A Type Theory for Robust Failure Handling in Distributed Systems*

Tzu-Chun Chen[1], Malte Viering[1], Andi Bejleri[1], Lukasz Ziarek[2], and Patrick Eugster[1,3]

[1] Department of Computer Science, TU Darmstadt, Darmstadt, Germany
{tc.chen,viering,bejleri,peugster}@dsp.tu-darmstadt.de
[2] Department of Computer Science and Engineering, SUNY Buffalo, New York, USA
lziarek@buffalo.edu
[3] Department of Computer Science, Purdue University, West Lafayette, USA

Abstract. This paper presents a formal framework for programming distributed applications capable of handling partial failures, motivated by the non-trivial interplay between failure handling and messaging in asynchronous distributed environments. Multiple failures can affect protocols at the level of individual interactions (**alignment**). At the same time, only participants affected by a failure or involved in its handling should be informed of it, and its handling should not be mixed with that of other failures **(precision)**. This is particularly challenging, as through the structure of protocols, failures may be linked to others in subsequent or concomitant interactions **(causality)**. Last but not least, no central authority should be required for handling failures **(decentralisation)**. Our goal is to give developers a description language, called protocol types, to specify robust failure handling that accounts for alignment, precision, causality, and decentralisation. A type discipline is built to statically ensure that asynchronous failure handling among multiple endpoints is free from orphan messages, deadlocks, starvation, and interactions never get stuck.

Keywords: Session Types, Partial Failure Handling, Distributed Systems

1 Introduction

For distributed systems where application components interact asynchronously and concurrently, the design and verification of communication protocols is critical. These systems are prone to *partial failures*, where some components or interactions may fail, while others must continue while respecting certain invariants. Since not all failures can be simply *masked* [10], programmers must *explicitly* deal with failures.

To ease the burden on programmers for constructing resilient communication protocols in the presence of partial failures, we propose a framework for *robust failure handling*. Our framework ensures safety during normal execution and in case of failures. In particular our framework provides the following properties:

* Financially supported by ERC grant FP7-617805 "LiVeSoft – Lightweight Verification of Software".

1. **P1 (alignment)**: The occurrences of failures are specified at the level of individual interactions, which they be raised.
2. **P2 (precision)**: If a failure occurs, an endpoint is informed iff it is affected by the failure or involved in handling it, and its handling is not mixed with that of other failures.
3. **P3 (causality)**: Dependencies between failures are considered, i.e., a failure can affect (enable, disable) others which may occur in subsequent or concomitant interactions.
4. **P4 (decentralisation)**: No central authority or component controls the decisions or actions of the participants to handle a failure.

Inspired by session types [12,19], we introduce *protocol types* to achieve these properties. The basic design is shown in Eq. (1):

$$\mathbf{T}[p_1 \to p_2 : \widetilde{S} \vee f_1, ..., f_n; \mathbf{G}]\mathbf{H}[f_1 : \mathbf{G}_1, ..., f_n : \mathbf{G}_n, ...] \tag{1}$$

where the first term expresses that a participant p_1 either sends a message of type $\widetilde{S}$ to another participant p_2, or raises one of several failures (i.e., $f_1, ..., f_n$). $\mathbf{G}$ specifies the subsequent interactions and $\mathbf{G}_i$ specifies the handling protocols for f_i, $i = 1..n$ in $\mathbf{H}$. In short, failures are thus associated with elementary interactions (**P1**), only participants affected by such a failure in $\mathbf{G}$ (e.g., they are expecting communication from p_2) or those involved in the corresponding failure handling activity are informed (**P2**). There is at most one f appearing in $\mathbf{H}$ that will be handled/raised (**P3**). Our semantics ensure that there is no central authority (**P4**), meaning, notifications of failures (and absence thereof) are delivered asynchronously from failure sources and processes are typed by local (i.e., endpoint) types achieved by the projection of participants over protocol types.

This design results in a distinguishable type system from the design of Eq. (2):

$$\mathbf{T}[p_1 \to p_2 : \widetilde{S}; \mathbf{G}]\mathbf{H}[\mathbf{G}'] \tag{2}$$

In Eq. (2), only one failure may occur in a try-block and where/when it will occur is not specified; once a failure — no matter which one — occurs somewhere in $p_1 \to p_2 : \widetilde{S}$ or $\mathbf{G}$, the handling activity $\mathbf{G}'$ simply takes over. Previous works on dealing with failures [2,3,4,7] are based on Eq. (2) and/or centralised authorities, and hence do not satisfy all of **P1** to **P4** (see Section 7 for details).

Just as multiparty session types aim to specify the interactions among participants and verify implementations of these participants, protocol types specify the global interplay of failures in interactions among participants and verify the failure-handling activities of these participants. To the best of our knowledge, this is the first work that presents a type system for statically checking fine-grained failure handling activities across asynchronous/concurrent processes for partial failures in practical distributed systems. We define a calculus of processes with the ability to not only raise ("throw") and handle ("catch") failures, but to also automatically notify processes of failures or absence thereof at runtime.

Our framework also gives protocol designers and endpoint application developers simple and intuitive description/programming abstractions, and ensures safe interactions among endpoint applications in concurrent environments.

Paper structure. Section 2 gives motivating examples to introduce the design of protocol types, which capture the properties **P1** to **P4**; then we introduce an operation called *transformation* to transform a protocol type to local (i.e., endpoint) types while preserving the desired properties. Section 3 gives a process calculus with de-centralised multiple-failure-handling capability, including the syntax for programs and networks, and the operational semantics for runtime. Section 4 gives a type system for local processes and Section 5 gives a type system for networks to maintain communication coherent. Section 6 states the property of safety, including subject reduction and communication safety, and the property of progress. Section 7 discusses related works. Finally Section 8 concludes our work.

Detailed formal and auxiliary definitions, lemmas and proofs are in Optional Appendices A and B respectively.

2 Protocol Types, Local Types, and Transformation

This section uses examples to show the design behind protocol types and uses Figure 1 to illustrate an operation called *transformation*, which generates local types from protocol types. All formal definitions are in Optional Appendix A.

The first example, visualised by Figure 1, shows the properties **P1** and **P2**. Assume that in a network all outgoing traffic passes through a proxy (Proxy), monitoring the traffic and logging general information, e.g., consumed bandwidth. The proxy sends this information to a log server (Log). If the proxy detects suspicious behaviour in the traffic, it raises a SuspiciousB failure and notifies Log and a supervision server (SupServer) to handle the failure by having Log send the traffic logs to SupServer; if the proxy detects that the quota of Client is low, it raises a QuotaWarn failure and notifies Log and Client to handle the failure by having Log send quota information to Client. Then Proxy forwards the traffic from Client to an external server (EServer). We propose the following type to formalise the above scenario:

$$\begin{aligned}\mathbf{G}_{proxy} = \mathbf{T}[&\mathsf{Client} \to \mathsf{Proxy} : \mathsf{str};\\ &\mathsf{Proxy} \to \mathsf{Log} : \mathsf{str} \vee \mathsf{SuspiciousB}, \mathsf{QuotaWarn}; \mathsf{Proxy} \to \mathsf{EServer} : \mathsf{str}]\\ &\mathbf{H}[\mathsf{SuspiciousB} : \mathsf{Log} \to \mathsf{SupServer} : \mathsf{str},\ \mathsf{QuotaWarn} : \mathsf{Log} \to \mathsf{Client} : \mathsf{str}]; \mathsf{end}\end{aligned}$$

It specifies that either SuspiciousB or QuotaWarn may occur at interaction Proxy $\to$ Log (**P1**). Proxy can raise the failure and correspondingly sends *failure/non-failure notifications* to all relevant parties. SuspiciousB affects Log and SupServer since they both handle that failure; it also affects Client because the occurrence of SuspiciousB implies that QuotaWarn will never occur, and thus Client will not yield to the handling activity for QuotaWarn. When QuotaWarn occurs, the situation is similar to SuspiciousB's. Once one of them occurs, Proxy sends failure notifications carrying the occurred failure to Log, SupServer, and Client; when no failures occur, Proxy sends non-failure notifications carrying both failures to the same participants to inform them not to yield to handling activity. No failures will affect Proxy and EServer (**P2**) because Proxy and EServer are not involved in any failure handling activities: Proxy continues sending to EServer after it raises a failure and EServer still receives a message from Proxy as expected.

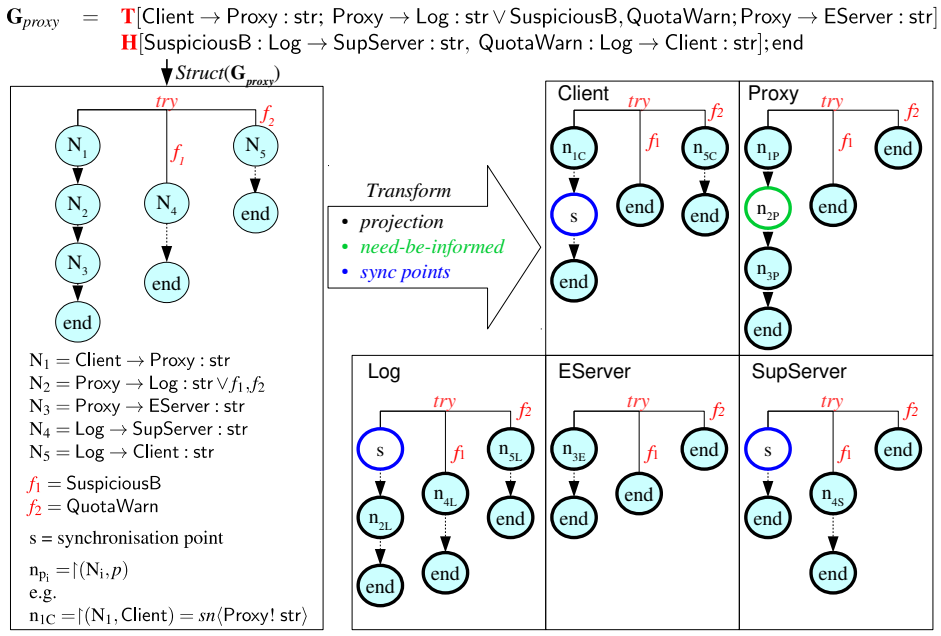

Fig. 1: Overview of *transformation* to obtain local types from a protocol type. The left-hand-side figure visualises a protocol type, $\mathbf{G}_{proxy}$, as a global structure by function *Struct*. The right-hand-side figures are local types for participants in $\mathbf{G}_{proxy}$. They are gained by an operation called *transformation*, which firstly *projects* the global structure onto participants to get *simple* local types, then adds the information of *need-be-informed* participants to the positions with green rings, and finally adds *synchronisation points* to the positions with blue rings. After *transformation*, local types for robust failure handling are reached.

The next example shows **P2** and **P3**. Assume a resource provider (RP) informs a coordinator (Coord) of which resources it can provide. The library (Lib) can get the resource by requesting Coord only if RP sends a list of resources to Coord. If failure NoRes occurs, meaning absence of resource, Coord informs Lib of this; otherwise, Lib places a request to Coord or raises a failure Abort (due to one of many possible local problems) and Coord invokes Record to record this failure:

$$\begin{aligned}\mathbf{G}_{coord} = \ & \mathbf{T}[\mathsf{RP} \to \mathsf{Coord} : \mathsf{str} \vee \mathsf{NoRes}; \mathsf{Lib} \to \mathsf{Coord} : \mathsf{str} \vee \mathsf{Abort}; \mathsf{Coord} \to \mathsf{Lib} : \mathsf{int}] \\ & \mathbf{H}[\mathsf{NoRes} : \mathsf{Coord} \to \mathsf{Lib} : \mathsf{bool},\ \mathsf{Abort} : \mathsf{Coord} \to \mathsf{Record} : \mathsf{str}]; \mathsf{end}\end{aligned}$$

Although it may seem that interactions RP → Coord and Lib → Coord can run concurrently, this is not the case because, Lib can only get the resource if RP gives Coord a resource list, which implies failures NoRes and Abort are dependent (**P3**). Additionally, we constrain that failure handling should be not be mixed (**P2**). *Synchronisation* of Lib to yield to the completion of RP → Coord is thus needed. This helps programmers

$$\text{(Sorts) } S ::= \mathsf{bool} \mid \mathsf{unit} \mid \mathsf{int} \mid \mathsf{str} \quad \text{(Failure) } f ::= \mathsf{AuthFail} \mid \mathsf{Abort} \mid \mathsf{ExecFail} \mid \ldots$$
$$\text{(Handlers) } h ::= \emptyset \mid h, f : g \qquad \text{(Set of Failures) } F ::= \emptyset \mid F, f$$
$$\text{(Inter. Types) } g ::= \varepsilon \mid p \to p : \widetilde{S} \vee F \mid \mathbf{T}[g]\mathbf{H}[h] \mid g;g \mid t \mid \mu t.g \quad \text{(Protocol Types) } \mathbf{G} ::= g;\mathsf{end}$$

Fig. 2: Syntax of protocol types.

tremendously in reasoning about the states that participants are in after failures. Note that, if $\mathsf{RP} \to \mathsf{Coord}$ and $\mathsf{Lib} \to \mathsf{Coord}$ have no failures specified, then $\mathsf{RP} \to \mathsf{Coord}$ and $\mathsf{Lib} \to \mathsf{Coord}$ can run concurrently because there is no failure dependency.

2.1 Protocol Types

To handle *partial failures* in interactions which exhibit the properties **P1** to **P4** , Figure 2 defines protocol types based on the definition of session types given in the work of Bettini *et al.* [1]. Protocol types, denoted by $\mathbf{G}$, are composed of interaction types g and terminated by end. We use $(p, \ldots)$ to range over identifiers, $(S, \ldots)$ to range over basic types like bool, unit, int, and str, and $(F, \ldots)$ to range over sets of failures. We highlight the key concepts:

(A) $p_1 \to p_2 : \widetilde{S} \vee F$ is a *failure-raising interaction tagged with* F, or F*-raising interaction* for short. When $\widetilde{S} \neq \emptyset$ and $F \neq \emptyset$, either p_1 sends a content of type $\widetilde{S}$ or raises *one* failure in F to p_2. When one of $\widetilde{S}$ and F is empty, p_1 only makes an output based on the non-empty one. We do not allow both F and $\widetilde{S}$ to be empty.

(B) $\mathbf{T}[g]\mathbf{H}[h]$ defines default interaction in g, which is an interaction type, and a *handling environment*, $h = \{f_i : g_i\}_{i \in I}$, which maps failures f_i to a handling activity defined in g_i. Our design allows h to deal with different failures, with exactly one handler taking over once failures occur.

(C) In (A), if F is empty, we do not require the interaction to be enclosed in a try-handle term; otherwise, the interaction *must* appear within a try-handle term.

In the remaining syntax, we use ε for idle, and $g_1; g_2$ for sequential composition. A type variable is denoted by t, and a recursive type under an equi-recursive approach [18] is denoted by $\mu t.g$, assuming every t appearing in g is guarded by prefixes.

For brevity, our protocol types presentation omits parallel composition, thus we do not allow session interleaving or multi-threading at a local participant. Note that we can still implement two individual interactions running in parallel by implementing two disjoint groups of interacting participants who execute two respective protocols. We omit branching with multiple options of ongoing interactions, since the term $\mathbf{T}[p_1 \to p_2 : l_1, \ldots, l_n; g]\mathbf{H}[l_1 : g_1, \ldots, l_n : g_n]$ is able to encode the branching in multiparty session types [12] by using failures $l_1, \ldots, l_n$ as labels for branches. The full encoding is in Appendix C. As an example, consider how we can encode a classic session type definition of a branch:

$$p_1 \to p_2 : k\{l_j : g_j\}_{j \in J}$$

To do so, we simply leverage branching labels in try-handle terms:

$$\mathbf{T}[p_1 \to p_2 : l_1, \ldots, l_n]\mathbf{H}[l_1 : g_1, \ldots, l_n : g_n]$$

where $\{l_j\}_{j\in J} = \{l_1,...,l_n\}$ and $\{g_j\}_{j\in J} = \{g_1,...,g_n\}$, and channel k is implicitly identified by p_1 and p_2 when they join a session s (i.e., k can be simply encoded as a channel linking endpoint $s[p_1]$ to endpoint $s[p_2]$). Based on point (A), one element in $\{l_1,...,l_n\}$ will occur and p_2 will be informed of this occurrence. Since we can encode branching, hereafter we use "failures" for all possible "labels" used in a branching.

We leave unaffected participants to continue default actions regardless of the occurrence of failures; we do not inform them of the failure. Moreover, we do not have wellformedness constraints on the shape of interactions in branches as most multiparty session types and choreorgraphic programming related works require [3,4,5,13,16].

2.2 Local Types and Transformation

In order to achieve our desired properties we use an operation, called *transformation*, to synthesize a guidance to locally guide which participants need to coordinate with others once a failure occurs, or inversely to assert that none has occurred, before proceeding with the next action. The operation *transformation* includes the following steps:

1. **Generating a global structure** from a given protocol type and alpha renaming it.
2. **Projecting** the above structure to every participants to obtain *simple* local types, which are not yet sufficient for robust failure handling. The projection algorithm is similar to the mechanism in multiparty session types [1,12].
3. **Adding** the information of *need-(to)-be-informed* participants, who are those affected by or involved in handling failures, and synchronisation points to local types.

After these 3 steps, we obtain local types which are sufficient for our type system to ensure *robust failure handling*.

Figure 1 uses the example of $\mathbf{G}_{proxy}$ to demonstrate the operation of *transforming* a protocol type to each participants' local types, which are defined below:

Definition 1 (Local Types T).

$$\text{(Local Types) } \mathbf{T} ::= \mathbf{n} \mid try\{\mathbf{T},\ \mathbf{H}\} \mid \mathbf{n} \dashrightarrow \mathbf{T} \mid t \mid \mu t.\mathbf{T} \quad \text{(Handlers) } \mathbf{H} ::= \emptyset \mid \mathbf{H}, f : \mathbf{T}$$

$$\text{(Action)} \qquad \mathbf{n} ::= \varepsilon \mid \mathsf{end} \mid sn\langle p!\ \widetilde{S} \vee F, \widetilde{p}, \widetilde{p}\rangle \mid rn\langle p?\ \widetilde{S} \vee F\rangle \mid yield\langle F\rangle$$

A local type is either an action ($\mathbf{n}$), a try-handle type ($try\{\mathbf{T},\mathbf{H}\}$), a sequencing type ($\mathbf{n} \dashrightarrow \mathbf{T}$), a local type variable (t), or a recursive type ($\mu t.\mathbf{T}$). We use ε to type an idle action, while end types termination. A sending action, typed by $sn\langle p!\ \widetilde{S} \vee F, \widetilde{p}', \widetilde{p}''\rangle$, specifies a sending of normal content of type $\widetilde{S}$ to p or raising a failure in F. When a failure is raised, the sender also sends failure notifications to participants $\widetilde{p}'$. When normal content is sent, the sender also sends non-failure notifications to participants $\widetilde{p}''$. A receiving action, typed by $rn\langle p?\ \widetilde{S} \vee F\rangle$, specifies the reception of content of type $\widetilde{S}$ from p, who may raise a failure in F instead. An action yielding to the arrival of a non-failure notification informing that no failures in F occurred, is typed by $yield\langle F\rangle$. A handling environment in local types, denoted by $\mathbf{H} = \{f_i : \mathbf{T}_i\}_{i\in I}$, maps failures to corresponding local handling actions defined in local types.

In Figure 1, we firstly create a **global structure** for $\mathbf{G}_{proxy}$ by $Struct(\mathbf{G}_{proxy})$. Global structures, denoted by $\mathtt{T}$, consist of either a single interaction ($\mathtt{N}$), a try-handle structure

($try\{\mathtt{T},\mathtt{H}\}$) where $\mathtt{H}$ has a similar shape to handling environments in local types, a sequence ($\mathtt{N} \dashrightarrow \mathtt{T}$), or a recursive structure ($\mu t.\mathtt{T}$). We define $\mathtt{N}$ as either $p \to p : \widetilde{S} \vee F$ or ε or end. By defining $Struct(\text{single interaction}) = \mathtt{N}$, a try-handle structure is obtained by $Struct(\mathbf{T}[g]\mathbf{H}[\, f_1 : g_1, ..., f_n : g_n];\mathbf{G}) = try\{Struct(g;\mathbf{G}),\ f_1 : Struct(g_1;\mathbf{G}), ...,\ f_n : Struct(g_n;\mathbf{G})\}$, while a recursive structure is obtained by $Struct(\mu t.\mathbf{G}) = \mu t.Struct(\mathbf{G})$.

The *simple* local types are gained by *projecting* $\mathtt{T}$, created by $Struct(\mathbf{G})$, on each participants. The projection rules are defined below:

Definition 2 (Projecting $\mathtt{T}$ onto Endpoint p). Assume $\mathtt{T} = Struct(\mathbf{G})$ and $\mathtt{T}$ is alpha-renamed so that all failures in $\mathtt{T}$ are unique. Define $\upharpoonright(\mathtt{T},p)$ as generating a local type on p:

$$(1)\ \upharpoonright(\mathsf{end},p) = \mathsf{end} \quad (2)\ \upharpoonright(p_1 \to p_2 : \widetilde{S} \vee F, p) = \begin{cases} sn\langle p_2 !\ \widetilde{S} \vee F, \_, \_\rangle & \text{if } p = p_1 \neq p_2 \\ rn\langle p_1 ?\ \widetilde{S} \vee F\rangle & \text{if } p = p_2 \neq p_1 \\ \varepsilon_F & \text{otherwise} \end{cases}$$

$$(3)\ \upharpoonright(try\{\mathtt{T}, f_1 : \mathtt{T}_1, ..., f_n : \mathtt{T}_n\}, p) = try\{\upharpoonright(\mathtt{T},p), f_1 : \upharpoonright(\mathtt{T}_1,p), ..., f_n : \upharpoonright(\mathtt{T}_n,p)\}$$

$$(4)\ \upharpoonright(\mathtt{N} \dashrightarrow \mathtt{T}, p) = \upharpoonright(\mathtt{N},p) \dashrightarrow \upharpoonright(\mathtt{T},p) \quad (5)\ \upharpoonright(t,p) = t \quad (6)\ \upharpoonright(\mu t.\mathtt{T},p) = \mu t.\ \upharpoonright(\mathtt{T},p)$$

Others are undefined.

Rule (2) is for dually interacting participants. It introduces ε_F, which has equivalent meaning to ε (i.e. idle action) but is only used in *transformation* for adding synchronisation points. As we project an interaction $p_1 \to p_2 : \widetilde{S} \vee F$ with $p_1 \neq p_2$ onto p_1 (resp. p_2), we get an action $sn\langle p_2 !\ \widetilde{S} \vee F, \_, \_\rangle$ (resp. $rn\langle p_1 ?\ \widetilde{S} \vee F\rangle$). Note that the two slots $\_$ in the sending action are preserved for adding the need-be-informed participants as a failure occurs (the first slot), and those as no failures occur (the second slot). As we project the interaction to some participant who is *not* in the interaction, we get ε_F (idle action). The subscript F indicates that if p is affected by some failures in F a synchronisation point will be added at this position. Rule (3) simply projects every sub-structure in the try block and handlers onto the participant. Rule (4) sequences two local types projected from a global structure. Other rules are straightforward.

After projection, we add *need-be-informed participants* into the failure-raiser's sending actions (e.g., the one marked in green ring in Figure 1). We use $C(\mathtt{T},F)$ to get the set of need-be-informed participants regarding a unique F in a global structure $\mathtt{T}$. It is the least fixed point of $c(\mathtt{T},\mathtt{T},F,r)$, which recursively collects the need-be-informed participants regarding F based on $\mathtt{T}$. Since for every protocol the number of participants is finite, function c will converge to a fixed set of participants. The key calculation is done by the rule below

$$c(\mathtt{T},\mathtt{N},F,r) = \begin{cases} r \cup pid(\mathtt{N}) \cup C(\mathtt{T}, FSet(\mathtt{N})) & \text{if } (F \text{ appears before } \mathtt{N} \text{ in } \mathtt{T}) \wedge \\ & ((pid(\mathtt{N}) \cap r \neq \emptyset) \vee (FSet(\mathtt{N}) \neq \emptyset)) \\ r & \text{otherwise} \end{cases}$$

where we require the *initial* r to be the set containing the receiver of F-raising interaction and the participants involved in handling F in $\mathtt{T}$; later r acts as an accumulator collecting the participants causally related to the initial r. $\mathtt{N}$ is an interaction,

$pid(\mathtt{N})$ is the set of participants in $\mathtt{N}$, and $FSet(\mathtt{N})$ returns the failures tagged on $\mathtt{N}$. This rule says that if the interaction we are checking appears after the F-raising interaction, and some of its interacting participants are related to r or the interaction itself can raise another failure set (e.g. the interaction $\mathsf{Lib} \to \mathsf{Coord} : \mathsf{str} \vee \mathsf{Abort}$ in $\mathsf{RP} \to \mathsf{Coord} : \mathsf{str} \vee \mathsf{NoRes}; \mathsf{Lib} \to \mathsf{Coord} : \mathsf{str} \vee \mathsf{Abort}$ is related to $F = \{\mathsf{NoRes}\}$), then we collect its participants (i.e. $pid(\mathtt{N})$) and the need-be-informed participants with respect to the failures that can be raised by $\mathtt{N}$.

After adding those participants, we add synchronisation points $yield\langle F\rangle$ to the positions where a participant yields to the arrival of non-failure notifications (e.g. those marked in blue rings in Figure 1). The key rule is:

$$Sync(\mathtt{T},\mathbf{n},p) = \begin{cases} yield\langle FSet(\mathtt{N})\rangle \dashrightarrow \mathbf{n} & \text{if } (\mathbf{n} = \restriction(\mathtt{N},p)) \wedge (p \in C(\mathtt{T},FSet(\mathtt{N}))) \wedge p \neq snd(\mathtt{N}) \\ \mathbf{n} \dashrightarrow yield\langle FSet(\mathtt{N})\rangle & \text{if } (\mathbf{n} = \restriction(\mathtt{N},p)) \wedge (p \in C(\mathtt{T},FSet(\mathtt{N}))) \wedge p = snd(\mathtt{N}) \\ \mathbf{n} & \text{otherwise} \end{cases}$$

where $snd(\mathtt{N})$ is the sender for interaction $\mathtt{N}$. This rule says the followings: If p needs to be informed because of $F = FSet(\mathtt{N})$ (i.e. $p \in C(\mathtt{T},FSet(\mathtt{N}))$) then it must add a synchronisation point. If p's action regarding F is ε_F (e.g. in $\mathbf{G}_{coord}$ the participant Lib has $\varepsilon_{\mathsf{Abort}}$ by Definition 2), $yield\langle F\rangle$ is positioned ahead of p's action (e.g. a sending action of Lib to Coord specified in $\mathbf{G}_{coord}$), because p needs to wait for the notification regarding F before taking any action. If p is the receiver, we have $yield\langle F\rangle$ positioned before the receiving action because $yield\langle F\rangle$ is the point deciding whether the process will handle a failure regarding F or proceed. If p is the sender, we should have $yield\langle F\rangle$ positioned after the sending action, because as p is involved for some failure handling activity regarding F, it needs to first send out failure notifications then go back to execute the handling activity; otherwise the process will get stuck.

In Figure 1, green ring appears at Proxy's second action because, if a failure occurs, Proxy has to inform Log, SupServer, and Client about that failure. Blue rings appear at Client, Log, and SupServer's try blocks because they are involved in handling activity, and they can terminate only after getting the notifications that no failures occurred.

Overall, we define the operation of transformation as $Transform(\mathbf{G},p)$, which transforms $\mathbf{G}$ to a local type for p.

3 Processes for Decentralised Multiple-Failure-Handling

We abstract distributed systems as a finite set of processes communicating by outputting (resp. inputting) messages into (resp. from) the shared global queue asynchronously and concurrently. The semantics of the calculus is in the same style as that of the π-calculus and does not involve any centralised authority for specifying how messages are exchanged (**P4**). The shared queue is only *conceptually* global for convenience, and could be split into individual participant queues.

Syntax. In Figure 3, we define x as value variables, y as channel variables, a as shared names (e.g., names for services or protocol managers), and s as session names (i.e., session IDs), p as participant identifiers, and X as process variables. We use u for names

$$
\begin{array}{rl}
\multicolumn{2}{l}{\text{(Variables) } x,y \quad \text{(Shared Names) } a \quad \text{(Session Names) } s \quad \text{(Process IDs) } p \quad \text{(Process Variables) } X} \\
\text{(Values)} & v ::= \mathsf{unit} \mid \mathsf{false} \mid \mathsf{true} \mid 1 \mid 2 \mid ... \qquad \text{(Names) } u ::= a \mid s \\
\text{(Expressions)} & e ::= x \mid y \mid v \mid u \mid e+e \mid -e \mid e \vee e \mid ... \qquad \text{(Channels) } c ::= y \mid s[p] \\
\text{(Processes)} & P ::= \mathbf{0} \mid c!(p, \langle \tilde{e} \rangle^F) \mid c?(p, (\tilde{x})^F).P \mid c\ \mathsf{raise}(f)\ \mathsf{to}\ p \mid c \otimes F \\
& \quad \mid \mathsf{try}\{P\}\mathsf{h}\{H\} \mid P;P \mid X\langle \tilde{e}\ c \rangle \mid \mathsf{def}\ D\ \mathsf{in}\ P \mid \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ P \\
\text{(Handlers)} & H ::= \emptyset \mid H, f : P \qquad \text{(Declaration) } D ::= X(\tilde{x}\ c) = P \\
\text{(Messages)} & m ::= \varepsilon \mid \langle p,p, \langle \tilde{v} \rangle \rangle^F \mid \langle p,f \rangle \mid \langle\langle p,F \rangle\rangle \qquad \text{(Context) } \mathscr{E} ::= [\,] \mid \mathsf{try}\{\mathscr{E}\}\mathsf{h}\{H\} \mid \mathscr{E};P \\
\text{(Network)} & N ::= a[p](y).P \mid [P]_{\mathbf{T}} \mid s : q \mid N \| N \mid (\mathsf{new}\ s)N \quad \text{(Queues) } q ::= m \mid q \cdot m
\end{array}
$$

Fig. 3: Syntax for processes and networks.

and c for channels, which are either variables or a combination of s and p. The definition for expressions e is standard. We define the syntax of processes $(P,...)$ and that of networks $(N,...)$, which represent interactions of processes at runtime. Process $\mathbf{0}$ is inactive. Process $c!(p,\langle \tilde{e} \rangle^F)$ denotes an output, which may alternatively raise a failure f in F, sends a message with content $\tilde{e}$ to p via channel c; while $c?(p,(\tilde{x})^F).P$ denotes an input using c to receive a content from p, which may alternatively raise a failure from F. Every $\tilde{x}$ appearing in P is bound by the input prefix. When $F = \emptyset$, we omit F since the process will not raise/receive a failure. Process $c\ \mathsf{raise}(f)\ \mathsf{to}\ p$ raises f to p via channel c. Process $c \otimes F$ is guarded by $\otimes F$, a synchronisation point, yielding to non-failure notification for F. A try-handle process $\mathsf{try}\{P\}\mathsf{h}\{H\}$ executes P until a handler $f \in dom(H)$ is triggered, then the triggered handler takes over. A handling environment, denoted by H, maps failure names to handling processes. We write $P_1;P_2$ to represent a sequential composition where P_2 follows P_1. Process $\mathsf{def}\ D\ \mathsf{in}\ P$ defines a recursion, where declaration $D = (X(\tilde{x}\ c) = P)$ is a recursive call. The term $\mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ P$ is standard. We define evaluation context $\mathscr{E}$ over processes. It is either a hole, a context in a try-handle term, or a context sequencing next processes.

A network N is composed by linking points, denoted by $a[p](y).P$, and runtime processes, denoted by $[P]_{\mathbf{T}}$ with global transports (i.e., $s : q$) for proceeding communications in a private session (i.e., $(\mathsf{new}\ s)N$). Our framework asks a process to join one session at a time. A linking point $a[p](y).P$ is guarded by $a[p](y)$ for session initiation, where shared name a associates a service to a protocol type. $[P]_{\mathbf{T}}$ represents a runtime process which is guided by $\mathbf{T}$ for notifying need-be-informed participants.

A session queue, denoted by $s : q$, is a queue for messages floating in session s. Message $\langle p_1,p_2,\langle \tilde{v} \rangle \rangle^F$ carries content $\tilde{v}$, sent from p_1 to p_2 prone to failure $f \in F$. Message $\langle p,f \rangle$ (resp. $\langle\langle p,F \rangle\rangle$) carries a failure name f to indicate that *failure f occurred* (resp. a set of failures F to indicate that *no failures in F occurred*) to p. Conventionally we say $\langle\langle p,\emptyset \rangle\rangle = \varepsilon$. When session s is initiated for a network, a private (i.e., hidden) session is created, in which activities cannot be witnessed from the outside. We use structural congruence rules, defined by $\equiv$, which are standard according to the works of multiparty session types [1,12].

