. If no participant fails, the coordinator collects $\langle p_2, \psi\rangle^{\phi}$, ... with **(CollectDone)**; and the coordinator issues done notifications $\langle \psi, \{p_1, p_2, p_3\}\rangle^{\phi}$ via **(IssueDone)**. All participants will then finish the try-handle of ϕ and move to their next actions, as shown by (1) $\to^*$ (2)

$$\begin{array}{l}(2)\ \mathsf{s}[p_1] : \mathsf{t}(\underline{0})\mathsf{h}(\mathsf{H}_{p_1})^{\phi}.\eta_1 \mid \mathsf{s}[p_2] : \mathsf{t}(\underline{0})\mathsf{h}(\mathsf{H}_{p_2})^{\phi}.\eta_2 \mid \mathsf{s}[p_3] : \mathsf{t}(\underline{0})\mathsf{h}(\mathsf{H}_{p_3})^{\phi}.\eta_3 \mid \\ \quad \mathsf{s} : \langle \psi, p_1\rangle^{\phi} \cdot \langle \psi, p_2\rangle^{\phi} \cdot \langle \psi, p_3\rangle^{\phi} \\ \to^*\ \mathsf{s}[p_1] : \eta_1 \mid \mathsf{s}[p_2] : \eta_2 \mid \mathsf{s}[p_3] : \eta_3 \mid \mathsf{s} : \emptyset\end{array}$$

If P_2 fails at the moment of Eq. (1), notification $\langle p_2, \psi\rangle^{\phi}$ is removed from the queue and therefore the try-handle of ϕ cannot finish, as shown by (1) $\to^*$ (3)

$$(3)\ \mathsf{s}[p_1] : \mathsf{t}(\underline{0})\mathsf{h}(\mathsf{H}_{p_1})^{\phi}.\eta_1 \mid \mathsf{s}[p_3] : \mathsf{t}(\underline{0})\mathsf{h}(\mathsf{H}_{p_3})^{\phi}.\eta_3 \mid \mathsf{s} : \langle p_1, \psi\rangle^{\phi} \cdot \langle p_3, \psi\rangle^{\phi} \cdot \langle\!\langle \psi, \mathsf{crash}\ \{p_2\}\rangle\!\rangle$$

and a notification for the failure of p_2 is added to the queue. When the coordinator receives the failure notification $\langle\!\langle \psi, \mathsf{crash}\ \{p_2\}\rangle\!\rangle$, it issues $\langle\!\langle p_1, \mathsf{crash}\ \{p_2\}\rangle\!\rangle$ and $\langle\!\langle p_3, \mathsf{crash}\ \{p_2\}\rangle\!\rangle$ to P_1 and P_3. Now P_1 and P_3 will perform the handling and change the level of the try-handle to $\phi' = (\kappa, \{p_2\})$, as shown by (3) $\rightarrow^*$ (4)

$$(4)\ \mathsf{s}[p_1] : \mathsf{t}(\eta_1)\mathsf{h}(\mathsf{H}_{p_1})^{\phi'}.\eta_1 \mid \mathsf{s}[p_3] : \mathsf{t}(\eta_3)\mathsf{h}(\mathsf{H}_{p_3})^{\phi'}.\eta_3 \mid \mathsf{s} : \langle\!\langle p_1, \mathsf{crash}\ \{p_2\}\rangle\!\rangle \cdot \langle\!\langle p_3, \mathsf{crash}\ \{p_2\}\rangle\!\rangle$$

B Detailed Formal Definitions

This section contains additional detailed formal definitions.

B.1 Definitions for Section 3 (Global Types)

Well-formedness. In the following we provide formal definitions for those used inexplaining well-formed global tyeps in Section 3. We formally define the syntactical well-formedness conditions.

For the well-formedness definition we need to address nesting of try-handles. We define the global contexts $\mathcal{G}$ as follows:

$$\mathcal{G} ::= [\,] \mid \mathsf{t}(\mathcal{G})\mathsf{h}(H)^{\kappa}.G \mid \mathsf{t}(G)\mathsf{h}(F{:}\mathcal{G}, H)^{\kappa}.G' \mid \mathsf{t}(G)\mathsf{h}(H)^{\kappa}.\mathcal{G} \mid p \rightarrow q\{l_i(S_i).G_i, l(S).\mathcal{G}\}_{i\in I} \mid \mu t.\mathcal{G}$$

Such contexts allow us to reason about parts of global types. Based on these contexts we define a containment relation on global types as follows:

Definition 9 ($G' \in G$). If $\exists \mathcal{G}$ s.t. $G = \mathcal{G}[G']$, then $G' \in G$

$G' \in G$ means G' is a part of global type G. Analogously to $G' \in G$, we write $G \in H$ if the handling environment H contains G; $\kappa \in G$ if G contains κ (remember κ is shorthand for $\mathsf{t}(G_1)\mathsf{h}(H)^{\kappa}$); $\kappa \in \kappa'$; $l \in G$ if the label l appears inside G; and $l \in \mathcal{G}$ if the label l appears inside $\mathcal{G}$. We use a lookup function $outer_G(\kappa)$ for the set of all try-handles in G that enclose a given κ (including κ itself):

Definition 10 ($outer_G(\kappa)$). $outer_G(\kappa) = \{\kappa' \mid \kappa \in \kappa' \wedge \kappa' \in G\}$.

As we mentioned in Sec. 3, we require that all κ are unique.
Based on Def. 9 and Def. 10, we define well-formedness of global types where Conditions 1 and 2 are the same as those stated in Sec. 3 Def. 1, and Conditions 3–5 define other conditions informally described in Sec. 3.

Definition 11 (Well-formedness (Full Version)). Let κ stand for $\mathsf{t}(G_1)\mathsf{h}(H)^{\kappa}$, and κ' for $\mathsf{t}(G_1')\mathsf{h}(H')^{\kappa'}$. A top-level $[\tilde{p}]G$ is *well-formed* if it fulfills *all* following conditions. For all $\kappa \in G$:

1. For any two separate handler signatures of a handling environment of κ, there always exists a handler whose handler signature matches the union of the respective failure sets. This handler is either inside the handling environment of κ itself, or in the handling environment of an outer try-handle:

$$\forall F_1 \in dom(H).\forall F_2 \in dom(H).\exists \kappa' \in outer_G(\kappa) \text{ s.t. } F_1 \cup F_2 \in dom(H')$$

2. If the handling environment of a try-handle κ contains a handler for F, then there is no outer try-handle κ' with a handler for F' such that $F' \subseteq F$:

$$\nexists F \in dom(H).\exists \kappa' \in outer_G(\kappa).\exists F' \in dom(H') \text{ s.t. } \kappa' \neq \kappa \wedge F' \subseteq F$$

3. All κ in G are unique. In addition all labels which appear inside a default try-body or any handler body do not occur outside of the default block/the handler body:

$$\begin{aligned} G = \mathcal{G}[\mathtt{t}(G_1)\mathtt{h}(H)^{\kappa}.G_2] \Rightarrow \\ \forall l \in G_1.l \notin \mathcal{G}, G_2, H \text{ and} \\ \forall F, F' \in H. \text{ s.t. } F \neq F' \; \forall l' \in H(F).l' \notin \mathcal{G}, G_1, G_2, H(F') \end{aligned}$$

4. A role does not appear in the handling activity of its own failure:

$$\forall F \in dom(H).\forall p \in F \Rightarrow p \notin roles(H(F))$$

5. All branching types of non-robust participants ($p \notin \tilde{p}$) must be handled in G, i.e., must be enclosed by try-handles which can handle the potential failures of p:

$$\begin{aligned} \forall q \to q'\{l_i(S_i).G_i\}_{i\in I} \in G.\forall p \in \{q, q'\}.p \notin \tilde{p} \Rightarrow \\ \exists \kappa' \in G.q \to q'\{l_i(S_i).G_i\}_{i\in I} \in \kappa' \wedge \{p\} \in dom(H') \end{aligned}$$

Condition 1 ensures that if roles are active in different handlers of the same try-handle, there is a handler whose signature corresponds to the union over the signatures of those different handlers. Example 2 together with Example 3 in Sec. 4 show why this condition is needed. The reason for Condition 2 is that in case of nested try-handles the operational semantics (see **(TryHdl)** in Sec. 4 Fig. 6) allows multiple try-handles to start failure handling; eventually the outer-most try-handle will perform the handling and may interrupt failure handling at an inner try-handle. $G = \mathtt{t}(\mathtt{t}(G')\mathtt{h}(\{p_1, p_2\} : G_1)^2)\mathtt{h}(\{p_1\} : G_1')^1$ violates condition 2 because, when p_1 and p_2 both failed, the handler signature $\{p_1\}$ will still be triggered (i.e., the outer try-handle will eventually take over). It is not sensible to run G_1' instead of G_1 (which is for the crashes of p_1 and p_2). The reason for Condition 3 is that a try-handle is our handling basis, and we shall not confuse normal messages or done notifications from different try-handles. Condition 4 is straightforward. Condition 5 is also straightforward as we can only handle failures occurring in a try-handle therefore interaction that involves non-robust roles needs to be inside try-handles.

Examples of well-formed and ill-formed global types.

Example 10. $G_{ko} = \mathsf{t}(\mathsf{t}(G')\mathsf{h}(\{p_1\} : G_1)^2)\mathsf{h}(\{p_1\} : G_1')$ violates Condition 2 because there are two different handling activities for $\{p_1\}$ at different nesting levels. Since the outer try-handle will eventually take over, it is not sensible to have the handling activity $\{p_1\}$ at the inner one.

Example 11. $G_{ko} = \mathsf{t}(p_1 \to p_2\ l_1(S_1))\mathsf{h}(\{p_1\} : p_1 \to p_2\ l_2(S_2))$ violates Condition 4. It is not well-formed since the handling activity of $\{p_1\}$ has $p_1 \to p_2\ l_2(S_2)$ in which p_1 is expected to output a message, yet p_1 would have failed at that point.

Example 12. Consider

$$G_{ko} = \mathsf{t}(p_1 \to p_2\ l_1(S_1))\mathsf{h}(\{p_1\} : p_2 \to p_3\ l_3(S_3)).p_1 \to p_3\ l_2(S_2)$$
$$G_{ok} = \mathsf{t}(p_1 \to p_2\ l_1(S_1).p_1 \to p_3\ l_2(S_2))\mathsf{h}(\{p_1\} : p_2 \to p_3\ l_3(S_3))$$

G_{ko} violates Condition 5 since the non-robust p_1 is expected to perform a message send $p_1 \to p_3\ l_2(S_2)$ which is not enclosed by any try-handle. This is not safe because, if p_1 crashes before $p_1 \to p_3\ l_2(S_2)$ completes, then p_3 will get stuck. On the other hand, G_{ok} is well-formed because all interactions containing the non-robust participant p_1 are enclosed in appropriate $\mathsf{t}(..)\mathsf{h}(..)$ blocks and all other properties hold.

B.2 Definitions for Section 4 (Process Calculus)

In the following we provide formal definitions for those we have used in Sec. 4 but have not yet formally defined.

Structural congruence. Processes, applications, and systems are considered modulo structural equivalence, denoted by $\equiv$, and defined by the rules in Def. 12 along with α-renaming.

Definition 12 (Structural Congruence).

$$h \equiv \emptyset \cdot h \equiv h \cdot \emptyset \qquad h_1 \cdot (h_2 \cdot h_3) \equiv (h_1 \cdot h_2) \cdot h_3$$
$$\frac{m \cdot m' \frown m' \cdot m}{h \cdot m \cdot m' \cdot h' \equiv h \cdot m' \cdot m \cdot h'} \qquad \frac{h \equiv h'}{\mathsf{s} : h \equiv \mathsf{s} : h'}$$
$$\mathsf{def}\ D\ \mathsf{in}\ 0 \equiv 0$$
$$\mathsf{def}\ D\ \mathsf{in}\ (\mathsf{def}\ D'\ \mathsf{in}\ \eta) \equiv \mathsf{def}\ D'\ \mathsf{in}\ (\mathsf{def}\ D\ \mathsf{in}\ \eta)$$
$$\text{if } dpv(D) \cap (dpv(D') \cup fpv(\eta)) = dpv(D') \cap (dpv(D) \cup fpv(\eta)) = \emptyset$$
$$\Psi \blacklozenge 0 \equiv 0 \quad \frac{N \equiv N'}{\Psi \blacklozenge N \equiv \Psi \blacklozenge N'}$$
$$(\nu \mathsf{s})(\nu \mathsf{s}')\mathcal{S} \equiv (\nu \mathsf{s}')(\nu\ \mathsf{s})\mathcal{S} \quad (\nu \mathsf{s})\mathcal{S}|N \equiv (\nu \mathsf{s})(\mathcal{S}|N) \quad \textit{if } \mathsf{s} \notin fn(N)$$

The rule **(Str)** in Fig. 6 in Sec. 4 uses this definition.

In Def. 12, the rules in the first two lines allow the permutation of messages. $m \cdot m' \curvearrowright m' \cdot m$ means that the order of $m \cdot m'$ can be switched to $m' \cdot m$. Permutation is detailed below. The rules in the next two lines gives structural congruence over recursions. Function $dpv(D)$ gives the set of process variables in declarations, and $fpv(\eta)$ gives the set of process variables which occur free in η. The final two line of rules state structural congruence of systems, each of which combines an application N and a coordinator Ψ. Function $fn(N)$ gives the set of free names in N. The structural congruence rules for restriction and scope extension are standard.

Permutation is possible, in short, as soon as messages and notifications have different sources or destinations they can be permuted ; also the order of done notifications from different levels (e.g., $\phi \neq \phi'$) can be permuted.

Definition 13 (Permutable Messages). We define $m_i \cdot m_j \curvearrowright m_j \cdot m_i$, $i \neq j$, saying $m_i \cdot m_j$ can be permuted to $m_j \cdot m_i$, if *none* of the following conditions holds:

- $m_i = \langle p, q, l(v)\rangle$ and $m_j = \langle p, q, l'(v')\rangle$ for some l, l', v, v'.
- $m_i = \langle \psi, q\rangle^{\phi}$ and $m_j = \langle\!\langle q, \mathsf{crash}\ F\rangle\!\rangle$ for some ϕ, F.

Extracting labels. Rule **(Cln)** in Fig. 7 in Sec. 4 removes messages if the messages contains a label which cannot be reached. (This situation could happen if the process applied **(TryHdl)**).

Def. 14 $labels(\eta)$ extracts all labels of receiving actions in η which potentially can receive messages. The only receiving actions which cannot and never will be able to receive messages in η are those in a handler with signature F in a try-handle (κ, F') where $F \subseteq F'$ (**(TryHdl)** cannot switch the handling to F).

Definition 14 (Extracting Reachable Labels in η).

$$
\begin{aligned}
labels(p!\{l_i(e_i).\eta_i\}_{i\in I}) &= \textstyle\bigcup_{i\in I} labels(\eta_i)\\
labels(p?\{l_i(e_i).\eta_i\}_{i\in I}) &= \textstyle\bigcup_{i\in I}(\{l_i\} \cup labels(\eta_i))\\
labels(\mathsf{t}(\eta)\mathsf{h}(\mathsf{H})^{(\kappa,\emptyset)}.\eta') &= labels(\eta) \cup labels(\eta') \textstyle\bigcup_{(F:\eta'')\in\mathsf{H}} labels(\eta'')\\
labels(\mathsf{t}(\eta)\mathsf{h}(\mathsf{H})^{(\kappa,F)}.\eta') &= labels(\eta) \cup labels(\eta') \textstyle\bigcup_{(F':\eta'')\in\mathsf{H}\wedge F\subset F'} labels(\eta'')\\
labels(0) &= \emptyset\\
labels(\underline{0}) &= \emptyset\\
labels(X\langle e\rangle) &= \emptyset\\
labels(\mathsf{def}\ D\ \mathsf{in}\ \eta) &= labels(\eta) \cup labels(D)\\
labels(\mathsf{if}\ e\ \eta\ \mathsf{else}\ \eta') &= labels(\eta) \cup labels(\eta')\\
labels(X(x) = \eta) &= labels(\eta)
\end{aligned}
$$

B.3 Definitions for Section 6 (Type System)

We define $T \lfloor p$ in Def. 15 as a filter to get the partial type which only contains actions of p in T. For example

$$p_1!l'(S').p_2!l(S) \lfloor p_2 = p_2!l(S)$$

and

$$p_1!\{l_1(S_1).p_2?l(S), l_2(S_2).p_2?l(S)\} \lfloor p_2 = p_2?l(S)$$

Definition 15 (Behaviors for p in T).

$$\mathsf{t}(T)\mathsf{h}(F_1:T_1,...,F_n:T_n)^{\phi}.T' \lfloor p = \begin{cases} \mathsf{t}(T \lfloor p)\mathsf{h}(F_1:T_1 \lfloor p,...,F_n:T_n \lfloor p)^{\phi}.T' \lfloor p & \text{if } p \in \mathit{roles}(T) \cup \bigcup_{i=1}^{n} \mathit{roles}(T_i) \\ T' \lfloor p & \text{otherwise} \end{cases}$$

$$q!\{l_i(S_i).T_i\}_{i\in I} \lfloor p = \begin{cases} p!\{l_i(S_i).T_i \lfloor p\}_{i\in I} & \text{if } p = q \\ T_i \lfloor p & \text{otherwise} \end{cases}$$

$$q?\{l_i(S_i).T_i\}_{i\in I} \lfloor p = \begin{cases} p?\{l_i(S_i).T_i \lfloor p\}_{i\in I} & \text{if } p = q \\ T_i \lfloor p & \text{otherwise} \end{cases}$$

$$\mathsf{end} \lfloor p = \mathsf{end} \quad t \lfloor p = t \quad \mu t.T \lfloor p = \begin{cases} \mu t.T \lfloor p & \text{if } T \lfloor p \neq t' \text{for any } t' \\ \mathsf{end} & \text{otherwise} \end{cases}$$

$\mathit{roles}(T)$ is used (again by slight abuse of notation) to represent the set of roles appearing in T. By the final case of Def. 4 (Projection).(2) if p does not select or receive a label, then the action of p shall be the same for any branch.

Next we define $(\mathtt{h})_{p\to q}$ in Def. 16 to generate (1) the normal message types sent from p heading to q, and (2) the notifications heading to q:

Definition 16 (Message Types to q). We define $(\mathtt{h})_{p\to q}$ by selecting message types and notifications in $\mathtt{h}$ which are (sent from p) heading to q. Let

$$\mathtt{mt} ::= p?l(S) \mid \langle\!\langle F \rangle\!\rangle \mid \langle\psi\rangle^{\phi}$$

and

$$\mathtt{ht} ::= \emptyset \mid \mathtt{mt} \cdot \mathtt{mt}$$

and $\mathtt{ht}$ range over by $(\mathtt{h})_{p\to q}$

$$(\emptyset)_{p\to q} = \emptyset \quad (\mathtt{m}\cdot\mathtt{h})_{p\to q} = \begin{cases} p?l(S)\cdot(\mathtt{h})_{p\to q} & \text{if } \mathtt{m} = \langle p, q, l(S)\rangle \cdot \mathtt{h} \\ \langle\!\langle F \rangle\!\rangle \cdot (\mathtt{h})_{p\to q} & \text{if } \mathtt{m} = \langle\!\langle q, \mathsf{crash}\ F\rangle\!\rangle \cdot \mathtt{h} \\ \langle\psi\rangle^{\phi} \cdot (\mathtt{h})_{p\to q} & \text{if } \mathtt{m} = \langle\psi, q\rangle^{\phi} \cdot \mathtt{h} \\ (\mathtt{h})_{p\to q} & \text{otherwise} \end{cases}$$

We define T–$\mathtt{ht}$ in Def. 17, meaning the effect of $\mathtt{ht}$ on T. Its concept is similar to the *session remainder* defined in [36], which returns new local types of participants after participants consume messages in the global queue. Since failure notifications will not be consumed in our system, and we only have to observe the change of a participant's type after receiving or being triggered by some messages types or notifications in $\mathtt{ht}$, we say that T–$\mathtt{ht}$ represents the effect of $\mathtt{ht}$ on T.