Operational semantics. Figure 4 gives the operational semantics for networks (i.e., runtime processes) through the reduction relation $N \to N$. We have added boxes to

$$\frac{a:\langle \mathbf{G}\rangle \quad \forall p\in\{1..n\}.\mathbf{T}_p = \mathit{Transform}(\mathbf{G},p) \quad s \text{ fresh}}{a[1](y_1).P_1 \parallel ... \parallel a[n](y_n).P_n \to (\mathsf{new}\ s)([P_1[s[1]/y_1]]_{\mathbf{T}_1} \parallel ... \parallel [P_n[s[n]/y_n]]_{\mathbf{T}_n} \parallel s:\varepsilon)} \ \textbf{[link]}$$

$$[s[p_1]?(p_2,(\tilde{x})^F).P]_{\mathbf{T}} \parallel s:\langle p_2,p_1,\langle \tilde{v}\rangle\rangle^F\cdot q \to [P[\tilde{v}/\tilde{x}]]_{\mathbf{T}} \parallel s:q \ \textbf{[rcv]}$$

$$[s[p_1]!(p_2,\langle \tilde{e}\rangle);P]_{\mathbf{T}} \parallel s:q \to [P]_{\mathbf{T}} \parallel s:q\cdot\langle p_1,p_2,\langle \tilde{v}\rangle\rangle \quad \tilde{e}\Downarrow\tilde{v} \ \textbf{[snd]}$$

$$\frac{\tilde{e}\Downarrow\tilde{v} \quad \mathit{node}(\mathbf{T},F) = sn\langle p_2!\,\widetilde{S}\vee F,\widetilde{p},\{p_1',...,p_n'\}\rangle \quad F\neq\emptyset}{\begin{array}{l}[\mathsf{try}\{s[p_1]!(p_2,\langle \tilde{e}\rangle^F);P\}\mathsf{h}\{H\}]_{\mathbf{T}} \parallel s:q \\ \quad\to [\mathsf{try}\{P\}\mathsf{h}\{H\}]_{\mathbf{T}} \parallel s:q\cdot\langle p_1,p_2,\langle \tilde{v}\rangle\rangle^F\cdot\langle\langle p_1',F\rangle\rangle...\langle\langle p_n',F\rangle\rangle\end{array}} \ \textbf{[sndF]}$$

$$\frac{\mathit{node}(\mathbf{T},F) = sn\langle p_2!\,\widetilde{S}\vee F,\{p_2,p_1',...,p_n'\},\widetilde{p}\rangle \quad f\in F}{\begin{array}{l}[\mathsf{try}\{s[p_1]\ \mathsf{raise}(f)\ \mathsf{to}\ p_2;P\}\mathsf{h}\{H\}]_{\mathbf{T}} \parallel s:q \\ \quad\to [\mathsf{try}\{P\}\mathsf{h}\{H\}]_{\mathbf{T}} \parallel s:q\cdot\langle p_2,f\rangle\cdot\langle p_1',f\rangle...\langle p_n',f\rangle\end{array}} \ \textbf{[thwf]}$$

$$\frac{\begin{array}{c}\mathit{act}(P) = s[p] \quad [P]_{\mathbf{T}} \parallel s:q \to [P']_{\mathbf{T}} \parallel s:q' \\ q = \langle p',f'\rangle\cdot q' \Rightarrow (p'\neq p)\vee(f'\notin \mathit{dom}(H))\end{array}}{[\mathsf{try}\{P\}\mathsf{h}\{H\};P'']_{\mathbf{T}} \parallel s:q \to [\mathsf{try}\{P'\}\mathsf{h}\{H\};P'']_{\mathbf{T}} \parallel s:q'} \ \textbf{[try]}$$

$$\frac{f\in \mathit{dom}(H)\cap F}{[\mathsf{try}\{\mathscr{E}[s[p]\otimes F]\}\mathsf{h}\{H\};P']_{\mathbf{T}} \parallel s:\langle p,f\rangle\cdot q \to [H(f);P']_{\mathbf{T}} \parallel s:q} \ \textbf{[hdl]}$$

$$\frac{F'\subseteq F}{[s[p]\otimes F';P]_{\mathbf{T}} \parallel s:\langle\langle p,F\rangle\rangle\cdot q \to [P]_{\mathbf{T}} \parallel s:q} \ \textbf{[sync-done]}$$

$$\frac{F''=F'\setminus F\neq\emptyset}{[s[p]\otimes F';P]_{\mathbf{T}} \parallel s:\langle\langle p,F\rangle\rangle\cdot q \to [s[p]\otimes F'';P]_{\mathbf{T}} \parallel s:q} \ \textbf{[sync]}$$

$$[\mathsf{try}\{v\}\mathsf{h}\{H\}]_{\mathbf{T}} \to [\mathbf{0}]_{\mathbf{T}} \ \textbf{[try-end]}$$

$$[\mathsf{if\ true\ then}\ P_1\ \mathsf{else}\ P_2]_{\mathbf{T}} \to [P_1]_{\mathbf{T}} \quad [\mathsf{if\ false\ then}\ P_1\ \mathsf{else}\ P_2]_{\mathbf{T}} \to [P_2]_{\mathbf{T}} \ \textbf{[if]}$$

$$\frac{[P_1]_{\mathbf{T}} \parallel s:q \to [P_2]_{\mathbf{T}} \parallel s:q'}{[P_1;P]_{\mathbf{T}} \parallel s:q \to [P_2;P]_{\mathbf{T}} \parallel s:q'} \ \textbf{[seq]} \qquad \frac{\tilde{e}\Downarrow\tilde{v} \quad X(\tilde{x}\ c) = P\in D}{[\mathsf{def}\ D\ \mathsf{in}\ X\langle\tilde{e}\ c\rangle]_{\mathbf{T}} \to [\mathsf{def}\ D\ \mathsf{in}\ (P[\tilde{v}/\tilde{x}])]_{\mathbf{T}}} \ \textbf{[call]}$$

$$\frac{N_1\to N_2}{N_1 \parallel N \to N_2 \parallel N} \ \textbf{[net]} \qquad \frac{[P_1]_{\mathbf{T}} \parallel s:q \to [P_2]_{\mathbf{T}} \parallel s:q'}{[\mathsf{def}\ D\ \mathsf{in}\ P_1]_{\mathbf{T}} \parallel s:q \to [\mathsf{def}\ D\ \mathsf{in}\ P_2]_{\mathbf{T}} \parallel s:q'} \ \textbf{[defin]}$$

$$\frac{N_1\equiv N_2\to N_3\equiv N_4}{N_1\to N_4} \ \textbf{[str]} \qquad \frac{N_1\to N_2}{(\mathsf{new}\ s)N_1\to(\mathsf{new}\ s)N_2} \ \textbf{[new]}$$

Fig. 4: Reduction rules for networks (i.e., runtime processes).

those rules which differ from standard session type definitions. In rule **[link]**, a session is generated with a fresh name s through shared name a obeying protocol type $\mathbf{G}$. This indicates that all processes in the new session s will obey to the behaviours defined in $\mathbf{G}$. At the same time, a global queue $s:\varepsilon$ is generated, and the local process associated with p replaces y_p with $s[p]$; a local type $\mathbf{T}_p$ is generated by $\mathit{Transform}(\mathbf{G},p)$ to guide the local process associated with p for propagating notifications. Note that, as we enclose a local process with $\mathbf{T}$, together they become an element of a network. $\mathbf{T}$ is merely a local type and the reduction of the network does not change $\mathbf{T}$.

Rule **[rcv]** states that, in s, a process associated with p_1 is able to receive a value $\tilde{v}$ from participant associated with p_2 and message $\langle p_2,p_1,\langle\tilde{v}\rangle\rangle^F$ is on the top of q. Then

$\tilde{v}$ will replace the free occurrences of $\tilde{x}$ in P. The shape of $s[p_1]?(p_2,(\tilde{x})^F)$ indicates that its dual action may send it a failure from F; in other words, if $F \neq \emptyset$, a process should be structured by a try-handle term for possible failure handling. Rule [snd] is dual to [rcv]. We define $node(\mathbf{T},F)$ as a function returning an action tagged with F in a local type $\mathbf{T}$. Rule [sndF] states that, if there is an action in $\mathbf{T}$ matching $s[p_1]!(p_2,\langle\tilde{e}\rangle^F)$ and $\tilde{e} \Downarrow \tilde{v}$, then the process associated with p_1 in s is allowed to send a message with content $\tilde{v}$ to p_2 and non-failure notifications $\langle\langle p'_1,F\rangle\rangle ... \langle\langle p'_n,F\rangle\rangle$. Note that, non failure notifications are automatically generated at runtime. If a process follows the guidance of the attached $\mathbf{T}$, since $\mathbf{T}$ is alpha-renamed, every failure raised by the process is unique. Similarly, [thwf] states for a process associated with p_1 in s, to raise f to p_2 and other affected ones, $p'_1,..,p'_n$. Very importantly, in [sndF] and [thwf], q has no failure notification to trigger H because, as a failure-raising interaction is ready to fire (i.e. its sender is about to send), it implies that, globally, either this interaction is the first failure-raising interaction in s (thus no failure yet occurs in s), or its previous interactions did not raise a failure in $dom(H)$ (thus by **P2**, this interaction is able to raise a failure in $dom(H)$, and no failures in $dom(H)$ yet occurs in s). For convenience, we use act to extract the channel that a process or the set of handlers is acting on, i.e. $act(P) = s[p]$ says P is acting on channel $s[p]$, and $act(H) = act(H(f))$ for every $f \in dom(H)$.

In [try], if the H in a try-handle process associated with p in s will not be triggered by the top message in $s:q$, then the process in the try block will take action according to the process's interaction with the queue. In [hdl], as f arrives to a try-handle process associated with p in s whose try block is yielding to non-failure notification for F and H is able to handle f, the handling process $H(f)$ takes over. Due to asynchrony, other processes' handlers for f may become active before this process. Thus some messages in q may be sent from other processes' handlers of f for P. Note that none of the messages in q are for $\mathscr{E}$ because, all default sending actions in other processes are also guarded by synchronisation points.

Synchronisation either proceeds with [sync-done], where F is sufficient to remove F', or with [sync], where F is included in F' carried in the notification. For the former, some processes in the failure-handling activity only take care of partial failures in F, i.e. F', when they receive F, to ensure that no failures in F occurred. For the latter, further synchronisation is required by $F'' = (F' \setminus F) \neq \emptyset$. In [try-end], since we have added sufficient synchronisation points to guard processes who must yield to non-failure notifications, when a network reaches $[\mathsf{try}\{v\}\mathsf{h}\{H\}]_{\mathbf{T}}$, it is safe to be inactive because no more failure notifications will occur. In other rules, the operations enclosed in $\mathbf{T}$ are standard according to the works of multiparty session types [1,12].

4 Typing Local Processes

This section introduces rules to type user-defined processes. Based on the multiparty session types [1,12], type environments and typing rules for processes are given in Figure 5. A shared environment Γ is a finite mapping from variables to sorts and from process variables to local types; a session environment Δ is a finite mapping from session channels to local types. $\Gamma, x:S$ means that x does not occur in Γ, so does $\Gamma, X:(\tilde{x}\ \mathbf{T})$ and $\Delta, c:\mathbf{T}$. We assume that expressions are typed by sorts. $\Gamma \vdash e:S$ is the typing judg-

Γ (Shared Environments) $::= \emptyset \mid \Gamma, x : S \mid \Gamma, X : (\tilde{x}\ \mathbf{T})$ $\quad \Delta$ (Session Environments) $::= \emptyset \mid \Delta, c : \mathbf{T}$

$$\frac{\Delta \text{ end only}}{\Gamma \vdash \mathbf{0} \triangleright \Delta}\ \text{[T-0]} \qquad \frac{\forall i \in \{1,2\}.\Gamma \vdash P_i \triangleright \Delta_i}{\Gamma \vdash P_1; P_2 \triangleright \Delta_1 \circ \Delta_2}\ \text{[T-seq]}$$

$$\frac{\Gamma, \widetilde{x} : \widetilde{S} \vdash P \triangleright s[p_1] : \mathbf{T}}{\Gamma \vdash s[p_1]?(p_2, (\widetilde{x})^F).P \triangleright s[p_1] : rn\langle p_2?\ \widetilde{S} \vee F\rangle \dashrightarrow \mathbf{T}}\ \text{[T-rcv]}$$

$$\frac{\widetilde{S} \neq \emptyset \quad \Gamma \vdash \tilde{e} : \widetilde{S} \quad \Gamma \vdash P \triangleright s[p_1] : \mathbf{T}}{\Gamma \vdash s[p_1]!(p_2, \langle \tilde{e} \rangle^F); P \triangleright s[p_1] : sn\langle p_2!\ \widetilde{S} \vee F, \_, \_\rangle \dashrightarrow \mathbf{T}}\ \boxed{\text{[T-snd]}}$$

$$\frac{f \in F \quad \Gamma \vdash P \triangleright s[p_1] : \mathbf{T}}{\Gamma \vdash s[p_1]\ \mathsf{raise}(f)\ \mathsf{to}\ p_2; P \triangleright s[p_1] : sn\langle p_2!\ \widetilde{S} \vee F, \_, \_\rangle \dashrightarrow \mathbf{T}}\ \boxed{\text{[T-thwf]}}$$

$$\frac{\begin{array}{c} dom(H) = dom(\mathbf{H}) \quad \forall f \in dom(H).f \notin P' \\ \Gamma \vdash P; P' \triangleright act(H) : \mathbf{T} \quad \forall f \in dom(H).\Gamma \vdash H(f); P' \triangleright act(H) : H(f) \end{array}}{\Gamma \vdash \mathsf{try}\{P\}\mathsf{h}\{H\}; P' \triangleright act(H) : try\{\mathbf{T}, \mathbf{H}\}}\ \boxed{\text{[T-try]}}$$

$$\frac{\Gamma \vdash P \triangleright c : \mathbf{T}}{\Gamma \vdash c \otimes F; P \triangleright c : yield\langle F\rangle \dashrightarrow \mathbf{T}}\ \boxed{\text{[T-sync]}} \qquad \frac{\Gamma \vdash e : \mathsf{bool} \quad i = 1,2.\ \Gamma \vdash P_i; P \triangleright \Delta}{\Gamma \vdash \mathsf{if}\ e\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2; P \triangleright \Delta}\ \text{[T-if]}$$

$$\Gamma, X : (\widetilde{S}\ \mathbf{T}) \vdash X\langle \tilde{e}\ c\rangle \triangleright c : \mathbf{T}\ \text{[T-var]} \qquad \frac{\Gamma, \tilde{x} : \widetilde{S}, X : (\widetilde{S}\ \mathbf{T}) \vdash P \triangleright c : \mathbf{T} \quad \Gamma, X : (\widetilde{S}\ \mathbf{T}) \vdash P' \triangleright \Delta}{\Gamma \vdash \mathsf{def}\ X(\tilde{x}\ c) = P\ \mathsf{in}\ P' \triangleright \Delta}\ \text{[T-rec]}$$

Fig. 5: Typing rules for processes.

ment for expressions, whose typing rules are standard. The typing judgment $\Gamma \vdash P \triangleright \Delta$ for local processes reads as "Γ proves that P complies with abstract specification Δ".

Rule [T-0] states that an idle process is typed by end-only Δ, which means $\forall c \in dom(\Delta)$, $\Delta(c) = \mathsf{end}$. Rule [T-seq] types sequential composition by sequencing P_2's action in Δ_2 after P_1's action in Δ_1 as long as P_1 and P_2 are acting on the same channel. We define $\Delta_1 \circ \Delta_2$ as the one defined in the multiparty session types extended with failure-handling ability [3]. Rule [T-rcv] specifies that $s[p_1]?(p_2, (\widetilde{x})^F).P$ is valid as it corresponds to local type $rn\langle p_2?\ \widetilde{S} \vee F\rangle \dashrightarrow \mathbf{T}$ as long as P, associated with p_1 in session s, is well-typed by $\mathbf{T}$ under an environment which knows $\tilde{x} : \widetilde{S}$. In [T-snd] and [T-thwf], since the slots are not related to typing, their contents are omitted. Rules [T-snd] and [T-thwf] share the same action for typing because $sn\langle p_2!\ \widetilde{S} \vee F, \_, \_\rangle$ specifies two possible actions: a sending action $s[p_1]!(p_2, \langle \tilde{e} \rangle^F)$ in which $\tilde{e}$ must have type $\widetilde{S}$, and action $s[p_1]\ \mathsf{raise}(f)\ \mathsf{to}\ p_2$ in which f must be in F. Then the continuing process P is typed by the following $\mathbf{T}$.

For typing handling activities, rule [T-try] types a try-handle term if its default action (i.e., P) with its following process is well-typed, and those in handlers with their follow-up processes are all well-typed. We require the following process P' should not contain any failure appearing in H because a failure cannot be handled in a following process but can be handled by an outer handler in a nested try-handle term (e.g. for $\mathsf{try}\{\mathsf{try}\{P\}\mathsf{h}\{H\}\}\mathsf{h}\{H'\}$, a failure in P can be handled by H'). Since P and any processes in H are acting on the same channel and $act(H)$ represents the channel that every processes in H is acting on, we use $act(H)$ to get the channel in order to type $\mathsf{try}\{\mathbf{0}\}\mathsf{h}\{H\}$. Recall Figure 1 and projection rules defined in Definition 2, for local

types the sequencing action is linked at every leaf in a try-handle term; in other words, the type of P' is attached to the type of P and also to every handler in $\mathbf{H}$. Therefore, as we type a try-handle term, we also consider its following process.

Rule [T-sync] specifies that process $c \otimes F; P$ is well-typed if the local type for c has synchronisation point $yield\langle F \rangle$ and P is well-typed w.r.t. $\mathbf{T}$. The algorithm for adding synchronisation points (introduced in Section 2) automatically places the synchronisation points in local types and ensures that once a failure is raised, other possible failure-raising actions must not fire. Since the operational semantics defined in Figure 4 only deliver notifications regarding F to need-be-informed participants and only one failure in F can be raised, our type system ensures only one failure in a try-handle term is handled and all participants affected by F have consistent failure handling activities.

Rule [T-var] types a local process variable, and rule [T-rec] types a recursion with Δ, where the recursive call $X(\tilde{x}, c) = P$ is typed by $c : \mathbf{T}$, indicating that P follows behaviour $\mathbf{T}$ at c. Others are standard according to the works of multiparty session types [1,12].

5 Typing the Network

Ultimately our framework needs to ensure that the network is coherent. *Coherence*, according to the works of multiparty session types [1,12], describes an environment where all *interactions* are complying with the guidance of some $\mathbf{G}$, such that the behaviour of every participant in Δ, say $s[p]$, obeys to $Transform(\mathbf{G}, p)$, which denotes a local type. To reason about coherence of default and handling interactions in a session, we statically type check the interactions by modeling the outputs and inputs among local processes and the shared global queue.

$$\begin{array}{rl} \mathbf{M} \text{ (Message Types)} & ::= \varepsilon \mid \langle p_1, p_2, \langle \widetilde{S} \rangle \rangle^F \mid \langle p, f \rangle \mid \langle\langle p, F \rangle\rangle \\ \mathbf{q} \text{ (Queue Type)} & ::= \mathbf{M} \mid \mathbf{q} \cdot \mathbf{M} \\ \Delta \text{ (Extended Session Environments)} & ::= \ldots \mid \Delta, s : \mathbf{q} \end{array}$$

$$\Gamma \vdash s : \varepsilon \triangleright \{s : \varepsilon\} \ \text{[T-m}\varepsilon\text{]} \qquad \frac{\Gamma \vdash \widetilde{v} : \widetilde{S} \quad \Gamma \vdash s : q \triangleright \{s : \mathbf{q}\}}{\Gamma \vdash s : q \cdot \langle p_1, p_2, \langle \widetilde{v} \rangle \rangle^F \triangleright \{s : \mathbf{q} \cdot \langle p_1, p_2, \langle \widetilde{S} \rangle \rangle^F\}} \ \text{[T-m]}$$

$$\frac{\Gamma \vdash s : q \triangleright \{s : \mathbf{q}\}}{\Gamma \vdash s : q \cdot \langle p, f \rangle \triangleright \{s : \mathbf{q} \cdot \langle p, f \rangle\}} \ \text{[T-mf]} \qquad \frac{\Gamma \vdash s : q \triangleright \{s : \mathbf{q}\}}{\Gamma \vdash s : q \cdot \langle\langle p, F \rangle\rangle \triangleright \{s : \mathbf{q} \cdot \langle\langle p, F \rangle\rangle\}} \ \text{[T-mF]}$$

$$\frac{\Gamma \vdash a : \langle \mathbf{G} \rangle \qquad \Gamma \vdash P \triangleright y : Transform(\mathbf{G}, p)}{\Gamma \vdash a[p](y).P \triangleright \emptyset} \ \boxed{\text{[T-link]}}$$

$$\frac{\Gamma \vdash P \triangleright \Delta \quad act(P) = s[p] \quad \exists \mathbf{G}. \text{ s.t. } \mathbf{T} = Transform(\mathbf{G}, p) \quad \mathbf{T} \text{ contains } \Delta(s[p])}{\Gamma \vdash [P]_{\mathbf{T}} \triangleright \Delta} \ \boxed{\text{[T-guide]}}$$

$$\frac{\forall i \in 1, 2. \Gamma \vdash N_i \triangleright \Delta_i \quad dom(\Delta_1) \cap dom(\Delta_2) = \emptyset}{\Gamma \vdash N_1 \parallel N_2 \triangleright \Delta_1, \Delta_2} \ \text{[T-net]} \qquad \frac{\Gamma \vdash N \triangleright \Delta \quad \Delta\langle s \rangle \text{ coherent}}{\Gamma \vdash (\mathsf{new}\ s) N \triangleright \Delta \setminus \Delta\langle s \rangle} \ \text{[T-new]}$$

Fig. 6: Typing rules for networks.

The typing rules for networks are defined in Figure 6 by extending the session environments such as to map queues to queue types. A queue type, denoted by $\mathbf{q}$, is composed by message types, which are typed by their contents or shapes: Rule [T-mε] types an empty queue, while rule [T-m] types a message carrying a value under the assumption that $\Gamma \vdash \widetilde{v} : \widetilde{S}$ and the following queue is well-typed; rule [T-mf] types $\langle p,f \rangle$ by message type $\langle p,f \rangle$, while rule [T-mF] types $\langle\langle p,F \rangle\rangle$ by message type $\langle\langle p,F \rangle\rangle$. Rule [T-link] types a linking point $a[p](y).P$ by assuming that a provides a behaviour pattern defined in $\mathbf{G}$. For guiding P associated with p, [T-link] uses local type $\mathbf{T}$ generated by $\mathit{Transform}(\mathbf{G},p)$ to type P acting on channel y. Rule [T-guide] states that $[P]_{\mathbf{T}}$ is well-typed by Δ if P is well-typed by Δ, and $\mathbf{T}$, gained by some $\mathbf{G}$, contains the type which types P acting on channel $s[p]$. Note that, by rule [link] (see Figure 4), $[P]_{\mathbf{T}}$ is created after linking and $\mathbf{T}$ is not changed after any reduction; thus $\mathbf{G}$ in rule [T-guide] comes from rule [T-link]. Rule [T-net] ensures the parallel composition of two networks if each of them is well-typed and they do not share a common channel (i.e., $\mathit{dom}(\Delta_1) \cap \mathit{dom}(\Delta_2) = \emptyset$). The composed network exhibits the union of the session environments. Rule [T-new] types hiding (i.e., $(\mathsf{new}\ s)N$) when the session environment of networks under s, denoted by

$$\Delta\langle s \rangle \stackrel{\text{def}}{=} \{s'[p] : \mathbf{T} \mid s'[p] \in \mathit{dom}(\Delta), s' = s\} \cup \{s : \mathbf{q}\},$$

is coherent:

Definition 3 (Coherence). We say $\Delta\langle s \rangle$ is coherent if there exists $\mathbf{G}$ such that $\mathit{pid}(\mathbf{G}) = \{p \mid s'[p] \in \mathit{dom}(\Delta), s' = s\}$ and either (1) $\forall s[p] \in \mathit{dom}(\Delta\langle s \rangle)$ we have $\mathit{Transform}(\mathbf{G},p)$ is equal to the type of $\Delta(s[p])$ after $s[p]$ absorbs all messages heading to it; or (2) there exists $\Delta' \subset \Delta$ such that $\forall s[p] \in \mathit{dom}(\Delta\langle s \rangle \setminus \Delta'\langle s \rangle)$ we have that $\mathit{Transform}(\mathbf{G},p)$ is equal to the type of $\Delta(s[p])$ after $s[p]$ absorbs all messages heading to it, and $\Delta'\langle s \rangle$ is coherent.

Note that due to asynchrony, after a sender takes action, the type of the sender and its receiver may be temporarily incoherent if the sender has moved forward and the output is still in the global queue. Therefore, coherence holds only after a receiver has absorbed all messages heading to it.

As we aim to handle partial failure(s), either (1) no failures occurred such that there exists $\mathbf{G}$ defining interactions for every $s[p]$ in $\Delta\langle s \rangle$, or (2) a failure occurs such that the need-be-informed participants, who are in $\Delta'\langle s \rangle$, are handling that failure in a coherent way, and other unaffected ones, who are in $\Delta\langle s \rangle \setminus \Delta'\langle s \rangle$, still follow the behaviour defined in $\mathbf{G}$.

6 Properties

We prove that our typing discipline ensures the properties of *safety* and *progress*. All auxiliary definitions and proofs are in Appendix B. The property of safety is defined by *subject reduction* and *communication safety*. Firstly we define $\Delta \rightharpoonup \Delta'$ as reductions of session environments. Intuitively, the reductions correspond closely to the operational semantics defined in Figure 4. Subject reduction states that a well-typed network (resp. coherent session environment) is always well-typed (resp. coherent) after reduction:

Theorem 1 (Subject Congruence and Reduction).

1. (subject congruence) $\Gamma \vdash N \triangleright \Delta$ and $N \equiv N'$ imply that $\Gamma \vdash N' \triangleright \Delta$.
2. (subject reduction) $\Gamma \vdash N \triangleright \Delta$ with Δ coherent and $N \to N'$ imply that $\Gamma \vdash N' \triangleright \Delta'$ such that $\Delta \rightharpoonup \Delta'$ or $\Delta \equiv \Delta'$ and Δ' is coherent.

According to the definition of communication safety in the works of multiparty session types [1,12], it is a corollary of Theorem 1. Note that, since our calculus is based on the work of Bettini *et al.* [1], global linearity-check is not needed. For convenience, we define here contexts on networks:

$$\mathscr{C} ::= [\,] \mid \mathscr{C} \parallel N \mid N \parallel \mathscr{C} \mid (\mathsf{new}\ s)\mathscr{C}$$

Corollary 1 (Communication Safety). Suppose $\Gamma \vdash N \triangleright \Delta$ and Δ is coherent. Let $N_1 = \mathscr{C}_1[s : q \cdot \langle p_2, p_1, \langle \tilde{v} \rangle \rangle^F \cdot q']$ and $N_2 = \mathscr{C}_2[s : q \cdot \langle p_1, f \rangle \cdot q']$ and $N_3 = \mathscr{C}_3[s : q \cdot \langle\langle p_1, F \rangle\rangle \cdot q']$ and no messages in q is sending to p_1.

1. If $N = \mathscr{C}[\mathscr{E}[s[p_1]?(p_2, (\tilde{x})^F).P]_{\mathbf{T}}]$, then $N \equiv N_1$ or $N \to^* N_1$.
2. If $N = \mathscr{C}[\mathscr{E}[\mathsf{try}\{s[p_1] \otimes F'; P\}\mathsf{h}\{H\}]_{\mathbf{T}}]$ and $F' \subseteq F \neq \emptyset$, then either (a) $N \equiv N_2$ or $N \to^* N_2$ or (b) $N \equiv N_3$ or $N \to^* N_3$.
3. If $N = \mathscr{C}[\mathscr{E}[\mathsf{try}\{v\}\mathsf{h}\{H\}]_{\mathbf{T}}]$ and $f \in dom(H)$ and process $H(f)$ is acting on $s[p_1]$, then $N \not\equiv N_2$ and $N \not\to^* N_2$.

This corollary states that our system is *free from deadlock* and *starvation*: if there is a receiving action in N, then N is either structurally congruent to the network which contains the message for input, or N will reduce to such a network. We state that $[\mathsf{try}\{v\}\mathsf{h}\{H\}]_{\mathbf{T}}$ is safe to become idle by proving that no $f \in dom(H)$ is heading to it (Case 3).

Corollary 1 provides the means to prove that our system *never gets stuck and is free from orphan messages* (*property of progress*):

Theorem 2 (Progress). $\Gamma \vdash N \triangleright \Delta$ with Δ coherent and $N \to N'$ imply that N' is communication safe or $N' = \mathbf{0} \parallel s : \varepsilon$.

This theorem states that every interaction in a well-typed network is a safe interaction and reducible until the whole network terminates without any message left.

7 Related Works

Failure handling has been addressed in several process calculi and communication-centered programming languages. For instance, the conversation calculus [20] models exception handling in abstract service-based systems with message-passing based communication. It studies expressiveness and behaviour theory of bisimilarity rather than theory of types. Colombo and Pace [8] investigate several different process calculi for failure-recovery within long-running transactions. They give insight regarding the application of these failure-recovery formalisms in practice via comparing the design choices and formal notions of correctness properties. Both works do not provide a type system to statically type check local implementations.

Previous works for failure handling with type systems [3,4,5,13] extend the theory of session types to specify error handling under asynchronous interactions. These works

do not capture handling of partial failures and the scenarios which exhibit the properties **P1** to **P4**. They may be able to encode multiple possible failures at the interaction level (**P1**), for example, by (i) explicitly using a labeled branching inside the failure handler, or (ii) piggybacking a label with the failure notification ("multiplexing"). However, (i) implies double communication and synchronisation (once for the failure notification, then for the branch) and (ii) implies that either the well-formedness constraints on the shape of interactions in handlers are needed or any participants related to a failure handling activity should be informed as a failure occurs in order to know how to proceed. Our approach is different since we do not have such constrains and we do not inform the unaffected participants. Moreover, while the termination of try-handle terms in those works demands an agreement of all participants, ours allows local try-handle terms to terminate since we have locally added synchronisation points by *transformation* (see Section 2.2). Our approach can encode the global types for exception-handling proposed in the work by Capecchi *et al.* [3], which is the closest related work (and other related ones have similar try-handle syntax). The formal encoding can be found in Optional Appendix C.

Collet and Van Roy [7] informally present a distributed programming model of Oz for asynchronous failure handling and focus on programming applications in a distributed manner. Jaksic and Padovani [14] study a type theory for error handling for copy-less messaging and memory sharing to prevent memory leaks/faults through typing of exchange heaps. Lanese *et al.* [11,15] formalise a feature which can dynamically install fault and compensation handlers at execution time in an orchestration programming style. They investigate the interplay between fault handling and the request-response pattern. In contrast, our framework statically defines the handlers for non-trivial failure handling, which can only be done with a global perspective.

8 Concluding Remarks

Protocol types enable the design of protocols in an intuitive manner, and statically type check multiple failure-handling processes in a transparent way. Our type discipline exhibits the desirable properties of **P1(alignment)**, **P2(precision)**, **P3(causality)**, and **P4(decentralisation)** for robust failure handling, and ensures fundamental properties of safety and progress. We are currently implementing the proposed framework and are extending it to support system-induced failures as opposed to application-specific ones focused on in this paper, in addition to parameterisation and dynamic multiroles.

References

1. L. Bettini, M. Coppo, L. D'Antoni, M. D. Luca, M. Dezani-Ciancaglini, and N. Yoshida. Global progress in dynamically interleaved multiparty sessions. In *CONCUR'08*, volume 5201 of *LNCS*, pages 418–433. Springer, 2008.
2. L. Caires and H. T. Vieira. Conversation types. In *ESOP '09*, volume 5502 of *LNCS*, pages 285–300. Springer, 2009.
3. S. Capecchi, E. Giachino, and N. Yoshida. Global escape in multiparty sessions. *MSCS*, 29:1–50, 2015.

4. M. Carbone, K. Honda, and N. Yoshida. Structured interactional exceptions in session types. In *CONCUR'08*, volume 5201 of *LNCS*, pages 402–417. Springer, 2008.
5. M. Carbone, N. Yoshida, and K. Honda. Asynchronous session types: Exceptions and multiparty interactions. In *SFM'09*, volume 5569 of *LNCS*, pages 187–212. Springer, 2009.
6. T. Chen, M. Dezani-Ciancaglini, and N. Yoshida. On the preciseness of subtyping in session types. In *Proceedings of the 16th International Symposium on Principles and Practice of Declarative Programming, Kent, Canterbury, United Kingdom, September 8-10, 2014*, pages 135–146. ACM, 2014.
7. R. Collet and P. Van Roy. Advanced topics in exception handling techniques. chapter Failure Handling in a Network-transparent Distributed Programming Language, pages 121–140. Springer-Verlag, Berlin, Heidelberg, 2006.
8. C. Colombo and G. J. Pace. Recovery within long-running transactions. *ACM Comput. Surv.*, 45(3):28:1–28:35, July 2013.
9. P.-M. Deniélou and N. Yoshida. Dynamic multirole session types. In *POPL'11*, pages 435–446, 2011.
10. F. C. Gärtner. Fundamentals of fault-tolerant distributed computing in asynchronous environments. *ACM Comput. Surv.*, 31(1):1–26, Mar. 1999.
11. C. Guidi, I. Lanese, F. Montesi, and G. Zavattaro. On the interplay between fault handling and request-response service invocations. In *Application of Concurrency to System Design, 2008. ACSD 2008. 8th International Conference on*, pages 190–198, June 2008.
12. K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. In *POPL ' 08*, pages 273–284. ACM, 2008.
13. R. Hu, R. Neykova, N. Yoshida, R. Demangeon, and K. Honda. Practical interruptible conversations - distributed dynamic verification with session types and python. In *RV '13*, volume 8174 of *LNCS*, pages 130–148. Springer, 2013.
14. S. Jakšić and L. Padovani. Exception Handling for Copyless Messaging. *Science of Computer Programming*, 84:22–51, 2014.
15. I. Lanese and F. Montesi. Error handling: From theory to practice. In *ISoLA'10*, volume 6416 of *LNCS*, pages 66–81. Springer, 2010.
16. I. Lanese, F. Montesi, and G. Zavattaro. Amending choreographies. In *WWV'13*, volume 123 of *EPTCS*, pages 34–48, 2013.
17. D. Mostrous. *Session Types in Concurrent Calculi: Higher-Order Processes and Objects*. PhD thesis, Imperial College London, 2009.
18. B. C. Pierce. *Types and programming languages*. MIT Press, 2002.
19. K. Takeuchi, K. Honda, and M. Kubo. An Interaction-based Language and its Typing System. In *PARLE'94*, volume 817 of *LNCS*, pages 398–413, 1994.
20. H. T. Vieira, L. Caires, and J. C. Seco. The conversation calculus: A model of service-oriented computation. In *ESOP ' 08*, volume 4960 of *LNCS*, pages 269–283. Springer, 2008.
21. N. Yoshida and V. T. Vasconcelos. Language primitives and type discipline for structured communication-based programming revisited: Two systems for higher-order session communication. *Electr. Notes Theor. Comput. Sci.*, 171(4):73–93, 2007.

A Optional Appendix for Sections 2 to 6

This appendix provides formal and auxiliary definitions for Sections 2, 3, 4, 5, and 6.

A.1 Formal Treatment of Section 2: Failure-handling Protocol Types, Global Structures, and Transformation

This section gives a formal definition for protocol types which are able to describe robust failure handling and full formal definitions for global structures and *transformation*, which are mentioned but not formally represented in Section 2.2. Particularly, the part for *transformation* includes formal definitions for calculating what participants that we need to inform as a failure occurs, and at which point that a participant needs to yield to non-failure notification(s): in other words, at which position that we need to add synchronisation points.

Failure-handling Protocol Types A protocol type which is capable of robust failure-handling has several attributes. To conveniently describe the conditions which identify those attributes, we give the interaction context on protocol types:

Definition 4 (Interaction Context).

$$\mathscr{G} ::= [\,] \mid \mathbf{T}[\mathscr{G}]\mathbf{H}[h] \mid \mathbf{T}[g]\mathbf{H}[..., f : \mathscr{G}, ...] \mid \mathscr{G};g \mid g;\mathscr{G} \mid \mu t.\mathscr{G}$$

We say a protocol type is applicable for failure handling if for its interaction type (g) the following conditions hold: (1) Every failure f appearing in g is handled by a handling environment in g; (2) every failure f appearing in some handling environment in g is unique to that handling environment; (3) every failure f appearing in some handling environment in g must also appear in a failure-raising interaction, which can trigger the handling environment containing f.

The formal definitions for them are defined below:

Definition 5 (Failure-handling Interaction Types). We say g is a failure-handling interaction type if the following conditions hold:

(1) If $g = \mathscr{G}[p_1 \to p_2 : \widetilde{S} \vee F]$, then for each $f \in F$ there exists $\mathscr{G}', \mathscr{G}''$ such that $\mathscr{G}[\,] = \mathscr{G}'[\mathbf{T}[\mathscr{G}''[\,]]\mathbf{H}[h]]$ and f is handled in h and for every failure appearing in $\mathscr{G}$, point 1 holds.
(2) If $g = \mathscr{G}[\mathbf{T}[g']\mathbf{H}[f_1 : g_1, ..., f_n : g_n]]$, then $\forall i, j \in \{1..n\}$, $i \neq j$ implies $f_i \neq f_j$ and for every handling environment appearing in $\mathscr{G}$, point 2 holds.
(3) If $g = \mathscr{G}[\mathbf{T}[g']\mathbf{H}[h]]$, then $\forall f \in dom(h)$, $\exists \mathscr{G}'.\ g' = \mathscr{G}'[p_1 \to p_2 : \widetilde{S} \vee F]$ and $f \in F$, in addition $\forall \mathscr{G}''.\ \mathscr{G}' = \mathscr{G}''[\mathbf{T}[\mathscr{G}''']\mathbf{H}[h']]$ where $f \in dom(h')$, then we should have $p_1 \to p_2 : \widetilde{S} \vee F$ appear in $\mathscr{G}''$. For every $\mathscr{G}$ point 3 holds.

Example 1. The interaction types

$$g_a = \mathbf{T}[p_1 \to p_2 : \widetilde{S} \vee f_1, f_2]\mathbf{H}[f_1 : g_1, f_2 : g_2]$$

and

$$g_b = \mathbf{T}[\mathbf{T}[p_1 \to p_2 : \widetilde{S} \vee f_1; p_2 \to p_3 : \widetilde{S} \vee f_2]\mathbf{H}[f_1 : g_1]]\mathbf{H}[f_2 : g_2]$$

are failure-handling because they satisfy points 1, 2 and 3: every f_1,f_2 in g_a and g_b are handled by $f_1 : g_1$ and $f_2 : g_2$, every failure name in the handlers is unique and every failure handler handles a possible failure.

But

$$g_c = \mathbf{T}[\mathbf{T}[p_1 \to p_2 : \widetilde{S} \vee f_1, f_2]\mathbf{H}[f_1 : g_1]]\mathbf{H}[f_3 : g_3]$$

and

$$g_d = \mathbf{T}[p_1 \to p_2 : \widetilde{S} \vee f]\mathbf{H}[f : g_1, f : g_2]$$

and

$$g_e = \mathbf{T}[p_1 \to p_2 : \widetilde{S}]\mathbf{H}[f : g_1]$$

are not since g_c violates point 1 as f_2 is not handled, g_d violates point 2 as f is not unique in the handling environment and g_e violates point 3 as f does not handle a possible failure in the handling environment.

Global Structures Now we define global structures which visualise protocol types with explicit interaction flow in a tree-like structure. They are introduced because they provide better shape for analysing which participants need to be notified as a failure occurs and which need to be synchronised to yield to non-failure notification.

For example, the following protocol type is hard to analyse:

$$\mathbf{T}[\mathbf{T}[p_1 \to p_2 : \mathsf{str} \vee f_1, f_2]\mathbf{H}[f_1 : p_1 \to p_3 : \mathsf{bool}]; p_4 \to p_5 : \mathsf{bool}]\mathbf{H}[f_2 : p_2 \to p_3 : \mathsf{int}]; \mathsf{end}$$

It is not immediately clear whether interaction $p_4 \to p_5$ will be affected by f_1,f_2 once they occur or not (so whether we need to inform these participants once f_1 or f_2 occurs). As an interaction will be affected by some failure (e.g., will be aborted by it), we say this interaction and the participants on this interaction are *related to* this failure.

We want to transform this protocol type to the following shape, which clearly shows that $p_4 \to p_5$ is related to f_2: as f_2 occurs, $p_4 \to p_5$ disappears:

$$\begin{aligned} try\{try\{&p_1 \to p_2 : \mathsf{str} \vee f_1, f_2 \dashrightarrow p_4 \to p_5 : \mathsf{bool} \dashrightarrow \mathsf{end}, \\ &f_1 : p_1 \to p_3 : \mathsf{bool} \dashrightarrow p_4 \to p_5 : \mathsf{bool} \dashrightarrow \mathsf{end} \\ f_2 : p_2 \to p_3 &: \mathsf{int} \dashrightarrow \mathsf{end}\} \end{aligned}$$

Since $p_1 \to p_2$ is tagged with both f_1 and f_2 and which one will occur is not known, we say $p_4 \to p_5$ is affected by the whole interaction $p_1 \to p_2 : \mathsf{str} \vee f_1, f_2$ instead of only f_2.

Thus we define global structures as:

Definition 6 (Global Structure $\mathtt{T}$ and Global Nodes $\mathtt{N}$).

$$\begin{aligned} &\text{(Global Structure)} && \mathtt{T} ::= \mathtt{N} \mid try\{\mathtt{T}, \mathtt{H}\} \mid \mathtt{N} \dashrightarrow \mathtt{T} \mid t \mid \mu t.\mathtt{T} \\ &\text{(Global Nodes)} && \mathtt{N} ::= \varepsilon \mid \mathsf{end} \mid p \to p : \widetilde{S} \vee F \\ &\text{(Global Structure Handlers)} && \mathtt{H} ::= \emptyset \mid \mathtt{H}, f : \mathtt{T} \end{aligned}$$

A global structure can be a node, a try-handle structure, a node followed by a structure, a structure variable, or a recursive structure. A node is either inactive (ε), or a termination (end), or an interaction $p_1 \to p_2 : \widetilde{S} \vee F$. Handlers here are defined similarly as in protocol types.

Global Structures from Protocol Types We use $\bullet$ to denote sequences in global structures in Definition 7 (see below), which defines how to generate global structures from protocol types. Global structure sequencing $\bullet$ is defined as follows:

$$\mathtt{N} \bullet \mathtt{T} = \begin{cases} \mathtt{T} & \text{if } \mathtt{N} = \mathsf{end} \\ \mathtt{N} \dashrightarrow \mathtt{T} & \text{otherwise} \end{cases} \qquad (\mathtt{N} \dashrightarrow \mathtt{T}') \bullet \mathtt{T} = \mathtt{N} \dashrightarrow (\mathtt{T}' \bullet \mathtt{T})$$

$$try\{\mathtt{T}_0, f_1 : \mathtt{T}_1, ..., f_n : \mathtt{T}_n\} \bullet \mathtt{T} = try\{\mathtt{T}_0 \bullet \mathtt{T}, f_1 : \mathtt{T}_1 \bullet \mathtt{T}, ..., f_n : \mathtt{T}_n \bullet \mathtt{T}\}$$

Example 2. Assume

$$\mathtt{T} = try\{p_1 \to p_2 : \mathsf{int} \vee f_1 \dashrightarrow \mathsf{end}, f_1 : \varepsilon\}$$

and $\mathtt{T}' = try\{p_2 \to p_3 : \mathsf{bool} \vee f_2 \dashrightarrow \mathsf{end}, f_2 : \mathsf{end}\}$. Then composing these two global structures in sequence we get:

$$\begin{aligned} \mathtt{T} \bullet \mathtt{T}' = try\{p_1 \to p_2 : \mathsf{int} \vee f_1 \dashrightarrow try\{p_2 \to p_3 : \mathsf{bool} \vee f_2 \dashrightarrow \mathsf{end}, f_2 : \mathsf{end}\}, \\ f_1 : \varepsilon \dashrightarrow try\{p_2 \to p_3 : \mathsf{bool} \vee f_2 \dashrightarrow \mathsf{end}, f_2 : \mathsf{end}\}\} \end{aligned}$$

Definition 7 (Generating Global Structures). $Struct(\mathbf{G})$ generates a global structure from protocol type $\mathbf{G}$ as follows:

$$Struct(\mathbf{G}) = \begin{cases} STC(g) \bullet \mathsf{end} & \text{if } (\mathbf{G} = g; \mathsf{end}) \\ & \wedge (g \text{ is a failure-handling interaction type}) \\ \text{Undefined} & \text{otherwise} \end{cases}$$

where $STC(g)$ is defined as below:

$$STC(\varepsilon) = \varepsilon \qquad STC(p_1 \to p_2 : \widetilde{S} \vee F) = \begin{cases} \text{Undefined} & \text{if } (F = \emptyset) \wedge (\widetilde{S} = \emptyset) \\ p_1 \to p_2 : \widetilde{S} \vee F & \text{otherwise} \end{cases}$$

$$STC(\mathtt{T}[g]\mathtt{H}[f_1 : g_1, ..., f_n : g_n]) = try\{STC(g), f_1 : STC(g_1), ..., f_n : STC(g_n)\}$$

$$STC(g_1; g_2) = STC(g_1) \bullet STC(g_2) \quad STC(t) = t \quad STC(\mu t.g) = \mu t.STC(g)$$

Formal Definition for Alpha-Renaming Alpha-renaming a global structure (i.e., thus also alpha-renaming a protocol type) is formally defined in this section. For convenience, we define a stack of handlers for alpha-renaming nested try-handle structures:

Definition 8. Define $\mathbb{H} ::= \mathtt{H} \mid \mathtt{H} :: \mathbb{H}$ as a stack of handlers. In $\mathtt{H} :: \mathbb{H}$, $\mathbb{H}$ contains the set(s) of outer handlers relative to $\mathtt{H}$.

The function $Rewrite(\mathbb{H}, F, F') = (\mathbb{H}', F'')$ rewrites the handlers which are in our target F and also appear in the stack of handlers $\mathbb{H}$ (i.e. $dom(\mathbb{H}) \cap F$) with fresh failure names. It uses F' to store those fresh names for calling *Rewrite* recursively; then it returns an updated $\mathbb{H}'$ and the set of fresh names F'':

$$Rewrite(\emptyset, F, F') = (\emptyset, F') \text{ [rw-empty]} \qquad Rewrite(\mathbb{H}, \emptyset, F) = (\mathbb{H}, F) \text{ [rw-result]}$$

$$\frac{f' \text{ fresh} \quad \exists f \in dom(\mathtt{H}) \cap F \quad \mathtt{H}' = \mathtt{H}, f' : \mathtt{H}(f)}{Rewrite(\mathtt{H} :: \mathbb{H}, F, F') = Rewrite(\mathtt{H}' :: \mathbb{H}, F \setminus \{f\}, F' \cup \{f'\})} \text{ [rw-inner]}$$

$$\frac{dom(\mathtt{H}) \cap F = \emptyset \quad Rewrite(\mathbb{H}, F, F') = (\mathbb{H}', F'')}{Rewrite(\mathtt{H} :: \mathbb{H}, F, F') = (\mathtt{H} :: \mathbb{H}', F'')} \text{ [rw-outer]}$$

Firstly we use a function $\rho(\mathtt{T}, \mathbb{H}, A)$ (defined below) to *recursively* alpha-rename the failures in $\mathtt{T}$ according to $\mathbb{H}$ and A, which respectively contain the information of existing failures, and the failures we have checked.

Definition 9 (Alpha-renaming). Let A be a set of failures. It is a storage for collecting the failures which have appeared when we walk through a structure. Define $\rho(\mathtt{T}, \mathbb{H}, A)$ as a function which maps a global structure and its stack of handlers and A to a new global structure, a new stack of handlers, and an augmented storage:

$$\rho(\varepsilon, \mathbb{H}, A) = (\varepsilon, \mathbb{H}, A) \text{ [}\rho\text{-null]} \qquad \rho(\mathsf{end}, \mathbb{H}, A) = (\mathsf{end}, \mathbb{H}, A) \text{ [}\rho\text{-end]}$$

$$\frac{A \cap F = F'' \neq \emptyset \quad Rewrite(\mathbb{H}, F'', \emptyset) = (\mathbb{H}', F')}{\rho(p_1 \to p_2 : \widetilde{S} \vee F, \mathbb{H}, A) = (p_1 \to p_2 : \widetilde{S} \vee F', \mathbb{H}', A)} \text{ [}\rho\text{-rename]}$$

$$\frac{F \cap A = \emptyset}{\rho(p_1 \to p_2 : \widetilde{S} \vee F, \mathbb{H}, A) = (p_1 \to p_2 : \widetilde{S} \vee F, \mathbb{H}, A \cup F)} \text{ [}\rho\text{-skip]}$$

$$\frac{\begin{array}{c}\mathtt{H} = f_1 : \mathtt{T}_1, ..., f_n : \mathtt{T}_n \quad \rho(\mathtt{T}, \mathtt{H} :: \mathbb{H}, A) = (\mathtt{T}', \mathtt{H}' :: \mathbb{H}_1, A_1) \\ \forall i \in \{1..n\}.\rho(\mathtt{T}_i, \mathbb{H}_i, A_i) = (\mathtt{T}_i', \mathbb{H}_{i+1}, A_{i+1}) \quad \mathbb{H}' = \mathbb{H}_{n+1} \quad A' = A_{n+1}\end{array}}{\rho(try\{\mathtt{T}, \mathtt{H}\}, \mathbb{H}, A) = (try\{\mathtt{T}', \mathtt{H}'[\mathtt{T}_1', ..., \mathtt{T}_n'/\mathtt{T}_1, ..., \mathtt{T}_n]\}, \mathbb{H}', A')} \text{ [}\rho\text{-try1]}$$

$$\frac{\mathbb{H} \neq \emptyset \quad \rho(\mathtt{T}, \mathbb{H}, A) = (\mathtt{T}', \mathbb{H}', A')}{\rho(try\{\mathtt{T}, \emptyset\}, \mathbb{H}, A) = (try\{\mathtt{T}', \emptyset\}, \mathbb{H}', A')} \text{ [}\rho\text{-try2]}$$

$$\frac{\rho(\mathtt{N}, \mathbb{H}, A) = (\mathtt{N}', \mathbb{H}'', A'') \quad \rho(\mathtt{T}, \mathbb{H}'', A'') = (\mathtt{T}', \mathbb{H}', A')}{\rho(\mathtt{N} \dashrightarrow \mathtt{T}, \mathbb{H}, A) = (\mathtt{N}' \dashrightarrow \mathtt{T}', \mathbb{H}', A')} \text{ [}\rho\text{-next]}$$

$$\rho(t, \mathbb{H}, A) = (t, \mathbb{H}, A) \text{ [}\rho\text{-}t\text{]} \quad \frac{\rho(\mathtt{T}, \mathbb{H}, A) = (\mathtt{T}', \mathbb{H}', A')}{\rho(\mu t.\mathtt{T}, \mathbb{H}, A) = (\mu t.\mathtt{T}', \mathbb{H}', A')} \text{ [}\rho\text{-rec]}$$

Others are not defined.

Let $\mathtt{fst}$ be a function which returns the first element of the result. Define α as a function which alpha-renames a global structure by making every $f \in F$ tagging on $p_1 \to p_2 : \widetilde{S} \vee F$ unique in the structure. If $f \in F$ has appeared in the structure, α renames f to a fresh one:

$$\alpha(\mathtt{T}) = \mathtt{fst}(\rho(\mathtt{T}, \emptyset, \emptyset))$$

For convenience, when we apply ρ to a global structure, we index the handlers with $i \in \mathbb{N}$ (i.e., natural numbers) to indicate that fresh f_i', f_i'', f_i''', ..., f_i^* are created from f_i; and index the interaction structures at handlers f_i, f_i', f_i'', .., f_i^* with i. We explain the interesting rules for $\rho(\mathtt{T}, \mathbb{H}, A)$ below:

Rule [ρ-**rename**] renames the failures in node $p_1 \to p_2 : \widetilde{S} \vee F$ with fresh names if those failures have appeared in previous nodes (i.e. $A \cap F = F' \neq \emptyset$). By applying rule [**rw-inner**] (i.e., the inner handlers are for those failures) or [**rw-outter**] (i.e., the outer handlers are for those failures), the new handlers with those fresh names are added to the stack of handlers.

Rule [ρ-**skip**] is applied if every failures in the node have not appeared in the previous nodes, or if this node has no failures (i.e., $F = \emptyset$), or if this node is the first node.

Rule [ρ-**try1**] firstly renames $\mathtt{T}$ with $\mathtt{H} :: \mathbb{H}$ since the failures occuring in $\mathtt{T}$ can be handled by $\mathtt{H}$ or $\mathbb{H}$. As $\mathtt{H} = f_1 : \mathtt{T}_1, ..., f_n : \mathtt{T}_n$, [$\rho$-**try1**] secondly renames $\mathtt{T}_1$ with $\mathbb{H}_1$, which is updated from $\mathbb{H}$; then it sequentially renames $\mathtt{T}_i$, $i \in \{2..n\}$ with $\mathbb{H}_i$, which is updated from $\mathbb{H}_{i-1}$. Note that if there is a common failure, say f, appearing in both $\mathtt{H}$ and $\mathbb{H}$, e.g. $\mathtt{H}(f) = \mathtt{T}_{\text{inner}}$ and $\mathbb{H}(f) = \mathtt{T}_{\text{outer}}$, once f occurs in $\mathtt{T}$ due to the try-handle structure, $\mathtt{H}$ will handle it. If the failure with the same name f appears in the following actions in try-block, [ρ-**try1**] will alpha-rename it to a fresh name, and respectively add a handler with a fresh name, say f', and the outer handlers are augmented to $\mathbb{H}$, $f' : \mathtt{T}_{\text{outer}}$. Although it may make the $f' : \mathtt{T}_{\text{outer}}$ in $\mathbb{H}$ redundent, it makes sure that every node tagged with a set of failures in a structure is unique and failures are properly handled. Thus any updated $\mathtt{T}_i'$ from $\mathtt{T}_i$ will not affect $\mathbb{H}$ and its following updates, but will affect $\mathtt{H}'$. We replace every $\mathtt{T}_i$ in $\mathtt{H}'$ with $\mathtt{T}_i'$. As [ρ-**try1**] renames all $\mathtt{T}, \mathtt{T}_i$, $i \in \{1..n\}$, it returns the renamed structure $try\{\mathtt{T}', \mathtt{H}'[\mathtt{T}_1', ..., \mathtt{T}_n'/\mathtt{T}_1, ..., \mathtt{T}_n]\}$ and the last updated handlers $\mathbb{H}' = \mathbb{H}_{n+1}$ and the last updated $A' = A_{n+1}$.

Rule [ρ-**try2**] simply renames $try\{\mathtt{T}, \emptyset\}$ with its outer handlers $\mathbb{H}$.

Rule [ρ-**next**] uses the updated stack of handlers and storage (i.e., $\mathbb{H}''$ is updated from $\mathbb{H}$ and A'' is updated from A) to rename the following structure.

Other rules are straightforward.

Example 3. In the following interaction structure

$$try\{p_1 \to p_2 : \widetilde{S} \vee f \dashrightarrow try\{p_1 \to p_2 : \widetilde{S} \vee f \dashrightarrow \mathsf{end}, f : \mathtt{T}'\}, f : \mathtt{T}\}$$

the sub-structure $try\{p_1 \to p_2 : \widetilde{S} \vee f \dashrightarrow \mathsf{end},\ f : \mathtt{T}'\}$ following the first node $p_1 \to p_2 : \widetilde{S} \vee f$ has a common f which has appeared in the first node. The handler for the second-shown f theoretically can take different behaviour $\mathtt{T}'$ rather than $\mathtt{T}$ because of the nested structure. However, the first and second nodes are indistinguishable. We want a structure which can *immediately* return a clear guidance when we (statically) check an action taken by a process. Thus we alpha-rename a structure to ensure the failures are handled in correct nested handlers.

Example 4. Assume

$$\mathtt{T} = try\{p_1 \to p_2 : \widetilde{S} \vee f \dashrightarrow try\{p_1 \to p_2 : \widetilde{S} \vee f \dashrightarrow \mathsf{end}, f : \mathtt{T}'\}, f : \mathtt{T}\}$$

Then we alpha rename $\mathtt{T}$:

$$\alpha(\mathtt{T}) = try\{p_1 \to p_2 : \widetilde{S} \vee f \dashrightarrow try\{p_1 \to p_2 : \widetilde{S} \vee f' \dashrightarrow \mathsf{end}, f' : \mathtt{T}'\}, f : \mathtt{T}\}$$

After alpha-renaming, all interactions tagged with failures are unique and the structures provides a global guidance.

Transformation To obey to a protocol type which exhibits the properties of **P1 (alignment)**, **P2 (precision)**, **P3 (causality)** and **P4 (decentralisation)**, we need to calculate/determine the sets of *need-(to-)be-informed* participants, including the set of those who need to be informed as a failure occurs, and the set of those who need to be informed as no failures occur, and where to add synchronisation points among interactions. Ultimately we want local types to contain such information to verify participant (endpoint) implementations.

The Set of Need-(to)-be-informed Participants To precisely handle partial failures, we need to first understand which participants are affected by some particular failures; in other words, which participants who *need-(to)-be-informed* regarding those failures. Specifically, we use $C(\mathtt{T}, F)$ to extract the set of participants who need-(to)-be-informed regarding the set of failures $F \neq \emptyset$ tagged on a specific interaction in $\mathtt{T}$.

Before we introduce this definition, some auxiliary definitions are provided. They are $FSet(\mathtt{N})$, $snd(\mathtt{N})$, and $rcv(\mathtt{N})$, respectively returning the set of failures tagging on interaction $\mathtt{N}$, the sender of $\mathtt{N}$, and the receiver of $\mathtt{N}$. In addition, we define function $rH(\mathtt{T}, F)$ returning all participants which are involved in the handling for failures in F, $node(\mathtt{T}, F)$ returning the node tagged with $F \neq \emptyset$ in $\mathtt{T}$, and $\{F, \mathtt{N}\} \Subset \mathtt{T}$ stating that the interaction which can raise F or the interaction $\mathtt{N}$ is in $\mathtt{T}$. Furthermore, we define $[\![\mathtt{N}_1 < \mathtt{N}_2]\!]_{\mathtt{T}}$, stating $\mathtt{N}_1$ appears before $\mathtt{N}_2$ in $\mathtt{T}$.

Definition 10 ($FSet(\mathtt{N}), snd(\mathtt{N}), rcv(\mathtt{N})$). We define $FSet(\mathtt{N})$ as the set of failures appearing in $\mathtt{N}$, $snd(\mathtt{N})$ as the sender of $\mathtt{N}$, and $rcv(\mathtt{N})$ as the receiver of $\mathtt{N}$

$$FSet(\varepsilon) = FSet(\mathsf{end}) = \emptyset \quad FSet(p_1 \to p_2 : \widetilde{S} \vee F) = F$$
$$snd(\varepsilon) = snd(\mathsf{end}) = \emptyset \quad snd(p_1 \to p_2 : \widetilde{S} \vee F) = p_1$$
$$rcv(\varepsilon) = rcv(\mathsf{end}) = \emptyset \quad rcv(p_1 \to p_2 : \widetilde{S} \vee F) = p_2$$

Definition 11 (The Set of Participants Involved for Handling F). Let $\mathtt{T}$ be alpha renamed. We define function $rH(\mathtt{T}, F)$ returning the set of participants who are involved for handling F in $\mathtt{T}$:

$$rH(\mathtt{N}, F) = \emptyset \quad rH(try\{\mathtt{T}, \mathtt{H}\}, F) = rH(\mathtt{T}, F) \cup r \text{ where } r = \bigcup_{(f \in dom(\mathtt{H}) \cap F)} pid(\mathtt{H}(f))$$

$$rH(\mathtt{N} \dashrightarrow \mathtt{T}, F) = rH(\mathtt{T}, F) \quad rH(\mu t.\mathtt{T}, F) = rH(\mathtt{T}, F) \quad rH(t, F) = \emptyset$$

Definition 12. Assume $\mathtt{T}$ is alpha renamed. Define function $node(\mathtt{T},F)$ returning the node tagged with $F \neq \emptyset$ from an alpha renamed $\mathtt{T}$:

$$node(\mathtt{N},F) = \begin{cases} \mathtt{N} & \text{if } FSet(\mathtt{N}) = F \\ \emptyset & \text{otherwise} \end{cases}$$

$$node(try\{\mathtt{T},\mathtt{H}\},F) = \begin{cases} node(\mathtt{T},F) & \text{if } (node(\mathtt{T},F) \neq \emptyset) \\ node(\mathtt{H}(f),F) & \text{if } (node(\mathtt{H}(f),F) \neq \emptyset) \wedge (f \in dom(\mathtt{H})) \\ \emptyset & \text{otherwise} \end{cases}$$

$$node(\mathtt{N} \dashrightarrow \mathtt{T},F) = \begin{cases} \mathtt{N} & \text{if } FSet(\mathtt{N}) = F \\ node(\mathtt{T},F) & \text{otherwise} \end{cases}$$

$$node(\mu t.\mathtt{T},F) = node(\mathtt{T},F) \qquad node(t,F) = \emptyset$$

Definition 13 ($\rho \in \{F,\mathtt{N}\} \Subset \mathtt{T}$). We define the binary relation $\Subset$ between $\rho \in \{F,\mathtt{N}\}$ and $\mathtt{T}$ that ρ appears in $\mathtt{T}$. It is defined as follows:

$$\frac{(FSet(\mathtt{N}) = \rho) \vee (\mathtt{N} = \rho)}{\rho \Subset \mathtt{N}} \qquad \frac{\rho \Subset \mathtt{T}}{\rho \Subset try\{\mathtt{T},\mathtt{H}\}} \qquad \frac{\exists f \in dom(\mathtt{H}).\rho \Subset \mathtt{H}(f)}{\rho \Subset try\{\mathtt{T},\mathtt{H}\}}$$

$$\frac{\rho \Subset \mathtt{N}}{\rho \Subset \mathtt{N} \dashrightarrow \mathtt{T}} \qquad \frac{\rho \Subset \mathtt{T}}{\rho \Subset \mathtt{N} \dashrightarrow \mathtt{T}} \qquad \frac{\rho \Subset \mathtt{T}}{\rho \Subset \mu t.\mathtt{T}}$$

Definition 14 ($\mathtt{N}_1$ Appears Before $\mathtt{N}_2$ in $\mathtt{T}$: $[\![\mathtt{N}_1 < \mathtt{N}_2]\!]_{\mathtt{T}}$). Assume $\mathtt{T}$ is alpha renamed. The relation $[\![\mathtt{N}_1 < \mathtt{N}_2]\!]_{\mathtt{T}}$ states that $\mathtt{N}_1$ is executed before $\mathtt{N}_2$ in $\mathtt{T}$:

$$\frac{\mathtt{N}_1 \dashrightarrow \mathtt{T} \quad \mathtt{N}_2 \Subset \mathtt{T}}{[\![\mathtt{N}_1 < \mathtt{N}_2]\!]_{\mathtt{N}_1 \dashrightarrow \mathtt{T}}} \qquad \frac{[\![\mathtt{N}_1 < \mathtt{N}_2]\!]_{\mathtt{T}}}{[\![\mathtt{N}_1 < \mathtt{N}_2]\!]_{\mathtt{N} \dashrightarrow \mathtt{T}}} \qquad \frac{[\![\mathtt{N}_1 < \mathtt{N}_2]\!]_{\mathtt{T}}}{[\![\mathtt{N}_1 < \mathtt{N}_2]\!]_{try\{\mathtt{T},\mathtt{H}\}}}$$

$$\frac{\exists f \in dom(\mathtt{H}).[\![\mathtt{N}_1 < \mathtt{N}_2]\!]_{\mathtt{H}(f)}}{[\![\mathtt{N}_1 < \mathtt{N}_2]\!]_{try\{\mathtt{T},\mathtt{H}\}}} \qquad \frac{(\mathtt{N}_1 \Subset \mathtt{T}) \wedge (\exists f \in dom(\mathtt{H}).\mathtt{H}(f) \Subset \mathtt{N}_2)}{[\![\mathtt{N}_1 < \mathtt{N}_2]\!]_{try\{\mathtt{T},\mathtt{H}\}}} \qquad \frac{[\![\mathtt{N}_1 < \mathtt{N}_2]\!]_{\mathtt{T}}}{[\![\mathtt{N}_1 < \mathtt{N}_2]\!]_{\mu t\mathtt{T}}}$$

Others are undefined

Now we formally define the set of participants who need-be-informed regarding the set of failures F (i.e. who becomes aborted or takes over failure handling activity when a failure in F occurs) in $\mathtt{T}$:

Definition 15 (Need-be-informed Participants $C(\mathtt{T},F)$ (Full Definition)). Let $r = \{rcv(node(\mathtt{T},F))\} \cup rH(\mathtt{T},F)$. We define function $C(\mathtt{T},F)$ returning the set of participants who need-be-infomred regarding F in $\mathtt{T}$:

$$C(\mathtt{T},F) = \begin{cases} C_{rec}(\mathtt{T},F,r) & \text{if } F \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases}$$

where $C_{rec}(\mathtt{T},F,r)$ is the least fixpoint of $c(\mathtt{T},\mathtt{T},F,r)$:

$$C_{rec}(\mathtt{T},F,r) = \begin{cases} r & \text{if } r = c(\mathtt{T},\mathtt{T},F,r) \\ C_{rec}(\mathtt{T},F,c(\mathtt{T},\mathtt{T},F,r)) & \text{otherwise} \end{cases}$$

and $c(\mathtt{T},\mathtt{T}',F,r)$ is defined by the following rules:

$$(1)\ c(\mathtt{T},\mathtt{N},F,r) = \begin{cases} r \cup pid(\mathtt{N}) \cup C(\mathtt{T},FSet(\mathtt{N})) & \text{if } [\![node(\mathtt{T},F) < \mathtt{N}]\!]_{\mathtt{T}} \wedge \\ & ((pid(\mathtt{N}) \cap r \neq \emptyset) \vee (FSet(\mathtt{N}) \neq \emptyset)) \\ r & \text{otherwise} \end{cases}$$

$$(2)\ c(\mathtt{T},try\{\mathtt{T}',\ \mathtt{H}\},F,r) = \begin{cases} c(\mathtt{T},\mathtt{T}_i,F,r) & \text{if } \mathtt{H} = \{f_1 : \mathtt{T}_1,...,f_n : \mathtt{T}_n\} \wedge F \Subset \mathtt{T}_i \\ c(\mathtt{T},\mathtt{T}',F,r) \bigcup_{f \in dom(\mathtt{H}) \cap F} c(\mathtt{T},\mathtt{H}(f),F,r) & \text{otherwise} \end{cases}$$

$$(3)\ c(\mathtt{T},\mathtt{N} \dashrightarrow \mathtt{T}',F,r) = c(\mathtt{T},\mathtt{N},F,r) \cup c(\mathtt{T},\mathtt{T}',F,r)$$

$$(4)\ c(\mathtt{T},t,F,r) = \emptyset \qquad (5)\ c(\mathtt{T},\mu t.\mathtt{T}',F,r) = pid(\mathtt{T}')$$

In summary, function c calculates the need-be-informed participants in a sub- or equivalent structure in $\mathtt{T}$ based on r, the set of participants that have been known as affected by F. Then C_{rec} returns the fixpoint of c. The fixpoint calculation is required because actions that occur later in a protocol can induce causal relationships between participants in previous interactions. For instance in

$$\mathtt{T} = try\{p_1 \to p_2 : \mathsf{int} \vee f \dashrightarrow p_4 \to p_3 : \mathsf{bool} \dashrightarrow p_2 \to p_4 : \mathsf{str},\ f : p_2 \to p_5 : \mathsf{int}\},$$

we discover that p_3 and p_4 are affected by f when we check $p_2 \to p_4 : \mathsf{str}$. If we do not count p_3 and p_4 as affected by f, once f occurs and p_2 switches to handling it, p_3 and p_4 will both starve.

Function $C(\mathtt{T},F)$ returns the result by initialising r (which is the base for calculating c) with the participants who are either the receiver of a failure in F or involved in handling failures in F (i.e. $r = \{rcv(node(\mathtt{T},F))\} \cup rH(\mathtt{T},F)$).

For the main calculuation c, rule (1) is the key rule: if the current interaction that we are checking (i.e. $\mathtt{N}$) appears after the interaction which may raise a failure in F (i.e. $[\![node(\mathtt{T},F) < \mathtt{N}]\!]_{\mathtt{T}}$), and some of its interacting participants (either the sender or the receiver) is related to r or the interaction itself can raise another failure set (i.e. the failure it may raise is dependent on the previous occurrence of failures), then we collect its interacting participants and those who need-be-informed regarding the failures that can be raised by $\mathtt{N}$. Rule (2), for a try-handle term, collects those in $\mathtt{T}_i$ if F is tagged on some interaction for handling f_i in $\mathtt{H}$; otherwise, it collects those in the default interactions and those in the failure handling activities for those $f \in F$ in $\mathtt{H}$. Rule (5) collects all participants if there is a recursive call. Other rules are trivial.

Example 5. Recall $\mathbf{G}_{proxy}$ in Section 2 in the main content:

$$\begin{aligned} \mathbf{G}_{proxy} = \mathbf{T}[&\mathsf{Client} \to \mathsf{Proxy} : \mathsf{str}; \\ &\mathsf{Proxy} \to \mathsf{Log} : \mathsf{str} \vee \mathsf{SuspiciousB}, \mathsf{QuotaWarn}; \mathsf{Proxy} \to \mathsf{EServer} : \mathsf{str}] \\ \mathbf{H}[&\mathsf{SuspiciousB} : \mathsf{Log} \to \mathsf{SupServer} : \mathsf{str},\ \mathsf{QuotaWarn} : \mathsf{Log} \to \mathsf{Client} : \mathsf{str}]; \mathsf{end} \end{aligned}$$

Let $\mathit{Struct}(\mathbf{G}_{proxy}) = \mathtt{T}_{proxy}$. We use C to check which participants are affected by $F = \{\mathsf{SuspiciousB}, \mathsf{QuotaWarn}\}$:

$$
\begin{aligned}
& C(\mathit{Struct}(\mathbf{G}_{proxy}), F) \\
= {} & C(\mathit{try}\{\mathsf{Client} \to \mathsf{Proxy} : \mathsf{str} \dashrightarrow \\
& \qquad \mathsf{Proxy} \to \mathsf{Log} : \mathsf{str} \vee \mathsf{SuspiciousB}, \mathsf{QuotaWarn} \dashrightarrow \\
& \qquad \mathsf{Proxy} \to \mathsf{EServer} : \mathsf{str} \dashrightarrow \mathsf{end}, \\
& \quad \mathsf{SuspiciousB} : \mathsf{Log} \to \mathsf{SupServer} : \mathsf{str} \dashrightarrow \mathsf{end}, \\
& \quad \mathsf{QuotaWarn} : \mathsf{Log} \to \mathsf{Client} : \mathsf{str} \dashrightarrow \mathsf{end}\},\ F) \\
= {} & C(\mathtt{T}_{proxy}, F) \\
= {} & C_{rec}(\mathtt{T}_{proxy}, F, \{\mathsf{Log}\} \cup \{\mathsf{Log}, \mathsf{SuperServer}, \mathsf{Client}\}) \\
= {} & C_{rec}(\mathtt{T}_{proxy}, F, \{\mathsf{Log}, \mathsf{SuperServer}, \mathsf{Client}\}) \\
= {} & C_{rec}(\mathtt{T}_{proxy}, F, c(\mathtt{T}_{proxy}, \mathtt{T}_{proxy}, F, \{\mathsf{Log}, \mathsf{SuperServer}, \mathsf{Client}\})) \\
= {} & \{\mathsf{Log}, \mathsf{SuperServer}, \mathsf{Client}\}
\end{aligned}
$$

Since we say that it is always a sender who can raise a failure when making an output action, to notify need-be-informed participants, we only need to add the information of whom need to be notified when a failure occurs into a sending node. Assume $\mathtt{T}$ is alpha renamed.

$$
\mathit{Causal}(\mathtt{T}, \mathbf{n}) = \begin{cases} sn\langle p_2 ! \widetilde{S} \vee F, C(\mathtt{T}, F), C(\mathtt{T}, F)\rangle & \text{if } (\mathbf{n} = sn\langle p_2 ! \widetilde{S} \vee F\rangle) \wedge (F \neq \emptyset) \\ & \wedge (\mathit{node}(\mathtt{T}, F) = \mathtt{N}) \\ \mathbf{n} & \text{otherwise} \end{cases}
$$

If $\mathbf{n}$ is a sending node tagged with $F \neq \emptyset$ and it matches a particular node in an alpha renamed $\mathtt{T}$, then $\mathit{Causal}(\mathtt{T}, \mathbf{n})$ adds the set of participants of $C(\mathtt{T}, F)$ which is defined in Definition 10. Otherwise it keeps $\mathbf{n}$ unchanged.

Note that, in this paper, *because the synchronisation points are not reduced (i.e. not optimised), the set of those who need-(to)-be-informed as a failure occurs, and the set of those who need-(to)-be-informed as no failure occurs are the same*. However, since we are going to reduce the synchronisation points in our future works (and it will result two different sets), and we aim to provide a more general version for the process calculus and the theorems (which are proved under the assumption that these two sets can be different), for a sending action specification (i.e. $sn\langle p_2 ! \widetilde{S} \vee F, \_, \_\rangle$) we give two slots: one for those who need to be informed as a failure occurs, another for those who need to be informed as no failures occur.