First we define session environment contexts $\mathcal{E} ::= [\,] \mid \mathsf{t}(\mathcal{E})\mathsf{h}(\mathcal{H})^{\phi}.T \mid \mu t.\mathcal{E}$ and define $\mathit{Fset}(\mathtt{ht}, p)$ similarly to Def. 2 by replacing queues with $\mathtt{ht}$. The permutation of $\mathtt{ht}$ is similarly defined as Def. 13 for queues.

Definition 17 (The Effect of $\mathtt{ht}$ on T). We define T–$\mathtt{ht}$ as follows:

$$T\text{–}\emptyset = T \quad \frac{l \notin \mathit{labels}(T)}{T\text{–}q?l(S_k)\cdot \mathtt{ht} = T\text{–}\mathtt{ht}} \quad \frac{\phi \notin T \quad \langle\psi\rangle^{\phi} \in \mathtt{ht}}{T\text{–}\mathtt{ht} = T\text{–}\mathtt{ht} \setminus \langle\psi\rangle^{\phi}}$$

$$\frac{F' = \cup\{A \mid A \in \mathit{dom}(\mathcal{H}) \wedge F \subset A \subseteq \mathit{Fset}(\mathtt{ht}, p)\} \quad F' : T' \in \mathcal{H}}{\mathcal{E}[\mathsf{t}(T)\mathsf{h}(\mathcal{H})^{(\kappa,F)}.T'']\text{–}\mathtt{ht} = \mathcal{E}[\mathsf{t}(T')\mathsf{h}(\mathcal{H})^{(\kappa,F')}.T'']\text{–}\mathtt{ht}}$$

$$\frac{\langle\psi\rangle^{\phi} \in \mathtt{ht}}{\mathcal{E}[\mathsf{t}(\underline{\mathsf{end}})\mathsf{h}(\mathcal{H})^{\phi}.T']\text{–}\mathtt{ht} = \mathcal{E}[T']\text{–}(\mathtt{ht} \setminus \langle\psi\rangle^{\phi})}$$

$$\frac{k \in I}{\mathcal{E}[q?\{l_i(S_i).T_i\}_{i\in I}]\text{–}q?l_k(S_k)\cdot \mathtt{ht} = \mathcal{E}[T_k]\text{–}\mathtt{ht}} \quad \frac{\mathtt{ht} \equiv p?l(S)\cdot \mathtt{ht}' \quad \exists k \in I.l = l_k}{\mathcal{E}[q!\{l_i(S_i).T_i\}_{i\in I}]\text{–}\mathtt{ht} = \mathcal{E}[q!\{l_i(S_i).(T_i\text{–}p?l(S))\}_{i\in I}]\text{–}\mathtt{ht}'}$$

Other cases are undefined.

Notice that those rules are for *general* session environments (i.e., Δ), not particularly for coherent session environments. Here we use the following example to explain the final rule particularly.

Example 13. Assume in Δ we have $\mathsf{s}[p] : T \in \Delta$ such that

$$T = p_1!\{l_1(\mathsf{int}).p_2?l(\mathsf{str}).p_1?l_3(\mathsf{int}),\ l_2(\mathsf{str}).p_2?l(\mathsf{str}).p_1?l_4(\mathsf{str})\}$$

Then, for analyzing the interactions of $\mathsf{s}[p]$ and $\mathsf{s}[p_1]$ only, we focus on p_1's actions by applying Def. 15: $T \upharpoonright p_1 = p_1!\{l_1(\mathsf{int}).p_1?l_3(\mathsf{int}),\ l_2(\mathsf{str}).p_1?l_4(\mathsf{str})\}$.

Assume now we have messages $\langle p_2, p, l(\mathsf{str})\rangle \cdot \langle p_1, p, l_3(\mathsf{int})\rangle$ heading to $\mathsf{s}[p]$. By applying Def. 16, we simplify those messages to only $p_1?l_3(\mathsf{int})$ and ignore the other one because it is sent by p_2. Because an output action $p_1!$ is not able to consume any message, we leave this message to be consumed by the type following $p_1!$, which is either $p_1?l_3(\mathsf{int})$ or $p_1?l_4(\mathsf{str})$. So the result after consuming $\langle p_1, p, l_3(\mathsf{int})\rangle$, we have $p_1!\{l_1(\mathsf{int}).p_1?l_3(\mathsf{int}),\ l_2(\mathsf{str}).p_1?l_4(\mathsf{str})\}\text{–}p_1?l_3(\mathsf{int}) = p_1!\{l_1(\mathsf{int}).\mathsf{end},\ l_2(\mathsf{str}).\mathsf{end}\}$.

However, very importantly, the existence of $\langle p_1, p, l_3(\mathsf{int})\rangle$ and T at the same time shall *never* happen in any coherent session environment. Duality, defined in Def. 7 in Sec. 6, makes sure that a coherent session environment can never have this case: If we can have $\langle p_1, p, l_3(\mathsf{int})\rangle$ before T executes the output $p_1!$, it implies that the local type of $\mathsf{s}[p_1]$ is $T_1 = p!\{l_3(\mathsf{int}).p?l_1(\mathsf{int}), l_4(\mathsf{str}).p?l_2(\mathsf{str})\}$ so that $\mathsf{s}[p] : p_1!.T'\text{–}\emptyset \not\bowtie \mathsf{s}[p_1] : p!.T_1'\text{–}\emptyset$ from the beginning!

Other rules are trivially defined based on our operational semantics of applications and the system in Fig. 6, Fig. 7, and Fig. 8.

C Implementation Details

Based on the calculus presented previously we developed a domain-specific language (DSL) and corresponding runtime system in Scala (without endpoint type checking), using ZooKeeper as the coordinator. In the following we show that our introduced abstractions match our implementation. In particular we show that Zookeeper together with our runtime system provide the coordinator abstraction and that our failure detector assumption (which in practice can be weakened by program-controlled crash) is reasonable.

ZooKeeper as coordinator. Like the global queue which is an abstraction for pairwise channels (e.g., TCP sockets) and can not model a message that is currently in the network vs a message that is in an endpoint buffer, our coordinator is also an abstraction. Before describing how ZooKeeper and our runtime implement the coordinator abstraction we provide a high level description of important ZooKeeper functionalities we use. In essence ZooKeeper provides a hierarchical name space similar to a classical file system which guarantees sequential consistency and atomicity among other things. Clients that work with ZooKeeper enter into a ZooKeeper session (the runtime system ensures that during **(Link)** every linking process enters a ZooKeeper session). ZooKeeper sessions are important for so-called *ephemeral nodes*, which allow clients to save non-persistent data for as long as their ZooKeeper session is running, in contrast to normal nodes which are persisted. ZooKeeper exchanges heartbeats with all clients and if it does not receive heartbeats from a client for a configuration timespan it disconnects that clients (i.e., ends the ZooKeeper session of that client and removes all its ephemeral nodes).

When participants create a new session via the **(Link)** rule the runtime system creates a unique name space for this session and saves the following information in it (amont other): (i) a persistent queue (the notification queue) for all done and failure notification sent to the coordinator, which ensure that all participant see notifications sent to that queue in the same order; (ii) for every participant a node which contains the information whether that participant has terminated (i.e., successfully finished its involvement in the session); (iii) an ephemeral node for every participant that indicates if the participant has an active ZooKeeper session.

Done and failure notification. ZooKeeper by itself does not directly implement the reduction rules **(CollectDone)**,**(IssueDone)** and **(F)**, however ZooKeeper ensures that every participant sees the done and failure notification in the same order. That makes it straightforward for the runtime system to provide the coordinator behavior by applying these rules in order of the notifications in the notification queue. The rules **(CollectDone)**,**(IssueDone)** and **(F)** provide a deterministic outcome (**(IssueDone)** process done notification based on the order in which they arrive). In cases where more then one of these rules is applicable, the runtime system selects a rule based on the following order: **(CollectDone)** over **(IssueDone)** over **(F)**.

Failure detection and false suspicions. The operational semantics model a perfect asynchronous failure detector, in that upon a process crash the **(Crash)** rule adds a failure notification to the queue which the coordinator eventually processes. In practice false suspicions (i.e., non-perfect failure detection) can happen. The failure detection in our implementation works as follows: if the runtime system of a participant p suspects that participant q has failed it only issues a failure notification if participant q has no active ZooKeeper session (the ephemeral node of q does not exist) and the saved status of q in ZooKeeper is not terminated.

The key point here is that a failure notification for a participant q will only be issued if q lost its ZooKeeper session.

However it is possible that a participant q loses its connection with ZooKeeper without failing, e.g., the network drops too many heartbeat messages. After q loses its ZooKeeper session other participants can issue failure notification for q therefore q realizes that it was suspected and stops, i.e., it performs a controlled crash. As mentioned, this occurs very infrequently in practice but is a feature used in many distributed systems as last resort.

If a participant finishes its actions inside a session it sets its status to terminated before disconnecting from ZooKeeper. This ensures that other participants can not detect it as failed.

Automatically inferring try-handle levels and semi-unique labels. For the simplification of the technique developed we require well-formed global types which require that every try-handle is annotated with a unique level. For the formal development we believe those are reasonable assumptions, however our runtime system provides functionalities to remove this burden from the programmer. Concretely, we implemented a deterministic traversal for global types which assigns unique and deterministic levels to all try-handle in a given global type and we allow writing a global type which only contains labels required for branching, adding others automatically.

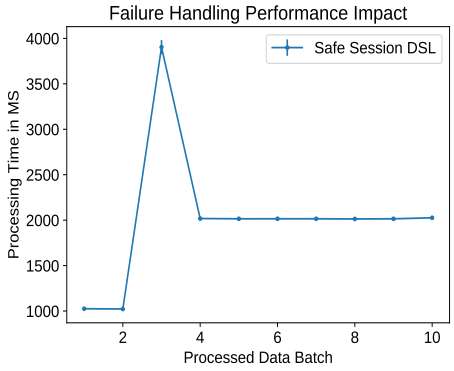

Fig. 14. Here two workers implemented in our DSL read data from ZooKeeper, simulate work on it, and write a result back. We inject a failure during the processing of batch number 3, which is clearly visible in the spike. At that point the first worker starts handling the failure of worker two, performing both works.

Evaluation. We evaluated our prototype on a microbenchmark, derived from the big data task shown in Fig. 2. For the evaluation all participants and the different ZooKeeper replicas run on different nodes. In all evaluation ZooKeeper runs in a three-server setting, i.e., we have three different nodes which all have a

ZooKeeper replica running and therefore the ZooKeeper setup can tolerate one process crash while still remaining operational. In this task two workers (w_1, w_2) read data from a distributed file system (*dfs*), simulate working on the data and then writes data back into the file system. The ZooKeeper setup provides reliable data source and sink (*dfs*) used in the benchmarks and the runtime systems are extended so that it can treat ZooKeeper like a regular robust participant. In order to measure the overhead of our system we deployed all nodes in distinct virtual machines on the same server, with the following spec: Intel(R) Xeon(R) CPU E5-2680 v3 2.50GHz and 126 GB of RAM; each node has 4 cores assigned and 8 GB of RAM. In addition ZooKeeper is configured to keep ZooKeeper session for at least 1000 milliseconds alive after the last successfully exchanged heartbeat. Our benchmark differs from the global type in Fig. 2 in that the workers read data, i.e., we have two communication steps instead of one. For evaluation purposes the actions inside the try-block and the handlers are repeated 10 times. Every repetition uses a distinct data batch and each worker reads one partition from this batch. Each batch contains two partitions, the worker w_1 works on partition one and worker w_2 work on partition two. In case of a failure the non-crashed worker processes both partitions.

The microbenchmark measures the cost of handling and recovering from a failure if work is performed. We simulate the processing of a partition of a batch as taking 1000 milliseconds. In this evaluation we manually stop the node of w_2 (the Linux system in which w_2 runs) when w_2 is processing the data from the batch. The graph in Fig. 14 shows a spike at batch three where the crash is injected. This spike shows the combined time of: crash detection, involvement of coordinator and worker on doing the processing of the entire batch three (i.e., also doing the work of w_2). For batches 4 to 10 the processing time doubles since w_1 also does the work of w_2. Apart from the time where failure handling is triggered one can see that the runtime is entirely dominated by the work, i.e., the overhead in the non-failure case is negligible.

D Proofs of Properties

This part of appendix provides the proofs and auxiliary definitions and lemmas for the proposition and theorems in the paper.

D.1 Proof for Section 4 (Process Calculus)

Proposition 1. Given $\mathsf{s} : h$ with $h = h' \cdot \langle \psi, p \rangle^{\phi} \cdot h''$ and $Fset(h, p) \neq \emptyset$, the rule **(TryHdl)** is not applicable for the try-handle ϕ at the process playing role p.

Proof. W.o.l.g. assume the coordinator is $\Psi = G : (F_q, d)$. Since the failure and done notifications sent by Ψ to the same participant have an order (not permutable by Def. 13), we have the following two cases:

1. If $\exists \langle\!\langle p, \mathsf{crash}\ F'' \rangle\!\rangle \in h'$, then we prove the statement by contradiction. Assume we have $F \in hdl(G, \phi)$ and $F = \cup\{A \mid A \in dom(\mathsf{H}) \wedge {F_s}' \subset A \subseteq Fset(h, p)\}$. Since $Fset(h, p) \subseteq F_q$, immediately rule **(IssueDone)** is violated.
2. If $\exists \langle\!\langle p, \mathsf{crash}\ F'' \rangle\!\rangle \in h''$, then we prove the statement by Def. 2 ($Fset$). By Def. 2, $\langle\!\langle p, \mathsf{crash}\ F'' \rangle\!\rangle \notin Fset(h, p)$, therefore $F'' \notin \cup\{A \mid A \in dom(\mathsf{H}) \wedge {F_s}' \subset A \subseteq Fset(h, p)\}$.

D.2 Proofs for Section 7 (Properties)

To prove the properties of subject congruence, subject reduction, and progress, we need to first prove one theorem, Theorem 4 (Preservation of Coherence), which is the basis for proving those properties. This section goes as follows: we prove Theorem 4 (Preservation of Coherence), then prove Theorem 1 (Subject Congruence) and Theorem 2 (Subject Reduction), and finally prove Theorem 3 (Progress). For each proof, several auxiliary formal definitions are introduced, and useful lemmas are stated and proved.

Preservation of Coherence. Before stating and proving preservation of coherence, we first define the reduction relation over session environments

$$\Psi \vdash \ \Delta \rightarrow_T \ \Psi' \vdash \ \Delta'$$

which is defined based on the operational semantics of applications and systems defined in Sec. 4.

Definition 18 (Reduction relation over session environments).

$$\mathsf{s}[p] : \mathcal{E}[q!\{l_i(S_i).T_i\}_{i\in I}], \mathsf{s} : \mathtt{h} \to_T \mathsf{s}[p] : \mathcal{E}[T_k], \mathsf{s} : \mathtt{h} \cdot \langle p, q, l_k(S_k)\rangle \quad k \in I \qquad [\![\texttt{Snd}]\!]$$

$$\mathsf{s}[p] : \mathcal{E}[q?\{l_i(S_i).T_i\}_{i\in I}], \mathsf{s} : \langle q, p, l_k(S_k)\rangle \cdot \mathtt{h} \to_T \mathsf{s}[p] : \mathcal{E}[T_k], \mathsf{s} : \mathtt{h} \quad k \in I \qquad [\![\texttt{Rcv}]\!]$$

$$\Delta, \{\mathsf{s} : \mathtt{h}\} \to \Delta \setminus \{\mathsf{s}[p] : T\}, \{\mathsf{s} : \mathit{remove}(\mathtt{h}, p) \cdot (\!|\psi, \mathsf{crash}\ p|\!)\} \quad p \text{ non-robust} \qquad [\![\texttt{Crash}]\!]$$

$$\frac{F' = \cup\{A \mid A \in \mathit{dom}(\mathsf{H}) \wedge F \subset A \subseteq \mathit{Fset}(\mathtt{h}, p)\} \quad F' : T' \in \mathcal{H}}{\mathsf{s}[p] : \mathcal{E}[\mathsf{t}(T)\mathsf{h}(\mathcal{H})^{(\kappa,F)}.T''], \mathsf{s} : \mathtt{h} \to \mathsf{s}[p] : \mathcal{E}[\mathsf{t}(T')\mathsf{h}(\mathcal{H})^{(\kappa,F')}.T''], \mathsf{s} : \mathtt{h}} \qquad [\![\texttt{TryHdl}]\!]$$

$$\mathsf{s}[p] : \mathcal{E}[\mathsf{t}(\mathsf{end})\mathsf{h}(\mathcal{H})^{\phi}.T],\ \mathsf{s} : \mathtt{h} \to \mathsf{s}[p] : \mathcal{E}[\mathsf{t}(\underline{\mathsf{end}})\mathsf{h}(\mathcal{H})^{\phi}.T],\ \mathsf{s} : \mathtt{h} \cdot \langle p, \psi\rangle^{\phi} \qquad [\![\texttt{SndDone}]\!]$$

$$\frac{\langle \psi, p\rangle^{\phi} \in \mathtt{h}}{\mathsf{s}[p] : \mathcal{E}[\mathsf{t}(\underline{\mathsf{end}})\mathsf{h}(\mathcal{H})^{\phi}.T],\ \mathsf{s} : \mathtt{h} \to \mathsf{s}[p] : \mathcal{E}[T],\ \mathsf{s} : \mathtt{h} \setminus \{\langle \psi, p\rangle^{\phi}\}} \qquad [\![\texttt{RcvDone}]\!]$$

$$\mathsf{s}[p] : \mathcal{E}[T],\ \mathsf{s} : \langle q, p, l(S)\rangle \cdot \mathtt{h} \to \mathsf{s}[p] : \mathcal{E}[T],\ \mathsf{s} : \mathtt{h} \quad l \notin \mathit{labels}(\mathcal{E}[T]) \qquad [\![\texttt{Cln}]\!]$$

$$\frac{\langle \psi, p\rangle^{\phi} \in \mathtt{h} \quad \phi \notin \mathcal{E}[T]}{\mathsf{s}[p] : \mathcal{E}[T],\ \mathsf{s} : \mathtt{h} \to \mathsf{s}[p] : \mathcal{E}[T] \ \mid\ \mathsf{s} : \mathtt{h} \setminus \{\langle \psi, p\rangle^{\phi}\}} \qquad [\![\texttt{ClnDone}]\!]$$

$$\frac{\tilde{p} = \mathit{roles}(G) \setminus (F \cup \{p\}) \quad \mathtt{h}' = \mathtt{h} \cdot (\!|\tilde{p}, \mathsf{crash}\ \{p\}|\!)}{G : (F, d) \vdash\ N \ \mid\ \mathsf{s} : (\!|\psi, \mathsf{crash}\ \{p\}|\!) \cdot \mathtt{h} \to_T G : (F \cup \{p\}, d) \vdash\ N \ \mid\ \mathsf{s} : \mathtt{h}'} \qquad [\![\texttt{F}]\!]$$

$$\frac{d' = d \cdot \langle p, \psi\rangle^{\phi}}{G : (F, d) \vdash\ \mathsf{s} : \langle p, \psi\rangle^{\phi} \cdot \mathtt{h} \to_T G : (F, d') \vdash\ \mathsf{s} : \mathtt{h}} \qquad [\![\texttt{CollectDone}]\!]$$

$$\frac{\mathit{roles}(d, \phi) \supseteq \mathit{roles}(G, \phi) \setminus F \quad \forall F' \in \mathit{hdl}(G, \phi).(F' \not\subseteq F) \quad d' = \mathit{remove}(d, \phi)}{G : (F, d) \vdash\ \mathsf{s} : \mathtt{h} \to_T G : (F, d') \vdash\ \mathsf{s} : \mathtt{h} \cdot \langle \psi, \mathit{roles}(G, \phi) \setminus F\rangle^{\phi}} \qquad [\![\texttt{IssueDone}]\!]$$

$$\frac{\Delta \to_T \Delta'}{\Delta, \Delta_0 \to_T \Delta', \Delta_0} \qquad [\![\texttt{Str}]\!]$$

When Ψ is not changed during reduction, i.e., $\Psi \vdash\ \Delta \to_T \Psi \vdash\ \Delta'$, we simply write the reduction relation as $\Delta \to_T \Delta'$. If $\Psi \vdash\ \Delta \to_T \Psi' \vdash\ \Delta'$ and $\Gamma = \Gamma_0, \Psi$ and $\Gamma' = \Gamma_0, \Psi'$, then we write $\Gamma \vdash\ \Delta \to_T \Gamma' \vdash\ \Delta'$. Remember we have defined the context of local types in App. B.3 $\mathcal{E} ::= [\,] \mid \mathsf{t}(\mathcal{E})\mathsf{h}(\mathcal{H})^{\phi}.T \mid \mu t.\mathcal{E}$.