Then we update every sending node tagged with $F \neq \emptyset$ in a *simple* local type according to a given global structure:

Definition 16 (Adding the Set of Need-(to)-be-informed Participants). Assume $\mathtt{T}$ is alpha renamed. We define $\mathit{CStruct}(\mathtt{T}, \mathbf{T})$ as a function updating every sending node(s) tagged with some $F \neq \emptyset$ in $\mathbf{T}$ by the given $\mathtt{T}$:

$$\mathit{CStruct}(\mathtt{T}, \mathbf{n}) = \mathit{Causal}(\mathtt{T}, \mathbf{n})$$

$$\mathit{CStruct}(\mathtt{T}, \mathit{try}\{\mathbf{T}, f_1 : \mathbf{T}_1, ..., f_n : \mathbf{T}_n\}) = \mathit{try}\{\mathit{CStruct}(\mathtt{T}, \mathbf{T}), f_1 : \mathit{CStruct}(\mathtt{T}, \mathbf{T}_1), ..., f_n : \mathit{CStruct}(\mathtt{T}, \mathbf{T}_n)\}$$

$$\mathit{CStruct}(\mathtt{T}, \mathbf{n} \dashrightarrow \mathbf{T}) = \mathit{CStruct}(\mathtt{T}, \mathbf{n}) \dashrightarrow \mathit{CStruct}(\mathtt{T}, \mathbf{T})$$

$$\mathit{CStruct}(\mathtt{T}, t) = t \qquad \mathit{CStruct}(\mathtt{T}, \mu t.\mathbf{T}) = \mu t.\mathit{CStruct}(\mathtt{T}, \mathbf{T})$$

Example 6. Recall $\mathtt{T}_{proxy}$ in the above Example 5 and let $F = \{\mathsf{SuspiciousB}, \mathsf{QuotaWarn}\}$ We firstly generate local types for each participants in $\mathtt{T}_{proxy}$ by Definition 2 (Projection):

$$\begin{aligned}
\upharpoonright(\mathtt{T}_{proxy}, \mathsf{Client}) &= try\{sn\langle \mathsf{Proxy!\ str}, _\ , _\rangle \dashrightarrow \varepsilon_F \dashrightarrow \varepsilon \dashrightarrow \mathsf{end},\\
&\qquad \mathsf{SuspiciousB} : \varepsilon \dashrightarrow \mathsf{end},\\
&\qquad \mathsf{QuotaWarn} : rn\langle \mathsf{Log?\ str}\rangle \dashrightarrow \mathsf{end}\} = \mathbf{T}_{\mathsf{Client}}\\
\upharpoonright(\mathtt{T}_{proxy}, \mathsf{Proxy}) &= try\{rn\langle \mathsf{Client?\ str}\rangle \dashrightarrow sn\langle \mathsf{Log!\ str} \vee F, _\ , _\rangle\\
&\qquad \dashrightarrow sn\langle \mathsf{EServer!\ str}, _\ , _\rangle \dashrightarrow \mathsf{end},\\
&\qquad \mathsf{SuspiciousB} : \varepsilon \dashrightarrow \mathsf{end},\\
&\qquad \mathsf{QuotaWarn} : \varepsilon \dashrightarrow \mathsf{end}\} = \mathbf{T}_{\mathsf{Proxy}}\\
\upharpoonright(\mathtt{T}_{proxy}, \mathsf{Log}) &= try\{\varepsilon \dashrightarrow rn\langle \mathsf{Proxy?\ str} \vee F\rangle \dashrightarrow \varepsilon \dashrightarrow \mathsf{end},\\
&\qquad \mathsf{SuspiciousB} : sn\langle \mathsf{SupServer!\ str}\rangle \dashrightarrow \mathsf{end},\\
&\qquad \mathsf{QuotaWarn} : sn\langle \mathsf{Client!\ str}\rangle \dashrightarrow \mathsf{end}\} = \mathbf{T}_{\mathsf{Log}}\\
\upharpoonright(\mathtt{T}_{proxy}, \mathsf{EServer}) &= try\{\varepsilon \dashrightarrow \varepsilon_F \dashrightarrow rn\langle \mathsf{Proxy?\ str}\rangle \dashrightarrow \mathsf{end},\\
&\qquad \mathsf{SuspiciousB} : \varepsilon \dashrightarrow \mathsf{end},\\
&\qquad \mathsf{QuotaWarn} : \varepsilon \dashrightarrow \mathsf{end}\} = \mathbf{T}_{\mathsf{EServer}}\\
\upharpoonright(\mathtt{T}_{proxy}, \mathsf{SupServer}) &= try\{\varepsilon \dashrightarrow \varepsilon_F \dashrightarrow \varepsilon \dashrightarrow \mathsf{end},\\
&\qquad \mathsf{SuspiciousB} : rn\langle \mathsf{Log?\ str}\rangle \dashrightarrow \mathsf{end},\\
&\qquad \mathsf{QuotaWarn} : \varepsilon \dashrightarrow \mathsf{end}\} = \mathbf{T}_{\mathsf{SupServer}}
\end{aligned}$$

Now we add the set of need-be-informed participants regarding F into each local types based on $\mathtt{T}_{proxy}$:

$$\begin{aligned}
CStruct(\mathtt{T}_{proxy}, \mathbf{T}_{\mathsf{Client}}) &= \mathbf{T}_{\mathsf{Client}}\\
CStruct(\mathtt{T}_{proxy}, \mathbf{T}'_{\mathsf{Proxy}}) &= try\{rn\langle \mathsf{Client?\ str}\rangle\\
&\qquad \dashrightarrow sn\langle \mathsf{Log!\ str} \vee F, \{\mathsf{Client}, \mathsf{Log}, \mathsf{SupServer}\},\\
&\qquad\qquad \{\mathsf{Client}, \mathsf{Log}, \mathsf{SupServer}\}\rangle\\
&\qquad \dashrightarrow sn\langle \mathsf{EServer!\ str}, _\ , _\rangle \dashrightarrow \mathsf{end},\\
&\qquad \mathsf{SuspiciousB} : \varepsilon \dashrightarrow \mathsf{end},\\
&\qquad \mathsf{QuotaWarn} : \varepsilon \dashrightarrow \mathsf{end}\} = \mathbf{T}'_{\mathsf{Proxy}}\\
CStruct(\mathtt{T}_{proxy}, \mathbf{T}_{\mathsf{Log}}) &= \mathbf{T}_{\mathsf{Log}}\\
CStruct(\mathtt{T}_{proxy}, \mathbf{T}_{\mathsf{EServer}}) &= \mathbf{T}_{\mathsf{EServer}}\\
CStruct(\mathtt{T}_{proxy}, \mathbf{T}_{\mathsf{SupServer}}) &= \mathbf{T}_{\mathsf{SupServer}}
\end{aligned}$$

Then we add synchronisation points in *simple* local types at the right positions to ensure a local participant can safely proceed the default actions after these points.

Definition 17 (Adding Synchronisation Points). Assume $\mathtt{T}$ is alpha renamed. Define *Sync* as a function inserting synchronisation points into a local type based on $\mathtt{T}$:

[sync-node]

$$Sync(\mathtt{T},\mathbf{n},p) = \begin{cases} yield\langle FSet(\mathtt{N})\rangle \dashrightarrow \mathbf{n} & \text{if } (\mathbf{n} = \lceil(\mathtt{N},p)) \wedge (p \in C(\mathtt{T},FSet(\mathtt{N}))) \wedge p \neq snd(\mathtt{N}) \\ \mathbf{n} \dashrightarrow yield\langle FSet(\mathtt{N})\rangle & \text{if } (\mathbf{n} = \lceil(\mathtt{N},p)) \wedge (p \in C(\mathtt{T},FSet(\mathtt{N}))) \wedge p = snd(\mathtt{N}) \\ \mathbf{n} & \text{otherwise} \end{cases}$$

[sync-try]

$$Sync(\mathtt{T},try\{\mathbf{T},\mathbf{H}\},p) = try\{Sync(\mathtt{T},\mathbf{T},p), \{f : Sync(\mathtt{T},\mathbf{H}(f),p) \mid f \in dom(\mathbf{H})\}\}$$

[sync-seq]

$$Sync(\mathtt{T},\mathbf{n} \dashrightarrow \mathbf{T},p) = Sync(\mathtt{T},\mathbf{n},p) \dashrightarrow Sync(\mathtt{T},\mathbf{T},p)$$

In rule **[sync-node]**, by Definition 2 (Projection), $\mathbf{n} \in \{sn\langle p''!\ \widetilde{S} \vee F\ \_\ ,\ \_\rangle, rn\langle p'?\ \widetilde{S} \vee F\rangle, \varepsilon_F\}$). Since $\mathtt{T}$ is alpha renamed, rule **[sync-node]** can find a unique node $\mathtt{N}$ in $\mathtt{T}$ matching $\mathbf{n}$ that we are checking. Note that, if p is not in $\mathtt{N}$, the action of p at the moment should be $\mathbf{n} = \varepsilon_F$ (by projection rule), which is still able to trace back that corresponding $\mathtt{N}$ (tagging with F). If p is affected by F (i.e. a need-be-informed participant) we add a synchronisation point $yield\langle F\rangle$, where F is the set of failures tagging on $\mathtt{N}$ (i.e. $F = FSet(\mathtt{N})$). If p is not in the node, then $\mathbf{n} = \varepsilon_F$ and it does not matter if $yield\langle F\rangle$ is inserted before or after $\mathbf{n}$. However if $p \in pid(\mathtt{N})$, then the position of the $yield\langle F\rangle$ is important. If p is the receiver, we have $yield\langle F\rangle$ positioned before the receiving action because $yield\langle F\rangle$ is the point deciding whether the process will handle a failure regarding F or proceed. If p is the sender, we should have $yield\langle F\rangle$ positioned after the sending action because, as p is involved for some failure handling activity regarding F (by Definition 15), it needs to firstly send out that failure notification then goes back to execute the handling activity; otherwise the process will get stuck. Other rules are trivial.

Example 7. Recall $\mathtt{T}_{proxy}$ in Example 5 and the updated local types in Example 6.

We apply function *Sync* to every updated local types based on $\mathtt{T}_{proxy}$:

$$\begin{aligned}
Sync(\mathtt{T}_{proxy}, \mathbf{T}_{Client}, \mathsf{Client}) = try\{&sn\langle \mathsf{Proxy!\ str}, _, _\rangle \\
&\dashrightarrow yield\langle F\rangle \dashrightarrow \varepsilon_F \dashrightarrow \varepsilon \dashrightarrow \mathsf{end}, \\
&\mathsf{SuspiciousB} : \varepsilon \dashrightarrow \mathsf{end}, \\
&\mathsf{QuotaWarn} : rn\langle \mathsf{Log?\ str}\rangle \dashrightarrow \mathsf{end}\} = \mathbf{T}'_{\mathsf{Client}} \\
Sync(\mathtt{T}_{proxy}, \mathbf{T}'_{Proxy}, \mathsf{Proxy}) = try\{&rn\langle \mathsf{Client?\ str}\rangle \\
&\dashrightarrow sn\langle \mathsf{Log!\ str} \vee F, \{\mathsf{Client}, \mathsf{Log}, \mathsf{SupServer}\}, \\
&\qquad\qquad \{\mathsf{Client}, \mathsf{Log}, \mathsf{SupServer}\}\rangle \\
&\dashrightarrow sn\langle \mathsf{EServer!\ str}, _, _\rangle \dashrightarrow \mathsf{end}, \\
&\mathsf{SuspiciousB} : \varepsilon \dashrightarrow \mathsf{end}, \\
&\mathsf{QuotaWarn} : \varepsilon \dashrightarrow \mathsf{end}\} = \mathbf{T}'_{\mathsf{Proxy}} \\
Sync(\mathtt{T}_{proxy}, \mathbf{T}_{Log}, \mathsf{Log}) = try\{&\varepsilon \dashrightarrow yield\langle F\rangle \\
&\dashrightarrow rn\langle \mathsf{Proxy?\ str} \vee F\rangle \dashrightarrow \varepsilon \dashrightarrow \mathsf{end}, \\
&\mathsf{SuspiciousB} : sn\langle \mathsf{SupServer!\ str} \vee F, _, _\rangle \\
&\dashrightarrow \mathsf{end}, \\
&\mathsf{QuotaWarn} : sn\langle \mathsf{Client!\ str} \vee F, _, _\rangle \\
&\dashrightarrow \mathsf{end}\} = \mathbf{T}'_{\mathsf{Log}} \\
Sync(\mathtt{T}_{proxy}, \mathbf{T}_{\mathsf{EServer}}, \mathsf{EServer}) = try\{&\varepsilon \dashrightarrow \varepsilon_F \dashrightarrow rn\langle \mathsf{Proxy?\ str}\rangle \dashrightarrow \mathsf{end}, \\
&\mathsf{SuspiciousB} : \varepsilon \dashrightarrow \mathsf{end}, \\
&\mathsf{QuotaWarn} : \varepsilon \dashrightarrow \mathsf{end}\} = \mathbf{T}_{\mathsf{EServer}} \\
Sync(\mathtt{T}_{proxy}, \mathbf{T}_{\mathsf{SupServer}}, \mathsf{SupServer}) = try\{&\varepsilon \dashrightarrow yield\langle F\rangle \dashrightarrow \varepsilon_F \dashrightarrow \varepsilon \dashrightarrow \mathsf{end}, \\
&\mathsf{SuspiciousB} : rn\langle \mathsf{Log?\ str}\rangle \dashrightarrow \mathsf{end}, \\
&\mathsf{QuotaWarn} : \varepsilon \dashrightarrow \mathsf{end}\} = \mathbf{T}'_{\mathsf{SupServer}}
\end{aligned}$$

Only the local types of Client, Log, and SupServer are added with synchronisation points $yield\langle F\rangle$.

Up to now, we have reached local types for every participants with sufficient information for guiding them to coordinate with others and where if a failure occurs, or inversely to assert that none has occurred, before proceeding with the next action. The whole operation from Example 5 to Example 7 is called *transformation*, which is formally defined below:

Definition 18 (Transformation).

$$Transform(\mathbf{G}, p) = \begin{cases} Sync(\mathtt{T}, CStruct(\mathtt{T}, \mathbf{T}), p) & \text{if } (\mathtt{T} = \alpha(Struct(\mathbf{G})) \text{ is defined}) \\ & \wedge (\mathbf{T} = \upharpoonright(\mathtt{T}, p) \text{ is defined}) \\ \text{Undefined} & \text{otherwise} \end{cases}$$

By $Transform(\mathbf{G}, p)$ we generate local types with the guidance to comply with the properties of **P1** to **P4** given in $\mathbf{G}$ for every participant appearing in $\mathbf{G}$.

$$(\text{The Set of Process Variables Declared in } D)\ dpv(D)$$
$$(\text{The Set of Names Occurring Free in } N)\ fn(N)$$

$$N \parallel [\mathbf{0}]_{\mathbf{T}} \equiv N \equiv [\mathbf{0}]_{\mathbf{T}} \parallel N \quad N_1 \parallel N_2 \equiv N_2 \parallel N_1 \quad (N_1 \parallel N_2) \parallel N_3 \equiv N_1 \parallel (N_2 \parallel N_3)$$
$$(\mathsf{new}\ s\ s')N \equiv (\mathsf{new}\ s'\ s)N \quad (\mathsf{new}\ s)\mathbf{0} \equiv \mathbf{0} \quad (\mathsf{new}\ s)N_1 \parallel N_2 \equiv (\mathsf{new}\ s)(N_1 \parallel N_2)\ \textit{if}\ s \notin fn(N_2)$$

$$\mathsf{def}\ D\ \mathsf{in}\ \mathbf{0} \equiv \mathbf{0} \quad \frac{dpv(D) \cap dpv(D') = \emptyset}{\mathsf{def}\ D\ \mathsf{in}\ \mathsf{def}\ D'\ \mathsf{in}\ P \equiv \mathsf{def}\ D'\ \mathsf{in}\ \mathsf{def}\ D\ \mathsf{in}\ P}$$

$$\varepsilon \cdot q \equiv q \equiv q \cdot \varepsilon \quad (q_1 \cdot q_2) \cdot q_3 \equiv q_1 \cdot (q_1 \cdot q_2) \quad \frac{m_1 \cdot m_2 \frown m_2 \cdot m_1}{q \cdot m_1 \cdot m_2 \cdot q' \equiv q \cdot m_2 \cdot m_1 \cdot q'} \quad \frac{q \equiv q'}{s : q \equiv s : q'}$$

Fig. 7: Structural congruence rules for the network.

Example 8. Based on Examples 6 and 7, we reach the local types generated by *Transform* from $\mathbf{G}_{proxy}$ for each participants:

$$\begin{aligned} \mathit{Transform}(\mathbf{G}_{proxy}, \mathsf{Client}) &= \mathbf{T}'_{\mathsf{Client}} \\ \mathit{Transform}(\mathbf{G}_{proxy}, \mathsf{Proxy}) &= \mathbf{T}'_{\mathsf{Proxy}} \\ \mathit{Transform}(\mathbf{G}_{proxy}, \mathsf{Log}) &= \mathbf{T}'_{\mathsf{Log}} \\ \mathit{Transform}(\mathbf{G}_{proxy}, \mathsf{EServer}) &= \mathbf{T}_{\mathsf{EServer}} \\ \mathit{Transform}(\mathbf{G}_{proxy}, \mathsf{supServer}) &= \mathbf{T}'_{\mathsf{SupServer}} \end{aligned}$$

A.2 Definitions for Section 3: Processes for Decentralised Multiple-Failure-Handling

This section gives formal definitions for functions which have been used in Section 3.

Figure 7 defines the structural congruence rules over processes and the network. Structural congruence is associative and commutative as usual. Following [9,12], $m_1 \cdot m_2$ is permutable to $m_2 \cdot m_1$, denoted by $m_1 \cdot m_2 \frown m_2 \cdot m_1$, meaning that, regarding asynchrony, they are structurally congruent even after linking q as the head and q' as the tail. We say two messages are permutable if they are sent from the same failure-raising interaction, e.g. $\langle p_1, p_2, \langle \widetilde{v} \rangle \rangle^F \cdot \langle\langle p, F \rangle\rangle \frown \langle\langle p, F \rangle\rangle \cdot \langle p_1, p_2, \langle \widetilde{v} \rangle \rangle^F$, or one of them is a failure notification and both are for the same receiver, or both are normal messages and either their senders or receivers are different.

Permutation of queues is defined below. Definition 19 captures the nature of concurrency and asynchrony by queuing messages in different possible orders:

Definition 19. We say q is permutable to q', denoted by $q \frown q'$, if one of the following conditions holds:

1. $q \frown q$.
2. $q = m_1 \cdot m_2$ and $q' = m_2 \cdot m_1$ and
   (a) $(m_1, m_2) = (\langle p_1, p_2, \langle \widetilde{v} \rangle \rangle^F, \langle p'_1, p'_2, \langle \widetilde{v'} \rangle \rangle^{F'})$ and $((p_1 \neq p'_1) \vee (p_2 \neq p'_2))$; or
   (b) $(m_1, m_2) = (\langle p_1, p_2, \langle \widetilde{v} \rangle \rangle^F, \langle\langle p, F \rangle\rangle)$; or
   (c) $(m_1, m_2) = (\langle\langle p, F \rangle\rangle, \langle p_1, p_2, \langle \widetilde{v} \rangle \rangle^F)$; or
   (d) $(m_1, m_2) = (\langle p_1, p_2, \langle \widetilde{v} \rangle \rangle^F, \langle p, f \rangle)$ and $f \in F$; or

(e) $(m_1,m_2) = (\langle p,f\rangle, \langle p_1,p_2,\langle\widetilde{v}\rangle\rangle^F)$ and $f \in F$; or
(f) $(m_1,m_2) = (\langle\langle p_1,F\rangle\rangle, \langle\langle p_2,F\rangle\rangle)$; or
(g) $(m_1,m_2) = (\langle p_1,f\rangle, \langle p_2,f\rangle)$.

The following definition is used in rules [try] and [hdl] in Figure 4 in the main content to map a process to the channel that it is acting on:

Definition 20. We define *act* as an operation that extracts the channel that a process or a network or a set of handlers is acting on in a session:

$$act(\mathbf{0}) = act(v) = \star \qquad act(a[p](y).P) = y$$

$$act(c!(p,\langle\tilde{e}\rangle^F)) = act(c?(p,(\tilde{x})^F).P) = act(c\ \mathsf{raise}(f)\ \mathsf{to}\ p) = act(c \otimes F) = c$$

$$act(\mathsf{try}\{P\}\mathsf{h}\{H\}) = \begin{cases} act(H) = act(H(f_i)) & \text{if } (act(P) \neq \star) \wedge \\ & (\forall f_i, f_j \in dom(H).act(H(f_i)) = act(H(f_j)) = act(P)) \\ act(H) = act(H(f_i)) & \text{if } (act(P) = \star) \wedge \\ & (\forall f_i, f_j \in dom(H).act(H(f_i)) = act(H(f_j))) \\ \text{Undefined} & \text{otherwise} \end{cases}$$

$$act(P_1;P_2) = \begin{cases} act(P_2) & \text{if } act(P_1) = \star \\ act(P_1) & \text{if } act(P_1) = act(P_2) \neq \emptyset \\ \text{Undefined} & \text{otherwise} \end{cases}$$

$$act(\mathsf{def}\ D\ \mathsf{in}\ P) = act(P) \quad act(X\langle\tilde{e}\ c\rangle) = c$$

$$act(\mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q) = \begin{cases} act(P) & \text{if } act(P) = act(Q) \\ \text{Undefined} & \text{otherwise} \end{cases}$$

$$act([P]_{\mathbf{T}}) = act(P) \quad act(N_1 \parallel N_2) = act(N_1) \cup act(N_2)$$

$$act(s:q) = s \qquad act((\mathsf{new}\ s)N) = act(N)$$

Others are undefined.

We use $\star$ to denote an arbitrary channel.

A.3 Definitions for Sections 4 and 5: Type System and the Network Coherence

In this section we provide the formal definitions for rules and functions which have been used in Sections 4 and 5.

Typing Rules for Expressions

$$\Gamma, x:S \vdash x:S \qquad \Gamma \vdash \mathsf{unit} : \mathsf{unit} \qquad \Gamma \vdash 0,1,\ldots : \mathsf{int}\ \text{[}\alpha\text{-axiom/unit/int]}$$

$$\Gamma \vdash \mathsf{true},\mathsf{false} : \mathsf{bool} \quad \frac{\Gamma \vdash e_i : \mathsf{bool} \quad i \in \{1,2\}}{\Gamma \vdash e_1 \vee e_2 : \mathsf{bool}} \quad \text{[}\alpha\text{-bool/or]}$$

The binary relation $\mathbf{T} \ni \mathbf{T}'$ has appeared in rule [T-guide] in Figure 6 in the main content, which defines the typing rules for networks. This relation describes that $\mathbf{T}$ can be a guidance for $\mathbf{T}'$ because $\mathbf{T}$, which contains $\mathbf{T}'$, has more information than $\mathbf{T}'$ does.

Definition 21. We define a binary relation $\ni$ over $\mathbf{T}$:

$$\mathbf{T} \ni \mathbf{T} \quad \frac{\mathbf{T} \ni \mathbf{T}' \quad \mathbf{T}' \ni \mathbf{T}''}{\mathbf{T} \ni \mathbf{T}''}$$

$$try\{\mathbf{T},\mathbf{H}\} \ni \mathbf{T} \quad \frac{\exists f \in dom(\mathbf{H}).h(f) \ni \mathbf{T}'}{try\{\mathbf{T},\mathbf{H}\} \ni \mathbf{T}'} \quad \frac{\mathbf{T} \ni \mathbf{T}'}{try\{\mathbf{T},\mathbf{H}\} \ni try\{\mathbf{T}',\mathbf{H}\}}$$

$$\frac{dom(\mathbf{H}') = dom(\mathbf{H}) \quad \exists f \in dom(\mathbf{H}').\mathbf{H}(f) \ni \mathbf{H}'(f)}{try\{\mathbf{T},\mathbf{H}\} \ni try\{\mathbf{T},\mathbf{H}'\}}$$

$$\mathbf{n} \dashrightarrow \mathbf{T} \ni \mathbf{T} \quad \frac{\mathbf{T} \ni \mathbf{n}}{\mathbf{n}' \dashrightarrow \mathbf{T} \ni \mathbf{n}} \quad \frac{\mathbf{T} \ni \mathbf{T}'}{\mathbf{n}' \dashrightarrow \mathbf{T} \ni \mathbf{T}'} \quad \frac{\mathbf{T} \ni \mathbf{T}'}{\mathbf{n} \dashrightarrow \mathbf{T} \ni \mathbf{n} \dashrightarrow \mathbf{T}'}$$

$$\mu t.\mathbf{T} \ni \mathbf{T} \quad \frac{\mathbf{T} \ni \mathbf{T}'}{\mu t.\mathbf{T} \ni \mu t.\mathbf{T}'}$$

$\Delta \rightharpoonup \Delta'$: Reduction of Session Environments To prove the property of subject reduction, we formalise the reduction of session environments, denoted by $\rightharpoonup$, in Figure 8. We write $\Delta \rightharpoonup \Delta'$ to represent that a session environment Δ is reduced to Δ' (due to the occurrence of interactions). For convenience, we define context on local types:

$$\mathscr{C} ::= [\,] \mid try\{\mathscr{C},\mathbf{H}\}$$

The rules abstractly correspond to the operational semantics of networks defined in Figure 4 in the main content, where the $\mathbf{T}$s are generated by *Transform*; here we also use local types to associate channels. Recall that, by Definition 18, the set of need-be-informed participants are added to local types. The most interesting rule is [R-sndthw], which is non-deterministic. It depends on the internal choice of p_1 in session s to send either a normal content or to raise a failure. This rule corresponds to the rules [sndF] and [thwf] defined in Figure 4 in the main content.

Coherence *Coherence* describes an environment where all *interactions* comply with the guidance of some $\mathbf{G}$, where $\forall p \in pid(\mathbf{G})$, the local type $Transform(\mathbf{G},p)$ is defined; in other words, in a coherent environment, every participant's behaviour obeys $Transform(\mathbf{G},p)$.

Due to asynchrony, after a sender takes action, its output may be still in the global queue. At this moment, the type of the sender and its receiver are not coherent since the sender has moved forward. We should check their coherence only after the receiver has absorbed all messages heading to it.

Thus the following functions $\upharpoonright(\mathbf{M},p)$ and $\mathbf{T}\!-\upharpoonright(\mathbf{q},p)$, where $\mathbf{q}$ is the type of a queue, are defined: the former extracts the message types heading to p, and the latter returns

$$s[p] : \varepsilon \dashrightarrow \mathbf{T} \rightharpoonup s[p] : \mathbf{T} \quad \textbf{[R-}\varepsilon\textbf{]} \qquad \frac{\Delta_1 \rightharpoonup \Delta_1'}{\Delta_1, \Delta_2 \rightharpoonup \Delta_1', \Delta_2} \quad \textbf{[R-res]}$$

$$\frac{\widetilde{S} \neq \emptyset}{s[p_1] : sn\langle p_2!(\widetilde{S})\rangle \dashrightarrow \mathbf{T},\ s : \mathbf{q} \rightharpoonup s[p_1] : \mathbf{T},\ s : \mathbf{q} \cdot \langle p_1, p_2, \langle \widetilde{S}\rangle\rangle} \quad \textbf{[R-snd]}$$

$$\frac{\begin{array}{c} F \neq \emptyset \quad \widetilde{p}' = p_2, p_1', ..., p_m' \quad \widetilde{p}'' = p_1'', ..., p_n'' \\ (\mathbf{q}' = \langle p_1, p_2, \langle \widetilde{S}\rangle\rangle^F \cdot \langle\langle p_1'', F\rangle\rangle ... \langle\langle p_n'', F\rangle\rangle) \wedge (\widetilde{S} \neq \emptyset)) \vee \\ (\mathbf{q}' = \langle p_2, f\rangle \cdot \langle p_1', f\rangle ... \langle p_m', f\rangle) \wedge (f \in F)) \end{array}}{s[p_1] : try\{sn\langle p_2!(\widetilde{S}) \vee F, \widetilde{p}', \widetilde{p}''\rangle \dashrightarrow \mathbf{T}, \mathbf{H}\},\ s : \mathbf{q} \rightharpoonup s[p_1] : try\{\mathbf{T}, \mathbf{H}\},\ s : \mathbf{q} \cdot \mathbf{q}'} \quad \boxed{\textbf{[R-sndthw]}}$$

$$s[p_1] : rn\langle p_2?(\widetilde{S}) \vee F\rangle \dashrightarrow \mathbf{T},\ s : \langle p_2, p_1, \langle \widetilde{S}\rangle\rangle^F \cdot \mathbf{q} \rightharpoonup s[p_1] : \mathbf{T}, s : \mathbf{q} \quad \textbf{[R-rcv]}$$

$$\frac{\mathbf{q} = \langle p', f'\rangle \cdot \mathbf{q}'' \Rightarrow (p' \neq p) \vee (f' \notin dom(\mathbf{H})) \quad s[p] : \mathbf{T}, s : \mathbf{q} \rightharpoonup s[p] : \mathbf{T}', s : \mathbf{q}'}{s[p] : try\{\mathbf{T}, \mathbf{H}\},\ s : \mathbf{q} \rightharpoonup s[p] : try\{\mathbf{T}', \mathbf{H}\}, s : \mathbf{q}'} \quad \boxed{\textbf{[R-try]}}$$

$$\frac{f \in dom(\mathbf{H}) \cap F}{s[p] : try\{\mathscr{C}[yield\langle F\rangle \dashrightarrow \mathbf{T}],\ \mathbf{H}\},\ s : \langle p, f\rangle \cdot \mathbf{q} \rightharpoonup s[p] : \mathbf{H}(f),\ s : \mathbf{q}} \quad \boxed{\textbf{[R-hdl]}}$$

$$\frac{F' \subseteq F}{s[p] : yield\langle F'\rangle \dashrightarrow \mathbf{T},\ s : \langle\langle p, F\rangle\rangle \cdot \mathbf{q},\ \rightharpoonup s[p] : \mathbf{T},\ s : \mathbf{q}} \quad \boxed{\textbf{[R-sync-done]}}$$

$$\frac{F'' = F' \setminus F \neq \emptyset}{s[p] : yield\langle F'\rangle \dashrightarrow \mathbf{T},\ s : \langle\langle p, F\rangle\rangle \cdot \mathbf{q}\ \rightharpoonup s[p] : yield\langle F''\rangle \dashrightarrow \mathbf{T},\ s : \mathbf{q}} \quad \boxed{\textbf{[R-sync]}}$$

$$s[p] : try\{\mathsf{end}, \mathbf{H}\}\ \rightharpoonup s[p] : \mathsf{end} \quad \textbf{[R-try-end]}$$

Fig. 8: Reduction of session environments.

the resulting type after $\mathbf{T}$ absorbs those (in queue type $\mathbf{q}$) heading to p. In summary, $\mathbf{T}\!\upharpoonright\!(\mathbf{q}, p)$ formally returns the *local type after absorbing messages in the global queue*, and gives a clearer figure for checking property of coherence.

We define $\upharpoonright(\mathbf{M}, p)$ as a type specifying a message heading to p. It ranges over

$$\mathtt{M} ::= \varepsilon \mid \langle\!\langle p, \langle \widetilde{S}\rangle\rangle\!\rangle^F \mid \langle\!\langle f\rangle\!\rangle \mid (\!(F)\!).$$

Definition 22 (Heading-to-p).

$$\upharpoonright(\varepsilon, p) = \varepsilon \qquad \upharpoonright(\mathbf{M} \cdot \mathbf{q}, p) = \begin{cases} \langle\!\langle p_1, \langle \widetilde{S}\rangle\rangle\!\rangle^F \cdot \upharpoonright(\mathbf{q}, p) & \text{if } p = p_2, \mathbf{M} = \langle p_1, p_2, \langle \widetilde{S}\rangle\rangle^F \\ \langle\!\langle f\rangle\!\rangle \cdot \upharpoonright(\mathbf{q}, p) & \text{if } p = p_2, \mathbf{M} = \langle p_2, f\rangle \\ (\!(F)\!) \cdot \upharpoonright(\mathbf{q}, p) & \text{if } p = p_2, \mathbf{M} = \langle\langle p_2, F\rangle\rangle \\ \upharpoonright(\mathbf{q}, p) & \text{otherwise} \end{cases}$$

Following [6,17], we define $\mathbf{T} - \mathtt{M}$ as $\mathbf{T}$ absorbing $\mathtt{M}$ in a straightforward way: only receivers absorb messages which she can absorb, while senders ignore the messages. For convenience, we define

$$\mathtt{q} ::= \mathtt{M} \mid \mathtt{q} \cdot \mathtt{M}$$

Conventionally, we say $(\!|\emptyset|\!) = \varepsilon$.

Definition 23 (Session Remainder $\mathbf{T}-\mathtt{q}$ (Formal)).

$$\mathbf{T}-\varepsilon = \mathbf{T} \quad \mathsf{end}-\mathtt{q} = \mathsf{end} \qquad \textsf{[rm-q}\varepsilon\textsf{ / end]}$$

$$(\varepsilon \dashrightarrow \mathbf{T})-\mathtt{q} = \mathbf{T}-\mathtt{q} \qquad \textsf{[rm-}\varepsilon\textsf{]}$$

$$(sn\langle p!(\widetilde{S}) \vee F, \widetilde{p}', \widetilde{p}''\rangle \dashrightarrow \mathbf{T})-\mathtt{q} = sn\langle p!(\widetilde{S}) \vee F, \widetilde{p}', \widetilde{p}''\rangle \dashrightarrow (\mathbf{T}-\mathtt{q}) \qquad \textsf{[rm-snd]}$$

$$(rn\langle p?(\widetilde{S}) \vee F\rangle \dashrightarrow \mathbf{T})-(\!|p, \langle \widetilde{S}\rangle|\!)\cdot\mathtt{q} = \mathbf{T}-\mathtt{q} \qquad \textsf{[rm-rcv]}$$

$$\frac{F' \subseteq F}{(yield\langle F'\rangle \dashrightarrow \mathbf{T})-(\!|F|\!)\cdot\mathtt{q} = \mathbf{T}-\mathtt{q}} \qquad \textsf{[rm-sync-done]}$$

$$\frac{F' \setminus F = F'' \neq \emptyset}{(yield\langle F'\rangle \dashrightarrow \mathbf{T})-(\!|F|\!)\cdot\mathtt{q} = (yield\langle F''\rangle \dashrightarrow \mathbf{T})-\mathtt{q}} \qquad \textsf{[rm-sync]}$$

$$\frac{\forall f \in dom(\mathbf{H}).(\mathtt{M} \neq (\!|f|\!))}{(try\{\mathbf{T},\ \mathbf{H}\})-\mathtt{M}\cdot\mathtt{q} = try\{\mathbf{T}-\mathtt{M}, \mathbf{H}\}-\mathtt{q}} \qquad \textsf{[rm-try]}$$

$$(try\{\mathsf{end},\ \mathbf{H}\})-\mathtt{q} = \mathsf{end} \qquad \textsf{[rm-try-end]}$$

$$\frac{f \in dom(\mathbf{H}) \cap F}{(try\{\mathscr{C}[yield\langle F\rangle \dashrightarrow \mathbf{T}], \mathbf{H}\})-(\!|f|\!)\cdot\mathtt{q} = \mathbf{H}(f)-\mathtt{q}} \qquad \textsf{[rm-hdl]}$$

Others are undefined.

The rules [rm-qε, rm-end, rm-ε, rm-snd, rm-rcv] are defined similarly as those in [17,6] except that here we define on tree-like local types. What [17,6] do not have are the rules for synchronisation points (i.e., [rm-sync-done, rm-sync]) and for try-handle terms (i.e., [rm-try, rm-try-end, rm-hdl]). Note that, in $\mathbf{T}-\mathtt{M}$, $\mathbf{T}$ absorbs $\mathtt{M}$ only when $\mathbf{T}$ is able to do so. Rules [rm-end] and [rm-snd] are the cases that $\mathbf{T}$ is not able to absorb the top message. For [rm-try], since $\mathbf{T}$ in $try\{\mathbf{T},\mathbf{H}\}$ contains all ongoing *default* actions, the action after absorbing $\mathtt{M}$ should become $try\{\mathbf{T}-\mathtt{M},\mathbf{H}\}-\mathbf{q}$. For [rm-try-end], we define $try\{\mathsf{end},\mathbf{H}\}$ is not able to absorb any messages and result end. For [rm-hdl], when the type has a synchronisation point for F in its try-block and a handler for $f \in F$ and a failure $(\!|f|\!)$ occurs, it returns the local type for handling f.

G *Guides* Δ Since all participants in a coherent environment should have consistent interactions, there must exist a protocol type which is able to guide every participant's behaviour; thus we define:

Definition 24. $s[p]$ is guided by $\mathbf{G}$ in Δ if $Transform(\mathbf{G},p) \equiv_{type} \Delta(s[p])-\upharpoonright(\Delta(s),p)$.

where $\equiv_{type}$ is defined by the followings:

Definition 25 (Types Equivalence). We define the relation $\equiv_{type}$ over $\mathbf{T}$ by the following rules:

$$\mathbf{T} \equiv_{type} \mathbf{T} \quad \varepsilon \dashrightarrow \mathbf{T} \equiv_{type} \mathbf{T} \quad \varepsilon_F \dashrightarrow \mathbf{T} \equiv_{type} \mathbf{T}$$

$$\frac{dom(\mathbf{H}) = dom(\mathbf{H}') \quad \forall f \in dom(\mathbf{H}) \wedge dom(\mathbf{H}').\mathbf{H}(f) \equiv_{type} \mathbf{H}'(f)}{try\{\mathbf{T},\mathbf{H}\} \equiv_{type} try\{\mathbf{T},\mathbf{H}'\}}$$

$$\frac{\mathbf{T} \equiv_{type} \mathbf{T}'}{try\{\mathbf{T},\mathbf{H}\} \equiv_{type} try\{\mathbf{T}',\mathbf{H}\}} \quad \frac{\mathbf{T} \equiv_{type} \mathbf{T}'}{\mathbf{n} \dashrightarrow \mathbf{T} \equiv_{type} \mathbf{n} \dashrightarrow \mathbf{T}'} \quad \frac{\mathbf{T} \equiv_{type} \mathbf{T}'}{\mu t.\mathbf{T} \equiv_{type} \mu t.\mathbf{T}'}$$

We say $\mathbf{G}$ guides p if $\mathbf{G}$ gives p a local guidance after p absorbs all of its messages in the global queue. We say an environment is totally guided by $\mathbf{G}$ if every participant is guided by $\mathbf{G}$:

Definition 26. We say that $\Delta\langle s\rangle$ is *totally* guided by $\mathbf{G}$ if $\{p \mid s[p] \in dom(\Delta)\} = pid(\mathbf{G})$ and $\forall s[p] \in dom(\Delta)$, $s[p]$ is guided by $\mathbf{G}$.