Assume G is well-formed and there is a session s such that $\Delta = \{\mathsf{s}[p_1] : G\upharpoonright p_1, ..., \mathsf{s}[p_n] : G\upharpoonright p_n\} \cup \{\mathsf{s} : \emptyset\}$ and $\mathit{roles}(G) = \{p_1, ..., p_n\}$. The immediate question for Def. 8 (Coherence) is whether this definition ensures the interactions among endpoints do progress with the guidance of G or not.

It is not trivial because, for the endpoints, they can take actions concurrently, failures can occur randomly, and message types/notifications are transmitted in an asynchronous manner. Consider a simple example:

$$G = p_1 \to p_2\ l_1(\mathsf{int}).p_3 \to p_2\ l_2(\mathsf{str})$$

for a s that obeys to G, we will have

$$\Delta = \{\mathsf{s}[p_1] : p_2!l_1(\mathsf{int}).\mathsf{end}, \mathsf{s}[p_2] : p_1?l_1(\mathsf{int}).p_3?l_2(\mathsf{str}).\mathsf{end}, \mathsf{s}[p_3] : p_2!l_2(\mathsf{str}).\mathsf{end}, \mathsf{s} : \emptyset\}$$

and $\Delta \to_T \Delta'$ such that

$$\Delta' = \{\mathsf{s}[p_1] : p_2!l_1(\mathsf{int}).\mathsf{end}, \mathsf{s}[p_2] : p_1?l_1(\mathsf{int}).p_3?l_2(\mathsf{str}).\mathsf{end}, \mathsf{s}[p_3] : \mathsf{end}, \mathsf{s} : \langle p_3, p_2, l_2(\mathsf{str})\rangle\}$$

At this step, $\langle p_3, p_2, l_2(\mathsf{str})\rangle$ cannot trigger its receiver $\mathsf{s}[p_2]$ because the corresponding receiving action $p_3?l_2(\mathsf{str})$ is guarded by $p_1?l_1(\mathsf{int})$.

Note that, Δ' is still able to reduce because $\mathsf{s}[p_1]$ can take action. When $\mathsf{s}[p_1]$ takes action and outputs $\langle p_1, p_2, l_2(\mathsf{int})\rangle$, its receiver $\mathsf{s}[p_2]$ can absorb $\langle p_3, p_2, l_2(\mathsf{str})\rangle$ immediately.

Then in the next reduction, $\mathsf{s}[p_2]$ will absorb $\langle p_3, p_2, l_2(\mathsf{str})\rangle$.

We observe that, not every sending action fired by a well-typed behavior can trigger its expected receiver immediately; but this will happen eventually.

The following simple lemmas and definitions will be used in the proof for Theorem 4 (Preservation of Coherence).

Lemma 1. If $T \lfloor p = \mathcal{E}[p\ w\ \{l_i(S_i).T_i\}_{i\in I}]$ and $w \in \{!, ?\}$ for some $\mathcal{E}$ and $T_i, i \in I$, then for any $p' \neq p$ we have for any $j, k \in I, j \neq k, \mathcal{E}[T_j] \lfloor p' = \mathcal{E}[T_k] \lfloor p'$.

Proof. Immediately by Def. 4 and Def. 7 (Duality).

Lemma 2. For any G, $p \in \mathit{roles}(G)$ and $q \notin \mathit{roles}(G)$ imply $\mathsf{end} = G\upharpoonright q \lfloor p \bowtie G\upharpoonright p \lfloor q = \mathsf{end}$.

Proof. Immediately by Def. 4 (Projection) and Def. 15.

The next lemma states that, given a session environment, if each of its endpoint's type is projected from a well-formed global type, then this session environment is coherent.

Lemma 3. Given G is well-formed and $\Delta_{\mathsf{s}} = \{\mathsf{s}[p_1] : T_1, ..., \mathsf{s}[p_n] : T_n, \mathsf{s} : \emptyset\}$ and $\{p_1, ..., p_n\} = \mathit{roles}(G)$ and $T_i = G\upharpoonright p_i$ for $i \in \{1..n\}$. Then $\Gamma, G : (\emptyset, \emptyset) \vdash \Delta_{\mathsf{s}}$ is coherent.

Proof. By the structure of a well-formed G, the proof is immediate by Def. 4 (Projection), Def. 7 (Duality), and Def. 8 (Coherence).

And, after a failure occurs, if a coordinator has not yet issued failure notifications to endpoints in a coherent Δ for handling this failure, the types of all non-failed endpoints are still dual to each other in Δ.

Lemma 4. Assuming $\Delta = \Delta_0, \{\mathsf{s} : \mathtt{h}\}$ is coherent and $\Delta \rightarrow_T \Delta_0 \setminus \{\mathsf{s}[p] : T\}, \{\mathsf{s} : \mathit{remove}(\mathtt{h}, p) \cdot \langle\!\langle \psi, \mathsf{crash}\ p\rangle\!\rangle\} = \Delta'$. Then $\forall \mathsf{s}[p] : T \in \Delta$, if $\mathsf{s}[q] : T' \in \Delta$ then $\mathsf{s}[p] : T \lfloor q{-}(\mathtt{h})_{q\to p} \bowtie \mathsf{s}[q] : T' \lfloor p{-}(\mathtt{h})_{p\to q}$.

Proof. Assume the non-failed endpoints are $\{\mathsf{s}[p_i] : T_i\}_{i\in I}$. Since Δ is coherent, $\{\mathsf{s}[p_i] : T_i\}_{i\in I}$ in Δ are dual to each other after considering the effect of $\mathtt{h}$ on each of them. By the rules defined in Def. 17, we know $\langle\!\langle \psi, \mathsf{crash}\ p\rangle\!\rangle$ will not affect any non-failed endpoints. So the endpoints of $\{\mathsf{s}[p_i] : T_i\}_{i\in I}$ in Δ' are still dual to each other after considering the effect of $\mathit{remove}(\mathtt{h}, p) \cdot \langle\!\langle \psi, \mathsf{crash}\ p\rangle\!\rangle$ on each of them.

To reason about asynchrony in Δ, we define message types permutation, which is very similar to Def. 13.

Definition 19 (Permutable Message Types). We define $\mathtt{m}_i \cdot \mathtt{m}_j \curvearrowright \mathtt{m}_j \cdot \mathtt{m}_i$, $i \neq j$, saying $\mathtt{m}_i \cdot \mathtt{m}_j$ can be permuted to $\mathtt{m}_j \cdot \mathtt{m}_i$, if *none* of the following conditions holds:

- $\mathtt{m}_i = \langle p, q, l(S) \rangle$ and $\mathtt{m}_j = \langle p, q, l'(S') \rangle$ for some l, l', S, S'.
- $\mathtt{m}_i = \langle \psi, q \rangle^{\phi}$ and $\mathtt{m}_j = \langle\!\langle q, \mathsf{crash}\ F \rangle\!\rangle$ for some ϕ, F.

For convenience we say that participants currently performing actions in try-handle ϕ are *partners*. We also give the following function *HdlLogic* to extract a particular try-handle for a handler. This function is used for proving Case $[\![\mathtt{F}]\!]$ in Theorem 4.

When $\mathsf{t}(G_0 \upharpoonright p)\mathsf{h}(F_1\ G_1 \upharpoonright p, ..., F_n\ G_n \upharpoonright p)^{(\kappa,\emptyset)}$ is generated by Def. 4 (Projection), for other handler signatures in the try-handle of κ can be automatically generated by

Definition 20. Given $\mathsf{t}(T)\mathsf{h}(\mathcal{H})^{(\kappa,\emptyset)}$ and $\mathcal{H} = F_1\ T_1, ..., F_n\ T_n$. We define

$$\mathit{HdlLogic}(\mathsf{t}(T)\mathsf{h}(\mathcal{H})^{(\kappa,\emptyset)}, F) = \mathsf{t}(T_i)\mathsf{h}(\mathcal{H})^{(\kappa,F)} \text{ if } F = F_i, i \in \{1..n\}$$

Also for convenience, we define

$$\mathit{notify}(p) ::= \langle\!\langle p, \mathsf{crash}\ F \rangle\!\rangle \mid \langle\!\langle \psi, \mathsf{crash}\ F \rangle\!\rangle \mid \langle \psi, p \rangle^{\phi} \mid \langle p, \psi \rangle^{\phi}$$

The following proof states that a coherent session environment will reduce to another coherent session environment.

Theorem 4 (Preservation of Coherence). $\Gamma \vdash \Delta$ coherent and $\Gamma = G : (F, d), \Gamma'$ and $G : (F, d) \vdash \Delta \rightarrow_T G : (F', d') \vdash \Delta'$ imply that $\Gamma', G : (F', d') \vdash \Delta'$ is coherent.

Proof. We prove the statement by mechanically proving each cases.

1. Case $\mathsf{s} : \emptyset$. We have the following subcases.

   (I) Case $[\![\mathtt{Snd}]\!]$, there exists $\mathsf{s}[p] : T \in \Delta, T = \mathcal{E}[q!\{l_i(S_i).T_i\}_{i \in I}]$.

   W.o.l.g. assume

   $$\Delta = \Delta_0, \mathsf{s}[p] : T, \mathsf{s} : \emptyset$$

   such that

   $$G : (F_q, d) \vdash \Delta_0, \mathsf{s}[p] : T, \mathsf{s} : \emptyset \rightarrow_T G : (F_q, d) \vdash \Delta_0, \mathsf{s}[p] : \mathcal{E}[T_k], \mathsf{s} : \langle p, q, l_k(S_k) \rangle$$

   for some $k \in I$. Let $\Delta' = \Delta_0, \mathsf{s}[p] : \mathcal{E}[T_k], \mathsf{s} : \langle p, q, l_k(S_k) \rangle$.

   In Δ, by Def. 8.(2), for any $\mathsf{s}[q'] : T' \in \Delta$, we have

   $$\mathsf{s}[p] : T \downharpoonright q' \bowtie \mathsf{s}[q'] : T' \downharpoonright p$$

In Δ, if $q' = q$, then let $T \lfloor q = \mathcal{E}''[q!\{l_i(S_i).T_i''\}_{i\in I}]$ for some $\mathcal{E}''$ and $T_i'', i \in I$. By Def. 7 $T' \lfloor p = \mathcal{E}'[p?\{l_i(S_i).T_i'\}_{i\in I}]$ for some $\mathcal{E}'$ and $T_i', i \in I$ and

$$\forall i \in I.\ \mathsf{s}[p] : \mathcal{E}''[T_i''] \bowtie \mathsf{s}[q] : \mathcal{E}'[T_i'].$$

In Δ', by Def. 17 (The Effect of $\mathtt{ht}$) and Def. 7 (Duality), we have $\mathsf{s}[p] : \mathcal{E}[T_k]$ and $\mathsf{s}[q] : T'$ such that

$$\begin{aligned}\mathsf{s}[p] : \mathcal{E}[T_k] \lfloor q - (\langle p, q, l_k(S_k)\rangle)_{q\to p} &= \mathsf{s}[p] : \mathcal{E}''[T_k''] \\ &\bowtie \mathsf{s}[q] : \mathcal{E}'[T_k'] \\ &= \mathsf{s}[q] : T' \lfloor p - (\langle p, q, l_k(S_k)\rangle)_{p\to q}\end{aligned}$$

In Δ, for any $q' \neq q$, $q' \in \mathit{roles}(G)$ and $\mathsf{s}[q'] : T'' \in \Delta$, by Lemma 1, the fact that $T \lfloor q' = \mathcal{E}[q!\{l_i(S_i).T_i\}_{i\in I}] \lfloor q'$ is defined implies that for any $j, k \in I$, $j \neq k$, we have $\mathcal{E}[T_j] \lfloor q' = \mathcal{E}[T_k] \lfloor q'$ and

$$\mathsf{s}[p] : T \lfloor q' = \mathsf{s}[p] : \mathcal{E}[T_j] \lfloor q' \bowtie \mathsf{s}[q'] : T'' \lfloor p$$

In Δ' , by Def. 17 (The Effect of $\mathtt{ht}$) and Def. 7 (Duality), we have $\mathsf{s}[p] : \mathcal{E}[T_k]$ and $\mathsf{s}[q'] : T''$ such that

$$\begin{aligned}\mathsf{s}[p] : \mathcal{E}[T_k] \lfloor q' - (\langle p, q, l_k(S_k)\rangle)_{q'\to p} &= \mathsf{s}[p] : \mathcal{E}[T_k] \lfloor q' \\ &\bowtie \mathsf{s}[q'] : T'' \lfloor p - (\langle p, q, l_k(S_k)\rangle)_{p\to q'} \\ &= \mathsf{s}[q'] : T'' \lfloor p\end{aligned}$$

Since the types of other endpoints are not changed, $\Gamma', G : (F_q, d) \vdash \Delta'$ is coherent.

(II) Case $[\![\mathtt{Crash}]\!]$, some endpoint in Δ crashes. W.o.l.g. assume $\Delta = \Delta_0, \mathsf{s}[p] : T, \mathsf{s} : \emptyset$ such that

$$G : (F_q, d) \vdash \Delta_0, \mathsf{s}[p] : T, \mathsf{s} : \emptyset \to_T G : (F_q, d) \vdash \Delta_0, \mathsf{s} : \langle\!\langle \psi, \mathsf{crash}\ F\rangle\!\rangle$$

Let $\Delta' = \Delta_0, \mathsf{s} : \langle\!\langle \psi, \mathsf{crash}\ F\rangle\!\rangle$.

By Lemma 4 we have $\Gamma \vdash \Delta'$ is coherent

(III) Case $[\![\mathtt{SndDone}]\!]$, some endpoint in Δ finishes its default action. W.o.l.g. assume $\Delta = \Delta_0, \mathsf{s}[p] : T, \mathsf{s} : \emptyset$ and $T = \mathcal{E}[\mathsf{t}(\mathsf{end})\mathsf{h}(\mathcal{H})^\phi.T']$ such that

$$\begin{aligned}&G : (F_q, d) \vdash \Delta_0, \mathsf{s}[p] : T, \mathsf{s} : \emptyset \\ &\quad\to_T G : (F_q, d) \vdash \Delta_0, \mathsf{s}[p] : \mathcal{E}[\mathsf{t}(\underline{\mathsf{end}})\mathsf{h}(\mathcal{H})^\phi.T'], \mathsf{s} : \langle p, \psi\rangle^\phi\end{aligned}$$

Let $\Delta' = \Delta_0, \mathsf{s}[p] : \mathcal{E}[\mathsf{t}(\underline{\mathsf{end}})\mathsf{h}(\mathcal{H})^\phi.T'], \mathsf{s} : \langle p, \psi\rangle^\phi$.

Since this kind of message type will not affect any endpoints in Δ_0 and by Def. 8.(2) for any $\mathsf{s}[q] : T' \in \Delta$ we have $\mathsf{s}[p] : T \lfloor q \bowtie \mathsf{s}[q] : T' \lfloor p$ so $\Gamma \vdash \Delta'$ is coherent.

2. Case $\mathtt{s} : \mathtt{h} \neq \emptyset$ and $\Gamma \vdash \Delta$ is coherent.[7]
   (I) Case $[\![\mathtt{Snd}]\!]$, there exists $\mathtt{s}[p] : T \in \Delta, T = \mathcal{E}[q!\{l_i(S_i).T_i\}_{i\in I}]$.

   W.o.l.g. assume

$$\Delta = \Delta_0, \mathtt{s}[p] : T, \mathtt{s} : \mathtt{h}$$

   such that

$$\begin{aligned} &G : (F_q, d) \vdash \Delta_0, \mathtt{s}[p] : T, \mathtt{s} : \mathtt{h} \\ &\quad \to_T G : (F_q, d) \vdash \Delta_0, \mathtt{s}[p] : \mathcal{E}[T_k], \mathtt{s} : \mathtt{h} \cdot \langle p, q, l_k(S_k)\rangle \end{aligned}$$

   for some $k \in I$. Let $\Delta' = \Delta_0, \mathtt{s}[p] : \mathcal{E}[T_k], \mathtt{s} : \mathtt{h} \cdot \langle p, q, l_k(S_k)\rangle$.

   Then we have the following cases:

   (A) Case $\forall p \in \mathit{roles}(G).\ \mathit{notify}(p) \notin \mathtt{h}$ which implies $\langle\!\langle \psi, \mathsf{crash}\ F]\!] \notin \mathtt{h}$ (i.e. $\mathtt{h}$ contains only normal message types).

   In Δ, by Def. 17 (The Effect of $\mathtt{ht}$) and Def. 8 (2). (a), for any $\mathtt{s}[q'] : T' \in \Delta$ we have

$$\begin{aligned} \mathtt{s}[p] : T \upharpoonright q' - (\mathtt{h})_{q'\to p} &= \mathtt{s}[p] : \mathcal{E}[q!\{l_i(S_i).T_i\}_{i\in I}] \upharpoonright q' - (\mathtt{h})_{q'\to p} \\ &\bowtie \mathtt{s}[q'] : T' \upharpoonright p - (\mathtt{h})_{p\to q'} \end{aligned}$$

   In Δ, if $q' = q$, we have

$$T \upharpoonright q - (\mathtt{h})_{q\to p} = \mathcal{E}[q!\{l_i(S_i).T_i\}_{i\in I}] \upharpoonright q - (\mathtt{h})_{q\to p} = \mathcal{E}''[q!\{l_i(S_i).T_i'''\}_{i\in I}]$$

   for some $\mathcal{E}''$ and $T_i''', i \in I$ and by Def. 7 (Duality), we have $T' \upharpoonright p - (\mathtt{h})_{p\to q} = \mathcal{E}'[p?\{l_i(S_i).T_i'\}_{i\in I}]$ for some $\mathcal{E}'$ and

$$\forall i \in I.\ \mathtt{s}[p] : \mathcal{E}''[T_i'''] \bowtie \mathtt{s}[q] : \mathcal{E}'[T_i'].$$

   By Def. 17 (The Effect of $\mathtt{ht}$) and Def. 7 (Duality)

$$\begin{aligned} &\mathtt{s}[p] : \mathcal{E}[T_k] \upharpoonright q - (\mathtt{h} \cdot \langle p, q, l_k(S_k)\rangle)_{q\to p} \\ &= \mathtt{s}[p] : \mathcal{E}[T_k] \upharpoonright q \\ &= \mathtt{s}[p] : \mathcal{E}''[T_k'''] \\ &\bowtie \mathtt{s}[q] : \mathcal{E}'[T_k'] \\ &= \mathtt{s}[q] : T' \upharpoonright p - (\mathtt{h} \cdot \langle p, q, l_k(S_k)\rangle)_{p\to q} \end{aligned}$$

   In Δ, for any $q' \neq q$, let $\mathtt{s}[q'] : T'' \in \Delta$. By Lemma 1, $\forall j, k \in I, j \neq k$ we have $\mathcal{E}[T_j] \upharpoonright q' - (\mathtt{h})_{q'\to p} = \mathcal{E}[T_k] \upharpoonright q' - (\mathtt{h})_{q'\to p}$ and

$$\begin{aligned} \mathtt{s}[p] : T \upharpoonright q' - (\mathtt{h})_{q'\to p} &= \mathtt{s}[p] : \mathcal{E}[q!\{l_i(S_i).T_i\}_{i\in I}] \upharpoonright q' - (\mathtt{h})_{q'\to p} \\ &= \mathtt{s}[p] : \mathcal{E}[T_j] \upharpoonright q' - (\mathtt{h})_{q'\to p} \\ &\bowtie \mathtt{s}[q'] : T'' \upharpoonright p - (\mathtt{h})_{p\to q'} \end{aligned}$$

[7] For convenience we sometimes write $\mathtt{s}[p] : T \upharpoonright p - (\mathtt{h})_{p\to q} = \mathtt{s}[p] : T \upharpoonright p$ for $(\mathtt{h})_{p\to q} \neq \emptyset$ if $(\mathtt{h})_{p\to q}$ contains only failure notification and they have no effect on $\mathtt{s}[p] : T \upharpoonright p$

Thus in Δ' we have

$$\begin{aligned}
&\mathsf{s}[p] : \mathcal{E}[T_k] \lfloor q' - (\mathtt{h} \cdot \langle p, q, l_k(S_k)\rangle)_{q' \to p} \\
=\ &\mathsf{s}[p] : \mathcal{E}[T_k] \lfloor q' - (\mathtt{h})_{q' \to p} \\
\bowtie\ &\mathsf{s}[q'] : T'' \lfloor p - (\mathtt{h})_{p \to q'} \\
=\ &\mathsf{s}[q'] : T'' \lfloor p - (\mathtt{h} \cdot \langle p, q, l_k(S_k)\rangle)_{p \to q'}
\end{aligned}$$

Since the types of other endpoints are not changed, $\Gamma \vdash \Delta'$ is coherent.