Now we formally define network coherence by defining a coherent protocol type, a coherent session environment for a particular session, and a coherent session environment for the whole network.

Definition 27 (Coherence (Formal)).

(1) We say $\mathbf{G}$ is coherent if $Transform(\mathbf{G},p)$ is defined for every $p \in pid(\mathbf{G})$.
(2) We say $\Delta\langle s\rangle$ is coherent if there exists a coherent $\mathbf{G}$ such that either $\Delta\langle s\rangle$ is totally guided by $\mathbf{G}$ or there exists $\Delta' \subset \Delta$ such that $\forall s[p] \in dom(\Delta) \setminus dom(\Delta')$, $s[p]$ is guided by $\mathbf{G}$ and $\Delta'\langle s\rangle$ is coherent and there exist coherent $\Delta'' \subseteq \Delta'$ and $\mathbf{G}'$ such that $\Delta''\langle s\rangle$ is totally guided by $\mathbf{G}'$.
(3) Δ is coherent if $\Delta\langle s\rangle$ is coherent for every s appearing in Δ.

Point (1) states that a coherent $\mathbf{G}$ is able to give local guidance (i.e., local types) for all of its participants. Point (2) states that, for a coherent $\Delta\langle s\rangle$, when no failures occur, we always find a coherent $\mathbf{G}$ to *totally* guide all participants in $\Delta\langle s\rangle$; once a failure occurs and triggers a handler, those participants affected by that failure will be aborted or switch to handling activities, while other unaffected ones still continue default interactions. As a protocol type is nested with several try-handle blocks, the occurrences of failures will trigger different handling activities and the affected participants will change their behaviours accordingly. Note that, if there exists a coherent $\mathbf{G}$ to *totally* guide Δ, then Δ must be coherent; however, a coherent Δ does not imply there exists such a coherent $\mathbf{G}$ to *totally* guide it but implies that there exist one or more than one protocol type to guide its participants.

Figure 9 uses an example to illustrate Point (2). Assume p_i, $i = 1..5$ are participants in a session s, whose actions obey to protocol type $\mathbf{G}_0$, while $\mathbf{G}_j$ respectively specifies handling protocols for f_j, $j = 1..3$ and guides interactions for corresponding failures. Then session environment $\{s[p_i] : \mathbf{T}_i\}_{i=1..5} \cup \{s : \mathbf{q}\}$ is coherent. In other words, every $s[p] \in dom(\Delta\langle s\rangle)$ is guided by some coherent protocol type. Notice that, as failure f_1 occurs, p_1 and p_2 switch to, while p_5 is invoked to, execute the handling activity defined by protocol $\mathbf{G}_1$, while p_3 and p_4, who are not affected by the occurrences of f_1, f_2, and f_3, continue their interaction defined by $\mathbf{G}_0$.

Point (3) is straightforward.

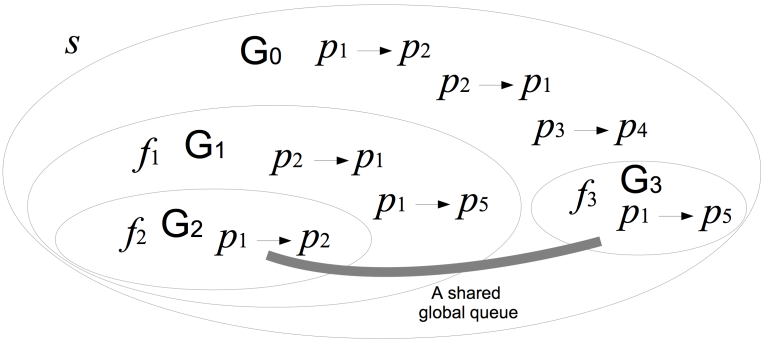

Fig. 9: Given $\mathbf{G}_0 = \mathbf{T}[p_1 \to p_2 : \mathsf{int} \vee f_1; p_2 \to p_1 : \mathsf{bool} \vee f_3; p_3 \to p_4 : \mathsf{str}]\mathbf{H}[f_1 : \mathbf{G}_1, f_3 : \mathbf{G}_3]$ and $\mathbf{G}_1 = \mathbf{T}[p_2 \to p_1 : \mathsf{bool} \vee f_2; p_1 \to p_5 : \mathsf{int}]\mathbf{H}[f_2 : \mathbf{G}_2]$ and $\mathbf{G}_2 = p_1 \to p_2 : \mathsf{str}$ and $\mathbf{G}_3 = p_1 \to p_5 : \mathsf{str}$. We observe that $\mathbf{G}_2 \subset \mathbf{G}_1 \subset \mathbf{G}_0$ and $\mathbf{G}_3 \subset \mathbf{G}_0$, and default interactions among p_1, p_2, p_3 and p_4 are guided by $\mathbf{G}_0$, handling for f_1 among p_1, p_2 and p_5 are guided by $\mathbf{G}_1$, handling for f_2 among p_1 and p_2 is guided by $\mathbf{G}_2$, while handling for $f_3$among p_1 and p_5 is guided by $\mathbf{G}_3$.

Auxiliary Definitions for Session Environments We define $dom(\Delta)$ as the set of session channels appearing in Δ, and $dom_{\mathsf{q}}(\Delta)$ as the set of global queues occurring in Δ.

Definition 28. We write Δ_1, Δ_2 for $\Delta_1 \cup \Delta_2$ which is the disjoint union of Δ_1 and Δ_2.

Definition 29. 1. $\mathbf{M} \curvearrowright \mathbf{M}$.

2. $\mathbf{q} = \mathbf{M}_1 \cdot \mathbf{M}_2$ and $\mathbf{q}' = \mathbf{M}_2 \cdot \mathbf{M}_1$ and
   (a) $(\mathbf{M}_1, \mathbf{M}_2) = (\langle p_1, p_2, \langle\widetilde{S}\rangle\rangle^F, \langle p_1', p_2', \langle\widetilde{S'}\rangle\rangle^{F'})$ and $(p_1 \neq p_1') \vee (p_2 \neq p_2')$; or
   (b) $(\mathbf{M}_1, \mathbf{M}_2) = (\langle p_1, p_2, \langle\widetilde{S}\rangle\rangle^F, \langle\langle p, F\rangle\rangle)$; or
   (c) $(\mathbf{M}_1, \mathbf{M}_2) = (\langle\langle p, F\rangle\rangle, \langle p_1, p_2, \langle\widetilde{S}\rangle\rangle^F)$; or
   (d) $(\mathbf{M}_1, \mathbf{M}_2) = (\langle p_1, p_2, \langle\widetilde{S}\rangle\rangle^F, \langle p, f\rangle)$ and $f \in F$; or
   (e) $(\mathbf{M}_1, \mathbf{M}_2) = (\langle p, f\rangle, \langle p_1, p_2, \langle\widetilde{S}\rangle\rangle^F)$ and $f \in F$; or
   (f) $(\mathbf{M}_1, \mathbf{M}_2) = (\langle\langle p_1, F\rangle\rangle, \langle\langle p_2, F\rangle\rangle)$; or
   (g) $(\mathbf{M}_1, \mathbf{M}_2) = (\langle p_1, f\rangle, \langle p_2, f\rangle)$.

Definition 30 ($\Delta_1 \equiv \Delta_2$)**.** We write $\Delta_1 \equiv \Delta_2$ if all the following conditions hold:

$c \in dom(\Delta_1) \wedge dom(\Delta_2)$ implies $\Delta_1(c) = \Delta_2(c)$

$c \in dom(\Delta_1) \wedge c \notin dom(\Delta_2)$ implies $\Delta_1(c) = \mathsf{end}$

$c \notin dom(\Delta_1) \wedge c \in dom(\Delta_2)$ implies $\Delta_2(c) = \mathsf{end}$

$s \in dom(\Delta_1)$ implies $s \in dom(\Delta_2)$ and $\Delta_1(s) = q \curvearrowright q' = \Delta_2(s)$ and vise versa

Definition 31 (Δ **end-only).** We say that Δ is end-only if for every $c \in dom(\Delta)$, $\Delta(c) = \mathsf{end}$.

It is easy to verify that Δ is end-only iff $\Delta \equiv \emptyset$.

Before introducing the rules for sequentially composing session environments, we introduce the rules for sequencing local types. The rules are similar to those for sequencing global structures. We abuse the notation $\bullet$, which has been used for sequecing global structures, to denote the sequencing of local types.

Definition 32 (Sequencing Local Types $\mathbf{T} \bullet \mathbf{T}$).

$$\mathbf{n} \bullet \mathbf{T} = \begin{cases} \mathbf{T} & \text{if } \mathbf{n} = \mathsf{end} \\ \mathbf{n} \dashrightarrow \mathbf{T} & \text{otherwise} \end{cases} \qquad (\mathbf{n} \dashrightarrow \mathbf{T}') \bullet \mathbf{T} = \mathbf{n} \dashrightarrow (\mathbf{T}' \bullet \mathbf{T})$$

$$try\{\mathbf{T}_0, f_1 : \mathbf{T}_1, ..., f_n : \mathbf{T}_n\} \bullet \mathbf{T} = \begin{cases} \mathbf{T} & \text{if } (\mathbf{T}_0 \equiv_{type} \mathsf{end}) \\ try\{\mathbf{T}_0 \bullet \mathbf{T}, f_1 : \mathbf{T}_1 \bullet \mathbf{T}, ..., f_n : \mathbf{T}_n \bullet \mathbf{T}\} & \text{otherwise} \end{cases}$$

The following Definition 33 is standard as the original rule in [3].

Definition 33 (Sequential Composition of Session Environments). We define $\Delta_1 \circ \Delta_2$ as the sequential composition of the session environments:

$$\Delta_1 \circ \Delta_2(c) = \begin{cases} \mathbf{T} & \text{if } \Delta_1(c) = \mathbf{T} \text{ and } \Delta_2(c) = \mathsf{end} \\ \mathbf{T}_1 \bullet \mathbf{T}_2 & \text{if } \Delta_1(c) = \mathbf{T}_1 \text{ and } \Delta_2(c) = \mathbf{T}_2 \\ \Delta_1(c) & \text{if } \Delta_1(c) \text{ defined and } \Delta_2(c) \text{ undefined} \\ \Delta_2(c) & \text{if } \Delta_1(c) \text{ udefined and } \Delta_2(c) \text{ defined} \\ \text{Undefined} & \text{otherwise} \end{cases}$$

B Optional Appendix for Proofs

Coherence Remains after Reduction of Session Environments We firstly give several auxiliary lemmas for proving that coherence is remained after reduction. From Proposition 1 to Lemma 3, we simply state that a coherent Δ will anyway be able to reduce unless Δ is end-only.

Proposition 1. Assume $\Delta\langle s\rangle$ is totally guided by a coherent $\mathbf{G}$ and $\mathbf{G} \neq \mathsf{end}$. Then there exists $s[p] \in dom(\Delta)$ such that $\Delta = \Delta_0, s[p] : \mathbf{T} \rightharpoonup \Delta_0', s[p] : \mathbf{T}'$ for some Δ_0, Δ_0' and $\mathbf{T} \neq \mathbf{T}'$.

Proof. By Definition 27, $\forall s[p] \in dom(\Delta\langle s\rangle)$ we have $Transform(\mathbf{G}, p) = \Delta(s[p])- \upharpoonright (\Delta(s), p)$ is defined. We have the following cases:

(a) If $\Delta(s) \neq \varepsilon$, then without loss of generality there exists p' such that a message heading to p' is on the *top* of $\Delta(s)$. If $\Delta(s[p'])$ is able to input (including receiving a normal message, receiving a failure, or receiving a synchronisation point), then by rules defined in Figure 8, it will input this message and $\Delta(s[p'])$ will reduce immediately; otherwise, $\Delta(s[p'])$ is able to output immediately. So we take $p = p'$ and then this case is done.

(b) If $\forall s[p] \in dom(\Delta\langle s\rangle)$ we have $\upharpoonright(\Delta(s), p) = \varepsilon$, which implies that $\forall s[p] \in dom(\Delta\langle s\rangle)$ we have $Transform(\mathbf{G}, p) = \Delta(s[p])$. Then we have the following cases:
   1. Case $\mathbf{G} = p_1 \to p_2 : \widetilde{S} \vee F; \mathbf{G}'$ or $\mathbf{G} = p_1 \to p_2 : \widetilde{S}; \mathbf{G}'$ or $\mathbf{G} = p_1 \to p_2 : F; \mathbf{G}'$. We take p as the sender in the first interaciton of $\mathbf{G}$, i.e. $p = p_1$.
   2. Case $\mathbf{G} = \mathbf{T}[\mathbf{G}']\mathbf{H}[h]; \mathbf{G}''$ and $\mathbf{G}' \neq \varepsilon$. We take p as the sender in the first interaction in $\mathbf{G}'$.
   3. Case $\mathbf{G} = \mu t.\mathbf{G}'$. We take p as the sender in the first interaction in $\mathbf{G}'$.
   4. Case $\mathbf{G} = \mathbf{T}[\varepsilon]\mathbf{H}[h]; \mathbf{G}'$. We take p as the sender in the first interaction in $\mathbf{G}'$.

   Then we conclude this case because we find $s[p]$ is able to reduce immediately.

□

Lemma 1. Assume Δ is not end-only and that $\Delta\langle s\rangle$ and $\Delta'\langle s\rangle$ are coherent and $dom(\Delta'\langle s\rangle) \subset dom(\Delta\langle s\rangle)$ and $dom(\Delta\langle s\rangle) \setminus dom(\Delta'\langle s\rangle) \neq \emptyset$. Then (1) $\forall s[p] \in dom(\Delta'\langle s\rangle)$, $s[p]$ does not interact with $\forall s[p'] \in dom(\Delta\langle s\rangle) \setminus dom(\Delta'\langle s\rangle)$, and (2) there exists $s[p] \in dom(\Delta\langle s\rangle)$ such that $\Delta = \Delta_0, s[p] : \mathbf{T} \rightharpoonup \Delta_0', s[p] : \mathbf{T}'$ for some Δ_0, Δ_0' and $\mathbf{T} \neq \mathbf{T}'$.

Proof. By Definition 27, without loss of generality, we assume there are coherent $\mathbf{G} \neq \mathsf{end}$ and $\mathbf{G}'$ such that $pid(\mathbf{G}) = \{p \mid \ s[p] \in dom(\Delta\langle s\rangle)\}$ and $pid(\mathbf{G}') = \{p \mid \ s[p] \in dom(\Delta'\langle s\rangle)\}$ and $\forall s[p] \in dom(\Delta\langle s\rangle) \setminus dom(\Delta'\langle s\rangle)$, we have $Transform(\mathbf{G}, p) = \Delta(s[p])- \upharpoonright (\Delta(s), p)$.

For any $s[p] \in dom(\Delta'\langle s\rangle)$, she will not interact with any $s[p'] \in dom(\Delta\langle s\rangle) \setminus dom(\Delta'\langle s\rangle)$ because, if $s[p]$ interacts with such $s[p']$, we should have $p \to p'$ or $p' \to p$ appear in $\mathbf{G}'$, which implies that $p' \in pid(\mathbf{G}')$ and leads to a contradiction. In other words, the interactions in $dom(\Delta\langle s\rangle) \setminus dom(\Delta'\langle s\rangle)$ are not related to those in $dom(\Delta'\langle s\rangle)$.

Then the proof is similar to the one in Proposition 1. We have the following cases:

(a) If $\Delta(s) \neq \varepsilon$, then without loss of generality there exists p' such that a message heading to p' is on the top of $\Delta(s)$, then by assumption, $\forall s[p'] \in dom(\Delta\langle s\rangle) \setminus dom(\Delta'\langle s\rangle)$, we have $Transform(\mathbf{G}, p') = \Delta(s[p']) - \restriction(\Delta(s), p')$ defined, and $\forall s[p'] \in dom(\Delta'\langle s\rangle)$, we have $Transform(\mathbf{G}', p') = \Delta(s[p']) - \restriction(\Delta(s), p')$ defined; no matter $s[p']$ is in which domain, $\Delta(s[p'])$ must be able to receive this top message at the end. With the same reasoning as the one in Proposition 1 (a), we take $p = p'$ and then this case is done.

(b) If $\forall s[p] \in dom(\Delta\langle s\rangle)$ we have $\restriction(\Delta(s), p) = \varepsilon$, which implies $\forall s[p] \in dom(\Delta\langle s\rangle) \setminus dom(\Delta'\langle s\rangle)$ we have $Transform(\mathbf{G}, p) = \Delta(s[p])$ and $\forall s[p] \in dom(\Delta'\langle s\rangle)$ we have $Transform(\mathbf{G}', p) = \Delta(s[p])$. As long as $\mathbf{G}' \neq \mathsf{end}$, by Proposition 1, there exists $s[p] \in dom(\Delta'\langle s\rangle)$ will reduce; otherwise, we check every possible $\mathbf{G} \neq \mathsf{end}$ by excluding those participants appearing in $\mathbf{G}'$ through examing Cases 1 to 5:

1. Case $\mathbf{G} = p_1 \to p_2 : \widetilde{S} \vee F; \mathbf{G}''$ or $\mathbf{G} = p_1 \to p_2 : \widetilde{S}; \mathbf{G}''$ or $\mathbf{G} = p_1 \to p_2 : F; \mathbf{G}''$. If $p_1 \in pid(\mathbf{G}')$, we check $\mathbf{G}''$. Otherwise we take $p = p_1$.
2. Case $\mathbf{G} = \mathbf{T}[\mathbf{G}'']\mathbf{H}[h]; \mathbf{G}'''$ and $\mathbf{G}'' \neq \varepsilon$. If $pid(\mathbf{G}'') \subseteq pid(\mathbf{G}')$, then we check $\mathbf{G}'''$. Otherwise we check $\mathbf{G}''$.
3. Case $\mathbf{G} = \mu t.\mathbf{G}''$. We check $\mathbf{G}''$.
4. Case $\mathbf{G} = \mathbf{T}[\varepsilon]\mathbf{H}[h]; \mathbf{G}''$. We check $\mathbf{G}''$.

Since $dom(\Delta\langle s\rangle) \setminus dom(\Delta'\langle s\rangle) \neq \emptyset$ implies that $pid(\mathbf{G}) \setminus pid(\mathbf{G}') \neq \emptyset$, as all cases will finally be analysed by Case 1, we know there is some $p'' \to p'''$ appearing in $\mathbf{G}''$ such that $p'', p''' \in pid(\mathbf{G}) \setminus pid(\mathbf{G}')$. We take $p = p''$, then we conclude this case because $s[p]$ is able to reduce immediately.

□

Lemma 2. Assume Δ is not end-only and $\Delta\langle s\rangle$ is coherent. Then there exists $s[p] \in dom(\Delta\langle s\rangle)$ such that $\Delta = \Delta_0, s[p] : \mathbf{T} \rightharpoonup \Delta_0', s[p] : \mathbf{T}'$ for some Δ_0, Δ_0' and $\mathbf{T} \neq \mathbf{T}'$.

Proof. It is proved directly by Definition 27, Proposition 1, and Lemma 1. □

Lemma 3. Assume Δ is not end-only and is coherent. Then there exists $s[p] \in dom(\Delta)$ such that $\Delta = \Delta_0, s[p] : \mathbf{T} \rightharpoonup \Delta_0', s[p] : \mathbf{T}'$ and $\mathbf{T} \neq \mathbf{T}'$.

Proof. It is proved directly by Definition 27 and Lemma 2. □

To conveniently describe the action/interaction which is able to fire immediately, we define $\mathscr{F}$ on interaction types:

$$\mathscr{F} ::= [\,] \mid \mathbf{T}[\mathscr{F}]\mathbf{H}[h] \mid \mathscr{F}; g \mid \mu t.\mathscr{F}$$

It will be used in the following proofs.

Theorem 3. If Δ is not end-only and coherent, then $\Delta \rightharpoonup \Delta'$ and Δ' is coherent.

Proof. It is proved by mechanically checking every rule in Figure 8. Many are done by Definition 27 (Coherence), Definition 22 (Heading-to-p) and Definition 23 (Session Remainder).

Since those participants who appear in Δ will never disappear after Δ reduces, and only those in $\Delta\langle s\rangle$ are affected by the reduction according to rules in Figure 8, we only need to consider the possible changes of those in $\Delta\langle s\rangle$.

1. Case [R-ε]. Let $\Delta = \Delta_0, s[p] : \varepsilon \dashrightarrow \mathbf{T}$. The immediate reduction is at channel $s[p]$. By [R-ε], we have $\Delta' = \Delta_0, s[p] : \mathbf{T}$.

   By Definition 27 and Definition 23 and the fact that the global queue of s is not affected by the reduction, we have

   $$\begin{array}{rcl} \mathit{Transform}(\mathbf{G},p) & = & \Delta(s[p])\!\upharpoonright\!(\Delta(s),p) = (\varepsilon \dashrightarrow \mathbf{T})\!\upharpoonright\!(\Delta(s),p) \\ & \equiv_{type} & \mathbf{T}\!\upharpoonright\!(\Delta(s),p) \qquad\qquad \textit{By Definition 23 or 25} \\ & = & \Delta'(s[p])\!\upharpoonright\!(\Delta(s),p) = \Delta'(s[p])\!\upharpoonright\!(\Delta'(s),p) \end{array}$$

   Since for any $s[p'] \in \mathit{dom}(\Delta), p' \neq p$ is not affected by the reduction, we have

   $$\mathit{Transform}(\mathbf{G},p') = \Delta(s[p'])\!\upharpoonright\!(\Delta(s),p') = \Delta'(s[p'])\!\upharpoonright\!(\Delta'(s),p')$$

   So there exists $\mathbf{G}' = \mathbf{G}$ to totally guide Δ' and thus Δ' is coherent.

2. Case [R-snd]. Let $\Delta = \Delta_0, s[p_1] : sn\langle p_2!\,\widetilde{S}\rangle \dashrightarrow \mathbf{T}, s : \mathbf{q}$ and $\widetilde{S} \neq \emptyset$ and Δ be coherent. The immediate reduction is at channel $s[p_1]$.

   By Definition 27, there exists coherent $\mathbf{G}$ to guide p_1 such that

   $$\begin{array}{rcll} \mathit{Transform}(\mathbf{G},p_1) & = & \Delta(s[p_1])\!\upharpoonright\!(\mathbf{q},p_1) & \\ & = & (sn\langle p_2!\,\widetilde{S}\rangle \dashrightarrow \mathbf{T})\!\upharpoonright\!(\mathbf{q},p_1) & \\ & = & sn\langle p_2!\,\widetilde{S}\rangle \dashrightarrow (\mathbf{T}\!\upharpoonright\!(\mathbf{q},p_1)) \textit{ by Definition 23} & (3) \end{array}$$

   By Definition 18 and Definition 2 (Projection) and Eq. (3), we know $\mathit{Transform}(\mathbf{G},p_1)$ is generated from $\mathbf{G}$, which has the following shape:

   $$\mathbf{G} = p_1 \to p_2 : \widetilde{S}; \mathbf{G}' \tag{4}$$

   Thus by Eq. (3) and Eq. (4) we have

   $$\begin{array}{rcll} \mathit{Transform}(\mathbf{G},p_1) & = & \mathit{Transform}(p_1 \to p_2 : \widetilde{S}; \mathbf{G}', p_1) & \\ & = & sn\langle p_2!\,\widetilde{S}\rangle \dashrightarrow \mathit{Transform}(\mathbf{G}', p_1) & \\ & = & sn\langle p_2!\,\widetilde{S}\rangle \dashrightarrow (\mathbf{T}\!\upharpoonright\!(\mathbf{q},p_1)) & (5) \end{array}$$

   Note that by Definition 17, no synchronisation point is added when we compute $\mathit{Transform}(p_1 \to p_2 : \widetilde{S}, p)$; in other words, if there is a synchronisation point in $\mathbf{T}$,

it is from $\mathit{Transform}(\mathbf{G}',p)$ but not from $\mathit{Transform}(p_1 \to p_2 : \widetilde{S},p)$.

Since the output action is heading to p_2 in session s, we have $s[p_2] \in \mathit{dom}(\Delta_0) \subset \mathit{dom}(\Delta)$. By Definition 27, we have

$$\begin{aligned} & \mathit{Transform}(\mathbf{G},p_2) \\ = \ & \Delta(s[p_2]) \upharpoonright (\mathbf{q},p_2) \\ = \ & \mathit{Transform}(p_1 \to p_2 : \widetilde{S};\mathbf{G}',p_2) \\ = \ & rn\langle p_1 ? \widetilde{S}\rangle \dashrightarrow \mathit{Transform}(\mathbf{G}',p_2) \end{aligned} \tag{6}$$

By [R-snd], we have

$$\Delta' = \Delta_0, s[p_1] : \mathbf{T},\ s : \mathbf{q} \cdot \langle p_1, p_2, \langle \widetilde{S} \rangle \rangle \tag{7}$$

Then we have

$$\begin{aligned} & \Delta'(s[p_1]) \upharpoonright (\mathbf{q} \cdot \langle p_1, p_2, \langle \widetilde{S} \rangle \rangle, p_1) \\ = \ & \mathbf{T} \upharpoonright (\mathbf{q}, p_1) \quad \textit{By Definition 23} \\ = \ & \mathit{Transform}(\mathbf{G}', p_1) \quad \textit{By Eq. (5)} \end{aligned}$$

Similarly, we have

$$\begin{aligned} & \Delta'(s[p_2]) \upharpoonright (\mathbf{q} \cdot \langle p_1, p_2, \langle \widetilde{S} \rangle \rangle, p_2) \\ = \ & \Delta(s[p_2]) \upharpoonright (\mathbf{q}, p_2) \cdot \upharpoonright (\langle p_1, p_2, \langle \widetilde{S} \rangle \rangle, p_2) \\ = \ & (rn\langle p_1 ? \widetilde{S}\rangle \dashrightarrow \mathit{Transform}(\mathbf{G}', p_2)) \upharpoonright (\langle p_1, p_2, \langle \widetilde{S} \rangle \rangle, p_2) \ \textit{By Eq. (6)} \\ = \ & \mathit{Transform}(\mathbf{G}', p_2) \qquad \textit{By Definition 23} \end{aligned} \tag{8}$$

For other $s[p] \in \mathit{dom}(\Delta_0) \subset \mathit{dom}(\Delta), p \neq p_1$ and $p \neq p_2$, since they are not affected by the reduction, we have

$$\begin{aligned} & \Delta'(s[p]) \upharpoonright (\mathbf{q} \cdot \langle p_1, p_2, \langle \widetilde{S} \rangle \rangle, p) \\ = \ & \Delta(s[p]) \upharpoonright (\mathbf{q}, p) \quad \textit{By Definition 23} \\ = \ & \mathit{Transform}(\mathbf{G}, p) \quad \textit{By Definition 27} \\ = \ & \mathit{Transform}(\mathbf{G}', p) \quad \textit{By Definition 2} \end{aligned} \tag{9}$$

So there exists $\mathbf{G}'$ to totally guide Δ' and thus Δ' is coherent.

3. Case [R-sndthw]. Let $\Delta = \Delta_0, s[p_1] : \mathit{try}\{sn\langle p_2 ! \widetilde{S} \vee F, \widetilde{p}', \widetilde{p}'' \rangle \dashrightarrow \mathbf{T}, \mathbf{H}\}, s : \mathbf{q}$ and $F \neq \emptyset$ and $\widetilde{p}' = p_2, p_1', ..., p_m'$ and $\widetilde{p}'' = p_1'', ..., p_n''$ and Δ be coherent. The immediate reduction is at channel $s[p_1]$.

   Note that by Definition 17 $s[p_1]$ is ready to take a failure-raising output implies that in $\mathbf{q}$ there is no failure notifications to trigger $\mathbf{H}$ because a failure-raising interaction

will fire only when globally it is the first failure-raising interaction in the session or its previous interactions, which are tagged with failures in the same handling environment (i.e. H), do not raise a failure. In either case, there is no failure in $\mathbf{q}$ heading to the current sender.

By Definition 27, there exists a coherent $\mathbf{G}$ to guide p_1 such that

$$\begin{aligned} \mathit{Transform}(\mathbf{G}, p_1) &= \Delta(s[p_1]) - \restriction(\mathbf{q}, p_1) \\ &= (\mathit{try}\{sn\langle p_2 ! \widetilde{S} \vee F, \widetilde{p}', \widetilde{p}''\rangle \dashrightarrow \mathbf{T}, \mathbf{H}\}) - \restriction(\mathbf{q}, p_1) \end{aligned} \tag{10}$$

By Definition 18 and Definition 2 (Projection) and Eq.(10), we know $\mathit{Transform}(\mathbf{G}, p_1)$ is generated from $\mathbf{G}$; thus we deduce $\mathbf{G}$ should have the shape:

$$\mathbf{G} = \mathbf{T}[p_1 \to p_2 : \widetilde{S} \vee F; \mathbf{G}']\mathbf{H}[h] \tag{11}$$

where $\mathit{dom}(h) = \mathit{dom}(\mathbf{H})$. Thus by Eq. (10) and Eq.(11) we have

$$\begin{aligned} \mathit{Transform}(\mathbf{G}, p_1) &= \mathit{Transform}(\mathbf{T}[p_1 \to p_2 : \widetilde{S} \vee F; \mathbf{G}']\mathbf{H}[h], p_1) \\ &= (\mathit{try}\{sn\langle p_2 ! \widetilde{S} \vee F, \widetilde{p}', \widetilde{p}''\rangle \dashrightarrow \mathbf{T}, \mathbf{H}\}) - \restriction(\mathbf{q}, p_1) \end{aligned} \tag{12}$$

Since the output action is heading to p_2 in session s, we have $s[p_2] \in \mathit{dom}(\Delta_0) \subset \mathit{dom}(\Delta)$. By Definition 27, we have

$$\mathit{Transform}(\mathbf{G}, p_2) = \Delta(s[p_2]) - \restriction(\mathbf{q}, p_2)$$

By Eq.(11), we have

$$\begin{aligned} \mathit{Transform}(\mathbf{G}, p_2) &= \mathit{Transform}(\mathbf{T}[p_1 \to p_2 : \widetilde{S} \vee F; \mathbf{G}']\mathbf{H}[h], p_2) \\ &= \mathit{try}\{\mathit{yield}\langle F\rangle \dashrightarrow \mathit{rn}\langle p_1 ? \widetilde{S} \vee F\rangle \dashrightarrow \mathit{Transform}(\mathbf{G}', p_2), \mathbf{H}'\} \\ &= \Delta(s[p_2]) - \restriction(\mathbf{q}, p_2) \quad \textit{By Definition 17} \end{aligned} \tag{13}$$

where $\mathit{dom}(\mathbf{H}') = \mathit{dom}(\mathbf{H})$.

By [R-sndthw], we have

$$\Delta' = \Delta_0, s[p_1] : \mathit{try}\{\mathbf{T}, \mathbf{H}\}, s : \mathbf{q} \cdot \mathbf{q}'' \tag{14}$$

We have the following cases:

(a) $\widetilde{S} \neq \emptyset$ and $\mathbf{q}'' = \langle p_1, p_2, \langle \widetilde{S} \rangle\rangle^F \cdot \langle\langle p_1'', F\rangle\rangle \ldots \langle\langle p_n'', F\rangle\rangle$ and we have the following cases:

i $p_1 \notin \{p_1'', \ldots, p_n''\}$. Then we have:

$$\begin{aligned} & \mathit{Transform}(\mathbf{G}, p_1) \\ &= \mathit{Transform}(\mathbf{T}[p_1 \to p_2 : \widetilde{S} \vee F; \mathbf{G}']\mathbf{H}[h], p_1) \\ &= \mathit{try}\{sn\langle p_2 ! \widetilde{S} \vee F, \widetilde{p}', \widetilde{p}''\rangle \dashrightarrow \mathit{Transform}(\mathbf{G}', p_1), \mathbf{H}\} \\ &= \mathit{try}\{sn\langle p_2 ! \widetilde{S} \vee F, \widetilde{p}', \widetilde{p}''\rangle \dashrightarrow \mathbf{T}, \mathbf{H}\} - \restriction(\mathbf{q}, p_1) \;\; \textit{By Eq.(12)} \end{aligned} \tag{15}$$

where no synchronisation point is added between $sn\langle p_2!\ \widetilde{S}\vee F,\widetilde{p}',\widetilde{p}''\rangle$ and $Transform(\mathbf{G}',p_1)$.

Moreover, since no messages in $\mathbf{q}''$ are for p_1, we have

$$\begin{aligned}
&\Delta'(s[p_1])-\restriction(\mathbf{q}\cdot\mathbf{q}'',p_1)\\
&= try\{\mathbf{T},\mathbf{H}\}-\restriction(\mathbf{q}\cdot\mathbf{q}'',p_1)\\
&= try\{\mathbf{T},\mathbf{H}\}-\restriction(\mathbf{q},p_1) && \textit{By Definition 23}\\
&= try\{Transform(\mathbf{G}',p_1),\mathbf{H}\} && \textit{By Eq. (15)}\\
&= Transform(\mathbf{T}[\mathbf{G}']\mathbf{H}[h],p_1) && (16)
\end{aligned}$$

Again, as we noted at the beginning, in $\mathbf{q}$ there is no message who carries a failure notification to trigger $\mathbf{H}$.

ii $p_1\in\{p_1'',...,p_n''\}$. By Definition 17, we have:

$$\begin{aligned}
&Transform(\mathbf{G},p_1)\\
&= Transform(\mathbf{T}[p_1\rightarrow p_2:\widetilde{S}\vee F;\mathbf{G}']\mathbf{H}[h],p_1)\\
&= try\{sn\langle p_2!\ \widetilde{S}\vee F,\widetilde{p}',\widetilde{p}''\rangle \dashrightarrow yield\langle F\rangle \dashrightarrow Transform(\mathbf{G}',p_1),\mathbf{H}\}\\
&= try\{sn\langle p_2!\ \widetilde{S}\vee F,\widetilde{p}',\widetilde{p}''\rangle \dashrightarrow \mathbf{T},\mathbf{H}\}-\restriction(\mathbf{q},p_1) \quad \textit{By Eq.(12)} && (17)
\end{aligned}$$

So we have

$$\begin{aligned}
&\Delta'(s[p_1])-\restriction(\mathbf{q}\cdot\mathbf{q}'',p_1)\\
&= try\{\mathbf{T},\mathbf{H}\}-\restriction(\mathbf{q}\cdot\mathbf{q}'',p_1)\\
&= try\{yield\langle F\rangle \dashrightarrow Transform(\mathbf{G}',p_1),\mathbf{H}\}-\restriction(\mathbf{q}'',p_1) \quad \textit{By Eq. (17)}\\
&= try\{Transform(\mathbf{G}',p_1),\mathbf{H}\} && \textit{By Definition 23}\\
&= Transform(\mathbf{T}[\mathbf{G}']\mathbf{H}[h],p_1) && (18)
\end{aligned}$$

The third equivalent equation is done because we have $\langle\langle p_1,F\rangle\rangle\in\mathbf{q}''$ and other messages in $\mathbf{q}''$ are not for p_1.