(B) Case $\exists \langle\!\langle p_0, \mathsf{crash}\ F' \rangle\!\rangle \in \mathtt{h}$ for some F' and p_0.

Since $\langle\!\langle p_0, \mathsf{crash}\ F' \rangle\!\rangle$ is issued by $[\![\mathtt{F}]\!]$, we have $\forall \mathsf{s}[p_i] \in \mathit{dom}(\Delta_{\mathsf{s}})$, $\langle\!\langle p_i, \mathsf{crash}\ F' \rangle\!\rangle \in \mathtt{h}$ (by $[\![\mathtt{Crash}]\!]$, only the alive endpoints).

W.o.l.g assume $\Delta = \Delta_0, \mathsf{s}[p] : T, \mathsf{s}[q] : T', \mathsf{s} : \mathtt{h}$ and assume $T = \mathcal{E}[q!\{l_i(S_i).T_i''\}_{i \in I}]$ and

$$\begin{aligned}
&G : (F_q, d) \vdash \Delta_0, \mathsf{s}[p] : T, \mathsf{s}[q] : T', \mathsf{s} : \mathtt{h} \to_T \\
&\quad G : (F_q, d) \vdash \Delta_0, \mathsf{s}[p] : \mathcal{E}[T_k], \mathsf{s}[q] : T', \mathsf{s} : \mathtt{h} \cdot \langle p, q, l_k(S_k)\rangle
\end{aligned}$$

In Δ we observe

* If $T = \mathcal{E}'''[\mathtt{t}(\mathcal{E}''[q!\{l_i(S_i).T_i''\}_{i \in I}])\mathtt{h}(F : T_F, \mathcal{H})^{(\kappa, F_s)}]$, where T_F is the handler bodyfor F, and $F = \cup\{A \mid A \in \mathit{dom}(\mathcal{H}) \wedge F_s \subset A \subseteq \mathit{Fset}(\mathtt{h}, p)\}$ (notice that $\langle \psi, p\rangle^{(\kappa, F_s)} \notin \mathtt{h}$ because endpoint $\mathsf{s}[p]$ who is still taking actions in ϕ), then by Def. 8 and Def. 7 (Duality) and Def. 17 (The Effect of $\mathtt{ht}$), we have

$$\begin{aligned}
&\mathsf{s}[p] : T \lfloor q - (\mathtt{h})_{q \to p} \\
=\ &\mathsf{s}[p] : \mathcal{E}'''[\mathtt{t}(\mathcal{E}''[q!\{l_i(S_i).T_i''\}_{i \in I}])\mathtt{h}(F : T_F, \mathcal{H})^{(\kappa, F_s)}] \lfloor q - (\mathtt{h})_{q \to p} \\
=\ &\mathsf{s}[p] : \mathcal{E}'''[\mathtt{t}(T_F)\mathtt{h}(F : T_F, \mathcal{H})^{(\kappa, F)}] \lfloor q - (\mathtt{h})_{q \to p} \\
\bowtie\ &\mathsf{s}[q] : T' \lfloor p - (\mathtt{h})_{p \to q} \\
=\ &\mathsf{s}[q] : \mathcal{E}''''[\mathtt{t}(T'_F)\mathtt{h}(F : T'_F, \mathcal{H}')^{(\kappa, F)}] \lfloor p - (\mathtt{h})_{p \to q}
\end{aligned}$$

where $T_F \bowtie T'_F$ by Def. 7.

Now $l_k \notin T'$ due to the occurrence of failure

In Δ' we observe

$$\begin{aligned}
&\mathsf{s}[p] : \mathcal{E}[T_k] \lfloor q - (\mathtt{h} \cdot \langle p, q, l_k(S_k)\rangle)_{q \to p} \\
=\ &\mathsf{s}[p] : \mathcal{E}'''[\mathtt{t}(T_F)\mathtt{h}(F : T_F, \mathcal{H})^{(\kappa, F)}] \lfloor q - (\mathtt{h})_{q \to p} \\
\bowtie\ &\mathsf{s}[q] : \mathcal{E}''''[\mathtt{t}(T'_F)\mathtt{h}(F : T'_F, \mathcal{H}')^{(\kappa, F)}] \lfloor p - (\mathtt{h})_{p \to q} \\
=\ &\mathsf{s}[q] : T' \lfloor p - (\mathtt{h})_{p \to q} \\
=\ &\mathsf{s}[q] : T' \lfloor p - (\mathtt{h} \cdot \langle p, q, l_k(S_k)\rangle)_{p \to q}
\end{aligned}$$

because $\langle p, q, l_k(S_k)\rangle$ is removed by Def. 11 (Well-formedness).(3) and rule $[\![\mathtt{cln}]\!]$ since $l_k \notin \mathit{labels}(T')$ (i.e., $\langle p, q, l_k(S_k)\rangle$ is a message sent out after failures which can be handled by $\mathsf{s}[p]$ and $\mathsf{s}[q]$

occur).

* A case like $T = \mathcal{E}'''[\mathsf{t}(\mathcal{E}''[q!\{l_i(S_i).T_i''\}_{i\in I}])\mathsf{h}(F : T_F)^{(\kappa,F_s)}]$ and $\langle\psi, p\rangle^{(\kappa,F_s)} \in \mathtt{h}$ is not possible, immediately by rules $[\![\texttt{SndDone}]\!]$ and $[\![\texttt{IssueDone}]\!]$.

* Otherwise (i.e., T cannot handle failures in $\mathtt{h}$), it goes to Case (2).(I).(A) or Case (2).(I).(C): Case (2).(I).(A) is for no any done notifications sent from the coordinator in $\mathtt{h}$, while Case (2).(I).(C) is for some done notifications sent from the coordinator in $\mathtt{h}$.

For other endpoints the proof is similar as above.

Thus $G : (F_q, d) \vdash \Delta'$ is coherent.

(C) Case $\exists\langle\psi, p'\rangle^{\phi} \in \mathtt{h}$ for some p' and $\phi = (\kappa, F_s)$.

Let the try-handle of κ in G be $\mathsf{t}(G_0)\mathsf{h}(H)^{\kappa}$. If $\langle\psi, p_i\rangle^{\phi} \in \mathtt{h}$, then $p_i \in \mathit{roles}(\mathsf{t}(G_0)\mathsf{h}(H)^{\kappa})$.

W.o.l.g assume $\Delta = \Delta_0, \mathtt{s}[p] : T, \mathtt{s}[q] : T', \mathtt{s} : \mathtt{h}$ and assume $T = \mathcal{E}[q!\{l_i(S_i).T_i''\}_{i\in I}]$

$$\begin{array}{l} G : (F_q, d) \vdash \Delta_0, \mathtt{s}[p] : T, \mathtt{s}[q] : T', \mathtt{s} : \mathtt{h} \rightarrow_T \\ \quad G : (F_q, d) \vdash \Delta_0, \mathtt{s}[p] : \mathcal{E}[T_k''], \mathtt{s}[q] : T', \mathtt{s} : \mathtt{h} \cdot \langle p, q, l_k(S_k)\rangle \end{array}$$

Let $\mathit{Labels}(T, \phi)$ as a set of labels appearing in the try-handle of $\phi = (\kappa, F_s)$ in T. If $l_k \in \mathit{Labels}(T, \phi)$, by $[\![\texttt{IssueDone}]\!]$ it leads to contradiction because the actions in the try-handle of ϕ is not yet finished.

So we have $l_k \notin \mathit{Labels}(T, \phi)$, which implies that $l_k \notin \mathsf{t}(G_0)\mathsf{h}(H)^{\kappa}$ by Def. 11(3)(Well-formedness) and $[\![\texttt{IssueDone}]\!]$. The rest of the argumentation is similar to case (A) ($\mathtt{s}[q] : T' \lfloor p - (\mathtt{h})_{p\rightarrow q} = T''$ and $\nexists \mathsf{t}(T''')\mathsf{h}(\mathcal{H})^{(\kappa,F)}] \in T''$).

Thus $\Gamma \vdash \Delta'$ is coherent.

(D) Case $\exists\langle\!|\psi, \mathsf{crash}\ F|\!\rangle \in \mathtt{h}$ for some F.
W.o.l.g assume $\Delta = \Delta_0, \mathtt{s}[p] : T, \mathtt{s} : \mathtt{h}$.

By assumption, we have

$$\begin{array}{l} G : (F_q, d) \vdash \Delta_0, \mathtt{s}[p] : T, \mathtt{s} : \mathtt{h} \rightarrow_T \\ \quad G : (F_q, d) \vdash \Delta_0, \mathtt{s}[p] : T', \mathtt{s} : \mathtt{h} \cdot \langle p, q, l_k(S_k)\rangle \end{array}$$

and $\Delta' = \Delta_0, \mathtt{s}[p] : T, \mathtt{s} : \mathtt{h} \cdot \langle p, q, l_k(S_k)\rangle$.

Note that, in Δ, if the reduction comes from $[\![\texttt{Snd}]\!]$ (by the assumption of this case), then $p \notin F$ (The case that coordinator $G : (F_q, d)$ is affected is Case 2. (V), where $[\![\texttt{F}]\!]$ is applied).

* If $F = \{q\}$: In Δ by Def. 8 we have $\forall \mathsf{s}[q'] : T' \in \Delta$, $\mathsf{s}[p] : T \lfloor q' - (\mathtt{h})_{q' \to p} \bowtie \mathsf{s}[q'] : T' \lfloor p - (\mathtt{h})_{p \to q'}$ and $\mathsf{s}[p] \notin \Delta$.
* If $F \neq \{q\}$: Analogous to case (A-C).

Thus $G : (F_q, d) \vdash \Delta'$ is coherent.

(E) Case $\exists \langle p', \psi \rangle^{\phi} \in \mathtt{h}$ for some p' and ϕ.

By Def. 17, this kind of message type will not trigger any endpoints, the proof goes back to Cases 2.(I).(A,B,C,D).

(II) Case $[\![\texttt{Crash}]\!]$, there exists some $\mathsf{s}[p] : T \in \Delta$ that crashes, and failure notification $\langle\!\langle \psi, \mathsf{crash}\ \{p\} \rangle\!\rangle$ is sent to the coordinator. The proof is similar to Case 1.(II).

(III) Case $[\![\texttt{SndDone}]\!]$, there exists $\mathsf{s}[p] : T \in \Delta, T = \mathcal{E}[\mathsf{t}(\mathsf{end})\mathsf{h}(\mathcal{H})^{\phi}.T']$ and the reduction sends done notification $\langle p, \psi \rangle^{\phi}$ to the coordinator. The proof is similar to Case 1.(III).

Note that, this notification at this moment, which is abstracted as $\langle p, \psi \rangle^{\phi}$, is not able to affect any endpoints; moreover, the coordinator $G : (F_q, d)$ will eventually handle this notification.

(V) Case $[\![\texttt{Rcv}]\!]$. Every receiving case is trivial because the coherence has been ensured when every message/notification is sent out. We only prove the standard receiving case $[\![\texttt{Rcv}]\!]$, where a reduction comes from an endpoint receives a message in the queue. We do not need to consider here that the sender failed, since by $[\![\texttt{Crash}]\!]$ we know that there is no message to be consumed (i.e., $[\![\texttt{Rcv}]\!]$ is not applicable).

W.o.l.g. assume $\mathsf{s}[p] : \mathcal{E}[q?\{l_i(S_i).T_i\}_{i \in I}] \in \Delta$ and $\mathtt{h} = \langle q, p, l_k(S_k) \rangle \cdot \mathtt{h}'$ and $k \in I$ and $\Delta' = \Delta'_0, \mathsf{s} : \mathtt{h}'$ such that $\mathsf{s}[p] : \mathcal{E}[T_k] \in \Delta'$.

By Def. 8.(2) and Def. 17 (The Effect of $\mathtt{ht}$), for any $p' \neq p, \mathsf{s}[p'] : T' \in \Delta$, we have

$$
\begin{aligned}
&\mathsf{s}[p] : \mathcal{E}[q?\{l_i(S_i).T_i\}_{i \in I}] \lfloor p' - (\langle q, p, l_k(S_k) \rangle \cdot \mathtt{h}')_{p' \to p} \\
&= \mathsf{s}[p] : \mathcal{E}[T_k] \lfloor p' - (\mathtt{h}')_{p' \to p} \\
&\bowtie \mathsf{s}[p'] : T' \lfloor p - (\langle q, p, l_k(S_k) \rangle \cdot \mathtt{h}')_{p \to p'} \\
&= \mathsf{s}[p'] : T' \lfloor p - (\mathtt{h}')_{p \to p'}
\end{aligned}
$$

Thus in Δ', we immediately have

$$\mathsf{s}[p] : \mathcal{E}[T_k] \lfloor p' - (\mathtt{h}')_{p' \to p} \bowtie \mathsf{s}[p'] : T' \lfloor p - (\mathtt{h}')_{p \to p'}$$

Thus $\Gamma \vdash \Delta'$ is coherent.

(VI) Case $[\![\mathrm{F}]\!]$. If the reduction comes from $[\![\mathrm{F}]\!]$, in which the coordinator adds a notification $\langle\!\langle p, \mathsf{crash}\ F \rangle\!\rangle$ into the queue type to notify the endpoint acting as role p that F occurs.

This case, particularly, gives reasons for

(a) Def. 11 (Well-formedness).(5) ensures either F will be handled or there is no more interactions involving $\forall q \in F$.

(b) Def. 11 (Well-formedness).(1),(2) work for avoiding any confusion among partners for handling some F.

(c) With the above reasons, a coordinator needs to keep a well-formed G as a global guidance.

In the following we give our proofs for (a) and (b).

W.o.l.g assume

- $\Delta = \Delta_0, \mathsf{s} : \langle\!\langle \psi, \mathsf{crash}\ F \rangle\!\rangle \cdot \mathtt{h}''$,
- $\Delta' = \Delta_0, \mathsf{s} : \mathtt{h}'' \cdot \langle\!\langle \tilde{p}, \mathsf{crash}\ F \rangle\!\rangle$,
- $\mathtt{h}' = \langle\!\langle \tilde{p}, \mathsf{crash}\ F \rangle\!\rangle$,
- $\mathtt{h} = \mathtt{h}'' \cdot \mathtt{h}'$
- $\mathtt{h}''' = \langle\!\langle \psi, \mathsf{crash}\ F \rangle\!\rangle \cdot \mathtt{h}''$

$$G : (F_q, d) \vdash \Delta_0, \mathsf{s} : \langle\!\langle \psi, \mathsf{crash}\ F \rangle\!\rangle \cdot \mathtt{h}'' \to_T G : (F_q \cup F, d) \vdash \Delta_0, \mathsf{s} : \mathtt{h}'' \cdot \langle\!\langle \tilde{p}, \mathsf{crash}\ F \rangle\!\rangle$$

(a) If G is well-formed $\langle\!\langle p, \mathsf{crash}\ F \rangle\!\rangle \in \mathtt{h}$ means $\forall q \in F$ is non-robust, so a well-formed G shall take care of it (somewhere) by Def. 11.5.
For convenience we call all participant which occur in a try-handle partners for that try-handle.
Moreover, by Def. 4 (Projection), for every *alive* partners (ensured by Def. 11 (Well-formedness).(1)) for handlers we have (for some try-handle of ϕ)
  - The same set of handler signature and
  - The same try-handles of ϕ and
  - They are dual to each other in the handler body by given Def. 8.(b) holds.

They imply that, once a participant is triggered for handling F, all partners will soon or later be triggered for handling F too.

(b) By Def. 11 (Well-formedness).(1) and rules **(TryHdl)** and $[\![\texttt{TryHdl}]\!]$, a participant's try-block always gets triggered by the largest set of failures which currently it is able to handle.

Moreover, Def. 11 (Well-formedness).(2) ensures that the structure of G never results in some partners are handling F in an inner try block but some are handling F in the outer try-handle. The following is reasoning about this point.

By Def. 11 (Well-formedness), we know the possible shapes of G for any F are as follows:

(1) There is only one handler for F: $\exists \mathsf{t}(G_0)\mathsf{h}(H)^{\kappa} \in G, F \in dom(H)$ and $H(F) = G'$ and $\forall \mathsf{t}(G'_0)\mathsf{h}(H')^{\kappa'} \in G, \kappa' \neq \kappa.\ F \notin dom(H')$.

(2) There are two handlers for F such that

$$G = \mathcal{G}[\mathsf{t}(G_0)\mathsf{h}(F : G'_0, H_0)^{\kappa}].\mathcal{G}'[\mathsf{t}(G_1)\mathsf{h}(F : G'_1, H_1)^{\kappa'}]]$$

Note that, by Def. 11 (Well-formedness), the following shape

$$G = \mathcal{G}[\mathsf{t}(\mathcal{G}'[\mathsf{t}(G_0)\mathsf{h}(F : G'_0, H_0)^{\kappa}])\mathsf{h}(F : G'_1, H_1)^{\kappa'}]$$

is forbidden.

(3) There are more than two handlers for F. This case is by inductively applying Case (2).

In the following we prove (1) and (2) respectively.

For (1)

(i) If $\forall \mathtt{m} \in \mathtt{h}'', \mathtt{m} \neq \langle\!\langle p, \mathsf{crash}\ F' \rangle\!\rangle, \mathtt{m} \neq \langle \psi, p \rangle^{\phi}$ for some p, ϕ, F'. We can have $\mathtt{m} = \langle p', q', l(S) \rangle$ for some p', q', l, S,
Consider $G = \mathcal{G}[\mathsf{t}(...)\mathsf{h}(F : G', H)^{\kappa}]$. Let $G'' = \mathsf{t}(...)\mathsf{h}(F : G', H)^{\kappa}$.

In Δ, for any $p, q \in roles(G'')$ and $\mathsf{s}[p], \mathsf{s}[q] \in \Delta$, let $\Delta(\mathsf{s}[p]) = T$ and $\Delta(\mathsf{s}[q]) = T'$.
W.o.l.g assume $\mathsf{s}[p] : T \lfloor q - (\mathtt{h}''')_{q \to p} = \mathsf{s}[p] : (\mathcal{E}[G'' \restriction p.T'']) \lfloor q$ for all p and q.
By Def. 8 (Coherence).(2), we have

$$\begin{aligned} \mathsf{s}[p] : T \lfloor q - (\mathtt{h}''')_{q \to p} &= \mathsf{s}[p] : (\mathcal{E}[G'' \restriction p.T'']) \lfloor q \\ &\bowtie \mathsf{s}[q] : (\mathcal{E}[G'' \restriction q.T''']) \lfloor p \\ &= \mathsf{s}[q] : T' \lfloor p - (\mathtt{h}''')_{p \to q} \end{aligned}$$

In Δ', for those $p, q \in \mathit{roles}(G'')$ and and $\mathtt{s}[p], \mathtt{s}[q] \in \Delta$, by Def. 20 and Def. 7 (Duality), we have

$$\mathtt{s}[p] : (\mathit{HdlLogic}(G''\restriction p, F).T'') \lfloor q \bowtie \mathtt{s}[q] : (\mathit{HdlLogic}(G''\restriction q, F).T''') \lfloor p$$

In Δ, for any $p, q \notin \mathit{roles}(G'')$, they will not be affected by F, so they remain being dual to each other and so as their endpoint types in Δ'.

In Δ, for any $p \in \mathit{roles}(G''), q \notin \mathit{roles}(G'')$, they do not interact with each other when p is affected by F and $k \notin T \lfloor q$,$\kappa \notin T^{IV}$ and $\kappa \notin T^{V}$

$$\begin{aligned}\mathtt{s}[p] : T \lfloor q - (\mathtt{h}''')_{q \to p} &= \mathtt{s}[p] : T^{IV} \\ &\bowtie \mathtt{s}[q] : T^{V} \\ &= \mathtt{s}[q] : T' \lfloor p - (\mathtt{h}''')_{p \to q}\end{aligned}$$

In Δ', their endpoint types hold this relation because they still do not interact to each other in κ.

(ii) If $\mathit{Fset}(\mathtt{h}'', p) = F' \neq \emptyset$ for some p and $\forall \mathtt{m} \in \mathtt{h}'', \mathtt{m} \neq \langle \psi, p' \rangle^{\phi}$ for some p' and some ϕ.

Then we consider G with the following shapes:

- $G = \mathcal{G}[\mathtt{t}(\mathcal{G}'[\mathtt{t}(G_0)\mathtt{h}(F : G', H_0)^{\kappa}])\mathtt{h}(F' : G'', H_1)^{\kappa'}]$.

  In Δ, for any $p, q \in \mathit{roles}(\mathtt{t}(\mathcal{G}'[\mathtt{t}(G_0)\mathtt{h}(F : G', H_0)^{\kappa}])\mathtt{h}(F' : G'', H_1)^{\kappa'})$ and $\mathtt{s}[p], \mathtt{s}[q] \in \Delta$, let $\mathtt{s}[p] : \mathcal{E}[T_0]$ and $\mathtt{s}[q] : \mathcal{E}'[T_0']$ w.o.l.g such that T_0 and T_0' both have the try-handle of (κ', F_s) for some F_s at T_0 and T_0'.

  When $\nexists F'' \supset F \cup F'$ such that $F'' \in G$, then by Def. 8 (Coherence).(2) and Def. 20, we have in Δ

  $$\begin{aligned}&\mathtt{s}[p] : \mathcal{E}[T_0] \lfloor q - (\mathtt{h}''')_{q \to p} \\ =\ &\mathtt{s}[p] : \mathcal{E}[\mathit{HdlLogic}(T_0, F')] \lfloor q - (\mathtt{h}''')_{q \to p} \\ \bowtie\ &\mathtt{s}[q] : \mathcal{E}'[T_0'] \lfloor p - (\mathtt{h}''')_{p \to q} \\ =\ &\mathtt{s}[q] : \mathcal{E}'[\mathit{HdlLogic}(T_0', F')] \lfloor p - (\mathtt{h}''')_{p \to q}\end{aligned}$$

  This case shows that, if there exists failure notifications to trigger outer and inner handler, the outer one takes over.

  In Δ' their endpoint types hold

  $$\begin{aligned}&\mathtt{s}[p] : \mathcal{E}[\mathit{HdlLogic}(T_0, F')] \lfloor q - (\mathtt{h}''' \cdot \mathtt{h}')_{q \to p} \\ &\quad \bowtie \mathtt{s}[q] : \mathcal{E}'[\mathit{HdlLogic}(T_0', F')] \lfloor p - (\mathtt{h}''' \cdot \mathtt{h}')_{p \to q}\end{aligned}$$

When $\exists F'' \supset F \cup F'$ such that $F'' \in G$, by Def. 11. (2), F'' must be in a try-handle of $outer_G(\kappa')$ (i.e., F'' is in an outer try-handle or the same try-handle of F'). The proof is similar to the previous case.

- $G = \mathcal{G}[\mathtt{t}(\mathcal{G}'[\mathtt{t}(G_0)\mathtt{h}(F' : G'', H_0)^{\kappa}])\mathtt{h}(F : G', H_1)^{\kappa'}]$.

  In Δ, for any $p, q \in roles(\mathtt{t}(\mathcal{G}'[\mathtt{t}(G_0)\mathtt{h}(F' : G'', H_0)^{\kappa}])\mathtt{h}(F : G', H_1)^{\kappa'})$ and $\mathtt{s}[p], \mathtt{s}[q] \in \Delta$, let $\mathtt{s}[p] : \mathcal{E}[T_0]$ and $\mathtt{s}[q] : \mathcal{E}'[T_0']$ such that T_0 and T_0' both have the try-handle of (κ', F_s) for some F_s.

  By Def. 17 (The Effect of $\mathtt{ht}$) and Def. 7 (Duality),

  $$\begin{array}{l} \mathtt{s}[p] : T_0 \upharpoonright q - (\mathtt{h} \cdot \mathtt{h}')_{q \to p} \\ = \mathtt{s}[p] : \mathcal{E}[\mathit{HdlLogic}(T_0, F)] \upharpoonright q - (\mathtt{h}'' \cdot \mathtt{h}')_{q \to p} \\ \bowtie \mathtt{s}[q] : T_0' \upharpoonright p - (\mathtt{h} \cdot \mathtt{h}')_{p \to q} \\ = \mathtt{s}[q] : \mathcal{E}'[\mathit{HdlLogic}(T_0', F)] \upharpoonright p - (\mathtt{h}'' \cdot \mathtt{h}')_{p \to q} \end{array}$$

  Again, in Δ' their endpoint types hold

  $$\begin{array}{l} \mathtt{s}[p] : \mathcal{E}[\mathit{HdlLogic}(T_0, F)] \upharpoonright q - (\mathtt{h}'' \cdot \mathtt{h}')_{q \to p} \\ \quad \bowtie \mathtt{s}[q] : \mathcal{E}'[\mathit{HdlLogic}(T_0', F)] \upharpoonright p - (\mathtt{h}'' \cdot \mathtt{h}')_{p \to q} \end{array}$$

(iii) If $\exists \mathtt{m} \in \mathtt{h}'', \mathtt{m} = \langle \psi, p \rangle^{\phi}$ for some p and some $\phi = (\kappa, F_s)$.

Then we consider G having the following shapes:

- $G = \mathcal{G}[\mathtt{t}(\mathcal{G}'[\mathtt{t}(G_0)\mathtt{h}(H_0)^{\kappa}])\mathtt{h}(F : G', H_1)^{\kappa'}]$ and $F_s \in dom(H_0)$

  W.o.l.g assume
  $\mathtt{s}[p] : \mathcal{E}[\mathtt{t}(\mathcal{E}'[\mathtt{t}(T)\mathtt{h}(\mathcal{H})^{(\kappa, F_s)}.T''']])\mathtt{h}(\mathcal{H})^{(\kappa, F'')}.T'']$ for all $\mathtt{s}[p] \in \Delta$ and $p \in \mathtt{t}(G_0)\mathtt{h}(H_0)^{\kappa}$

  $$\begin{array}{l} \mathtt{s}[p] : \mathcal{E}[\mathtt{t}(\mathcal{E}'[\mathtt{t}(T)\mathtt{h}(\mathcal{H})^{(\kappa, F_s)}.T''']])\mathtt{h}(\mathcal{H})^{(\kappa, F'')}.T''] \upharpoonright q - (\mathtt{h}''')_{q \to p} \\ = \mathtt{s}[p] : \mathcal{E}[\mathtt{t}(\mathcal{E}'[T''']])\mathtt{h}(\mathcal{H})^{(\kappa, F'')}.T''] \upharpoonright q - (\mathtt{h}''' \setminus \langle \psi, p \rangle^{\phi})_{q \to p} \end{array}$$

  The rest of the proofs follows the reasoning in (i) and (ii).
- $G = \mathcal{G}[\mathtt{t}(G_0)\mathtt{h}(F : G', H)^{\kappa}]$ and $F_s \in dom(H)$.

  In this case $Fset(\mathtt{h}, p)$ (Def. 2 second condition) will not collect $\langle\!\langle p, \mathsf{crash}\ F \rangle\!\rangle$ to trigger p's handler in the try-handle of κ. Done notification $\langle \psi, p \rangle^{\phi}$, $\phi = (\kappa, F_s)$, will finish the try-handle of κ.

For (b).(2), there are two handlers for F, say

$$G = \mathcal{G}[\mathtt{t}(G_0)\mathtt{h}(F:G', H_0)^{\kappa}].\mathcal{G}'[\mathtt{t}(G_1)\mathtt{h}(F:G'', H_1)^{\kappa'}]]$$

If $\mathit{roles}(\mathtt{t}(G_0)\mathtt{h}(F:G', H_0)^{\kappa}) \oplus \mathit{roles}(\mathtt{t}(G_1)\mathtt{h}(F:G'', H_1)^{\kappa'}) = \emptyset$[8] the proof is the same as the part in (b).(1).

W.o.l.g assume $p \in \mathit{roles}(\mathtt{t}(G_0)\mathtt{h}(F:G', H_0)^{\kappa})$, $q \in \mathit{roles}(\mathtt{t}(G_1)\mathtt{h}(F:G'', H_1)^{\kappa'})$, $q \notin \kappa$, and $\mathtt{s}[p], \mathtt{s}[q] \in \Delta$, $\Delta(\mathtt{s}[p]) = T$, $\Delta(\mathtt{s}[q]) = T'$ and $T = \mathcal{E}[\mathtt{t}(T_{F_s})\mathtt{h}(\mathcal{H})^{(\kappa,F_s)}.T'']$.

(a) $\Delta(\mathtt{s}[q]) = T' = \mathcal{E}'[\mathtt{t}(T_{F'_s})\mathtt{h}(\mathcal{H}')^{(\kappa',F'_s)}.T''']$
Assume $\exists F''_s \supset \{F'_s, F''_s\} \in \mathit{dom}(\mathcal{H}') \wedge F''_s \subseteq \mathit{Fset}(\mathtt{h}, q) \wedge F \in F''_s$ otherwise $\mathtt{h}'$ has no affect on $\Delta(\mathtt{s}[q])$.
In Δ by Def. 8

$$\begin{aligned} &\mathtt{s}[p] : \mathcal{E}[\mathtt{t}(T_{F_s})\mathtt{h}(\mathcal{H})^{(\kappa,F_s)}.T''] \lfloor q - (\mathtt{h}''')_{q\to p} \\ =\ &\mathtt{s}[p] : \mathcal{E}''[T'' \lfloor q] - (\mathtt{h}^*)_{q\to p} \\ \bowtie\ &\mathtt{s}[q] : \mathcal{E}'[\mathtt{t}(T_{F'_s})\mathtt{h}(\mathcal{H})^{(\kappa',F'_s)}.T'''] \lfloor p - (\mathtt{h}''')_{p\to q} \end{aligned}$$

If $p \notin \kappa'$ then $\mathcal{E}'[\mathtt{t}(T_{F'_s})\mathtt{h}(\mathcal{H})^{(\kappa',F'_s)}.T'''] \lfloor p - (\mathtt{h}''')_{p\to q} = \mathcal{E}'''[T''' \lfloor p] - (\mathtt{h}''')_{p\to q}$ and $\mathcal{E}'''[T''' \lfloor p] - (\mathtt{h}''')_{p\to q} = \mathcal{E}'''[T''' \lfloor p] - (\mathtt{h})_{p\to q}$

Otherwise ($p \in \kappa'$): Then
$\mathcal{E}''[T'' \lfloor q] - (\mathtt{h}^*)_{q\to p} = \mathcal{E}''[\mathtt{t}(T_{F'''_s})\mathtt{h}(\mathcal{H})^{(\kappa',F'''_s)}.T^{IV}] - (\mathtt{h}^*)_{q\to p}$, by Def. 7

$$\begin{aligned} &\mathcal{E}''[\mathit{HdlLogic}(\mathtt{t}(T_{F'''_s})\mathtt{h}(\mathcal{H})^{(\kappa',F'''_s)}, F''_s).T^{IV}] - (\mathtt{h}^*)_{q\to p} \\ &\bowtie \\ &\mathcal{E}'[\mathit{HdlLogic}(\mathtt{t}(T_{F'_s})\mathtt{h}(\mathcal{H})^{(\kappa',F'_s)}, F''_s).T'''] - (\mathtt{h}''')_{p\to q} \end{aligned}$$

(b) $\Delta(\mathtt{s}[q]) = T'$ and $\kappa' \in T'$ and $\nexists \mathcal{E}'.T' = \mathcal{E}'[\mathtt{t}(T_{F'_s})\mathtt{h}(\mathcal{H}')^{(\kappa',F'_s)}.T''']$
$T' \lfloor p - (\mathtt{h}''')_{p\to q} = T' \lfloor p - (\mathtt{h})_{p\to q}$ ($\langle\!\langle \tilde{p}, \mathsf{crash}\ F \rangle\!\rangle$ has no immediately effect on $\Delta(\mathtt{s}[q])$)

(c) $\Delta(\mathtt{s}[q]) = T'$ and $\kappa' \notin T'$
$T' \lfloor p - (\mathtt{h}''')_{p\to q} = T' \lfloor p - (\mathtt{h})_{p\to q}$ ($\langle\!\langle \tilde{p}, \mathsf{crash}\ F \rangle\!\rangle$ has no effect on $\Delta(\mathtt{s}[q])$ since the try-handle has already finished)

Thus overall, $\Gamma', G : (F_q{}', d) \vdash \Delta'$ is coherent.

(C) Case $[\![\texttt{CollectDone}]\!]$. This case is trivial because only the coordinator collects done notifications sent from participants; no endpoints will be affected.

(D) Case $[\![\texttt{IssueDone}]\!]$. By $[\![\texttt{IssueDone}]\!]$,

$$G : (F_q, d) \vdash \Delta = \Delta_0, \mathtt{s} : \mathtt{h} \to_T G : (F_q, d') \vdash \Delta' = \Delta_0, \mathtt{s} : \mathtt{h}'' \cdot \mathtt{h}'$$

[8] We use $\oplus$ for the symmetric difference

such that $\mathtt{h}''$ contains no $\langle p'', \psi\rangle^{(\kappa,F_s)}$ for some p'' and $\forall \mathtt{m} \in \mathtt{h}', \mathtt{m} = \langle \psi, p'\rangle^{(\kappa,F_s)}$ and, let $\mathtt{t}(...)\mathsf{h}(H)^{\kappa} \in G$, we have $p' \in \mathit{roles}(H(F_s))$.

(a) If there exists $\langle\!|p, \mathsf{crash}\ F|\!\rangle \in \mathtt{h}'' \cdot \mathtt{h}'$ such that $p \in \mathit{roles}(H(F_s))$, then by Proposition 1, endpoint $\mathsf{s}[p] : T$ will not be affected by failures in $\mathtt{h}'' \cdot \mathtt{h}'$. Therefore, $\langle \psi, p'\rangle^{(\kappa,F_s)}$ is the one that will affect $\mathsf{s}[p] : T$ and finishes T's try-handle of (κ, F_s).

(b) If there exists $\langle q, p, l(S)\rangle \in \mathtt{h}$, then by the structure of try-handles, this message type have the following relation with the try-handle of $\phi = (\kappa, F_s)$:

- if it comes from a try-handle of $\phi' = (\kappa', {F_s}')$ such that $\kappa \in \mathit{outer}_G(\kappa')$ then it is already orphan; by Def. 11 (Well-formedness).(3), $[\![\mathtt{Cln}]\!]$ will be applied.
- if it comes from a try-handle of $\phi' = (\kappa', {F_s}')$ such that $\kappa' \in \mathit{outer}_G(\kappa)$, then by rule $[\![\mathtt{RcvDone}]\!]$, $\langle \psi, p'\rangle^{(\kappa,F_s)}$ will finish T's try-handle of F_s.
- if it comes from a try-handle of $\phi' = (\kappa', {F_s}')$ such that $\kappa' \notin \mathit{outer}_G(\kappa)$ and , $\kappa \notin \mathit{outer}_G(\kappa')$ then by rule $[\![\mathtt{RcvDone}]\!]$, $\langle \psi, p'\rangle^{(\kappa,F_s)}$ will finish T's try-handle of F_s.

Then the rest of proof is similar to Case 2.(V).(B).

(E) Case $[\![\mathtt{TryHdl}]\!]$. In Case $[\![\mathtt{F}]\!]$, we prove that the failure notifications sent out by the coordinator with a well-formed G can trigger endpoints who are able to handle failures; moreover, Def. 8(Coherence).(2) ensures that the handler for any F are coherent in the sense that every participant has a dual interacting party to handle failures.

After triggering handler(s), the prove is as same as the one in Case 1.(I).(B).

(F) Case $[\![\mathtt{RcvDone}]\!]$. The proof is similar to Case 1.(I).(C).

Subject congruence and subject reduction. We first state the inversion lemma and several auxiliary lemmas which will be used in our proofs.

Lemma 5 (Inversion Lemma).

1. If $\Gamma \vdash \mathsf{c} : 0 \rhd \Delta$, then Δ is end-only.
2. If $\Gamma \vdash \mathsf{c} : \mathsf{if}\ e\ \eta_1\ \mathsf{else}\ \eta_2 \rhd \Delta$, then $\Gamma \vdash e : \mathsf{bool}$ and $\forall i \in \{1, 2\}$ we have $\Gamma \vdash \mathsf{c} : \eta_i \rhd \Delta$.
3. If $\Gamma \vdash a[p](y).P \rhd \Delta$, then $\Delta = \emptyset$ and $\Gamma \vdash a : \langle G\rangle$ and $\Gamma \vdash P \rhd \{\mathsf{c} : G\lceil p\}$.