For p_2, since $\Delta'(s[p_2])=\Delta_0(s[p_2])=\Delta(s[p_2])$, by Eq. 13 and Definition 17, we have

$$\begin{aligned}
&\Delta'(s[p_2])-\restriction(\mathbf{q}\cdot\mathbf{q}'',p_2)\\
&= (\Delta(s[p_2])-\restriction(\mathbf{q},p_2))-\restriction(\mathbf{q}'',p_2)\\
&= try\{(yield\langle F\rangle \dashrightarrow rn\langle p_1?\ \widetilde{S}\vee F\rangle \dashrightarrow Transform(\mathbf{G}',p_2)),\mathbf{H}'\}-\restriction(\mathbf{q}'',p_2) \quad \textit{By Eq. (13)}\\
&= try\{(Transform(\mathbf{G}',p_2),\mathbf{H}'\} && \textit{By Definition 23}\\
&= Transform(\mathbf{T}[\mathbf{G}']\mathbf{H}[h],p_2) && (19)
\end{aligned}$$

The third equivalent equation is done by the fact that $\langle p_1,p_2,\widetilde{\langle S\rangle}\rangle^F\in\mathbf{q}''$ and $\langle\langle p_2,F\rangle\rangle\in\mathbf{q}''$ and, except them, all other messages in $\mathbf{q}''$ are not heading to p_2.

iii For other $s[p] \in dom(\Delta_0) \subset dom(\Delta), p \neq p_1$ and $p \neq p_2$, by Definition 27, we have $\Delta(s[p]) = \Delta_0(s[p]) = \Delta'(s[p])$ and $Transform(\mathbf{G}, p) = \Delta(s[p]) - \restriction(\mathbf{q}, p)$. We consider the following cases:

– If $p \notin \{p''_1, ..., p''_n\}$, then we have

$$\begin{aligned}
\Delta'(s[p]) - \restriction(\mathbf{q} \cdot \mathbf{q}'', p) &= \Delta(s[p]) - \restriction(\mathbf{q} \cdot \mathbf{q}'', p) \\
&= (\Delta(s[p]) - \restriction(\mathbf{q}, p)) - \restriction(\mathbf{q}'', p) \\
&= \Delta(s[p]) - \restriction(\mathbf{q}, p) \quad \textit{By Definition 23} \\
&= Transform(\mathbf{G}, p) \quad \textit{By Definition 27} \\
&= \varepsilon \dashrightarrow Transform(\mathbf{T}[\mathbf{G}']\mathbf{H}[h], p) \quad \textit{By Definition 2} \\
&\equiv_{type} Transform(\mathbf{T}[\mathbf{G}']\mathbf{H}[h], p) \qquad (20)
\end{aligned}$$

– If $p \in \{p''_1, ..., p''_n\}$, then, by Definition 17, we have:

$$\begin{aligned}
& Transform(\mathbf{G}, p) \\
= {} & Transform(\mathbf{T}[p_1 \rightarrow p_2 : \widetilde{S} \vee F; \mathbf{G}']\mathbf{H}[h], p) \\
= {} & try\{\varepsilon_F \dashrightarrow yield\langle F \rangle \dashrightarrow Transform(\mathbf{G}', p), \mathbf{H}\} \\
= {} & \Delta(s[p]) - \restriction(\mathbf{q}, p) \quad \textit{By Eq.(12)} \qquad (21)
\end{aligned}$$

So we have

$$\begin{aligned}
& \Delta'(s[p]) - \restriction(\mathbf{q} \cdot \mathbf{q}'', p) \\
= {} & \Delta(s[p]) - \restriction(\mathbf{q}, p) - \restriction(\mathbf{q}'', p) \\
= {} & try\{\varepsilon_F \dashrightarrow yield\langle F \rangle \dashrightarrow Transform(\mathbf{G}', p), \mathbf{H}\} - \restriction(\mathbf{q}'', p) \quad \textit{By Eq. (25)} \\
= {} & try\{Transform(\mathbf{G}', p), \mathbf{H}\} \quad \textit{By Definition 23} \\
= {} & Transform(\mathbf{T}[\mathbf{G}']\mathbf{H}[h], p) \qquad (22)
\end{aligned}$$

The third equivalent equation is done because we have $\langle\langle p, F \rangle\rangle \in \mathbf{q}''$ and other messages in $\mathbf{q}''$ are not for p.

Overall, for Δ' there exists a coherent $\mathbf{T}[\mathbf{G}']\mathbf{H}[h]$ to guide it. Thus Δ' is coherent.

(b) $\mathbf{q}'' = \langle p_2, f \rangle \cdot \langle p'_1, f \rangle ... \langle p'_m, f \rangle$ and we have the following cases:

i $p_1 \notin \{p'_1, ..., p'_m\}$. The result of this case is as same as Case (a).i.

Therefore, p_1 is guided by $\mathbf{T}[\mathbf{G}']\mathbf{H}[h]$.

ii $p_1 \in \{p'_1,...,p'_m\}$. So we have $\langle p_1, f\rangle \in \mathbf{q}''$ heading to p_1. Let $h(f) = \mathbf{G}''$. Then we have

$$\begin{aligned}
&\Delta'(s[p_1])- \restriction(\mathbf{q}\cdot\mathbf{q}'',p_1)\\
&= try\{\mathbf{T},\mathbf{H}\}- \restriction(\mathbf{q}\cdot\mathbf{q}'',p_1)\\
&= \mathbf{H}(f)\\
&= Transform(\mathbf{G}'',p_1) \qquad \textit{By Definition 23 [rm-hdl]} \qquad (23)
\end{aligned}$$

For p_2, with the simiar reasoning above, we have

$$\begin{aligned}
&\Delta'(s[p_2])- \restriction(\mathbf{q}\cdot\mathbf{q}'',p_2)\\
&= (\Delta(s[p_2])- \restriction(\mathbf{q},p_2))- \restriction(\mathbf{q}'',p_2)\\
&= try\{yield\langle F\rangle \dashrightarrow rn\langle p_1?\,\widetilde{S}\vee F\rangle \dashrightarrow Transform(\mathbf{G}',p_2),\mathbf{H}'\}- \restriction(\mathbf{q}'',p_2) \quad \textit{By Eq. (13)}\\
&= \mathbf{H}'(f) \qquad \textit{By Definition 23}\\
&= Transform(\mathbf{G}'',p_2) \qquad (24)
\end{aligned}$$

For other $s[p] \in dom(\Delta_0) \subset dom(\Delta), p \neq p_1$ and $p \neq p_2$, by Definition 27, we have $\Delta(s[p]) = \Delta_0(s[p]) = \Delta'(s[p])$ and $Transform(\mathbf{G},p) = \Delta(s[p])- \restriction(\mathbf{q},p)$. We consider the following cases:

- If $p \notin \{p'_1,...,p'_m\}$, then we have the same result as the first case in Case (a).iii.

- If $p \in \{p'_1,...,p'_m\}$, then, by Definition 17, we have:

$$\begin{aligned}
&Transform(\mathbf{G},p)\\
&= Transform(\mathbf{T}[p_1 \rightarrow p_2 : \widetilde{S}\vee F;\mathbf{G}']\mathbf{H}[h],p)\\
&= try\{\varepsilon_F \dashrightarrow yield\langle F\rangle \dashrightarrow Transform(\mathbf{G}',p),\mathbf{H}\}\\
&= \Delta(s[p])- \restriction(\mathbf{q},p) \quad \textit{By Eq.(12)} \qquad (25)
\end{aligned}$$

  So we have

$$\begin{aligned}
&\Delta'(s[p])- \restriction(\mathbf{q}\cdot\mathbf{q}'',p)\\
&= \Delta(s[p])- \restriction(\mathbf{q},p)- \restriction(\mathbf{q}'',p)\\
&= try\{\varepsilon_F \dashrightarrow yield\langle F\rangle \dashrightarrow Transform(\mathbf{G}',p),\mathbf{H}\}- \restriction(\mathbf{q}'',p) \quad \textit{By Eq. (25)}\\
&= \mathbf{H}(f) \qquad \textit{By Definition 23}\\
&= Transform(\mathbf{G}'',p) \qquad (26)
\end{aligned}$$

  The third equivalent equation is done because we have $\langle p,f\rangle \in \mathbf{q}''$ and other messages in $\mathbf{q}''$ are not for p.

Overall, for Δ', there exists $\Delta'' \subset \Delta'$ and a coherent $\mathbf{G}''$ to *totally* guide the participants who are in $\{p \mid s[p] \in dom(\Delta'')\} = pid(\mathbf{G}'')$ and there exists a coherent $\mathbf{T}[\mathbf{G}']\mathbf{H}[h]$ to guide other participants who are in $\{p \mid s[p] \in dom(\Delta') \setminus dom(\Delta'')\} = pid(\mathbf{T}[\mathbf{G}']\mathbf{H}[h]) \setminus pid(\mathbf{G}'')$.

Finally, after analysing all cases, we conclude that Δ' is coherent.

4. Case [R-rcv]. Let $\Delta = \Delta_0, s[p_1] : rn\langle p_2?\ \widetilde{S} \vee F\rangle \dashrightarrow \mathbf{T},\ s : \langle p_2, p_1, \langle\widetilde{S}\rangle\rangle^F \cdot \mathbf{q}$ and Δ be coherent.

   By Definition 27, there exists a coherent $\mathbf{G}$ to guide p_1 such that

$$\begin{aligned} \mathit{Transform}(\mathbf{G}, p_1) &= \Delta(s[p_1]) \upharpoonright (\langle p_2, p_1, \langle\widetilde{S}\rangle\rangle^F \cdot \mathbf{q}, p_1) \\ &= (rn\langle p_2?\ \widetilde{S} \vee F\rangle \dashrightarrow \mathbf{T}) \upharpoonright \langle\!\langle p_2, \langle\widetilde{S}\rangle\rangle\!\rangle^F \upharpoonright (\mathbf{q}, p_1) \\ &= \mathbf{T} \upharpoonright (\mathbf{q}, p_1) \qquad \textit{by Definition 23} \end{aligned} \tag{27}$$

   By [R-rcv], we have

$$\Delta' = \Delta_0, s[p_1] : \mathbf{T}, s : \mathbf{q} \tag{28}$$

   Then by Eq. (27) and Eq. (28), we have

$$\Delta'(s[p_1]) \upharpoonright (\mathbf{q}, p_1) = \mathit{Transform}(\mathbf{G}, p_1) \tag{29}$$

   Since the message is sent from p_2 in session s, by Definition 27, we should have $s[p_2] \in \mathit{dom}(\Delta_0) \subset \mathit{dom}(\Delta)$. By Definition 27, we have

$$\begin{aligned} \mathit{Transform}(\mathbf{G}, p_2) &= \Delta(s[p_2]) \upharpoonright (\langle p_2, p_1, \langle\widetilde{S}\rangle\rangle^F \cdot \mathbf{q}, p_2) \\ &= \Delta(s[p_2]) \upharpoonright (\mathbf{q}, p_2) \quad \textit{by Definition 23} \\ &= \Delta'(s[p_2]) \upharpoonright (\mathbf{q}, p_2) \end{aligned} \tag{30}$$

   For other $s[p] \in \mathit{dom}(\Delta_0) \subset \mathit{dom}(\Delta), p \neq p_1$ and $p \neq p_2$, by Definition 27, we have $\Delta(s[p]) = \Delta_0(s[p]) = \Delta'(s[p])$ and

$$\begin{aligned} \mathit{Transform}(\mathbf{G}, p) &= \Delta(s[p]) \upharpoonright (\langle p_2, p_1, \langle\widetilde{S}\rangle\rangle^F \cdot \mathbf{q}, p) \\ &= \Delta(s[p]) \upharpoonright (\mathbf{q}, p) \quad \textit{by Definition 23} \\ &= \Delta'(s[p]) \upharpoonright (\mathbf{q}, p) \end{aligned} \tag{31}$$

   So we conclude that there exists $\mathbf{G}' = \mathbf{G}$ to totally guide Δ' and thus Δ' is coherent.

5. Case [R-sync-done]. Let $\Delta = \Delta_0, s[p] : \mathit{yield}\langle F'\rangle \dashrightarrow \mathbf{T},\ s : \langle\langle p, F\rangle\rangle \cdot \mathbf{q}$ and $F' \subseteq F$ and Δ be coherent. The immediate reduction is at channel $s[p]$.

   By Definition 27, there exists a coherent $\mathbf{G}$ to guide p such that

$$\begin{aligned} \mathit{Transform}(\mathbf{G}, p) &= \Delta(s[p]) \upharpoonright (\langle\langle p, F\rangle\rangle \cdot \mathbf{q}, p) \\ &= (\mathit{yield}\langle F'\rangle \dashrightarrow \mathbf{T}) \upharpoonright (\!(F)\!) \upharpoonright (\mathbf{q}, p) \\ &= \mathbf{T} \upharpoonright (\mathbf{q}, p) \quad \textit{by Definition 23} \end{aligned} \tag{32}$$

By [R-sync-done], we have

$$\Delta' = \Delta_0, s[p] : \mathbf{T},\ s : \mathbf{q} \tag{33}$$

Then by Eq. (32) and Eq. (33), we have

$$\Delta'(s[p]) - \restriction(\mathbf{q}, p) = \mathbf{T} - \restriction(\mathbf{q}, p) = \mathit{Transform}(\mathbf{G}, p)$$

For other $s[p'] \in \mathit{dom}(\Delta_0) \subset \mathit{dom}(\Delta), p' \neq p$, by Definition 27, we have $\Delta(s[p']) = \Delta_0(s[p']) = \Delta'(s[p'])$ and

$$\begin{aligned}\mathit{Transform}(\mathbf{G}, p') &= \Delta(s[p']) - \restriction(\langle\langle p, F\rangle\rangle \cdot \mathbf{q}, p') \\ &= \Delta(s[p']) - \restriction(\mathbf{q}, p') \quad \textit{by Definition 23} \\ &= \Delta'(s[p']) - \restriction(\mathbf{q}, p')\end{aligned}$$

So we conclude that there exists $\mathbf{G}' = \mathbf{G}$ to totally guide Δ' and thus Δ' is coherent.

6. Case [R-sync]. Let $\Delta = \Delta_0, s[p] : \mathit{yield}\langle F'\rangle \dashrightarrow \mathbf{T},\ s : \langle\langle p, F\rangle\rangle \cdot \mathbf{q}$ and $F' \setminus F \neq \emptyset$ and Δ be coherent. The immediate reduction is at channel $s[p]$.

   By Definition 27, there exists a coherent $\mathbf{G}$ to guide p such that

$$\begin{aligned}\mathit{Transform}(\mathbf{G}, p) &= \Delta(s[p]) - \restriction(\langle\langle p, F\rangle\rangle \cdot \mathbf{q}, p) \\ &= (\mathit{yield}\langle F'\rangle \dashrightarrow \mathbf{T}) - (\!(F)\!) - \restriction(\mathbf{q}, p) \\ &= (\mathit{yield}\langle F' \setminus F\rangle \dashrightarrow \mathbf{T}) - \restriction(\mathbf{q}, p) \quad \textit{by Definition 23}\end{aligned} \tag{34}$$

   By [R-sync], we have

$$\Delta' = \Delta_0, s[p] : \mathit{yield}\langle F' \setminus F\rangle \dashrightarrow \mathbf{T},\ s : \mathbf{q} \tag{35}$$

   Then by Eq. (34) and Eq. (35), we have

$$\begin{aligned}\Delta'(s[p]) - \restriction(\mathbf{q}, p) &= (\mathit{yield}\langle F' \setminus F\rangle \dashrightarrow \mathbf{T}) - \restriction(\mathbf{q}, p) \\ &= \mathit{Transform}(\mathbf{G}, p)\end{aligned}$$

   For other $s[p'] \in \mathit{dom}(\Delta_0) \subset \mathit{dom}(\Delta), p' \neq p$, by Definition 27, we have $\Delta(s[p']) = \Delta_0(s[p']) = \Delta'(s[p'])$ and

$$\begin{aligned}\mathit{Transform}(\mathbf{G}, p') &= \Delta(s[p']) - \restriction(\langle\langle p, F\rangle\rangle \cdot \mathbf{q}, p') \\ &= \Delta(s[p']) - \restriction(\mathbf{q}, p') \quad \textit{by Definition 23} \\ &= \Delta'(s[p']) - \restriction(\mathbf{q}, p')\end{aligned}$$

   So we conclude that there exists $\mathbf{G}' = \mathbf{G}$ to totally guide Δ' and thus Δ' is coherent.

7. Case [R-try]. Let $\Delta = \Delta_0, s[p] : try\{\mathbf{T},\mathbf{H}\},\ s : \mathbf{q}$ and $\mathbf{q} = \mathbf{M}\cdot\mathbf{q}''$ and $\mathbf{M} \neq \langle p,f\rangle$ for any $f \in dom(\mathbf{H})$ and we know for any s',

$$s'[p] : \mathbf{T}, s' : \mathbf{q} \rightharpoonup s'[p] : \mathbf{T}', s' : \mathbf{q}' \tag{36}$$

and Δ be coherent. The immediate reduction is at channel $s[p]$.

By Definition 27, there exist coherent $\mathbf{G}$ and coherent Δ'' such that

$$pid(\mathbf{G}) = \{p' \mid s'[p'] \in \Delta'', s' = s\}$$

and $dom(\Delta'') \subseteq dom(\Delta)$ and $s[p] \in dom(\Delta'')$ and

$$\begin{aligned} Transform(\mathbf{G},p) &= \Delta''(s[p])\!\upharpoonright\!(\mathbf{q},p) \\ &= \Delta(s[p])\!\upharpoonright\!(\mathbf{q},p) \\ &= try\{\mathbf{T},\mathbf{H}\}\!\upharpoonright\!(\mathbf{q},p) \end{aligned} \tag{37}$$

(a) If $\mathbf{q}''$ has no failure notifications, since no failure will trigger any handler in $\mathbf{H}$, by Definition 23, Eq. (37) becomes

$$Transform(\mathbf{G},p) = try\{\mathbf{T}\!\upharpoonright\!(\mathbf{q},p),\mathbf{H}\} \tag{38}$$

Eq.(38) implies that $\mathbf{G}$ has the shape

$$\mathbf{G} = \mathbf{T}[g]\mathbf{H}[h];\mathsf{end} \tag{39}$$

By Definition 27, we assume there exists Δ_0'' such that $dom(\Delta_0'') \subset dom(\Delta'')$ and $s[p] \notin dom(\Delta_0'')$ and Δ_0'' is coherent and every participants there are guided by some other protocol types. Thus by Definition 27 and Eq. (39), we have

$$\begin{aligned} \forall s[p'] \in dom(\Delta'') \setminus dom(\Delta_0'').Transform(\mathbf{G},p') &= try\{Transform(g;\mathsf{end},p'),\mathbf{H}_{p'}\} \\ &= \Delta''(s[p'])\!\upharpoonright\!(\mathbf{q},p') \\ &= \Delta(s[p'])\!\upharpoonright\!(\mathbf{q},p') \end{aligned} \tag{40}$$

where $\mathbf{H}_{p'} = \{f : Transform(h(f);\mathsf{end},p')\}_{f\in dom(h)}$. The first equivalent equation simply comes from Definition 18 (Transformation).

By Eq. (38) and Eq. (40), we have

$$Transform(g;\mathsf{end},p) = \mathbf{T}\!\upharpoonright\!(\mathbf{q},p) \tag{41}$$

where $g;\mathsf{end}$ is coherent since, by Eq. (39), $\mathbf{T}[g]\mathbf{H}[h];\mathsf{end}$ is coherent. For any $p' \in pid(g;\mathsf{end})$, we have $p' \in pid(\mathbf{G})$ and $s[p'] \in dom(\Delta'')$, and $Transform(g;\mathsf{end},p')$ is defined.

By [R-try], we have $\Delta = \Delta_0, \Delta'' \rightharpoonup \Delta_0, \Delta''' = \Delta'$ such that

$$\Delta''' = \Delta_0', s[p] : try\{\mathbf{T}',\mathbf{H}\}, s : \mathbf{q}'$$

By Eq. (36) and the rules defined in Figure 8 (reduction rules of session environments) and the rules defined in Definition 2 (Projection), we observe:

i. If the reduction is from [R-ε] or [R-try-end], then by Eq. (41) we have

$$\begin{aligned} \mathit{Transform}(g;\mathsf{end},p) = \mathbf{T}-\upharpoonright(\mathbf{q},p) = \mathbf{T}'-\upharpoonright(\mathbf{q}',p) \\ \mathbf{q} = \mathbf{q}' \end{aligned} \tag{42}$$

Thus by Eq. (37) and Eq. (42)

$$\begin{aligned} \Delta''(s[p])-\upharpoonright(\mathbf{q}',p) &= \mathit{Transform}(\mathbf{T}[g]\mathbf{H}[h];\mathsf{end},p) \\ &= \mathit{try}\{\mathit{Transform}(g;\mathsf{end},p),\mathbf{H}\} \\ &= \mathit{try}\{\mathbf{T}-\upharpoonright(\mathbf{q},p),\mathbf{H}\} \\ &= \mathit{try}\{\mathbf{T}'-\upharpoonright(\mathbf{q}',p),\mathbf{H}\} \\ &= \mathit{try}\{\mathbf{T}',\mathbf{H}\}-\upharpoonright(\mathbf{q}',p)\ \textit{By Definition 23} \\ &= \Delta'''(s[p])-\upharpoonright(\mathbf{q}',p) \end{aligned}$$

Other $s[p'] \in \mathit{dom}(\Delta'') \setminus \mathit{dom}(\Delta''_0)$ and $p' \neq p$ are not affected.
So we take $\mathbf{G}' = \mathbf{T}[g]\mathbf{H}[h];\mathsf{end}$ such that $\mathit{pid}(\mathbf{G}') = \{p' \mid s'[p'] \in \mathit{dom}(\Delta'''), s = s'\}$ and $\forall s[p'] \in \mathit{dom}(\Delta''') \setminus \mathit{dom}(\Delta''_0) = \mathit{dom}(\Delta'') \setminus \mathit{dom}(\Delta''_0)$, we have

$$\begin{aligned} \mathit{Transform}(\mathbf{G}',p') &= \mathit{Transform}(\mathbf{G},p') \\ &= \Delta(s[p'])-\upharpoonright(\mathbf{q},p') \\ &= \Delta(s[p'])-\upharpoonright(\mathbf{M}\cdot\mathbf{q}',p')\ \ \textit{By Definition 23} \\ &= \Delta'''(s[p'])-\upharpoonright(\mathbf{q}',p') \end{aligned}$$

Note that, in the third equivalent equation, $\upharpoonright(\mathbf{M},p') = \varepsilon$ because $p' \neq p$ and $\mathbf{M}$ is received by p by assumption.

For those in Δ''_0, as they will not interact with those in Δ'', they are not affected. So, as we take $\mathbf{G}' = \mathbf{T}[g]\mathbf{H}[h];\mathsf{end}$, we conclude Δ''' is coherent.

ii. If the reduction is from an input, by Eq. (41) we have

$$\begin{aligned} \mathit{Transform}(g;\mathsf{end},p) &= \mathbf{T}-\upharpoonright(\mathbf{q},p) \\ &= \mathbf{T}-\upharpoonright(\mathbf{M}\cdot\mathbf{q}',p) \\ &= (\mathbf{T}-\upharpoonright(\mathbf{M},p))-\upharpoonright(\mathbf{q}',p) \\ &= \mathbf{T}'-\upharpoonright(\mathbf{q}',p) \end{aligned} \tag{43}$$

the last equivalent equation is due to the fact that, by Eq. (36), the reduction comes from $\mathbf{T}-\mathbf{M} = \mathbf{T}'$.

Thus by Eq. (37) and Eq. (43)

$$\begin{aligned}\Delta''(s[p])-\upharpoonright(\mathbf{q}',p) &= \mathit{Transform}(\mathbf{T}[g]\mathbf{H}[h];\mathsf{end},p)\\ &= \mathit{try}\{\mathit{Transform}(g;\mathsf{end},p),\mathbf{H}\}\\ &= \mathit{try}\{\mathbf{T}-\upharpoonright(\mathbf{q},p),\mathbf{H}\}\\ &= \mathit{try}\{\mathbf{T}'-\upharpoonright(\mathbf{q}',p),\mathbf{H}\}\\ &= \mathit{try}\{\mathbf{T}',\mathbf{H}\}-\upharpoonright(\mathbf{q}',p)\ \textit{By Definition 23}\\ &= \Delta'''(s[p])-\upharpoonright(\mathbf{q}',p)\end{aligned}$$

The remaining reasoning is as same as the one for Case (a).i.

iii. If the reduction is from an output, let $g = \mathscr{F}[g'';g''']$;$\mathsf{end}$ such that g'' is a singular interaction and has no try-handle structure, then by the rules for output defined in Figure 8 (reduction rules of session environments) and the rules defined in Definition 2 (Projection), we know the reduction comes from an interaction on top (the first interaction ready to fire), which is the sending action in g''. So for the rest interactions (i.e. g''') we have

$$\mathit{Transform}(\mathscr{F}[g''];\mathsf{end},p) = \mathbf{T}'-\upharpoonright(\mathbf{q}',p)\ \ \textit{By Eq. (36)} \tag{44}$$

and $\mathscr{F}[g''']$ is coherent because g, which contains $\mathscr{F}[g''']$, is coherent. Thus by Eq. (44) we have

$$\begin{aligned}\mathit{Transform}(\mathbf{T}[\mathscr{F}[g''']]\mathbf{H}[h];\mathsf{end},p) &= \mathit{try}\{\mathit{Transform}(\mathscr{F}[g''];\mathsf{end},p),\mathbf{H}\}\\ &= \mathit{try}\{\mathbf{T}'-\upharpoonright(\mathbf{q}',p),\mathbf{H}\}\end{aligned} \tag{45}$$

where $\mathbf{T}[\mathscr{F}[g''']]\mathbf{H}[h];\mathsf{end}$ is coherent because $\mathscr{F}[g''']$ is coherent and, by Eq. (39) and Eq. (40), for every participants in h, their handling activities are defined.

Eq. (45) implies that, for any kinds of output (including failure or non-failure notification) from $\mathbf{T}$ heading to p, it will be absorbed by $\mathbf{T}'$.

For every $s[p'] \in \mathit{dom}(\Delta'') \setminus \mathit{dom}(\Delta''_0)$ and $p' \neq p$, by Definition 27, we already know

$$\begin{aligned}\mathit{Transform}(\mathbf{T}[g]\mathbf{H}[h];\mathsf{end},p') &= \mathit{Transform}(\mathbf{T}[\mathscr{F}[g'';g''']]\mathbf{H}[h];\mathsf{end},p')\\ &= \Delta''(s[p'])-\upharpoonright(\mathbf{q},p')\\ &= \mathit{try}\{\mathit{Transform}(\mathscr{F}[g'';g''';\mathsf{end}],p'),\mathbf{H}_{p'}\}\end{aligned}$$

Let $\mathbf{q}''$ be outputed due to g'', say $g'' = p \to p'' : \widetilde{S} \vee F$, and thus $\mathbf{q}' = \mathbf{q}\cdot\mathbf{q}''$.

By Eq. (45), that all failure-/non-failure notifications heading to p are absorbed by $\mathbf{T}'$ implies that all failures are handled in $\mathscr{F}$. So the outer handler $\mathbf{H}_{p'}$, projected from h, will not be triggered.

For those $p' \neq p$ who are related to $\mathbf{q}''$ (i.e. they are the receivers of $\mathbf{q}''$), with the similar reasoning as those in the proofs for Case [R-snd] and Case [R-sndthw], $\Delta''(s[p'])$ should have a shape of $try\{\mathbf{T}_{p'}, \mathbf{H}_{p'}\}$, and $\mathbf{T}_{p'}$ must have an immediate receiving action or synchronisation point or the handler to handle the failure in $\mathbf{q}''$, and they only appear in $\mathbf{q}''$ (because $\mathbf{q}''$ is from one output reduction, see [R-snd] and [R-sndthw]). For those p', we have

$$\begin{aligned}\Delta'''(s[p'])\!-\!\upharpoonright\!(\mathbf{q}',p') &= \Delta''(s[p'])\!-\!\upharpoonright\!(\mathbf{q},p')\!-\!\upharpoonright\!(\mathbf{q}'',p')\\ &= try\{Transform(\mathscr{F}[g'';g''']\mathsf{;end},p'),\mathbf{H}_{p'}\}\!-\!\upharpoonright\!(\mathbf{q}'',p')\\ &= try\{Transform(\mathscr{F}[g''']\mathsf{;end},p'),\mathbf{H}_{p'}\}\\ &= Transform(\mathbf{T}[\mathscr{F}[g''']]\mathbf{H}[h]\mathsf{;end},p') \qquad (46)\end{aligned}$$

where $Transform(\mathscr{F}[g'';g''']\mathsf{;end},p') = \mathbf{T}_{p'}$, which is able to absorb those messages in $\mathbf{q}''$ heading to her. The third equivalent equation is by Definition 23 for receiving those messages heading to her.

For those $p' \neq p$ who are not related to $\mathbf{q}''$, we have

$$\begin{aligned}\Delta'''(s[p'])\!-\!\upharpoonright\!(\mathbf{q}',p') &= \Delta''(s[p'])\!-\!\upharpoonright\!(\mathbf{q},p')\!-\!\upharpoonright\!(\mathbf{q}'',p')\\ &= \Delta''(s[p'])\!-\!\upharpoonright\!(\mathbf{q},p')\\ &= try\{Transform(\mathscr{F}[g'';g''']\mathsf{;end},p'),\mathbf{H}_{p'}\}\\ &\equiv_{type} try\{Transform(\mathscr{F}[g''']\mathsf{;end},p'),\mathbf{H}_{p'}\}\\ &= Transform(\mathbf{T}[\mathscr{F}[g''']]\mathbf{H}[h]\mathsf{;end},p') \qquad (47)\end{aligned}$$

The forth equivalent equation is because, by Definition 2. $p' \notin pid(g'')$.

Overall, we take $\mathbf{G}' = \mathbf{T}[\mathscr{F}[g''']]\mathbf{H}[h]\mathsf{;end}$ and conclude Δ''' is coherent for this case.

(b) If $\mathbf{q}''$ has some failure notification(s), then, since some participants in $dom(\Delta)$ will be triggered by failure notifications for handling activity, Eq.(37) implies that, let $\Delta = \Delta_0$, some participants of s in $dom(\Delta) \setminus dom(\Delta_1)$, where $\Delta_1\langle s\rangle$ is coherent, are guided by some coherent $\mathbf{G}_0$, and some participants in $dom(\Delta_1) \setminus dom(\Delta_2)$, where $\Delta_2\langle s\rangle$ is coherent, are guided by some coherent $\mathbf{G}_1$, and so on, and finally there is $\Delta_n \subseteq \Delta_{n-1}.. \subset \Delta$ such that $\Delta_n\langle s\rangle$ is *totally* guided by some coherent $\mathbf{G}_n$. Let Δ_{n+1} be empty. As $\Delta \rightharpoonup \Delta'$, for some $i \in \{0..n\}$, $\Delta_i \setminus \Delta_{i+1} \rightharpoonup \Delta'_i \setminus \Delta_{i+1}$.

Assume $s[p] \in dom(\Delta_k) \setminus dom(\Delta_{k+1})$ for some $k \in \{0..n\}$. Take $\mathbf{G} = \mathbf{G}_k$.
For $s[p]$ and other participants in $dom(\Delta_k) \setminus dom(\Delta_{k+1})$ who are not affected by $\mathbf{q}''$ (i.e. they are not the receivers of $\mathbf{q}''$), the proof is as same as case (a).

For those $s[p'] \in dom(\Delta_k) \setminus dom(\Delta_{k+1})$ who are the receivers of failure messages in $\mathbf{q}''$, by the proof in Lemma 1, we know they only interact with those in $dom(\Delta_k) \setminus dom(\Delta_{k+1})$. If there is a failure message heading to some $s[p'] \in$

$dom(\Delta_k) \setminus dom(\Delta_{k+1})$, then the failure messages coming from the same failure-raising interaction will only send to those in $dom(\Delta_k) \setminus dom(\Delta_{k+1})$. By Definition 23, there exist $\mathbf{T}''$ and $\mathbf{q}'''$ such that for those in $dom(\Delta_k) \setminus dom(\Delta_{k+1})$ who are the receivers of failures in $\mathbf{q}''$, they have received all failure messages heading to them:

$$Transform(\mathbf{G}_k, p') = \Delta(s[p']) - \restriction(\Delta(s), p') = \mathbf{T}'' - \restriction(\mathbf{q}''', p') \tag{48}$$

where $\mathbf{q}'''$ has no failure messages for p'.

If $\mathbf{T}''$ has a try-handle structure, the proof goes back to this case (i.e. Case [R-try]); otherwise the proof goes to other cases. Thus we can prove that we can find a coherent protocol type to guide those in $dom(\Delta'_k) \setminus dom(\Delta_{k+1})$.

For all other $\mathbf{G}_i, i \in \{0..n\}$, we similarly prove that we can find a coherent protocol type to guide those in $dom(\Delta'_i) \setminus dom(\Delta_{i+1})$.

In summary, we prove that Δ' is coherent.

8. Case [R-hdl]. Let $\Delta = \Delta'_0, s[p] : try\{\mathscr{C}[yield\langle F\rangle \dashrightarrow \mathbf{T}],\ \mathbf{H}\},\ s : \langle p, f\rangle \cdot \mathbf{q}$ and $f \in dom(\mathbf{H}) \cap F$ and Δ be coherent. The immediate reduction is at channel $s[p]$. Let $\mathbf{H}(f) = \mathbf{T}'$.

   Since the failure will trigger handling activity, let $\Delta = \Delta_0$, some participants of s in $dom(\Delta) \setminus dom(\Delta_1)$, where $\Delta_1\langle s\rangle$ is coherent, are guided by some coherent $\mathbf{G}_0$, and some participants in $dom(\Delta_1) \setminus dom(\Delta_2)$, where $\Delta_2\langle s\rangle$ is coherent, are guided by some coherent $\mathbf{G}_1$, and so on, and finally there is $\Delta_n \subseteq \Delta_{n-1} .. \subset \Delta$ such that $\Delta_n\langle s\rangle$ is *totally* guided by some coherent $\mathbf{G}_n$. Let Δ_{n+1} be empty. As $\Delta \rightharpoonup \Delta'$, for some $i \in \{0..n\}$, $\Delta_i \setminus \Delta_{i+1} \rightharpoonup \Delta'_i \setminus \Delta_{i+1}$.

   Follow the assumption above. Assume $s[p] \in dom(\Delta_k) \setminus dom(\Delta_{k+1})$ for some $k \in \{0..n\}$. Then $\mathbf{G} = \mathbf{G}_k$ and

$$\begin{aligned} Transform(\mathbf{G}, p) &= \Delta(s[p]) - \restriction(\langle p, f\rangle \cdot \mathbf{q}, p) \\ &= try\{\mathscr{C}[yield\langle F\rangle \dashrightarrow \mathbf{T}],\ \mathbf{H}\} - \restriction(\langle p, f\rangle \cdot \mathbf{q}, p) \\ &= \mathbf{H}(f) - \restriction(\mathbf{q}, p) \quad \textit{By Definition 23} \\ &= \mathbf{T}' - \restriction(\mathbf{q}, p) \quad \textit{By assumption} \end{aligned} \tag{49}$$

   By [R-hdl], we have

$$\Delta \rightharpoonup \Delta' = \Delta'_0, s[p_1] : \mathbf{T}', s : \mathbf{q} \tag{50}$$

$$\Delta'(s[p]) - \restriction(\mathbf{q}, p) = \mathbf{T}' - \restriction(\mathbf{q}, p) = Transform(\mathbf{G}, p) \tag{51}$$

For other $p' \in dom(\Delta_k) \setminus dom(\Delta_{k+1})$ and $p' \neq p$, we have $\Delta(s[p']) = \Delta_0(s[p']) = \Delta'(s[p'])$ and

$$\begin{aligned} Transform(\mathbf{G}, p') &= \Delta(s[p]) - \restriction(\langle p, f\rangle \cdot \mathbf{q}, p) \\ &= \Delta(s[p]) - \restriction(\mathbf{q}, p) \quad \textit{By Definition 23} \\ &= \Delta'(s[p]) - \restriction(\mathbf{q}, p) \end{aligned} \tag{52}$$

The second equivalent equation is because f will not affect/trigger any other participant who is not p.

Thus for this reduction, we have $\Delta_k \setminus \Delta_{k+1} \rightharpoonup \Delta'_k \setminus \Delta_{k+1}$ and every participants in $\Delta'_k \setminus \Delta_{k+1}$ are guided by $\mathbf{G}$. Therefore $\Delta' = (\Delta \setminus \Delta_k \setminus \Delta_{k+1}) \cup (\Delta'_k \setminus \Delta_{k+1})$ is coherent because for other participants who are not in $dom(\Delta_k) \setminus dom(\Delta_{k+1})$ are not affected.

9. Case [R-try-end]. Let $\Delta = \Delta'_0, s[p] : try\{\mathsf{end}, \mathbf{H}\}$ and Δ be coherent. The immediate reduction is at channel $s[p]$.

   Let $\Delta_k \subseteq \Delta = \Delta_0$ and $s[p] \in dom(\Delta_k)$. By Definition 27, there exists a coherent $\mathbf{G}$ such that

$$\begin{aligned} Transform(\mathbf{G}, p) &= \Delta(s[p]) - \restriction(\mathbf{q}, p) \\ &= try\{\mathsf{end}, \mathbf{H}\} - \restriction(\mathbf{q}, p) \\ &= \mathsf{end} \qquad \textit{By Definition 23 [rm-try-end]} \end{aligned} \tag{53}$$

   By [R-try-end], we have

$$\Delta' = \Delta_0, s[p] : \mathsf{end} \tag{54}$$

   Then by Eq. (54), we have

$$\begin{aligned} \Delta'(s[p]) - \restriction(\mathbf{q}, p) &= \mathsf{end} - \restriction(\mathbf{q}, p) \\ &= \mathsf{end} \qquad \textit{By Definition 23 [rm-end]} \\ &= Transform(\mathbf{G}, p) \qquad \textit{By Eq. (53)} \end{aligned}$$

   So there exists a coherent $\mathbf{G}$ to guide $s[p] \in dom(\Delta'_k) \setminus dom(\Delta_{k+1}) \subseteq \Delta'$. Other participants in Δ (and thus also in Δ') are not affected by this reduction happened in Δ_k. Thus Δ' is coherent.

10. Case [R-res]. Let $\Delta_1, \Delta \rightharpoonup \Delta_2, \Delta$ and $\Delta_1 \rightharpoonup \Delta_2$ and Δ_1, Δ is coherent. By induction hypothesis on $\rightharpoonup$ and every cases we have examined above, if Δ_1 is coherent, then Δ_2 is coherent. As Δ_1 is coherent, we have Δ is coherent because Δ_1, Δ is coherent. Therefore we obtain Δ_2, Δ is also coherent.

□

Property of Subject Reduction Here we prove the property of subject reduction. Recall that we have formally define the binary relation $\mathbf{T} \ni \mathbf{T}$ on local types in Definition 21, Appendix A. We will use it in following lemma and the proof for Theorem 1.

Lemma 4 gives the inversion lemma for typing rules of processes and networks.

Lemma 4 (Inversion Lemma).

1. If $\Gamma \vdash \mathbf{0} \rhd \Delta$, then Δ is end-only.

2. If $\Gamma \vdash s[p_1]?(p_2,(\widetilde{x})^F).P \rhd \Delta$, then $\Delta = \{s[p_1] : rn\langle p_2?\,\widetilde{S} \vee F\rangle \dashrightarrow \mathbf{T}\}$ and $\Gamma, \widetilde{x} : \widetilde{S} \vdash P \rhd s[p_1] : \mathbf{T}$.

3. If $\Gamma \vdash s[p_1]!(p_2,\langle\widetilde{e}\rangle^F);P \rhd \Delta$, then $\Delta = \{s[p_1] : sn\langle p_2!\ \widetilde{S} \vee F,\ \_\ ,\_\ \rangle \dashrightarrow \mathbf{T}\}$ and $\Gamma \vdash \widetilde{e} : \widetilde{S}$ and $\widetilde{S} \neq \emptyset$ and $\Gamma \vdash P \rhd s[p_1] : \mathbf{T}$.

4. If $\Gamma \vdash s[p_1]\ \mathsf{raise}(f)\ \mathsf{to}\ p_2;P \rhd \Delta$, then $\Delta = \{s[p_1] : sn\langle p_2!\ \widetilde{S} \vee F,\ \_\ ,\ \_\ \rangle \dashrightarrow \mathbf{T}\}$ and $f \in F$ and $\Gamma \vdash P \rhd s[p_1] : \mathbf{T}$

5. If $\Gamma \vdash c \otimes F;P \rhd \Delta$, then $\Delta = \{c : yield\langle F\rangle \dashrightarrow \mathbf{T}\}$ and $\Gamma \vdash P \rhd \{c : \mathbf{T}\}$.

6. If $\Gamma \vdash \mathsf{try}\{P\}\mathsf{h}\{H\};P' \rhd \Delta$, then $\forall f \in dom(H)$ we have $f \notin P'$ and $\Delta = \{act(H) : try\{\mathbf{T},\mathbf{H}\}\}$ such that $dom(H) = dom(\mathbf{H})$ and $\Gamma \vdash P;P' \rhd \{act(H) : \mathbf{T}\}$ and $\forall f \in dom(H)$ we have $\Gamma \vdash H(f);P' \rhd \{act(H) : \mathbf{H}(f)\}$.

7. If $\Gamma \vdash P_1;P_2 \rhd \Delta$, then $\Gamma \vdash P_1 \rhd \Delta_1$ and $\Gamma \vdash P_2 \rhd \Delta_2$ such that $\Delta = \Delta_1 \circ \Delta_2$.

8. If $\Gamma \vdash X\langle\tilde{e}\ c\rangle \rhd \Delta$, then $\Delta = \{c : \mathbf{T}\}$ and $X : (\widetilde{S}\ \mathbf{T})$ is in Γ.

9. If $\Gamma \vdash \mathsf{def}\ D\ \mathsf{in}\ P' \rhd \Delta$ and $X(\widetilde{x}\ c) = P \in D$, then $\Gamma, \tilde{x} : \widetilde{S}, X : (\widetilde{S}\ \mathbf{T}) \vdash P \rhd \{c : \mathbf{T}\}$ and $\Gamma, X : (\widetilde{S}\ \mathbf{T}) \vdash P' \rhd \Delta$.

10. If $\Gamma \vdash \mathsf{if}\ e\ \mathsf{then}\ P_1\ \mathsf{else}\ Q \rhd \Delta$, then $\Gamma \vdash e : \mathsf{bool}$ and $\Gamma \vdash P_1 \rhd \Delta$ and $\Gamma \vdash P_2 \rhd \Delta$.

11. If $\Gamma \vdash s : \varepsilon \rhd \Delta$, then $\Delta = \{s : \varepsilon\}$.

12. If $\Gamma \vdash s : q \cdot \langle p_1, p_2, \langle\widetilde{v}\rangle\rangle^F \rhd \Delta$, then $\Delta = \{s : \mathbf{q} \cdot \langle p_1, p_2, \langle\widetilde{S}\rangle\rangle^F\}$ and $\Gamma \vdash \widetilde{v} : \widetilde{S}$ and $\Gamma \vdash s : q \rhd \{s : \mathbf{q}\}$.

13. If $\Gamma \vdash s : q \cdot \langle p, f\rangle \rhd \Delta$, then $\Delta = \{s : \mathbf{q} \cdot \langle p, f\rangle\}$ and $\Gamma \vdash s : q \rhd \{s : \mathbf{q}\}$.

14. If $\Gamma \vdash s : q \cdot \langle\langle p, F\rangle\rangle \rhd \Delta$, then $\Delta = \{s : \mathbf{q} \cdot \langle\langle p, F\rangle\rangle\}$ and $\Gamma \vdash s : q \rhd \{s : \mathbf{q}\}$.

15. If $\Gamma \vdash a[p](y).P \rhd \Delta$, then $\Delta = \emptyset$ and $\Gamma \vdash a : \langle\mathbf{G}\rangle$ and $\Gamma \vdash P \rhd \{y : Transform(\mathbf{G}, p)\}$.

16. If $\Gamma \vdash [P]_{\mathbf{T}} \rhd \Delta$, then $\Gamma \vdash P \rhd \Delta$ and $\mathbf{T} \ni \Delta(act(P))$.

17. If $\Gamma \vdash N_1 \parallel N_2 \rhd \Delta$, then $\Delta = \Delta_1, \Delta_2$ such that $dom(\Delta_1) \cap dom(\Delta_2) = \emptyset$ and $\forall i \in 1,2$ we have $\Gamma \vdash N_i \rhd \Delta_i$.

18. If $\Gamma \vdash (\mathsf{new}\ s)N \rhd \Delta$, then $\Delta = \Delta' \setminus \Delta'\langle s \rangle$ and $\Delta'\langle s \rangle$ is coherent and $\Gamma \vdash N \rhd \Delta'$.

Proof. By induction on derivations. □

Lemma 5 (Types of Queues).

1. If $\Gamma \vdash s : \langle p_1, p_2, \langle \tilde{v} \rangle \rangle^F \cdot q \rhd \Delta$, then $\Delta = \{s : \langle p_1, p_2, \langle \widetilde{S} \rangle \rangle^F \cdot \mathbf{q}\}$ and $\Gamma \vdash \tilde{v} : \widetilde{S}$ and $\Gamma \vdash s : q \rhd \{s : \mathbf{q}\}$.
2. If $\Gamma \vdash s : \langle p, f \rangle \cdot q \rhd \Delta$, then $\Delta = \{s : \langle p, f \rangle \cdot \mathbf{q}\}$ and $\Gamma \vdash s : q \rhd \{s : \mathbf{q}\}$.
3. If $\Gamma \vdash s : \langle\langle p, F \rangle\rangle \cdot q \rhd \Delta$, then $\Delta = \{s : \langle\langle p, F \rangle\rangle \cdot \mathbf{q}\}$ and $\Gamma \vdash s : q \rhd \{s : \mathbf{q}\}$.

Proof. We only prove the first case. Other cases are similar and simpler. By induction on n we show: If $\Gamma \vdash s : \langle p_1, p_2, \langle \tilde{v} \rangle \rangle^F \cdot m_1 ... m_n \rhd \Delta$ where $m_1 ... m_n$ are messages. Then $\Gamma \vdash \tilde{v} : \widetilde{S}$ and

$$\Delta = \{s : \langle p_1, p_2, \langle \widetilde{S} \rangle \rangle^F \cdot \mathbf{M}_1 ... \mathbf{M}_n\}$$

where $\forall i \in 1..n$, $\Gamma \vdash m_i \rhd \mathbf{M}_i$. The first step follows from Lemma 4.11 and then 4.12. The induction step follows from Lemma 4.12, 4.13, and 4.14. □

A substitution lemma is provided for two cases: (1) replacing values to content variables in a process, and (2) replacing a session channel to a channel variable in a network.

Lemma 6 (Substitution Lemma).

(1) If $\Gamma, \tilde{x} : \widetilde{S} \vdash P \rhd \Delta$ and $\Gamma \vdash \tilde{v} : \widetilde{S}$, then $\Gamma \vdash P \rhd \Delta$.
(2) If $\Gamma \vdash N \rhd \Delta, y : \mathbf{T}$ and $s \notin dom(\Delta)$, then $\Gamma \vdash N[s[p]/y] \rhd \Delta, s[p] : \mathbf{T}$.

Proof. 1. For (1), the proof is standard.[21].

2. For (2), the proof is done by induction on the derivation of

$$\Gamma \vdash N \rhd \Delta, y : \mathbf{T}.$$

The only interesting case is:

$$\frac{\Gamma \vdash N \rhd \Delta, y : \mathbf{T} \quad \Delta\langle s' \rangle \text{ coherent}}{\Gamma \vdash (\nu s')N \rhd \Delta \setminus \Delta\langle s' \rangle, y : \mathbf{T}}$$

By induction we have

$$\Gamma \vdash N[s[p]/y] \rhd \Delta, s[p] : \mathbf{T}$$

and $\Delta\langle s' \rangle$ remains coherent. Thus by [T-new], we conclude

$$\Gamma \vdash (\nu s')N \rhd \Delta \setminus \Delta\langle s' \rangle, s[p] : \mathbf{T}.$$

□

Theorem 1 (1) Preservation of Structural Congruence If $\Gamma \vdash N \rhd \Delta$ and $N \equiv N'$, then $\Gamma \vdash N' \rhd \Delta$.

Proof. We list the interesting cases for the rules in Figure 7 (Structural Congruence for Networks) in Section 3. The proof is done by induction on $\equiv$.

1. Case $N \parallel [\mathbf{0}]_{\mathbf{T}} \equiv N$. Let

$$\Gamma \vdash N \rhd \Delta, \tag{55}$$

$$\Gamma \vdash [\mathbf{0}]_{\mathbf{T}} \rhd \Delta'. \tag{56}$$

   By applying Lemma 4.16 and Lemma 4.1 to Eq. (56), we get that Δ' is end-only. By applying rule [T-net] to Eq. (55) and Eq. (56), we get

$$\Gamma \vdash N \parallel [\mathbf{0}]_{\mathbf{T}} \rhd \Delta, \Delta'.$$

   Since $\Delta, \Delta' \equiv \Delta$ (because Δ' is end-only), we conclude $\Gamma \vdash N \parallel [\mathbf{0}]_{\mathbf{T}} \rhd \Delta$.

   For the converse direction, let

$$\Gamma \vdash N \parallel [\mathbf{0}]_{\mathbf{T}} \rhd \Delta. \tag{57}$$

   By applying Lemma 4.17 to Eq. (57), we get

$$\Gamma \vdash N \rhd \Delta_1,$$

$$\Gamma \vdash [\mathbf{0}]_{\mathbf{T}} \rhd \Delta_2 \tag{58}$$

   where $\Delta_1, \Delta_2 = \Delta$. By applying Lemma 4.16 and Lemma 4.1 to Eq. (58), we have that Δ_2 is end-only. Since $\Delta_1, \Delta_2 \equiv \Delta$, we conclude $\Gamma \vdash N \parallel [\mathbf{0}]_{\mathbf{T}} \rhd \Delta$.

   The proof for case $[\mathbf{0}]_{\mathbf{T}} \parallel N \equiv N$ is similar.

2. Case $N_1 \parallel N_2 \equiv N_2 \parallel N_1$. By the symmetry of the rule we only have to show one direction. Let

$$\Gamma \vdash N_1 \parallel N_2 \rhd \Delta. \tag{59}$$

   By applying Lemma 4.17 to Eq. (59), we get

$$\Gamma \vdash N_1 \rhd \Delta_1 \text{ and } \Gamma \vdash N_2 \rhd \Delta_2, \tag{60}$$

   and $\Delta_1, \Delta_2 = \Delta$. By applying rule [T-net] to Eq. (60) we derive $\Gamma \vdash N_1 \parallel N_2 \rhd \Delta_2, \Delta_1$, where $\Delta = \Delta_1, \Delta_2 = \Delta_2, \Delta_1$. Thus we conclude $\Gamma \vdash N_1 \parallel N_2 \rhd \Delta$.

3. Case $N_1 \parallel (N_2 \parallel N_3) \equiv (N_1 \parallel N_2) \parallel N_3$. Let

$$\Gamma \vdash N_1 \parallel (N_2 \parallel N_3) \rhd \Delta. \tag{61}$$

By applying Lemma 4.17 to Eq. (61), we get

$$\Gamma \vdash N_1 \rhd \Delta_1, \tag{62}$$

$$\Gamma \vdash N_2 \parallel N_3 \rhd \Delta_2, \tag{63}$$

where $\Delta = \Delta_1, \Delta_2$. Again, by applying Lemma 4.17 to Eq. (63), we get

$$\Gamma \vdash N_2 \rhd \Delta_3, \tag{64}$$

$$\Gamma \vdash N_3 \rhd \Delta_4, \tag{65}$$

where $\Delta_2 = \Delta_3, \Delta_4$. By applying rule [T-**net**] to Eq. (62) and Eq. (64), we derive

$$\Gamma \vdash N_1 \parallel N_2 \rhd \Delta_1, \Delta_3. \tag{66}$$

Again, by applying rule [T-**net**] to Eq. (65) and (66), we derive

$$\Gamma \vdash (N_1 \parallel N_2) \parallel N_3 \rhd \Delta_1, \Delta_3, \Delta_4,$$

where $\Delta_1, \Delta_3, \Delta_4 = \Delta_1, \Delta_2 = \Delta$. Thus we conclude

$$\Gamma \vdash (N_1 \parallel N_2) \parallel N_3 \rhd \Delta.$$

The proof for the converse direction is similar.

4. Case $(\mathsf{new}\ s)N_1 \parallel N_2 \equiv (\mathsf{new}\ s)(N_1 \parallel N_2)$ if $s \notin fn(N_2)$. Let

$$\Gamma \vdash (\mathsf{new}\ s)N_1 \parallel N_2 \rhd \Delta. \tag{67}$$

By applying Lemma 4.17 to Eq .(67), we have

$$\Gamma \vdash (\mathsf{new}\ s)N_1 \rhd \Delta_1, \tag{68}$$

$$\Gamma \vdash N_2 \rhd \Delta_2, \tag{69}$$

where $\Delta_1, \Delta_2 = \Delta$. By applying Lemma 4.18 to Eq. (68), we have

$$\Gamma \vdash N_1 \rhd \Delta_1', \tag{70}$$

such that $\Delta_1 = \Delta_1' \setminus \Delta_1'\langle s \rangle$ and $\Delta_1'\langle s \rangle$ is coherent. By applying rule [T-**net**] to Eq. (69) and Eq. (70), we derive

$$\Gamma \vdash N_1 \parallel N_2 \rhd \Delta_1', \Delta_2. \tag{71}$$

By applying rule [T-new] and the fact that $s \notin fn(N_2)$ to Eq. (71), we derive

$$\Gamma \vdash (\mathsf{new}\ s)(N_1 \parallel N_2) \rhd \Delta_1, \Delta_2,$$

where $\Delta_1, \Delta_2 = \Delta$, thus we conclude

$$\Gamma \vdash (\mathsf{new}\ s)(N_1 \parallel N_2) \rhd \Delta.$$

For the converse direction, let

$$\Gamma \vdash (\mathsf{new}\ s)(N_1 \parallel N_2) \rhd \Delta. \tag{72}$$

By applying Lemma 4.18 to Eq .(72), we get

$$\Gamma \vdash N_1 \parallel N_2 \rhd \Delta', \tag{73}$$

such that $\Delta = \Delta' \setminus \Delta'\langle s \rangle$ and $\Delta'\langle s \rangle$ is coherent. By applying Lemma 4.17 to Eq. (73), we get

$$\Gamma \vdash N_1 \rhd \Delta_1, \tag{74}$$

$$\Gamma \vdash N_2 \rhd \Delta_2, \tag{75}$$

where $\Delta_1, \Delta_2 = \Delta'$. Because $s \notin fn(N_2)$, we have $\Delta\langle s \rangle \in \Delta_1$. Let $\Delta_1 = \Delta_1', \Delta\langle s \rangle$, then $\Delta_1 \setminus \Delta\langle s \rangle = \Delta_1'$ and $\Delta\langle s \rangle$ is coherent. By applying rule [T-new] to Eq. (74), we derive

$$\Gamma \vdash (\mathsf{new}\ s)N_1 \rhd \Delta_1'. \tag{76}$$

By applying rule [T-net] to Eq. (75) and (76),

$$\Gamma \vdash (\mathsf{new}\ s)N_1 \parallel N_2 \rhd \Delta_2, \Delta_1',$$

where

$$\Delta_2, \Delta_1' = \Delta_2, (\Delta_1 \setminus \Delta\langle s \rangle) = (\Delta_2, \Delta_1) \setminus \Delta\langle s \rangle = \Delta' \setminus \Delta\langle s \rangle = \Delta$$

thus we conclude $\Gamma \vdash (\mathsf{new}\ s)N_1 \parallel N_2 \rhd \Delta$.

5. Case $(\mathsf{new}\ s\ s')N \equiv (\mathsf{new}\ s'\ s)N$ if $s \neq s'$. By the symmetry of the rule we only have to show one direction. Let

$$\Gamma \vdash (\mathsf{new}\ s\ s')N \rhd \Delta. \tag{77}$$

By applying Lemma 4.18 to Eq. (77), we have

$$\Gamma \vdash (\mathsf{new}\ s')N \rhd \Delta'. \tag{78}$$

such that $\Delta = \Delta' \setminus \Delta'\langle s \rangle$ and $\Delta'\langle s \rangle$ is coherent. Since $s \neq s'$, by applying Lemma 4.18 to Eq. (78), we get

$$\Gamma \vdash N \rhd \Delta''. \tag{79}$$

such that $\Delta' = \Delta'' \setminus \Delta''\langle s' \rangle$ and $\Delta''\langle s' \rangle$ is coherent. Thus we have $\Delta'' = \Delta, \Delta'\langle s \rangle, \Delta''\langle s' \rangle$. By applying rule [T-new] to Eq. (79), we derive

$$\Gamma \vdash (\mathsf{new}\ s)N \rhd \Delta, \Delta'\langle s \rangle. \tag{80}$$

By applying rule [T-new] to Eq. (80), we conclude

$$\Gamma \vdash (\mathsf{new}\ s'\ s)N \rhd \Delta.$$

6. Case $\mathsf{def}\ D\ \mathsf{in}\ \mathbf{0} \equiv \mathbf{0}$. Let $X(\tilde{x}\ c) = P \in D$ and

$$\Gamma \vdash \mathsf{def}\ D\ \mathsf{in}\ \mathbf{0} \rhd \Delta, \tag{81}$$

$$\Gamma \vdash \mathbf{0} \rhd \Delta'. \tag{82}$$

In Eq. (82), Δ' is end-only by Lemma 4.1. By applying Lemma 4.9 to Eq. (81), we get

$$\Gamma, \tilde{x} : \widetilde{S},\ X : (\widetilde{S}\ \mathbf{T}) \vdash P \rhd \{c : \mathbf{T}\}$$

and

$$\Gamma, X : (\widetilde{S}\ \mathbf{T}) \vdash \mathbf{0} \rhd \Delta. \tag{83}$$

By applying Lemma 4.1 to Eq. (83), we have that Δ is end-only. Thus by rule [T-0] we conclude $\Delta' \equiv \Delta$.

For the converse direction, the proof is similar.

7. Case $\mathsf{def}\ D_2\ \mathsf{in}\ \mathsf{def}\ D_1\ \mathsf{in}\ P \equiv \mathsf{def}\ D_1\ \mathsf{in}\ \mathsf{def}\ D_2\ \mathsf{in}\ P$. Let $X(\tilde{x}\ c) = P_1 \in D_1$ and $X'(\tilde{x}'\ c') = P_2 \in D_2$ and $dpv(D_1) \cap dpv(D_2) = \emptyset$ and

$$\Gamma \vdash \mathsf{def}\ D_2\ \mathsf{in}\ \mathsf{def}\ D_1\ \mathsf{in}\ P \rhd \Delta \tag{84}$$

By applying Lemma 4.9 to Eq. (84), we have

$$\Gamma, X' : (\widetilde{S'}\ \mathbf{T'}) \vdash \mathsf{def}\ D_1\ \mathsf{in}\ P \rhd \Delta \tag{85}$$

and

$$\Gamma, \tilde{x}' : \widetilde{S'}, X' : (\widetilde{S'}\ \mathbf{T'}) \vdash P_2 \rhd \{c' : \mathbf{T'}\} \tag{86}$$

for some $\widetilde{S'}$ and $\mathbf{T'}$.

Eq. (86) can be weaken to

$$\Gamma, \tilde{x}' : \widetilde{S'}, X' : (\widetilde{S'}\ \mathbf{T'}), X : (\widetilde{S}\ \mathbf{T}) \vdash P_2 \rhd \{c' : \mathbf{T'}\} \tag{87}$$

By applying Lemma 4.9 to Eq. (85), we have

$$\Gamma, X' : (\widetilde{S'}\ \mathbf{T'}), X : (\widetilde{S}\ \mathbf{T}) \vdash P \rhd \Delta \tag{88}$$

and

$$\Gamma, X' : (\widetilde{S}'\ \mathbf{T}'), \tilde{x} : \widetilde{S}, X : (\widetilde{S}\ \mathbf{T}) \vdash P_1 \rhd \{c : \mathbf{T}\} \tag{89}$$

By applying [T-rec] to Eq.(87) and Eq. (88), we have

$$\Gamma, X : (\widetilde{S}\ \mathbf{T}) \vdash \mathsf{def}\ D_2\ \mathsf{in}\ P \rhd \Delta \tag{90}$$

By the fact that $dpv(D_1) \cap dpv(D_2) = \emptyset$, Eq. (90) can be weaken to

$$\Gamma, X' : (\widetilde{S}'\ \mathbf{T}'), X : (\widetilde{S}\ \mathbf{T}) \vdash \mathsf{def}\ D_2\ \mathsf{in}\ P \rhd \Delta \tag{91}$$

By applying [T-rec] to Eq.(89) and Eq. (91) and we have

$$\Gamma \vdash \mathsf{def}\ D_1\ \mathsf{in}\ \mathsf{def}\ D_2\ \mathsf{in}\ P \rhd \Delta$$

Thus we conclude this case.

For the converse direction, the proof is similar.

8. Case $q \equiv q'$ and let
$$\Gamma \vdash s : q \rhd \Delta.$$
The equivalence $q \equiv q'$ should come from one of the following cases:
(a) $q \equiv \varepsilon \cdot q = q'$, or
(b) $q \equiv q \cdot \varepsilon = q'$, or
(c) $q = q_1 \cdot (q_2 \cdot q_3) \equiv (q_1 \cdot q_2) \cdot q_3 = q'$.
For all cases, the messages in q and q' are the same and they are in the same order. Therefore, by Lemma 5, if $\Gamma \vdash s : q' \rhd \Delta'$, then $\Delta' = \Delta$. The proof for the converse direction is the same.

□

Theorem 1 Subject Reduction $\Gamma \vdash N \rhd \Delta$ with Δ coherent and $N \to N'$ imply that $\Gamma \vdash N' \rhd \Delta'$ such that $\Delta \rightharpoonup \Delta'$ or $\Delta \equiv \Delta'$ and Δ' is coherent.

Proof. Since we have proved Theorem 3 (Coherence), which says that a coherent session environment which is not end-only will always reduce and remain coherent after reductions, we use this theorem in every case for proving this statement.

1. Case [link]. Let
$$\Gamma \vdash a[1](y_1).P_1 \parallel ... \parallel a[n](y_n).P_n \rhd \Delta \tag{92}$$
By applying Lemma 4.17 n times to Eq. (92), we have,
$$\forall i \in \{1..n\}.\ \Gamma \vdash a[i](y_i).P_i \rhd \Delta_i \tag{93}$$
such that $\Delta_1, ..., \Delta_n = \Delta$.

By applying Lemma 4.15 n times to Eq. (93), we have for all $i \in \{1..n\}$,

$$\begin{aligned} &\Delta_i = \emptyset \\ &\Gamma \vdash a : \langle \mathbf{G} \rangle \\ &\Gamma \vdash P_i \rhd \{y_i : \mathit{Transform}(\mathbf{G}, i)\} \end{aligned} \tag{94}$$

for some $\mathbf{G}$. Note that now we know $\Delta = \Delta_1, ..., \Delta_n = \emptyset$.

By applying Lemma 6 to Eq. (94), we have

$$\begin{aligned} &\Gamma \vdash P_i[s[i]/y_i] \rhd \{y_i : \mathit{Transform}(\mathbf{G}, i)\}[s[i]/y_i] \\ &\{y_i : \mathit{Transform}(\mathbf{G}, i)\}[s[i]/y_i] = \{s[i] : \mathit{Transform}(\mathbf{G}, i)\} \end{aligned} \tag{95}$$

By taking $\mathbf{T}_i = \mathit{Transform}(\mathbf{G}, i)$ and applying rule [T-**guide**] to Eq. (95), we have

$$\Gamma \vdash [P_i[s[i]/y_i]]_{\mathbf{T}_i} \rhd s_i[i] : \mathit{Transform}(\mathbf{G}, i) \tag{96}$$

By Lemma 4.11, we have

$$\Gamma \vdash s : \varepsilon \rhd \{s : \varepsilon\} \tag{97}$$

After applying [T-**net**] to Eq. (96) n times and to Eq. (97), we have

$$\begin{aligned} \Gamma \vdash [P_1[s[1]/y_1]]_{\mathbf{T}_1} \parallel ... \parallel [P_n[s[n]/y_n]]_{\mathbf{T}_n} \parallel s : \varepsilon \rhd \\ \{s[1] : \mathit{Transform}(\mathbf{G}, 1), ..., s[n] : \mathit{Transform}(\mathbf{G}, n)\} \cup \{s : \varepsilon\} \end{aligned} \tag{98}$$

By the fact that $\{1, ..., n\} = \mathit{pid}(\mathbf{G})$ and Definition 27, we know

$$\{s[1] : \mathit{Transform}(\mathbf{G}, 1), ..., s[n] : \mathit{Transform}(\mathbf{G}, n)\} \cup \{s : \varepsilon\}$$

is coherent.

After applying [T-**new**] to Eq. (98), we have

$$\Gamma \vdash (\mathsf{new}\ s)([P_1[s[1]/y_1]]_{\mathbf{T}_1} \parallel ... \parallel [P_n[s[n]/y_n]]_{\mathbf{T}_n} \parallel s : \varepsilon) \rhd \emptyset$$

where $\emptyset = \Delta$. Thus we conclude this case.

2. Case [**rcv**]. Let

$$\Gamma \vdash [s[p_1]?(p_2, (\tilde{x})^F).P]_{\mathbf{T}} \parallel s : \langle p_2, p_1, \langle \tilde{v} \rangle \rangle^F \cdot q \rhd \Delta \tag{99}$$

Applying Lemma 4.17 and the rules for queues in Lemma 4 and Lemma 5 to Eq.(99), we get

$$\begin{aligned} &\Gamma \vdash [s[p_1]?(p_2, (\tilde{x})^F).P]_{\mathbf{T}} \rhd \Delta_1 \\ &\Gamma \vdash s : \langle p_2, p_1, \langle \tilde{v} \rangle \rangle^F \cdot q \rhd \Delta_2 \\ &\Gamma \vdash s : q \rhd \{s : \mathbf{q}\} \quad \text{for some } \mathbf{q} \\ &\Gamma \vdash \tilde{v} : \widetilde{S} \end{aligned} \tag{100}$$

and $\Delta_1, \Delta_2 = \Delta$ and $\Delta_2 = \{s : \langle p_2, p_1, \langle \widetilde{S} \rangle \rangle^F \cdot \mathbf{q}\}$.

As for Δ_1, we apply Lemma 4.16 and then Lemma 4.2 to Eq. (100). We have

$$\begin{aligned}
&\Gamma \vdash s[p_1]?(p_2, (\tilde{x})^F).P \rhd \Delta_1 \\
&\Delta_1 = \{s[p_1] : rn\langle p_2? \, \widetilde{S} \vee F \rangle \dashrightarrow \mathbf{T}'\} \\
&\Gamma, \tilde{x} : \widetilde{S} \vdash P \rhd \{s[p_1] : \mathbf{T}'\} \\
&\mathbf{T} \ni rn\langle p_2? \, \widetilde{S} \vee F \rangle \dashrightarrow \mathbf{T}'
\end{aligned} \tag{101}$$

By apply Lemma 6.(1) to Eq. (101), we have

$$\Gamma \vdash P[\tilde{v}/\tilde{x}] \rhd \{s[p_1] : \mathbf{T}'\} \tag{102}$$

By applying [T-**guide**] to Eq. (102) and the fact that $\mathbf{T} \ni rn\langle p_2? \, \widetilde{S} \vee F \rangle \dashrightarrow \mathbf{T}' \ni \mathbf{T}'$, we have

$$\Gamma \vdash [P[\tilde{v}/\tilde{x}]]_{\mathbf{T}} \rhd \{s[p_1] : \mathbf{T}'\} \tag{103}$$

By applying [T-**net**] to Eq. (100) and Eq. (103), we have

$$\Gamma \vdash [P[\tilde{v}/\tilde{x}]]_{\mathbf{T}} \parallel s : q \rhd \{s[p_1] : \mathbf{T}'\} \cup \{s : \mathbf{q}\} = \Delta' \tag{104}$$

By [**R-rcv**], $\Delta \rightharpoonup \Delta'$, which is coherent by Theorem 3. Thus we conclude this case.

3. Case [**snd**]. Let

$$\Gamma \vdash [s[p_1]!(p_2, \langle \tilde{e} \rangle); P]_{\mathbf{T}} \parallel s : q \rhd \Delta \tag{105}$$

and $\tilde{e} \downarrow \tilde{v}$.

By applying Lemma 4.17 to Eq. (105) we have

$$\begin{aligned}
&\Gamma \vdash [s[p_1]!(p_2, \langle \tilde{e} \rangle); P]_{\mathbf{T}} \rhd \Delta_1 \\
&\Gamma \vdash s : q \rhd \Delta_2
\end{aligned} \tag{106}$$

such that $\Delta = \Delta_1, \Delta_2$.

By applying Lemma 4.16 and then applying Lemma 4.3 to Eq. (106), we have

$$\begin{aligned}
&\Delta_1 = \{s[p_1] : sn\langle p_2! \, \widetilde{S} \dashrightarrow \mathbf{T}'\}\rangle \\
&\mathbf{T} \ni sn\langle p_2! \, \widetilde{S}\rangle \dashrightarrow \mathbf{T}' \\
&\Gamma \vdash P \rhd \{s[p_1] : \mathbf{T}'\} \\
&\Gamma \vdash \widetilde{e} : \widetilde{S}
\end{aligned} \tag{107}$$

By applying [T-**guide**] to Eq. (107) and the fact that $\mathbf{T} \ni sn\langle p_2! \, \widetilde{S}\rangle \dashrightarrow \mathbf{T}' \ni \mathbf{T}'$, we have

$$\Gamma \vdash [P]_{\mathbf{T}} \rhd \{s[p_1] : \mathbf{T}'\} \tag{108}$$

By applying Lemma 4 and Lemma 5 to Eq. (106), we have $\Delta_2 = \{s : \mathbf{q}\}$ for some $\mathbf{q}$. By [T-m], we have

$$\begin{array}{l} \Gamma \vdash \tilde{v} : \widetilde{S} \\ \Gamma \vdash s : q \cdot \langle p_1, p_2, \langle \tilde{v} \rangle \rangle^F \rhd \{s : \mathbf{q} \cdot \langle p_1, p_2, \langle \widetilde{S} \rangle \rangle^F\} \end{array} \tag{109}$$

By applying [T-net] to Eq. (108) and Eq. (109), we get

$$\Gamma \vdash [P]_{\mathbf{T}} \parallel s : q \cdot \langle p_1, p_2, \langle \tilde{v} \rangle \rangle^F \rhd \Delta' \tag{110}$$

such that $\Delta' = \{s[p_1] : \mathbf{T}'\} \cup \{s : \mathbf{q} \cdot \langle p_1, p_2, \langle \widetilde{S} \rangle \rangle^F\}$. By [R-snd], we have $\Delta \rightharpoonup \Delta'$, which is coherent. Thus we conclude this case.

4. Case [sndF]. Let

$$\Gamma \vdash [\mathsf{try}\{s[p_1]!(p_2, \langle \tilde{e} \rangle^F); P\}\mathsf{h}\{H\}]_{\mathbf{T}} \parallel s : q \rhd \Delta \tag{111}$$

and $\tilde{e} \downarrow \tilde{v}$ and $\mathbf{T} = \mathscr{L}[sn\langle p_2! \widetilde{S} \vee F, \widetilde{p}, \{p'_1, ..., p'_n\}\rangle]$ and $F \neq \emptyset$ and q does not have failure messages or $q = \langle p, f' \rangle \cdot q'$ but $p \neq p_1$ or $f' \notin dom(H)$.

By applying Lemma 4.17 to Eq. (111) we have

$$\begin{array}{l} \Gamma \vdash [\mathsf{try}\{s[p_1]!(p_2, \langle \tilde{e} \rangle^F); P\}\mathsf{h}\{H\}]_{\mathbf{T}} \rhd \Delta_1 \\ \Gamma \vdash s : q \rhd \Delta_2 \end{array} \tag{112}$$

such that $\Delta = \Delta_1, \Delta_2$.