4. If $\Gamma \vdash \ \mathsf{c} : p!\ l(e).\eta \rhd\ \Delta$, then $\Delta = \{\mathsf{c} : T\}$ $l = l_k$ and $k \in I$ and $T = p!\ \{l_i(S_i).T_i'\}_{i\in I}$ and $\Gamma \vdash\ e : S_k$ and $\Gamma\ \vdash\ \mathsf{c} : \eta_k \rhd\ \{\mathsf{c} : T_k'\}$.
5. If $\Gamma \vdash\ \mathsf{c} : p?\ \{l_i(e_i).\eta_i\}_{i\in I} \rhd\ \Delta$, then $\Delta = \{\mathsf{c} : T\}$ and $T = p?\ \{l_i(S_i).T_i\}_{i\in I}$ and $\forall i \in I.\ \Gamma, x_i : S_i\ \vdash\ \mathsf{c} : \eta_i \rhd\ \{\mathsf{c} : T_i\}$.
6. If $\Gamma\ \vdash\ \mathsf{c} : \underline{0}.\eta \rhd\ \Delta$, then $\Delta = \{\mathsf{c} : T\}$ and $T = \underline{\mathsf{end}}.\mathsf{end}$ and $\Gamma\ \vdash\ \mathsf{c} : \eta \rhd\ \{\mathsf{c} : \mathsf{end}\}$.
7. If $\Gamma\ \vdash\ \mathsf{c} : X\langle e\rangle \rhd\ \Delta$, then $\Delta = \{\mathsf{c} : T\}$ and $\Gamma = \Gamma', X : S\ T$ and $\Gamma' \vdash\ e : S$.
8. If $\Gamma\ \vdash\ \mathsf{c} : \mathsf{def}\ D\ \mathsf{in}\ \eta_2 \rhd\ \Delta$ and $X(x) = \eta_1 \in D$, then $\Delta = \{\mathsf{c} : T\}$ and $\Gamma, X : S\ \mu t.T'\ \vdash\ \mathsf{c} : \eta_2 \rhd\ \{\mathsf{c} : T\}$ and $\Gamma, X : S\ t, x : S\ \vdash\ \mathsf{c} : \eta_1 \rhd\ \{\mathsf{c} : T'\}$.
9. If $\Gamma\ \vdash\ \mathsf{c} : \mathsf{t}(\eta)\mathsf{h}(\mathsf{H})^{\phi}.\eta' \rhd\ \Delta$, then $\Delta = \{\mathsf{c} : T\}$ and $T = \mathsf{t}(T')\mathsf{h}(\mathcal{H})^{\phi}.T''$ and $\Gamma\ \vdash\ \mathsf{c} : \eta \rhd\ \{\mathsf{c} : T'\}$ and $\Gamma\ \vdash\ \mathsf{c} : \eta' \rhd\ \{\mathsf{c} : T''\}$ and $dom(\mathsf{H}) = dom(\mathcal{H})$ and $\forall F \in dom(\mathsf{H})$ we have $\Gamma\ \vdash\ \mathsf{c} : \mathsf{H}(F) \rhd\ \{\mathsf{c} : \mathcal{H}(F)\}$.
10. If $\Gamma \vdash\ \mathsf{s} : \emptyset \rhd\ \Delta$, then $\Delta = \{\mathsf{s} : \emptyset\}$.
11. If $\Gamma \vdash\ \mathsf{s} : h \cdot \langle p, q, l(e)\rangle \rhd\ \Delta$, then $\Delta = \{\mathsf{s} : \mathtt{h} \cdot \langle p, q, l(S)\rangle\}$ and $\Gamma \vdash\ \mathsf{s} : h \rhd\ \{\mathsf{s} : \mathtt{h}\}$ and $\Gamma \vdash\ e : S$.
12. If $\Gamma \vdash\ \mathsf{s} : h \cdot \langle p_1, p_2\rangle^{\phi} \rhd\ \Delta$, then $(p_1, p_2) \in \{(p, \psi), (\psi, p)\}$ for some p and $\Delta = \{\mathsf{s} : \mathtt{h} \cdot \langle p_1, p_2\rangle^{\phi}\}$ and $\Gamma \vdash\ \mathsf{s} : h \rhd\ \{\mathsf{s} : \mathtt{h}\}$.
13. If $\Gamma \vdash\ \mathsf{s} : h \cdot (\!|q, \mathsf{crash}\ F|\!) \rhd\ \Delta$, then $\Delta = \{\mathsf{s} : \mathtt{h} \cdot (\!|q, \mathsf{crash}\ F|\!)\}$ and $q \in \{p, \psi\}$ and $\Gamma \vdash\ \mathsf{s} : h \rhd\ \{\mathsf{s} : \mathtt{h}\}$.
14. If $\Gamma \vdash\ N_1 | N_2 \rhd\ \Delta$, then $\Gamma \vdash\ N_1 \rhd\ \Delta_1$ and $\Gamma \vdash\ N_2 \rhd\ \Delta_2$ and $\Delta = \Delta_1, \Delta_2$ such that $dom(\Delta_1) \cap dom(\Delta_2) = \emptyset$.
15. If $\Gamma \vdash\ (\nu \mathsf{s})\mathcal{S} \rhd\ \Delta$, then $\Gamma \vdash\ \mathcal{S} \rhd\ \Delta'$ and and $\Delta = \Delta' \setminus \Delta'_{\mathsf{s}}$ and $\Gamma \vdash\ \Delta'_{\mathsf{s}}$ is coherent.
16. If $\Gamma \vdash\ \Psi \blacklozenge N \rhd\ \Delta$, then $\Gamma = \Gamma'$, Ψ and $\Gamma' \vdash\ N \rhd\ \Delta$.

Proof. By induction on derivations.

Lemma 6 (Substitution Lemma).

1. If $\Gamma, x : S \vdash\ \mathcal{S} \rhd\ \Delta$ and $\Gamma \vdash\ v : S$, then $\Gamma \vdash\ \mathcal{S}\{v/x\} \rhd\ \Delta$.
2. If $\Gamma \vdash\ \mathcal{S} \rhd\ \Delta, y : T$, then $\Gamma \vdash\ \mathcal{S}\{\mathsf{s}[p]/y\} \rhd\ \Delta, \mathsf{s}[p] : T$.

Proof.

1 The proof is by induction on derivation of $\Gamma, x : S \vdash\ \rhd \Delta$.

2 The proof is by induction on derivation of $\Gamma \vdash\ \rhd \Delta, y : T$.

Lemma 7 (Types of Queues).

1. If $\Gamma \vdash\ \mathsf{s} : \langle p, q, l(v)\rangle \cdot h \rhd\ \Delta$, then $\Delta = \{\mathsf{s} : \langle p, q, l(S)\rangle \cdot \mathtt{h}\}$ and $\Gamma \vdash\ \mathsf{s} : h \rhd\ \{\mathsf{s} : \mathtt{h}\}$.

2. If $\Gamma\ \vdash\ \mathsf{s} : (\!|role, \mathsf{crash}\ F|\!) \cdot h \rhd\ \Delta$, then $role \in \{p, \psi\}$ for some p and $\Delta = \{\mathsf{s} : (\!|role, \mathsf{crash}\ F|\!) \cdot \mathtt{h}\}$ and $\Gamma \vdash\ \mathsf{s} : h \rhd\ \{\mathsf{s} : \mathtt{h}\}$.

3. If $\Gamma \vdash\ \mathsf{s} : \langle role, role'\rangle^{\phi} \cdot h \rhd\ \Delta$, then $(role, role') \in \{(p, \psi), (\psi, p)\}$ for some p and $\Delta = \{\mathsf{s} : \langle role, role'\rangle^{\phi} \cdot \mathtt{h}\}$ and $\Gamma \vdash\ \mathsf{s} : h \rhd\ \{\mathsf{s} : \mathtt{h}\}$.

Proof. For all cases, the first step follows from Lemma 5.10. The induction step follows from Lemma 5.11.

Lemma 8. If $\Gamma \vdash \mathsf{c} : E[\eta] \rhd \{\mathsf{c} : T\}$ and $\Gamma \vdash \mathsf{c} : \eta \rhd \{\mathsf{c} : T'\}$, then $T = \mathcal{E}[T']$ for some $\mathcal{E}$.

Proof. The proof is by structural induction on contexts E.

- $E = [\,]$, then immediately $\mathcal{E} = [\,]$.
- $E = \mathsf{def}\ D\ \mathsf{in}\ E'$ and $X(x) = \eta' \in D$ and $\Gamma \vdash \mathsf{c} : \eta \rhd \{\mathsf{c} : T'\}$. By applying Lemma 5.8 to
$$\Gamma \vdash \mathsf{c} : \mathsf{def}\ D\ \mathsf{in}\ E'[\eta] \rhd \{\mathsf{c} : T\}$$
we have
$\Gamma, X : S\ \mu t.T'' \vdash \mathsf{c} : E'[\eta] \rhd \{\mathsf{c} : T\}$ and $\Gamma, X : S\ t, x : S \vdash \mathsf{c} : \eta' \rhd \{\mathsf{c} : T''\}$
Since $\Gamma, X : S\ \mu t.T'' \vdash \mathsf{c} : \eta \rhd \{\mathsf{c} : T'\}$, by induction, we have $T = \mathcal{E}[T']$.
- $E = \mathsf{t}(E')\mathsf{h}(\mathsf{H})^{\phi}.\eta'$ and $\Gamma \vdash \mathsf{c} : \eta \rhd \{\mathsf{c} : T'\}$. By applying Lemma 5.9 to
$$\Gamma \vdash \mathsf{c} : \mathsf{t}(E'[\eta])\mathsf{h}(\mathsf{H})^{\phi}.\eta' \rhd \{\mathsf{c} : T\}$$
we have
$$T = \mathsf{t}(T'')\mathsf{h}(\mathcal{H})^{\phi}.T''' \text{ and } \Gamma \vdash \mathsf{c} : E'[\eta] \rhd \{\mathsf{c} : T''\}$$
By induction, we have $T'' = \mathcal{E}'[T']$ for some $\mathcal{E}'$. So we have
$$T = \mathsf{t}(\mathcal{E}'[T'])\mathsf{h}(\mathcal{H})^{\phi}.T''' = \mathsf{t}(T'')\mathsf{h}(\mathcal{H})^{\phi}.T'''$$
such that $\mathcal{E} = \mathsf{t}(\mathcal{E}')\mathsf{h}(\mathcal{H})^{\phi}.T'''$.

Theorem 1 (Subject Congruence)

(1) $\Gamma \vdash N \rhd \Delta$ and $N \equiv N'$ imply $\Gamma \vdash N' \rhd \Delta$
(2) $\Gamma \vdash \mathcal{S} \rhd \Delta$ and $\mathcal{S} \equiv \mathcal{S}'$ imply $\Gamma \vdash \mathcal{S} \rhd \Delta$.

Proof. Both proofs are by induction on $\equiv$. We only list the interesting cases.

- $\dfrac{h \equiv h'}{\mathsf{s} : h \equiv \mathsf{s} : h'}$.
Given $h \equiv h'$, we firstly prove that $\Gamma \vdash \mathsf{s} : h \rhd \Delta$ implies $\Gamma \vdash \mathsf{s} : h' \rhd \Delta$.
Then by $\dfrac{N \equiv N'}{\Psi\blacklozenge N \equiv \Psi\blacklozenge N'}$ (which is proved in the next case) and $\lfloor$`T-sys`$\rfloor$, we have $\Gamma, G : (F_q, d) \vdash \Psi\blacklozenge\mathsf{s} : h \rhd \Delta$ then $\Gamma', G : (F_q, d) \vdash \Psi\blacklozenge\mathsf{s} : h' \rhd \Delta$ and $G : (F_q, d) \in \Psi$.

Let $h \equiv h'$ and $\Gamma \vdash \ \mathtt{s} : h \rhd \ \Delta$. The equivalence $h \equiv h'$ should come from one of the following cases: (1) $h \equiv h \cdot \emptyset \equiv h'$ or (2) $h \equiv \emptyset \cdot h \equiv h'$ or (3) $h \equiv h_1 \cdot (h_2 \cdot h_3) \equiv (h_1 \cdot h_2) \cdot h_3 \equiv h'$ or (4) $h \equiv h_1 \cdot m \cdot m' \cdot h_2 \equiv h_1 \cdot m' \cdot m \cdot h_2 \equiv h'$ by given $m \cdot m' \curvearrowright m' \cdot m$.

If the structural congruence comes from cases (1), (2), or (3), since the messages in h and h' are the same and they are in the same order, then by Lemma 7, $\Gamma \vdash \ \mathtt{s} : h' \rhd \ \Delta$.

If the structural congruence comes from case (4), given $m \cdot m' \curvearrowright m' \cdot m$, then by Lemma 5.11, Lemma 5.12, Lemma 5.13 and Lemma 7,

$$\Gamma \vdash \ \mathtt{s} : h_1 \cdot m \cdot m' \cdot h_2 \rhd \ \{\mathtt{s} : \mathtt{h}_1 \cdot \mathtt{m} \cdot \mathtt{m}' \cdot \mathtt{h}_2\}$$

imply $\Gamma \vdash \ \mathtt{s} : h_1 \rhd \{\mathtt{s} : \mathtt{h}_1\}$ and $\Gamma \vdash \ \mathtt{s} : m \rhd \{\mathtt{s} : \mathtt{m}\}$ and $\Gamma \vdash \ \mathtt{s} : m' \rhd \{\mathtt{s} : \mathtt{m}'\}$ and $\Gamma \vdash \ \mathtt{s} : h_2 \rhd \ \{\mathtt{s} : \mathtt{h}_2\}$.

By $\lfloor$T-m$\rfloor$ or $\lfloor$T-D$\rfloor$ or $\lfloor$T-$F\rfloor$, we derive

$$\Gamma \vdash \ \mathtt{s} : h_1 \cdot m' \cdot m \cdot h_2 \rhd \ \{\mathtt{s} : \mathtt{h}_1 \cdot \mathtt{m}' \cdot \mathtt{m} \cdot \mathtt{h}_2\}.$$

By Definition 19 (Permutable Message Types),

$$\{\mathtt{s} : \mathtt{h}_1 \cdot \mathtt{m}' \cdot \mathtt{m} \cdot \mathtt{h}_2\} \equiv \{\mathtt{s} : \mathtt{h}_1 \cdot \mathtt{m} \cdot \mathtt{m}' \cdot \mathtt{h}_2\} = \Delta$$

Thus we conclude this case.

– $\dfrac{N \equiv N'}{\Psi \blacklozenge N \equiv \Psi \blacklozenge N'}$.

By Lemma 5.16, then $G : (F_q, d) \in \psi$ and $\Gamma = \Gamma', G : (F_q, d)$ and $\Gamma' \vdash N \rhd \ \Delta$.

By induction on $\equiv$, we have $\Gamma \vdash \ N' \rhd \ \Delta$. By $\lfloor$T-sys$\rfloor$, we have

$$\Gamma \vdash \ \ \Psi \blacklozenge N' \rhd \ \Delta$$

We conclude this case.

Before proving Theorem 2 (Subject Reduction), we give the following lemma for a special case that when $N \to \mathcal{S}$:

Lemma 9. $\Gamma \vdash \ N \rhd \ \Delta$ with $\Gamma \vdash \ \Delta$ coherent and $N \to \mathcal{S}$ imply that $\Gamma, \psi \vdash \mathcal{S} \rhd \ \Delta$.

Proof. The only case is **(Link)**. Assume $\Gamma \vdash \ a[p_1](y_1).P_1 \mid ... \mid a[p_n](y_n).P_n \rhd \ \Delta$ and $a : G$ and $roles(G) = \{p_1, ..., p_n\}$.

By applying Lemma 5.(14) n times, we have

$$\Delta = \{\Delta_i\}_{i \in \{1..n\}} \text{ and } \forall i \in \{1..n\}.\ \Gamma \vdash \ a[p_i](y_i).P_i \rhd \ \Delta_i \tag{3}$$

By applying Lemma 5.(3) on Eq. (3), we have

$$\begin{array}{l} \Delta_i = \emptyset \Rightarrow \Delta = \emptyset \\ \forall i \in \{1..n\}.\ \Gamma \vdash\ P_i \rhd\ \{y_i : G\upharpoonright p_i\} \text{ and } \Gamma \vdash\ a : \langle G \rangle \end{array} \tag{4}$$

By applying $\lfloor$T-pa$\rfloor$ n times on Eq. (4), we derive

$$\Gamma \vdash\ P_1 \mid ... \mid P_n \rhd\ \{y_1 : G\upharpoonright p_1, ..., y_n : G\upharpoonright p_n\} \tag{5}$$

By applying Lemma 6.(2) to Eq. (5), we derive

$$\Gamma \vdash\ P_1\{\mathsf{s}[p_1]/y_1\} \mid ... \mid P_n\{\mathsf{s}[p_n]/y_n\} \rhd\ \{\mathsf{s}[p_1] : G\upharpoonright p_1, ..., \mathsf{s}[p_n] : G\upharpoonright p_n\} \tag{6}$$

By applying $\lfloor$T-pa$\rfloor$ n and $\lfloor$T-$\emptyset\rfloor$ to Eq. (6), we derive

$$\begin{array}{r} \Gamma \vdash\ P_1\{\mathsf{s}[p_1]/y_1\} \mid ... \mid P_n\{\mathsf{s}[p_n]/y_n\} \mid \mathsf{s} : \emptyset \rhd \\ \{\mathsf{s}[p_1] : G\upharpoonright p_1, ..., \mathsf{s}[p_n] : G\upharpoonright p_n, \{\mathsf{s} : \emptyset\}\} \end{array} \tag{7}$$

By applying $\lfloor$T-sys$\rfloor$ to Eq. (7), we have

$$\begin{array}{r} \Gamma, G : (\emptyset, \emptyset) \vdash\ G : (\emptyset, \emptyset) \blacklozenge P_1\{\mathsf{s}[p_1]/y_1\} \mid ... \mid P_n\{\mathsf{s}[p_n]/y_n\}\ \mid\ \mathsf{s} : \emptyset \rhd \\ \{\mathsf{s}[p_1] : G\upharpoonright p_1, ..., \mathsf{s}[p_n] : G\upharpoonright p_n, \{\mathsf{s} : \emptyset\}\} \end{array} \tag{8}$$

By applying Lemma 3 to Eq. (8), we have

$$\Gamma, G : (\emptyset, \emptyset) \vdash\ \{\mathsf{s}[p_1] : G\upharpoonright p_1, ..., \mathsf{s}[p_n] : G\upharpoonright p_n, \{\mathsf{s} : \emptyset\}\} \quad \text{coherent} \tag{9}$$

By applying $\lfloor$T-s$\rfloor$ to Eq. (9) we derive

$$\Gamma, G : (\emptyset, \emptyset) \vdash\ (\nu \mathsf{s})(G : (\emptyset, \emptyset) \blacklozenge P_1\{\mathsf{s}[p_1]/y_1\} \mid ... \mid P_n\{\mathsf{s}[p_n]/y_n\}\ \mid\ \mathsf{s} : \emptyset) \rhd\ \emptyset$$

Thus $\Delta' = \emptyset = \Delta$. We conclude this case.

Now we prove subject reduction theorem.

Theorem 2 (Subject Reduction)

(a). $\Gamma \vdash\ \mathcal{S} \rhd \Delta$ with $\Gamma \vdash\ \Delta$ coherent and $\mathcal{S} \to^* \mathcal{S}'$ imply that $\exists \Delta'$ such that $\Gamma' \vdash\ \mathcal{S}' \rhd\ \Delta'$ and $\Gamma \vdash\ \Delta \to_T^* \Gamma' \vdash\ \Delta'$ or $\Delta \equiv \Delta'$ and $\Gamma' \vdash\ \Delta'$ coherent.
(b). $\Gamma \vdash\ \mathcal{S} \rhd\ \emptyset$ and $\mathcal{S} \to \mathcal{S}'$ imply that $\Gamma \vdash\ \mathcal{S}' \rhd\ \emptyset$.

Proof. **For (a).**

The proof is by induction on the derivation of $\mathcal{S} \to \mathcal{S}'$. We list rules in Fig. 6 (operational semantics of applications and system).

– **(Snd)**. Assume $\Gamma \vdash \ \mathtt{s}[p] : E[q!\ l(e).\eta] \ \mid \ \mathtt{s} : h \rhd \ \Delta$ and $e \Downarrow v$ and $\Gamma \vdash \ \Delta$ is coherent.

By applying Lemma 5.(14), we have

$$\Delta = \Delta_1, \Delta_2 \text{ and } \Gamma \vdash \ \mathtt{s}[p] : E[q!\ l(e).\eta] \rhd \ \Delta_1 \text{ and } \Gamma \vdash \ \mathtt{s} : h \rhd \ \Delta_2 \quad (10)$$

By applying Lemma 7 and Eq. (10) and $\lfloor$T-m$\rfloor$, we have

$$\Delta_2 = \{\mathtt{s} : \mathtt{h}\} \text{ and } \Gamma \vdash \ v : S \text{ and } \Gamma \vdash \ \mathtt{s} : h \cdot \langle p, q, l(v)\rangle \rhd \ \{\mathtt{s} : \mathtt{h} \cdot \langle p, q, l(S)\rangle\} \quad (11)$$

By applying Lemma 5.(4) and Lemma 8 to Eq. (10), we have

$$\Delta_1 = \{\mathtt{s}[p] : \mathcal{E}[q!\ l(S).T']\} \text{ and } \Gamma \vdash \ \mathtt{s}[p] : E[\eta] \rhd \ \{\mathtt{s}[p] : \mathcal{E}[T']\} \quad (12)$$

By $\lfloor$T-pa$\rfloor$ and Eq. (11) and Eq. (12), we derive

$$\Gamma \vdash \ \mathtt{s}[p] : E[\eta] \ \mid \ \mathtt{s} : h \cdot \langle p, q, l(v)\rangle \rhd \ \{\mathtt{s}[p] : \mathcal{E}[T'], \mathtt{s} : \mathtt{h} \cdot \langle p, q, l(S)\rangle\} \quad (13)$$

Let $\Delta' = \{\mathtt{s}[p] : \mathcal{E}[T'], \mathtt{s} : \mathtt{h} \cdot \langle p, q, l(S)\rangle\}$. By $[\![\mathtt{Snd}]\!]$, we have

$$G : (F_q, d) \vdash \Delta \rightarrow_T G : (F_q, d) \vdash \Delta'$$

and by Theorem 4 $\Gamma \vdash \ \Delta'$ is coherent. We conclude this case.

– **(Rcv)**. Assume $\Gamma \vdash \ \mathtt{s}[p] : E[q?\{l_i(x_i) : \eta_i\}_{i\in I}] \ \mid \ \mathtt{s} : \langle q, p, l(v)\rangle \cdot h \rhd \ \Delta$ and $\Gamma \vdash \ \Delta$ is coherent.

By applying Lemma 5.(14), we have $\Delta = \Delta_1, \Delta_2$

$$\Gamma \vdash \ \mathtt{s}[p] : E[q?\{l_i(x_i) : \eta_i\}_{i\in I}] \rhd \ \Delta_1 \text{ and } \Gamma \vdash \ \mathtt{s} : \langle q, p, l(v)\rangle \cdot h \rhd \ \Delta_2 \quad (14)$$

By applying Lemma 7 and Eq. (14), we have

$$\Delta_2 = \{\mathtt{s} : \langle q, p, l(v)\rangle \cdot \mathtt{h}\} \text{ and } \Gamma \vdash \ v : S \text{ and } \Gamma \vdash \ \mathtt{s} : h \rhd \ \{\mathtt{s} : \mathtt{h}\} \quad (15)$$

By applying Lemma 5.(5) and Lemma 8 to Eq. (14) we have

$$\begin{array}{l} \Gamma \vdash \ \mathtt{s}[p] : E[q?\{l_i(x_i) : \eta_i\}_{i\in I}] \rhd \ \{\mathtt{s}[p] : \mathcal{E}[q?\ \{l_i(S_i).T_i\}_{i\in I}]\} \\ \forall i \in I.\ \Gamma, x_i : S_i \ \vdash \ \mathtt{s}[p] : E[\eta_i] \rhd \ \{\mathtt{s}[p] : \mathcal{E}[T_i]\} \end{array} \quad (16)$$

By applying Lemma 6 (Substitution) and Eq. (16),

$$\Gamma \vdash \ \mathtt{s}[p] : E[\eta_k\{v_k/x_k\}] \rhd \ \{\mathtt{s}[p] : \mathcal{E}[T_k]\} \quad (17)$$

By $\lfloor$T-pa$\rfloor$ and Eq. (15) and Eq. (17), we derive

$$\Gamma \vdash \ \mathtt{s}[p] : E[\eta_k\{v_k/x_k\}] \ \mid \ \mathtt{s} : h \cdot \langle q, p, l(v)\rangle \rhd \ \{\mathtt{s}[p] : \mathcal{E}[T_k], \mathtt{s} : \mathtt{h}\}$$

Let $\Delta' = \{\mathtt{s}[p] : \mathcal{E}[T_k], \mathtt{s} : \mathtt{h}\}$. By $[\![\mathtt{Rcv}]\!]$, we have

$$G : (F_q, d) \vdash \Delta \rightarrow_T G : (F_q, d) \vdash \Delta'$$

and by Theorem 4 $\Gamma \vdash \ \Delta'$ is coherent. We conclude this case.

– **(Rec)**. Assume $\Gamma \vdash\ \mathsf{s}[p] : \mathsf{def}\ X(x) = \eta\ \mathsf{in}\ X\langle e\rangle \rhd\ \Delta$.

By applying Lemma 5.(8), we have

$$\begin{array}{l}\Delta = \{\mathsf{s}[p] : T\} \\ \Gamma, X : S\ \mu t.T' \vdash\ \mathsf{s}[p] : X\langle e\rangle \rhd\ \Delta \\ \Gamma, X : S\ t, x : S\ \vdash\ \mathsf{s}[p] : \eta \rhd\ \{\mathsf{s}[p] : T'\}\end{array} \tag{18}$$

By applying Lemma 5.(8) to Eq. (18), we have

$$\Gamma = \Gamma', X : S\ T \text{ and } \Gamma' \vdash\ e : S \tag{19}$$

By applying $\lfloor$`T-def`$\rfloor$ to Eq. (18) and Eq. (19), we derive

$$\Gamma, x : S\ \vdash\ \mathsf{s}[p] : \mathsf{def}\ X(x) = \eta\ \mathsf{in}\ X\langle e\rangle \rhd\ \Delta \tag{20}$$

By applying Lemma 6.(1) to Eq. (20), we have

$$\Gamma \vdash\ \mathsf{s}[p] : \mathsf{def}\ X(x) = \eta\ \mathsf{in}\ \eta\{v/x\} \rhd\ \Delta \text{ where } e \Downarrow v$$

Thus we conclude this case.

– **(TryHdl)**. Assume $\Gamma \vdash\ \mathsf{s}[p] : E[\mathsf{t}(\eta)\mathsf{h}(F : \eta', \mathsf{H})^{(\kappa, F_s)}.\eta'']\ \mid\ \mathsf{s} : h \rhd\ \Delta$ and $F = \cup\{A \mid A \in dom(\mathsf{H}) \wedge F_s \subset A \subseteq Fset(h, p)\}$ and $\Gamma \vdash\ \Delta$ is coherent.

By applying Lemma 5.(14), we have

$$\begin{array}{l}\Delta = \Delta_1, \Delta_2 \text{ and} \\ \Gamma \vdash\ \mathsf{s}[p] : E[\mathsf{t}(\eta)\mathsf{h}(F : \eta', \mathsf{H})^{(\kappa, F_s)}.\eta''] \rhd\ \Delta_1 \text{ and} \\ \Gamma \vdash\ \mathsf{s} : h \rhd\ \Delta_2\end{array} \tag{21}$$

By applying Lemma 8 and Lemma 5.(9) to Eq. (21), we have

$$\begin{array}{l}\Delta_1 = \{\mathsf{s}[p] : T\} \\ \Gamma\ \vdash\ \mathsf{s}[p] : E[\mathsf{t}(\eta)\mathsf{h}(F : \eta', \mathsf{H})^{(\kappa, F_s)}.\eta''] \rhd\ \{\mathsf{s}[p] : T\} \\ T = \mathcal{E}[\mathsf{t}(T')\mathsf{h}(F : T'', \mathcal{H})^{(\kappa, F_s)}.T'''] \\ \Gamma\ \vdash\ \mathsf{s}[p] : \eta \rhd\ \{\mathsf{s}[p] : T'\} \text{ and } \Gamma\ \vdash\ \mathsf{s}[p] : \eta'' \rhd\ \{\mathsf{s}[p] : T'''\} \text{ and} \\ dom(\mathsf{H}) = dom(\mathcal{H}) \text{ and } \forall F \in dom(\mathsf{H}).\ \Gamma\ \vdash\ \mathsf{s}[p] : \eta' \rhd\ \{\mathsf{s}[p] : T''\}\end{array} \tag{22}$$

By applying Lemma 7 to Eq. (21), we have

$$\Delta_2 = \{\mathsf{s} : \mathtt{h}\} \text{ and } \Gamma \vdash\ \mathsf{s} : h \rhd\ \{\mathsf{s} : \mathtt{h}\} \tag{23}$$

By $\lfloor$`T-th`$\rfloor$ and Eq. (22), we have

$$\begin{array}{rl}\Gamma\ \vdash & \mathsf{s}[p] : E[\mathsf{t}(\eta')\mathsf{h}(F : \eta', \mathsf{H})^{(\kappa, F)}.\eta''] \rhd \\ & \{\mathsf{s}[p] : \mathcal{E}[\mathsf{t}(T'')\mathsf{h}(F : T'', \mathcal{H})^{(\kappa, F)}.T''']\}\end{array} \tag{24}$$

By $\lfloor$T-pa$\rfloor$ and Eqs. (22), (23), (24), we derive

$$\begin{aligned}\Gamma \vdash\ & \mathtt{s}[p] : E[\mathtt{t}(\eta')\mathtt{h}(F{:}\eta', \mathsf{H})^{(\kappa,F)}.\eta''] \ \mid\ \mathtt{s} : h \\ & \rhd \{\mathtt{s}[p] : \mathcal{E}[\mathtt{t}(T'')\mathtt{h}(F{:}T'', \mathcal{H})^{(\kappa,F)}.T'''], \mathtt{s} : \mathtt{h}\}\end{aligned} \tag{25}$$

Let $\Delta' = \{\mathtt{s}[p] : \mathcal{E}[\mathtt{t}(T'')\mathtt{h}(F{:}T'', \mathcal{H})^{(\kappa,F)}.T'''], \mathtt{s} : \mathtt{h}\}$. By $[\![\texttt{TryHdl}]\!]$ we have

$$G : (F_q, d) \vdash \Delta \rightarrow_T G : (F_q, d) \vdash \Delta'$$

and by Theorem 4, $\Gamma \vdash\ \Delta'$ is coherent. We conclude this case.

– **(Cln)**. Assume $\Gamma \vdash\ \mathtt{s}[p] : E[\eta] \ \mid\ \mathtt{s} : \langle q, p, l(v)\rangle \cdot h \rhd\ \Delta$ and $l \notin \mathit{labels}(E[\eta])$ and Δ is coherent. (Note that, Def. 17 (The Effect of ht) defines that this kind of $\langle q, p, l(v)\rangle$ will not affect any endpoints in Δ).

By applying Lemma 5.(14), we have

$$\Delta = \Delta_1, \Delta_2 \text{ and } \Gamma \vdash\ \mathtt{s}[p] : E[\eta] \rhd\ \Delta_1 \text{ and } \Gamma \vdash\ \mathtt{s} : \langle q, p, l(v)\rangle \cdot h \rhd\ \Delta_2 \tag{26}$$

By applying Lemma 7 and Eq. (26), we have

$$\Delta_2 = \{\mathtt{s} : \langle q, p, l(e)\rangle \cdot \mathtt{h}\} \text{ and } \Gamma \vdash\ v : S \text{ and } \Gamma \vdash\ \mathtt{s} : h \rhd\ \{\mathtt{s} : \mathtt{h}\} \tag{27}$$

By $\lfloor$T-pa$\rfloor$ and Eq. (26) and Eq. (27), we have

$$\Gamma \vdash\ \mathtt{s}[p] : E[\eta] \ \mid\ \mathtt{s} : h \rhd\ \Delta_1, \{\mathtt{s} : \mathtt{h}\} \tag{28}$$

Let $\Delta' = \Delta_1, \{\mathtt{s} : \mathtt{h}\}$. By $[\![\texttt{Cln}]\!]$, we have

$$G : (F_q, d) \vdash \Delta \rightarrow_T G : (F_q, d) \vdash \Delta'$$

and by Theorem 4 $\Gamma \vdash\ \Delta'$ is coherent. We conclude this case.

– **(ClnDone)**. The proof is similar to the case **(Cln)**.

– **(Crash)**. Assume $\Gamma \vdash\ \mathtt{s}[p] : \eta \ \mid\ N \ \mid\ \mathtt{s} : h \rhd\ \Delta$.

By applying Lemma 5.(14) two times, we have

$$\begin{aligned}\Delta = \Delta_1, \Delta_2, \Delta_3 \text{ and } \Gamma \vdash\ & \mathtt{s}[p] : \eta \rhd\ \Delta_1 \text{ and} \\ & \Gamma \vdash\ N \rhd\ \Delta_2 \text{ and } \Gamma \vdash\ \mathtt{s} : h \rhd\ \Delta_3\end{aligned} \tag{29}$$

By applying Lemma 7 and $\lfloor$T-m$\rfloor$ to Eq. (29), we have

$$\Delta_3 = \{\mathtt{s} : \mathtt{h}\} \text{ and } \mathtt{h} = \mathtt{h}_0 \cdot \mathtt{m}_1 \cdot \mathtt{h}_1 ... \mathtt{h}_{n-1} \cdot \mathtt{m}_n \cdot \mathtt{h}_n \text{ where } p \in \mathtt{m}_i, i \in \{1..n\} \tag{30}$$

Let $\mathit{msg}(h, p)$ collect messages. By applying Lemma 7 and $\lfloor$T-m$\rfloor$ to Eq. (30), we have

$$\Gamma \vdash\ \mathtt{s} : \mathit{remove}(h, p) \rhd\ \{\mathtt{s} : (\mathit{remove}(\mathtt{h}, p)\} \tag{31}$$

By $\lfloor$T-pa$\rfloor$ and $\lfloor$T-$F\rfloor$ to Eq. (29) and Eq. (31), we derive

$$\Gamma \vdash \ N \ \mid \ \mathsf{s} : \mathit{remove}(h, p) \cdot \langle\!|\psi, \mathsf{crash}\ p|\!\rangle \rhd \tag{32}$$

$$\Delta_2, \{\mathsf{s} : \mathit{remove}(\mathtt{h}, p) \cdot \langle\!|\psi, \mathsf{crash}\ p|\!\rangle\} \tag{33}$$

Let $\Delta' = \Delta_2, \{\mathsf{s} : \mathtt{h} \setminus \mathit{msg}(\mathtt{h}, p) \cdot \langle\!|\psi, \mathsf{crash}\ p|\!\rangle\}$. By $[\![\mathtt{Crash}]\!]$ we have

$$G : (F_q, d) \vdash \Delta \rightarrow_T G : (F_q, d) \vdash \Delta'$$

and by Theorem 4, $\Gamma \vdash \ \Delta'$ is coherent. We conclude this case.

– **(SndDone)**. Assume $\Gamma \vdash \ \mathsf{s}[p] : E[\mathsf{t}(0)\mathsf{h}(\mathsf{H})^{\phi}.\eta'] \ \mid \ \mathsf{s} : h \rhd \ \Delta$

By applying Lemma 5.(14), we have

$$\begin{array}{l} \Delta = \Delta_1, \Delta_2 \text{ and} \\ \Gamma \vdash \ \mathsf{s}[p] : E[\mathsf{t}(0)\mathsf{h}(\mathsf{H})^{\phi}.\eta'] \rhd \ \Delta_1 \text{ and} \\ \Gamma \vdash \ \mathsf{s} : h \rhd \ \Delta_2 \end{array} \tag{34}$$

By applying Lemma 8 and Lemma 5.(9) and Lemma 5.(1) to Eq. (34), we have

$$\Delta_1 = \{\mathsf{s}[p] : T\} \tag{35}$$

$$\begin{array}{l} T = \mathcal{E}[\mathsf{t}(\mathsf{end})\mathsf{h}(\mathcal{H})^{\phi}.T'] \text{ and} \\ \Gamma \ \vdash \ \mathsf{s}[p] : 0 \rhd \ \{\mathsf{s}[p] : \mathsf{end}\} \text{ and } \Gamma \ \vdash \ \mathsf{s}[p] : \eta' \rhd \ \{\mathsf{s}[p] : T'\} \text{ and} \\ \mathit{dom}(\mathsf{H}) = \mathit{dom}(\mathcal{H}) \text{ and} \\ \forall F \in \mathit{dom}(\mathsf{H}).\ \Gamma \ \vdash \ \mathsf{s}[p] : \mathsf{H}(F) \rhd \ \{\mathsf{s}[p] : \mathcal{H}(F)\} \end{array} \tag{36}$$

By applying Lemma 7 and Eq. (34), we have

$$\Delta_2 = \{\mathsf{s} : \mathtt{h}\} \tag{37}$$

By $\lfloor$T-yd$\rfloor$ and $\lfloor$T-th$\rfloor$ and Eq. (35), we have

$$\Gamma \ \vdash \ \mathsf{s}[p] : E[\mathsf{t}(\underline{0})\mathsf{h}(\mathsf{H})^{\phi}.\eta'] \rhd \ \{\mathsf{s}[p] : \mathcal{E}[\mathsf{t}(\underline{0})\mathsf{h}(\mathcal{H})^{\phi}.T']\} \tag{38}$$

By $\lfloor$T-D$\rfloor$ and Eq. (37), we derive

$$\Gamma \vdash \ \mathsf{s} : h \cdot \langle p, \psi\rangle^{\phi} \rhd \ \{\mathsf{s} : \mathtt{h} \cdot \langle p, \psi\rangle^{\phi}\} \tag{39}$$

By $\lfloor$T-pa$\rfloor$ and Eqs. (35), (37), (38), (39), we derive

$$\begin{array}{l} \Gamma \vdash \ \mathsf{s}[p] : E[\mathsf{t}(\underline{0})\mathsf{h}(\mathsf{H})^{\phi}.\eta'] \ \mid \ \mathsf{s} : h \cdot \langle p, \psi\rangle^{\phi} \rhd \\ \qquad \{\mathsf{s}[p] : \mathcal{E}[\mathsf{t}(\underline{0})\mathsf{h}(\mathcal{H})^{\phi}.T'], \mathsf{s} : \mathtt{h} \cdot \langle p, \psi\rangle^{\phi}\} \end{array}$$

Let

$$\Delta' = \{\mathsf{s}[p] : \mathcal{E}[\mathsf{t}(\underline{0})\mathsf{h}(\mathcal{H})^{\phi}.T'], \mathsf{s} : \mathtt{h} \cdot \langle p, \psi\rangle^{\phi}\}.$$

By $[\![\mathtt{SndDone}]\!]$ we have

$$G : (F_q, d) \vdash \Delta \rightarrow_T G : (F_q, d) \vdash \Delta'$$

and by Theorem 4, $\Gamma \vdash \ \Delta'$ is coherent. We conclude this case.

– **(RcvDone)**. Assume $\Gamma \vdash \ \mathsf{s}[p] : E[\mathsf{t}(\underline{0})\mathsf{h}(\mathsf{H})^{\phi}.\eta] \ \mid \ \mathsf{s} : h \rhd \ \Delta$ and $\langle \psi, p \rangle^{\phi} \in h$.

By applying Lemma 5.(14), we have

$$\begin{array}{l} \Delta = \Delta_1, \Delta_2 \text{ and} \\ \Gamma \vdash \ \mathsf{s}[p] : E[\mathsf{t}(0)\mathsf{h}(\mathsf{H})^{\phi}.\eta'] \rhd \ \Delta_1 \text{ and} \\ \Gamma \vdash \ \mathsf{s} : h \rhd \ \Delta_2 \end{array} \tag{40}$$

By applying Lemma 8 and Lemma 5.(6) to Eq. (40), we have

$$\Delta_1 = \{\mathsf{s}[p] : T\} \tag{41}$$

$$\begin{array}{l} T = \mathcal{E}[\mathsf{t}(\underline{\mathsf{end}})\mathsf{h}(\mathcal{H})^{\phi}.T'] \text{ and} \\ \Gamma \ \vdash \ \mathsf{s}[p] : \underline{0} \rhd \ \{\mathsf{s}[p] : \underline{\mathsf{end}}\} \text{ and } \Gamma \ \vdash \ \mathsf{s}[p] : \eta' \rhd \ \{\mathsf{s}[p] : T'\} \text{ and} \\ dom(\mathsf{H}) = dom(\mathcal{H}) \text{ and } \forall F \in dom(\mathsf{H}).\ \Gamma \ \vdash \ \mathsf{H}(F) \rhd \ \mathcal{H}(F) \end{array} \tag{42}$$

By Lemma 7 and $\lfloor$T-D$\rfloor$ and Eq. (40) and the condition that $\langle \psi, p \rangle^{\phi} \in h$, we have

$$\Delta_2 = \{\mathsf{s} : \mathtt{h}\} \text{ and } \mathtt{h} = \mathtt{h}' \cdot \langle \psi, p \rangle^{\phi} \cdot \mathtt{h}'' \tag{43}$$

By applying Lemma 8 to Eq. (41), we have

$$\Gamma \ \vdash \ \mathsf{s}[p] : E[\eta'] \rhd \ \{\mathsf{s}[p] : \mathcal{E}[T']\} \tag{44}$$

By applying Eq. (43), we derive

$$\Gamma \vdash \ \mathsf{s} : h \setminus \langle \psi, p \rangle^{\phi} \rhd \ \{\mathsf{s} : \mathtt{h}' \cdot \mathtt{h}''\} \tag{45}$$

By $\lfloor$T-pa$\rfloor$ and Eqs. (44), (45), we derive

$$\Gamma \vdash \ \mathsf{s}[p] : E[\eta'] \ \mid \ \mathsf{s} : h \setminus \langle p, \psi \rangle^{\phi} \rhd \ \{\mathsf{s}[p] : \mathcal{E}[T'], \mathsf{s} : \ \mathtt{h}' \cdot \mathtt{h}''\}$$

Let $\Delta' = \{\mathsf{s}[p] : \mathcal{E}[T'], \mathsf{s} : \ \mathtt{h}' \cdot \mathtt{h}''\}$. By $[\![$RcvDone$]\!]$ we have

$$G : (F_q, d) \vdash \Delta \rightarrow_T G : (F_q, d) \vdash \Delta'$$

and by Theorem 4, $\Gamma \vdash \ \Delta'$ is coherent.

– **(F)**. Assume $\Gamma \vdash \ G : (F_q, d)\blacklozenge\mathsf{s} : \langle\!|\psi, \mathsf{crash}\ F|\!\rangle \cdot h \rhd \ \Delta$ and $\exists \mathsf{t}(...)\mathsf{h}(H)^{\kappa} \in G$ such that $F \in dom(H)$.

By applying Lemma 5.(16), we have

$$\begin{array}{r} G : (F_q, d) \in \Gamma \text{ and } \Gamma' \vdash \ \mathsf{s} : \langle\!|\psi, \mathsf{crash}\ F|\!\rangle \cdot h \rhd \ \Delta \\ \text{such that } \Gamma = \Gamma', G : (F_q, d) \end{array} \tag{46}$$

By applying Lemma 7 and Eq. (46), we have

$$\Delta = \{\mathsf{s} : \langle\!|\psi, \mathsf{crash}\ F|\!\rangle \cdot \mathtt{h}\} \text{ and } \Gamma' \vdash \ \mathsf{s} : h \rhd \ \{\mathsf{s} : \mathtt{h}\} \tag{47}$$

By $\lfloor$T-sys$\rfloor$ and $\lfloor$T-$F\rfloor$ and Eq. (47), we derive

$$\begin{aligned}\Gamma', G : (F_q \cup F, d) \vdash\ & G : (F_q \cup F, d)\blacklozenge \mathsf{s} : \\ & h \cdot \langle\!\langle roles(G) \setminus (F_q \cup F), \mathsf{crash}\ F \rangle\!\rangle \cdot \rhd \\ & \quad \{\mathsf{s} : \mathtt{h} \cdot \langle\!\langle roles(G) \setminus (F_q \cup F), \mathsf{crash}\ F \rangle\!\rangle\}\end{aligned} \tag{48}$$

Let

$$\Delta' = \{\mathsf{s} : \mathtt{h} \cdot \langle\!\langle roles(G) \setminus (F_q \cup F), \mathsf{crash}\ F \rangle\!\rangle\}$$

By $[\![\mathtt{F}]\!]$ we have

$$G : (F_q, d) \vdash \Delta \rightarrow_T G : (F_q \cup F, d) \vdash\ \Delta'$$

and by Theorem 4, $\Gamma', G : (F_q \cup F, d) \vdash\ \Delta'$ is coherent. We conclude this case.

– **(CollectDone)**. The proof is trivial.

– **(IssueDone)**. Assume $\Gamma\blacklozenge\mathsf{s} : h \rhd\ \Delta$ and $\Gamma = \Gamma', G : (F_q, d)$ and there exists ϕ such that $roles(d, \phi) \supseteq roles(G, \phi) \setminus F_q$ and $\forall F \in hdl(G, \phi).(F \not\subseteq F_q)$

By applying Lemma 5.(16), we have

$$\Gamma' \vdash\ \mathsf{s} : h \rhd\ \Delta \tag{49}$$

By applying Lemma 7 and Eq. (49) and $\lfloor$T-D$\rfloor$, we have

$$\begin{aligned}&\Delta = \{\mathsf{s} : \mathtt{h}\} \\ &\mathtt{h} = \mathtt{h}_0 \cdot \langle p_1, \psi\rangle^\phi \cdot \mathtt{h}_1 \cdot ... \cdot \langle p_n, \psi\rangle^\phi \cdot \mathtt{h}_n \text{ where } roles(d, \phi) = \{p_1, ..., p_n\}\end{aligned} \tag{50}$$

By $\lfloor$T-sys$\rfloor$ and $\lfloor$T-D$\rfloor$ and Eq. (50), we derive

$$\begin{aligned}\Gamma', G : (F_q, remove(d, \phi)) \vdash\ & G : (F_q, remove(d, \phi))\blacklozenge \mathsf{s} : \\ & h \cdot \langle \psi, roles(G, \phi) \setminus (F_q)\rangle^\phi \\ & \quad \rhd \{\mathsf{s} : (\mathtt{h} \cdot \langle \psi, roles(G, \phi) \setminus (F_q)\rangle^\phi\}\end{aligned} \tag{51}$$

Let

$$\Delta' = \{\mathsf{s} : \{\mathsf{s} : (\mathtt{h} \cdot \langle \psi, roles(G, \phi) \setminus (F_q)\rangle^\phi\}$$

By $[\![\mathtt{IssueDone}]\!]$ we have

$$G : (F_q, d) \vdash \Delta \rightarrow_T G : (F_q, d') \vdash\ \Delta'$$

where $d' = remove(d, \phi)$ and by Theorem 4, $\Gamma, G : (F_q, d') \vdash\ \Delta'$ is coherent. We conclude this case.

For (b).

The proof is immediately because $\Gamma \vdash\ \emptyset$ is always coherent.

Progress. This section proves the property of progress. We first state the following auxiliary lemmas which will be used in our proof.

Lemma 10. If $\Gamma \vdash \Delta$ is coherent and $\mathtt{s} : \mathtt{h} \in \Delta$ and $\Gamma = \Gamma', G : (F_q, d)$, then $\Gamma', G : (F_q, d) \vdash \Delta \rightarrow_T \Gamma', G : (F_q{}', d') \vdash \Delta'$ for some $F_q{}', d'$.

Proof.

(A) Firstly, if Δ is not end-only and it exists $\mathtt{s}[p] \in \mathit{dom}(\Delta)$ which is non-robust, by $[\![\texttt{Crash}]\!]$, it is always possible that $\mathtt{s}[p] \in \mathit{dom}(\Delta)$ crashes. Then $G : (F_q, d) \vdash \Delta, \mathtt{s}[p] : T, \mathtt{s} : \mathtt{h} \rightarrow_T G : (F_q, d) \vdash \Delta, \mathtt{s} : \mathtt{h} \cdot \langle\!\langle \psi, \mathsf{crash}\ F \rangle\!\rangle$.
If $\mathtt{h} \equiv \langle\!\langle \psi, \mathsf{crash}\ F \rangle\!\rangle \cdot \mathtt{h}'$, it implies crash F has happened; then either

(a) $\exists \mathtt{t}(G')\mathtt{h}(H)^{\kappa} \in G.F \in \mathit{dom}(H)$, then by $[\![\texttt{F}]\!]$, $G : (F_q, d) \vdash \Delta \rightarrow_T G : (F_q, d) \vdash \Delta'$ such that Δ' is the result of the trigger of some handler;

(b) Otherwise, it violates rule $[\![\texttt{Crash}]\!]$.

(B) If no processes crash and Δ is not end-only, we mechanically prove all cases in Fig. 18. By the structure of local types (defined in Section 5), we have the following possible types at endpoints in Δ:

1 There exists $\mathtt{s}[p] : \mathcal{E}[q!\{l_i(S_i).T_i\}_{i \in I}]$. By $[\![\texttt{Snd}]\!]$, $G : (F_q, d) \vdash \Delta \rightarrow_T G : (F_q, d) \vdash \Delta'$.

2 There exists $\mathtt{s}[p] : \mathcal{E}[\mathtt{t}(\mathsf{end})\mathtt{h}(\mathcal{H})^{\phi}.T']$. By $[\![\texttt{SndDone}]\!]$, $G : (F_q, d) \vdash \Delta \rightarrow_T G : (F_q, d) \vdash \Delta'$.

3 $\mathtt{h} \equiv \langle q, p, l_k(S_k) \rangle \cdot \mathtt{h}'$. If $\mathtt{s}[p] : T \in \Delta$ because $\Gamma \vdash \Delta$ is coherent, then either by $[\![\texttt{Rcv}]\!]$ or $[\![\texttt{Cln}]\!]$, $G : (F_q, d) \vdash \Delta \rightarrow_T G : (F_q, d) \vdash \Delta'$.

4 $\langle \psi, p \rangle^{\phi} \in \mathtt{h}$ by $[\![\texttt{CollectDone}]\!]$, $G : (F_q, d) \vdash \Delta, \mathtt{s} : \mathtt{h} \rightarrow_T G : (F_q, d \cup \{\langle \psi, p \rangle^{\phi}\}) \vdash \Delta, \mathtt{s} : \mathtt{h} \setminus \{\langle \psi, p \rangle^{\phi}\}$.

5 If $\mathit{roles}(d, \phi) \supseteq \mathit{roles}(G, \phi) \setminus F_q$ and $\forall F \in \mathit{hdl}(G, \phi)$ we have $F \not\subseteq F_q$, then $[\![\texttt{IssueDone}]\!]$ is applied and $G : (F_q, d) \vdash \Delta, \mathtt{s} : \mathtt{h} \rightarrow_T G : (F_q, \mathit{remove}(d, \phi)) \vdash \Delta, \mathtt{s} : \mathtt{h} \cdot \langle \psi, \mathit{roles}(G, \phi) \setminus F_q \rangle^{\phi}$.

6 $\langle \psi, p \rangle^{\phi} \in \mathtt{h}$. Because $\Gamma \vdash \Delta$ is coherent, so we have $G : (F_q, d) \in \Gamma$ and there exists $G' = \mathtt{t}(G'')\mathtt{h}(H)^{\phi}, G' \in G$ and $p \in \mathit{roles}(G') \setminus F_q$ and $\langle \psi, p \rangle^{\phi}$ is issued only when $\mathtt{s}[p]$ has sent out $\langle p, \psi \rangle^{\phi}$, which means, by $[\![\texttt{SndDone}]\!]$, either (a) there exists $\mathtt{s}[p] : \mathcal{E}[\mathtt{t}(\underline{\mathsf{end}})\mathtt{h}(\mathcal{H})^{\phi}.T]$. Moreover, $p \in \mathit{roles}(G') \setminus F_q$ implies that $\mathtt{s}[p]$ is alive at the moment. Thus we have $\mathtt{s}[p]$ exists to apply $[\![\texttt{RcvDone}]\!]$ such that $G : (F_q, d) \vdash \Delta, \mathtt{s}[p] : \mathcal{E}[\mathtt{t}(\underline{\mathsf{end}})\mathtt{h}(\mathcal{H})^{\phi}.T] \rightarrow_T G : (F_q, d) \vdash \Delta, \mathtt{s}[p] : T$. Or (b) $\mathtt{s}[p] = T$ and $\phi \notin T$ then by $[\![\texttt{ClnDone}]\!]$ $G : (F_q, d) \vdash \Delta, \mathtt{s}[p] : T, \mathtt{s} : \mathtt{h} \rightarrow_T G : (F_q, d) \vdash \Delta, \mathtt{s}[p] : T, \mathtt{s} : \mathtt{h} \setminus \langle \psi, p \rangle^{\phi}$.

7 $\mathtt{h} \equiv \langle\!\langle \psi, \mathsf{crash}\ F \rangle\!\rangle \cdot \mathtt{h}'$. Then $[\![\mathtt{F}]\!]$ is applied and $G : (F_q, d) \vdash\ \Delta, \mathsf{s} : \mathtt{h} \to_T G : (F_q \cup F, d) \vdash\ \Delta, \mathsf{s} : \mathtt{h}', \langle\!\langle \tilde{q}, \mathsf{crash}\ F \rangle\!\rangle (q = \mathit{roles}(G) \setminus (F_q \cup F))$

8 $\mathtt{h} \equiv \langle\!\langle p, \mathsf{crash}\ F \rangle\!\rangle \cdot \mathtt{h}'$. Then either

i. there exists $\mathsf{s}[p] : \mathcal{E}[\mathsf{t}(T)\mathsf{h}(F' : T', \mathcal{H})^{\phi}.T'']$ such that $F' = \cup\{A \mid A \in \mathit{dom}(\mathsf{H}) \wedge F_s \subset A \subseteq \mathit{Fset}(h, p)\}$. Then by $[\![\mathtt{TryHdl}]\!]$, we have $G : (F_q, d) \vdash\ \Delta \to_T G : (F_q, d) \vdash\ \Delta'$; or

ii. $\mathsf{s}[p] : T$ and $\forall \mathsf{t}(T')\mathsf{h}(\mathcal{H})^{\phi} \in T$, $F \notin \mathit{dom}(\mathcal{H})$. Then we shall have either $T = \mathsf{end}$ or T is one of Cases 1–8.

9 There exists $\mathsf{s}[p] : \mathcal{E}[q?\{l_i(S_i).T_i\}_{i \in I}]$ and $\mathsf{s}[q] \notin \Delta$ by rule $[\![\mathtt{Crash}]\!]$. Assume we have $q \in F$ and $\mathtt{h} \equiv \langle\!\langle p, \mathsf{crash}\ F \rangle\!\rangle \cdot \mathtt{h}$ (If that is not the case we either have $\mathtt{h} \equiv \langle\!\langle \psi, \mathsf{crash}\ F \rangle\!\rangle \cdot \mathtt{h}'$ (goes to case 7) or $\mathtt{h} \equiv \mathtt{h}' \cdot \langle \psi, p \rangle^{\phi} \cdot \mathtt{h}'' \cdot \langle\!\langle p, \mathsf{crash}\ F \rangle\!\rangle \cdot \mathtt{h}'''$ (goes to case (6))). By Def. 11 (Well-formedness) (5) $\mathcal{E}[q?\{l_i(S_i).T_i\}_{i \in I}]$ is enclosed by a try-handle which can handle F. The case goes to (8)i.

(C) If Δ is end-only, then we the statement is true.

We conclude the proof after checking all cases.

Lemma 11. If $\Gamma \vdash\ \mathcal{S} \rhd\ \Delta$ and $G : (F_q, d) \in \Gamma$ and $G : (F_q, d) \vdash\ \Delta_{\mathsf{s}} \to_T G : ({F_q}', d') \vdash\ \Delta'_{\mathsf{s}}$, then there exists $\mathcal{S}'$ such that $\mathcal{S} \to \mathcal{S}'$.

Proof. W.l.o.g, assume $\mathcal{S} = \Psi \blacklozenge N | N'$ and $\Gamma \vdash\ \Psi \blacklozenge N \rhd\ \Delta_{\mathsf{s}}$. By Lemma 5 and typing rules defined in Fig. 10 and Fig. 11, $G : (F_q, d) \vdash\ \Delta_{\mathsf{s}} \to_T G : ({F_q}', d') \vdash\ \Delta'_{\mathsf{s}}$ implies that there exists some $\mathsf{s}[p] : \eta \in N$ and $\mathsf{s} : h \in N$ such that $N = \mathsf{s}[p] : \eta \mid \mathsf{s} : h \mid N_0$ and $\mathsf{s}[p] : \eta \mid \mathsf{s} : h \to \mathsf{s}[p] : \eta' \mid \mathsf{s} : h'$ Thus there exists $\mathcal{S}' = \Psi \blacklozenge \mathsf{s}[p] : \eta' \mid \mathsf{s} : h' \mid N_0 \mid N'$.

Given $\mathcal{S} = (\nu\ \mathsf{s})(\Psi \blacklozenge N) \mid N'$ and $\mathsf{s} \notin \mathit{fn}(N')$ is possible, we can have some participants in N' which do not join session s but may later or immediately create another session concurrently.

To cater this kind of of situation, we define an *initializable* $\mathcal{S}$ such that, $\forall a[p].P \in \mathcal{S}$, **(Link)** is applicable:

Definition 21 (Initializable $\mathcal{S}$). Given $\Gamma \vdash\ \mathcal{S} \rhd\ \emptyset$. $\mathcal{S}$ is initializable under Γ if $\exists a[p].P \in \mathcal{S}$ implies that $\Gamma \vdash\ a : G$ for some G, $\forall p_i \in \mathit{roles}(G), a[p_i].P_i \in \mathcal{S}$ and **(Link)** is applicable to them.

Theorem 3(Progress) If $\Gamma \vdash\ \mathcal{S} \rhd\ \emptyset$ and $\mathcal{S}$ is initializable, then either $\mathcal{S} \to^* \mathcal{S}'$ and $\mathcal{S}'$ is initializable or $\mathcal{S}' = \Psi \blacklozenge \mathsf{s} : h \mid ... \mid \Psi' \blacklozenge \mathsf{s}' : h'$ and $h, ..., h'$ only contain failure notifications sent by coordinators and messages heading to failed participants.

Proof. The proof is by Lemma 10 and Theorem 1 and Theorem 2.

For convenience, we also use Q to range over processes.

1. Assume no sessions in $\mathcal{S}$ have started, for example

$$\Psi\blacklozenge a[p_1].P_1 \mid ... \mid a[p_n].P_n \mid N$$

Since $\mathcal{S}$ is initializable, $\mathcal{S}$ either terminates (with only global queue left) or $\mathcal{S} \to \mathcal{S}'$ and $\mathcal{S}'$ is initializable and $\Gamma \vdash \mathcal{S}' \rhd \emptyset$ by Theorem 2. (2).

2. Assume some session, say s, has started its communication.

W.o.l.g. assume $\mathcal{S} = (\nu \mathsf{s})(\Psi\blacklozenge N) \mid N'$ and $\mathsf{s} \notin fn(N')$, i.e., N' has no any session running in it.

For all $P \in N$ and $Q \in N'$, we have $P \neq Q$. E.g.

$$(\nu \mathsf{s})(\mathsf{s}[p] : \eta \mid \mathsf{s}[p'] : \eta' \mid \mathsf{s} : h) \mid a[q].Q \mid a[q'].Q'$$

where a is a shared name ready to start another session. For the processes in N', the proof goes to Case (1), which says that due to initializability those processes will start a session without interfering the existing s; for the processes in N, the proof goes to Case (3).

3. If all sessions in $\mathcal{S}$ have started by **(Link)**.

W.o.l.g. assume $\mathcal{S} = (\nu \mathsf{s})(\Psi\blacklozenge N) \mid \mathcal{S}''$.

Since $(\nu \mathsf{s})(\Psi\blacklozenge N)$ and $\mathcal{S}'$ do not interfere each other, we can analyze them independently:

(a) For the part of $(\nu \mathsf{s})(\Psi\blacklozenge N)$, by applying Lemma 5.(15) to $\mathcal{S}$, we have

$$\begin{aligned} &\Gamma \vdash (\nu \mathsf{s})(\Psi\blacklozenge N) \rhd \emptyset \\ &\Gamma \vdash \Psi\blacklozenge N \rhd \Delta = \Delta_{\mathsf{s}} \\ &\Gamma \vdash \Delta_{\mathsf{s}} \text{ coherent} \end{aligned} \tag{52}$$

By Lemma 10, we have

$$\Gamma \vdash \Delta \to_T^* \Gamma \vdash \Delta'$$

where Δ' is end-only for session s.

By Lemma 11 and Theorem 2, there exists N' such that

$$\Psi\blacklozenge N \to^* \Psi\blacklozenge N' \text{ and } \Gamma \vdash \Psi\blacklozenge N' \rhd \Delta' \text{ and } \mathcal{S} \to^* (\nu \mathsf{s})(\Psi\blacklozenge N') = \mathcal{S} \tag{53}$$

By applying $\lfloor \mathtt{T\text{-}s} \rfloor$ to Eq. (53), we have $\Gamma \vdash (\nu \mathsf{s})\mathcal{S}' = \mathcal{S}'' \rhd \emptyset$.

Since Δ' is end-only for session s, it implies all interactions at endpoint processes in s have finished.

Moreover, based on **(CollectDone)**, **(IssueDone)**, and **(F)**, since a coordinator always consume notifications of $\langle p, \psi \rangle^{\phi}$ and $\langle\!\langle \psi, \mathsf{crash}\ F \rangle\!\rangle$ for any p, ϕ, F, so if there is any this kind of message left, the coordinator will consume so $\mathcal{S}' \to \mathcal{S}''$ until there is no more such messages in $\mathcal{S}''$.

(b) For the part of $\mathcal{S}''$, since all participants have joined some sessions (by assumption), the proof is as same as the above.