By applying Lemma 4 and Lemma 5 to Eq. (112), we have $\Delta_2 = \{s : \mathbf{q}\}$ for some $\mathbf{q}$. By [T-m] and [T-mF], we have

$$\begin{array}{l} \Gamma \vdash \tilde{v} : \widetilde{S} \\ \Gamma \vdash s : q \cdot \langle p_1, p_2, \langle \tilde{v} \rangle \rangle^F \cdot \langle\langle p'_1, F \rangle\rangle ... \langle\langle p'_n, F \rangle\rangle \rhd \\ \qquad \{s : \mathbf{q} \cdot \langle p_1, p_2, \langle \widetilde{S} \rangle \rangle^F \cdot \langle\langle p'_1, F \rangle\rangle ... \langle\langle p'_n, F \rangle\rangle\} \end{array} \tag{113}$$

By applying Lemma 4.16 to Eq. (112), we have

$$\begin{array}{l} \Gamma \vdash \mathsf{try}\{s[p_1]!(p_2, \langle \tilde{e} \rangle^F); P\}\mathsf{h}\{H\} \rhd \Delta_1 \\ \mathbf{T} \ni \Delta_1(s[p_1]) \end{array} \tag{114}$$

Since by Definition 20 $act(\mathsf{try}\{s[p_1]!(p_2, \langle \tilde{e} \rangle^F); P\}\mathsf{h}\{H\}) = s[p_1]$, by applying Lemma 4.6 to Eq. (114), we have

$$\begin{array}{l} \Delta_1 = \{s[p_1] : try\{\mathbf{T}', \mathbf{H}\}\} \\ \Gamma \vdash s[p_1]!(p_2, \langle \tilde{e} \rangle^F); P \rhd \{s[p_1] : \mathbf{T}'\} \\ \Gamma \vdash \widetilde{e} : \widetilde{S} \\ dom(\mathbf{H}) = dom(H) \end{array} \tag{115}$$

By the fact that every node tagged with $F \neq \emptyset$ in $\mathbf{T}$ is unique and by applying Lemma 4.3 to Eq.(115),

$$\mathbf{T} = \mathscr{L}[sn\langle p_2 ! \widetilde{S} \vee F, \widetilde{p}, \{p'_1, ..., p'_n\}\rangle] \ni try\{\mathbf{T}', \mathbf{H}\}$$

and

$$\Gamma \vdash s[p_1]!(p_2, \langle \vec{e} \rangle^F); P \rhd \{s[p_1] : \mathbf{T}'\} \text{ where } \mathbf{T}' = sn\langle p_2 ! \widetilde{S} \vee F, \_, \_ \rangle \dashrightarrow \mathbf{T}''$$

imply that

$$\begin{array}{l} \mathbf{T}' = sn\langle p_2 ! \widetilde{S} \vee F, \widetilde{p}, \{p'_1, ..., p'_n\}\rangle \dashrightarrow \mathbf{T}'' \\ \Gamma \vdash P \rhd \{s[p_1] : \mathbf{T}''\} \end{array} \tag{116}$$

By taking $\mathbf{T}$ which $\mathbf{T} \ni \mathbf{T}' \ni \mathbf{T}''$ and applying [T-guide] to Eq. (116), we have

$$\Gamma \vdash [P]_{\mathbf{T}} \rhd \{s[p_1] : \mathbf{T}''\} \tag{117}$$

By applying [T-net] to Eq. (113) and Eq. (117), we get

$$\Gamma \vdash [P]_{\mathbf{T}} \parallel s : q \cdot \langle p_1, p_2, \langle \tilde{v} \rangle\rangle^F \cdot \langle\langle p'_1, F\rangle\rangle ... \langle\langle p'_n, F\rangle\rangle \rhd \Delta'$$

such that $\Delta' = \{s[p_1] : \mathbf{T}''\} \cup \{s : \mathbf{q} \cdot \langle p_1, p_2, \langle \widetilde{S} \rangle\rangle^F \cdot \langle\langle p'_1, F\rangle\rangle ... \langle\langle p'_n, F\rangle\rangle\}$. By [R-sndthw], we have $\Delta \rightharpoonup \Delta'$, which is coherent. Thus we conclude this case.

5. Case [thwf] The proof for this case is similar to the one for Case [sndF].

6. Case [try]. Let

$$\Gamma \vdash [\mathsf{try}\{P\}\mathsf{h}\{H\}; P'']_{\mathbf{T}} \parallel s : q \rhd \Delta \tag{118}$$

and $act(\mathsf{try}\{P\}\mathsf{h}\{H\}) = s[p]$ and $[P]_{\mathbf{T}} \parallel s : q \rightarrow [P']_{\mathbf{T}} \parallel s : q'$ and q does not have failure messages or $q = \langle p, f' \rangle \cdot q'$ but $p \neq p_1$ or $f' \notin dom(H)$.

By applying Lemma 4.17 to Eq. (118) we have

$$\begin{array}{l} \Gamma \vdash [\mathsf{try}\{P\}\mathsf{h}\{H\}; P'']_{\mathbf{T}} \rhd \Delta_1 \\ \Gamma \vdash s : q \rhd \Delta_2 \end{array} \tag{119}$$

such that $\Delta = \Delta_1, \Delta_2$.

By applying Lemma 4.16 to Eq. (119), we have

$$\begin{array}{l} \Gamma \vdash \mathsf{try}\{P\}\mathsf{h}\{H\}; P'' \rhd \Delta_1 \\ \mathbf{T} \ni \Delta_1(s[p]) \end{array} \tag{120}$$

By applying Lemma 4.6 to Eq. (120), we have

$$\begin{aligned}
&\Delta_1 = \{s[p] : try\{\mathbf{T}'', \mathbf{H}\}\} \\
&\Gamma \vdash P; P'' \rhd \{s[p] : \mathbf{T}''\} \\
&dom(H) = dom(\mathbf{H}) \\
&\forall f \in dom(H). \Gamma \vdash H(f) \rhd \{s[p] : \mathbf{H}(f)\}
\end{aligned} \tag{121}$$

With the assumption that

$$[P]_{\mathbf{T}} \parallel s : q \rightarrow [P']_{\mathbf{T}} \parallel s : q'$$

and by rule [seq], we have

$$[P; P'']_{\mathbf{T}} \parallel s : q \rightarrow [P'; P'']_{\mathbf{T}} \parallel s : q' \tag{122}$$

by induction on $\rightarrow$ and by Lemma 4 and Lemma 5, we have

$$\begin{aligned}
&\Gamma \vdash [P'; P'']_{\mathbf{T}} \rhd \Delta'' \\
&\Gamma \vdash s : q' \rhd \{s : \mathbf{q}'\}
\end{aligned} \tag{123}$$

for some $\mathbf{q}'$.

By applying Lemma 4.16 to Eq. (123), we have

$$\begin{aligned}
&\Gamma \vdash P'; P'' \rhd \Delta'' \\
&\Delta'' = \{s[p] : \mathbf{T}'''\} \text{ for some } \mathbf{T}''' \text{ and } \mathbf{T}'' \ni \mathbf{T}'''
\end{aligned} \tag{124}$$

By applying [T-try] to Eq. (121) and Eq. (124), which gives H is typed by $\mathbf{H}$, we get

$$\Gamma \vdash \mathsf{try}\{P'\}\mathsf{h}\{H\}; P'' \rhd \{s[p] : try\{\mathbf{T}''', \mathbf{H}\}\} \tag{125}$$

Since $\mathbf{T} \ni try\{\mathbf{T}'', \mathbf{H}\}$ and $\mathbf{T}'' \ni \mathbf{T}'''$, by Definition 21, we have

$$\mathbf{T} \ni try\{\mathbf{T}''', \mathbf{H}\} \tag{126}$$

Baesd on Eq. (126), by applying [T-guide] to Eq. (125), we have

$$\Gamma \vdash [\mathsf{try}\{P'\}\mathsf{h}\{H\}; P'']_{\mathbf{T}} \rhd \{s[p] : try\{\mathbf{T}''', \mathbf{H}\}\} \tag{127}$$

By applying [T-net] to Eq. (127) and Eq. (123), we get

$$\Gamma \vdash [\mathsf{try}\{P'\}\mathsf{h}\{H\}; P'']_{\mathbf{T}} \parallel s : q' \rhd \Delta'$$

such that $\Delta' = \{s[p] : try\{\mathbf{T}''', \mathbf{H}\}\} \cup \{s : \mathbf{q}'\}$. By [R-try], we have $\Delta \rightharpoonup \Delta'$, which is coherent. Thus we conclude this case.

7. Case [hdl]. Let

$$\Gamma \vdash [\mathsf{try}\{\mathscr{E}[s[p] \otimes F; P]\}\mathsf{h}\{H\}; P']_{\mathbf{T}} \parallel s : \langle p, f\rangle \cdot q \rhd \Delta \tag{128}$$

and $f \in dom(H) \cap F$. By Definition 20, we have $act(\mathsf{try}\{\mathscr{E}[s[p] \otimes F; P]\}\mathsf{h}\{H\}) = s[p]$.

By applying Lemma 4.17 to Eq. (128) we have

$$\begin{aligned} &\Gamma \vdash [\mathsf{try}\{\mathscr{E}[s[p] \otimes F; P]\}\mathsf{h}\{H\}; P']_{\mathbf{T}} \rhd \Delta_1 \\ &\Gamma \vdash s : \langle p, f\rangle \cdot q \rhd \Delta_2 \end{aligned} \tag{129}$$

such that $\Delta = \Delta_1, \Delta_2$.

By applying Lemma 4.16 to Eq. (129), we have

$$\begin{aligned} &\Gamma \vdash \mathsf{try}\{\mathscr{E}[s[p] \otimes F; P]\}\mathsf{h}\{H\}; P' \rhd \Delta_1 \\ &\mathbf{T} \ni \Delta_1(s[p]) \end{aligned} \tag{130}$$

By applying Lemma 4 and Lemma 5 to Eq. (129), we have

$$\begin{aligned} &\Delta_2 = \{s : \langle p, f\rangle \cdot \mathbf{q}\} \\ &\Gamma \vdash s : q \rhd \{s : \mathbf{q}\} \end{aligned} \tag{131}$$

By applying Lemma 4.6 to Eq. (130), we have

$$\begin{aligned} &\Delta_1 = \{s[p] : try\{\mathbf{T}'', \mathbf{H}\}\} \\ &\Gamma \vdash \mathscr{E}[s[p] \otimes F; P; P'] \rhd \{s[p] : \mathbf{T}''\} \\ &dom(H) = dom(\mathbf{H}) \\ \forall f \in dom(H) \quad &\Gamma \vdash H(f); P' \rhd \{s[p] : \mathbf{H}(f)\} \end{aligned} \tag{132}$$

By the fact that we have $\mathbf{T} \ni try\{\mathbf{T}'', \mathbf{H}\}$, and by applying [T-guide] to Eq. (132), based on Definition 21, we have

$$\begin{aligned} &\forall f \in dom(\mathbf{H}) \quad \mathbf{T} \ni \mathbf{H}(f) \\ &\forall f \in dom(H) \quad \Gamma \vdash [H(f); P']_{\mathbf{T}} \rhd \{s[p] : \mathbf{H}(f)\} \end{aligned} \tag{133}$$

By applying [T-net] to Eq. (131) and Eq. (133), we get

$$\Gamma \vdash [H(f); P']_{\mathbf{T}} \parallel s : q \rhd \{s : \mathbf{q}\} \rhd \Delta'$$

such that $\Delta' = \{s[p] : \mathbf{H}(f)\} \cup \{s : \mathbf{q}\}$. By [R-hdl], we have $\Delta \rightharpoonup \Delta'$, which is coherent. Thus we conclude this case.

8. Case [sync-done]. Let

$$\Gamma \vdash [s[p] \otimes F'; P]_{\mathbf{T}} \parallel s : \langle\langle p, F\rangle\rangle \cdot q \rhd \Delta \tag{134}$$

and $F' \subseteq F$.

Applying Lemma 4.17 and the rules for queues in Lemma 4 and Lemma 5 to Eq.(134), we get

$$\begin{aligned}
&\Gamma \vdash [s[p] \otimes F'; P]_{\mathbf{T}} \rhd \Delta_1 \\
&\Gamma \vdash s : \langle\langle p, F\rangle\rangle \cdot q \rhd \Delta_2 \\
&\Gamma \vdash s : q \rhd \{s : \mathbf{q}\} \quad \text{for some } \mathbf{q} \\
&\Gamma \vdash s : \langle\langle p, F\rangle\rangle \rhd \{s : \langle\langle p, F\rangle\rangle\}
\end{aligned} \tag{135}$$

and $\Delta_1, \Delta_2 = \Delta$ and thus $\Delta_2 = \{s : \langle\langle p, F\rangle\rangle \cdot \mathbf{q}\}$.

As for Δ_1, we apply Lemma 4.16 and then Lemma 4.5 to Eq. (135). We have

$$\begin{aligned}
&\Gamma \vdash s[p] \otimes F'; P \rhd \Delta_1 \\
&\Delta_1 = \{s[p] : \mathit{yield}\langle F'\rangle \dashrightarrow \mathbf{T}'\} \\
&\Gamma \vdash P \rhd \{s[p] : \mathbf{T}'\} \\
&\mathbf{T} \ni \mathit{yield}\langle F'\rangle \dashrightarrow \mathbf{T}'
\end{aligned} \tag{136}$$

By applying [T-**guide**] to Eq. (136) and the fact that $\mathbf{T} \ni \mathit{yield}\langle F'\rangle \dashrightarrow \mathbf{T}'$, we have

$$\Gamma \vdash [P]_{\mathbf{T}} \rhd \{s[p] : \mathbf{T}'\} \tag{137}$$

By applying [T-**net**] to Eq. (135) and Eq. (137), we have

$$\Gamma \vdash [P]_{\mathbf{T}} \parallel s : q \rhd \{s[p] : \mathbf{T}'\} \cup \{s : \mathbf{q}\} = \Delta' \tag{138}$$

By [**R-sync-done**], $\Delta \rightharpoonup \Delta'$, which is coherent by Theorem 3. Thus we conclude this case.

9. Case [**sync**]. The proof for this case is similar to the one for Case [**sync-done**].

10. Case [**try-end**]. Let

$$\Gamma \vdash [\mathsf{try}\{v\}\mathsf{h}\{H\}]_{\mathbf{T}} \rhd \Delta \tag{139}$$

By Definition 20, we have $\mathit{act}(\mathsf{try}\{v\}\mathsf{h}\{H\}) = \mathit{act}(H)$. Let $\mathit{act}(H) = s[p]$. By applying Lemma 4.16 to Eq. (139), we have

$$\begin{aligned}
&\Gamma \vdash \mathsf{try}\{v\}\mathsf{h}\{H\} \rhd \Delta \\
&\mathbf{T} \ni \Delta(s[p])
\end{aligned} \tag{140}$$

By applying Lemma 4.6 to Eq. (140), we have

$$\begin{aligned}
&\Delta = \{s[p] : \mathit{try}\{\mathbf{T}', \mathbf{H}\}\} \\
&\Gamma \vdash v \rhd \{s[p] : \mathbf{T}'\}
\end{aligned} \tag{141}$$

$$\tag{142}$$

Since $\Gamma \vdash v : S$ and that Eq. (140) implies there is no action in $\mathbf{T}'$ (i.e. idle), we should have

$$\begin{aligned} &\mathbf{T}' = \mathsf{end} \\ &\Delta = \{s[p] : try\{\mathsf{end}, \mathbf{H}\}\} \end{aligned} \tag{143}$$

By Lemma 4.1, we have

$$\Gamma \vdash v \rhd \Delta' \quad \Delta' \text{ end-only} \tag{144}$$

Since $\mathbf{T} \ni \mathsf{end}$, by applying [T-guide] to Eq.(144), we have

$$\Gamma \vdash [v]_{\mathbf{T}} \rhd \Delta' \quad \Delta' \text{ end-only}$$

By [R-try-end], we have $\Delta \rightharpoonup \Delta'$, which is coherent. Thus we conclude this case.

11. Case [call]. Let

$$\begin{aligned} &\Gamma \vdash [\mathsf{def}\ D\ \mathsf{in}\ X\langle \tilde{e}\ c\rangle]_{\mathbf{T}} \rhd \Delta \\ &X(\tilde{x}\ c) = P \in D \\ &\tilde{e} \Downarrow \tilde{v} \end{aligned} \tag{145}$$

By applying Lemma 4.16 to Eq. (145), we have

$$\begin{aligned} &\Gamma \vdash \mathsf{def}\ D\ \mathsf{in}\ X\langle \tilde{e}\ c\rangle \rhd \Delta \\ &\mathbf{T} \ni \Delta(act(\mathsf{def}\ D\ \mathsf{in}\ (X\langle \tilde{e}\ c\rangle))) = \Delta(c) \quad \textit{By Definition 20,} \end{aligned} \tag{146}$$

By applying Lemma 4.9 to Eq. (146), we have

$$\begin{aligned} &\Gamma, X : (\widetilde{S}\ \mathbf{T}) \vdash X\langle \tilde{e}\ c\rangle \rhd \Delta \\ &\Gamma, \tilde{x} : \widetilde{S}, X : (\widetilde{S}\ \mathbf{T}) \vdash P \rhd \{c : \mathbf{T}\} \end{aligned} \tag{147}$$

By applying Lemma 6.(1) to Eq. (147), we obtain

$$\Gamma, X : (\widetilde{S}\ \mathbf{T}) \vdash P[\tilde{v}/\tilde{x}] \rhd \{c : \mathbf{T}\} \tag{148}$$

By applying Lemma 4.8 to Eq. (147), we have

$$\begin{aligned} &\Delta = \{c : \mathbf{T}\} \\ &\Gamma, X : (\widetilde{S}\ \mathbf{T}) \vdash \tilde{e} : \widetilde{S} \end{aligned} \tag{149}$$

By applying [T-rec] to Eq.(147) and Eq. (148) and Eq. (149), we have

$$\Gamma \vdash \mathsf{def}\ D\ \mathsf{in}\ P[\tilde{v}/\tilde{x}] \rhd \Delta \tag{150}$$

By applying [T-guide] to Eq. (151) and the fact that $\mathbf{T} \ni \Delta(c) = \Delta(act(\mathsf{def}\ D\ \mathsf{in}\ P[\tilde{v}/\tilde{x}]))$, we have

$$\Gamma \vdash [\mathsf{def}\ D\ \mathsf{in}\ P[\tilde{v}/\tilde{x}]]_{\mathbf{T}} \rhd \Delta$$

Thus we conclude this case.

12. Case [defin]. Let

$$\Gamma \vdash [\mathsf{def}\ D\ \mathsf{in}\ P_1]_{\mathbf{T}} \parallel s : q \rhd \Delta \tag{151}$$

and $[P_1]_{\mathbf{T}} \parallel s : q \to [P_2]_{\mathbf{T}} \parallel s : q'$ and $X(\widetilde{x}\ c) = P \in D$.

By applying Lemma 4.17 to Eq. (151), we have

$$\begin{aligned} &\Gamma \vdash [\mathsf{def}\ D\ \mathsf{in}\ P_1]_{\mathbf{T}} \rhd \Delta_1 \\ &\Gamma \vdash s : q \rhd \Delta_2 \end{aligned} \tag{152}$$

where $\Delta_1, \Delta_2 = \Delta$.

By applying the rules for queues in Lemma 4 and Lemma 5 to Eq.(152), we have

$$\Delta_2 = \{s : \mathbf{q}\} \quad \text{for some } \mathbf{q} \tag{153}$$

By applying Lemma 4.16 to Eq.(152), we have

$$\begin{aligned} &\Gamma \vdash \mathsf{def}\ D\ \mathsf{in}\ P_1 \rhd \Delta_1 \\ &\mathbf{T} \ni \Delta_1(act(\mathsf{def}\ D\ \mathsf{in}\ P_1)) = \Delta_1(act(P_1)) \end{aligned} \tag{154}$$

Assume $X(\tilde{x}\ c) = P \in D$. By applying Lemma 4.9 to Eq. (154), we obtain

$$\begin{aligned} &\Gamma, X : (\widetilde{S}\ \mathbf{T}) \vdash P_1 \rhd \Delta_1 \\ &\Gamma, \tilde{x} : \widetilde{S}, X : (\widetilde{S}\ \mathbf{T}) \vdash P \rhd \{c : \mathbf{T}\} \end{aligned} \tag{155}$$

By the fact that $\mathbf{T} \ni \Delta_1(act(P_1))$, applying [T-guide] to Eq. (155), we obtain

$$\Gamma, X : (\widetilde{S}\ \mathbf{T}) \vdash [P_1]_{\mathbf{T}} \rhd \Delta_1 \tag{156}$$

By applying [T-net] to Eq.(153) and Eq.(156), we have

$$\Gamma, X : (\widetilde{S}\ \mathbf{T}) \vdash [P_1]_{\mathbf{T}} \parallel s : q \rhd \Delta_1, \Delta_2 = \Delta \tag{157}$$

Since $[P_1]_{\mathbf{T}} \parallel s : q \to [P_2]_{\mathbf{T}} \parallel s : q'$, by induction hypothesis on $\to$, we have

$$\Gamma, X : (\widetilde{S}\ \mathbf{T}) \vdash [P_2]_{\mathbf{T}} \parallel s : q' \rhd \Delta' \tag{158}$$

such that $\Delta \rightharpoonup \Delta'$ and Δ' is coherent.

Again, by applying Lemma 4.16 and Lemma 4.17 to Eq. (158), we have

$$\begin{aligned} &\Gamma, X : (\widetilde{S}\ \mathbf{T}) \vdash P_2 \rhd \Delta'_1 \\ &\Gamma, X : (\widetilde{S}\ \mathbf{T}) \vdash s : q' \rhd \Delta'_2 \end{aligned} \tag{159}$$

where $\Delta'_1, \Delta'_2 = \Delta'$.

By applying the rules for queues in Lemma 4 and Lemma 5 to Eq.(159), we have

$$\Delta_2' = \{s : \mathbf{q}'\} \quad \text{for some } \mathbf{q}' \tag{160}$$

By applying [T-rec] to Eq. (155) and Eq. (159), we obtain

$$\Gamma \vdash \mathsf{def}\ D\ \mathsf{in}\ P_2 \triangleright \Delta_1'$$

By Definition 21 and reduction rules of session environments (defined in Figure 8), since $\Delta_1(act(P_1)) \ni \Delta_1'(act(P_2))$, we know $\mathbf{T} \ni \Delta_1(act(\mathsf{def}\ D\ \mathsf{in}\ P_1)) \ni \Delta_1'(act(\mathsf{def}\ D\ \mathsf{in}\ P_2))$. By applying [T-guide] to Eq. (161), we have

$$\Gamma \vdash [\mathsf{def}\ D\ \mathsf{in}\ P_2]_{\mathbf{T}} \triangleright \Delta_1' \tag{161}$$

By applying [T-net] to Eq. (160) and Eq.(161), we obtain

$$\Gamma \vdash [\mathsf{def}\ D\ \mathsf{in}\ P_2]_{\mathbf{T}} \parallel s : q' \triangleright \Delta_1', \Delta_2' = \Delta'$$

By Eq. (158), we know $\Delta \rightharpoonup \Delta'$, which is coherent by Theorem 3. Thus we conclude this case.

13. Case [seq]. Let

$$\Gamma \vdash [P_1; P]_{\mathbf{T}} \parallel s : q \triangleright \Delta \tag{162}$$

and $[P_1]_{\mathbf{T}} \parallel s : q \to [P_2]_{\mathbf{T}} \parallel s : q'$.

Applying Lemma 4.17 to Eq.(162), we have

$$\begin{aligned} &\Gamma \vdash [P_1; P]_{\mathbf{T}} \triangleright \Delta_1 \\ &\Gamma \vdash s : q \triangleright \Delta_2 \end{aligned} \tag{163}$$

where $\Delta_1, \Delta_2 = \Delta$.

Applying Lemma 4.16 to Eq.(163), we have

$$\begin{aligned} &\Gamma \vdash P_1; P \triangleright \Delta_1 \\ &\mathbf{T} \ni \Delta_1(act(P_1; P)) \end{aligned} \tag{164}$$

By Definition 20, as $act(P_1; P)$ is defined, we have $act(P_1; P) = act(P)$.

By applying the rules for queues in Lemma 4 and Lemma 5 to Eq.(163), we have

$$\Delta_2 = \{s : \mathbf{q}\} \quad \text{for some } \mathbf{q} \tag{165}$$

Applying Lemma 4.7 to Eq.(164), we have

$$\begin{aligned} &\Gamma \vdash P_1 \triangleright \Delta_1' \\ &\Gamma \vdash P \triangleright \Delta_0 \\ &\Delta_1' \circ \Delta_0 = \Delta_1 \end{aligned} \tag{166}$$

Since $\mathbf{T} \ni \Delta_1(act(P_1;P)) \ni \Delta_1(act(P_1)) \ni \Delta_1'(act(P_1))$, by applying [T-guide] to Eq. (166), we have

$$\Gamma \vdash [P_1]_{\mathbf{T}} \rhd \Delta_1' \tag{167}$$

By applying [T-net] to Eq. (163) and Eq. (165) and Eq. (167), we have

$$\Gamma \vdash [P_1]_{\mathbf{T}} \parallel s : q \rhd \Delta_1', \Delta_2 \tag{168}$$

Since $[P_1]_{\mathbf{T}} \parallel s : q \to [P_2]_{\mathbf{T}} \parallel s : q'$, by induction hypothesis on $\to$, we have

$$\begin{array}{l} \Gamma \vdash [P_2]_{\mathbf{T}} \parallel s : q' \rhd \Delta'' \\ \Delta_1', \Delta_2 \rightharpoonup \Delta'' \end{array} \tag{169}$$

By applying Lemma 4.17 to Eq. (169), we have

$$\begin{array}{l} \Gamma \vdash [P_2]_{\mathbf{T}} \rhd \Delta_1'' \\ \Gamma \vdash s : q' \rhd \Delta_2'' \\ \Delta_1'', \Delta_2'' = \Delta'' \end{array} \tag{170}$$

Again, by applying Lemma 4.16 and the rules for queues in Lemma 4 and Lemma 5 to Eq.(170), we have

$$\begin{array}{l} \Gamma \vdash P_2 \rhd \Delta_1'' \\ \mathbf{T} \ni \Delta_2''(act(P_2)) \\ \Delta_2'' = \{s : \mathbf{q}'\} \quad \text{for some } \mathbf{q}' \end{array} \tag{171}$$

By applying [T-seq] to Eq. (166) and Eq. (171), we have

$$\Gamma \vdash P_2;P \rhd \Delta_1'' \circ \Delta_0 \tag{172}$$

Since by Eq. (164) and Eq. (171) we have $\mathbf{T} \ni \Delta_1(act(P_1;P)) = \Delta_1' \circ \Delta_0(act(P_1;P)) \ni \Delta_0(act(P_1;P)) = \Delta_0(act(P))$ and $\mathbf{T} \ni \Delta_1''(act(P_2))$, by By Definition 21, we know $\mathbf{T} \ni \Delta_1'' \circ \Delta_0(act(P_2;P))$. By applying [T-guide] to Eq. (172), we have

$$\Gamma \vdash [P_2;P]_{\mathbf{T}} \rhd \Delta_1'' \circ \Delta_0 \tag{173}$$

By applying [T-net] to Eq. (170) and Eq. (171) and Eq. (173), we have

$$\Gamma \vdash [P_2;P]_{\mathbf{T}} \parallel s : q' \rhd \Delta_1'' \circ \Delta_0, \Delta_2'' = \Delta' \tag{174}$$

By Eq. (169) and Eq. (170), we know

$$\Delta_1', \Delta_2 \rightharpoonup \Delta_1'', \Delta_2'' \tag{175}$$

Since $\Delta = \Delta_1, \Delta_2 = \Delta_1' \circ \Delta_0, \Delta_2$, by Eq. (175) and by mechanically examining all cases in Definition 32 and Definition 33 (sequential composition of session environments), we have

$$\Delta = \Delta_1' \circ \Delta_0, \Delta_2 \rightharpoonup \Delta_1'' \circ \Delta_0, \Delta_2'' = \Delta'$$

which is coherent by Theorem 3. Thus we conclude this case.

14. Case [net]. Let

$$\Gamma \vdash N_1 \parallel N \rhd \Delta \tag{176}$$

Applying Lemma 4.17 to Eq.(176), we have

$$\begin{aligned} &\Gamma \vdash N_1 \rhd \Delta_1 \\ &\Gamma \vdash N \rhd \Delta_2 \end{aligned} \tag{177}$$

such that $\Delta_1, \Delta_2 = \Delta$.

Since $N_1 \to N_2$, by induction hypothesis on $\to$, we have

$$\begin{aligned} &\Gamma \vdash N_2 \rhd \Delta_1' \qquad \Delta_1' \text{ coherent} \\ &\Delta_1 \rightharpoonup \Delta_1' \end{aligned} \tag{178}$$

Since $dom(\Delta_1) = dom(\Delta_1')$, by applying [T-net] to Eq. (177) and Eq. (178), we have

$$\Gamma \vdash N_2 \parallel N \rhd \Delta_1', \Delta_2 = \Delta' \tag{179}$$

By applying [R-res] to Eq. (178) and Eq. (179), we have $\Delta \rightharpoonup \Delta'$, which is coherent by Theorem 3. Thus we conclude this case.

15. Case [new]. Let

$$\Gamma \vdash (\mathsf{new}\ s) N_1 \rhd \Delta \tag{180}$$

Applying Lemma 4.18 to Eq.(180), we have

$$\begin{aligned} &\Gamma \vdash N_1 \rhd \Delta_1' \\ &\Delta = \Delta_1' \setminus \Delta_1' \langle s \rangle \end{aligned} \tag{181}$$

Since $N_1 \to N_2$, by induction hypothesis on $\to$, we have

$$\begin{aligned} &\Gamma \vdash N_2 \rhd \Delta_2' \qquad \Delta_2' \text{ coherent} \\ &\Delta_1' \rightharpoonup \Delta_2' \end{aligned} \tag{182}$$

By applying [T-new] to Eq. (181) and Eq. (182), we have

$$\Gamma \vdash (\mathsf{new}\ s) N_2 \rhd \Delta_2' \setminus \Delta_2' \langle s \rangle = \Delta' \tag{183}$$

Since $dom(\Delta_1') = dom(\Delta_2')$ and $dom(\Delta_1' \langle s \rangle) = dom(\Delta_2' \langle s \rangle)$, by applying [R-res] to Eq. (182) and Eq. (183), we have $\Delta \rightharpoonup \Delta'$, which is coherent by Theorem 3. Thus we conclude this case.

□

Property of Communication Safety Recall that we have defined

$$\mathscr{E} ::= [\,] \mid \mathscr{E};P \mid \mathsf{try}\{\mathscr{E}\}\mathsf{h}\{H\} \quad \mathscr{C} ::= [\,] \mid \mathscr{C} \parallel N \mid N \parallel \mathscr{C} \mid (\mathsf{new}\ s)\mathscr{C}$$

in main the content, Section 3 and Section 6.

Corollary 1 Communication Safety Suppose $\Gamma \vdash N \triangleright \Delta$ and Δ is coherent. Let $N_1 = \mathscr{C}_1[s : q \cdot \langle p_2, p_1, \langle \tilde{v} \rangle^F \cdot q']$ and $N_2 = \mathscr{C}_2[s : q \cdot \langle p_1, f \rangle \cdot q']$ and $N_3 = \mathscr{C}_3[s : q \cdot \langle\langle p_1, F \rangle\rangle \cdot q']$. and no messages in q is sending to p_1.

1. If $N = \mathscr{C}[\mathscr{E}[s[p_1]?(p_2,(\tilde{x})^F).P]_{\mathbf{T}}]$, then $N \equiv N_1$ or $N \to^* N_1$.
2. If $N = \mathscr{C}[\mathscr{E}[\mathsf{try}\{s[p_1] \otimes F'; P\}\mathsf{h}\{H\}]_{\mathbf{T}}]$ and $F' \subseteq F \neq \emptyset$, then either (a) $N \equiv N_2$ or $N \to^* N_2$ or (b) $N \equiv N_3$ or $N \to^* N_3$.
3. If $N = \mathscr{C}[\mathscr{E}[\mathsf{try}\{v\}\mathsf{h}\{H\}]_{\mathbf{T}}]$ and $f \in dom(H)$ and process $H(f)$ is acting on $s[p_1]$, then $N \not\equiv N_2$ and $N \not\to^* N_2$.

Proof. The proof is done by Definition 27 (Coherence) and Theorem 1(Subject Reduction) and Theorem 3. Recall that we have defined the following contexts in Appendix A and Appendix B:

$$\mathscr{G} ::= [\,] \mid \mathbf{T}[\mathscr{G}]\mathbf{H}[h] \mid \mathbf{T}[g]\mathbf{H}[..., f : \mathscr{G}, ...] \mid \mathscr{G};g \mid g;\mathscr{G} \mid \mu t.\mathscr{G}$$

$$\mathscr{F} ::= [\,] \mid \mathbf{T}[\mathscr{F}]\mathbf{H}[h] \mid \mathscr{F};g \mid \mu t.\mathscr{F}$$

For convenience, define an additional contexts on local types:

$$\mathscr{B} ::= [\,] \mid try\{\mathscr{B}, \mathbf{H}\} \mid \mathscr{B} \dashrightarrow \mathbf{T} \mid \mu t.\mathscr{B}$$

$$\mathscr{L} ::= [\,] \mid try\{\mathscr{L}, \mathbf{H}\} \mid try\{\mathbf{T}, ..., f : \mathscr{L}, ...\} \mid \mathbf{n} \dashrightarrow \mathscr{L} \mid \mathscr{L} \dashrightarrow \mathbf{T} \mid \mu t.\mathscr{L}$$

Case 1. $N = \mathscr{C}[\mathscr{E}[s[p_1]?(p_2,(\tilde{x})^F).P]_{\mathbf{T}}]$. Since $\Gamma \vdash N \triangleright \Delta$, we deduce that $\Delta(s[p_1]) = \mathscr{B}_1[rn\langle p_2 ? \widetilde{S} \vee F \rangle]$ for some $\mathscr{B}_1$ and $\widetilde{S}$. Let $\mathscr{B}_1[rn\langle p_2 ? \widetilde{S} \vee F \rangle] = \mathbf{T}_1$.

Assume the type of q is $\mathbf{q}$. By the assumption that no messages in q is heading to p_1, we know $s : \mathbf{q}$ and $\Delta(s[p_1])$ will not reduce.

By Theorem 3, since Δ is not end-only, Δ will reduce.

Assume $\Delta = \Delta_0, s[p_1] : \mathbf{T}_1, s[p_2] : \mathbf{T}_2$ and $\Delta_0 = \Delta_0', s : \mathbf{q}_0$ for some $\mathbf{q}_0 = \mathbf{q} \cdot \mathbf{q}_1$ and Δ_0 is not able to reduce. We have the following cases:

(I) $s[p_2]$ and Δ_0 can reduce but $s[p_1]$ and Δ_0 *cannot reduce*. By Definition 27, there exists a coherent $\mathbf{G}$ such that $pid(\mathbf{G}) = \{p | s'[p] \in dom(\Delta), s' = s\}$ and $Transform(\mathbf{G}, p_1) = \Delta(s[p_1]) \upharpoonright (\mathbf{q}_0, p_1) = \Delta(s[p_1]) = \mathbf{T}_1$ (that $s[p_1]$ and Δ_0 cannot reduce implies no messages in $\mathbf{q}_0$ is for p_1, i.e. $\upharpoonright(\mathbf{q}_0, p_1) = \varepsilon$). By the shape of $\Delta(s[p_1])$, we have $\mathbf{G} = \mathscr{G}[p_2 \to p_1 : \widetilde{S} \vee F]$ and by Definition 18 (Transformation) and Definition 27:

$$Transform(\mathbf{G}, p_2) = \mathscr{L}[sn\langle p_1 ! \widetilde{S} \vee F, \widetilde{p}', \widetilde{p}'' \rangle] = \Delta(s[p_2]) \upharpoonright (\mathbf{q}_0, p_2)$$

which says, by Definition 23 (Remainder), $\Delta(s[p_2]) = \mathbf{T}_2$ contains an action $sn\langle p_1!\,\widetilde{S} \vee F, \widetilde{p}', \widetilde{p}''\rangle$.

(i) If $\mathbf{T}_2 = \mathscr{L}_2[sn\langle p_1!\,\widetilde{S} \vee F, \widetilde{p}', \widetilde{p}''\rangle]$ for some $\mathscr{L}_2$ and $\mathscr{L}_2$ contains some receiving action or synchronisation points, and

$$\Delta = \Delta_0, s[p_1] : \mathbf{T}_1, s[p_2] : \mathbf{T}_2 \rightharpoonup^* \Delta_0'', s[p_1] : \mathbf{T}_1, s[p_2] : \mathscr{B}_2[sn\langle p_1!\,\widetilde{S} \vee F, \widetilde{p}', \widetilde{p}''\rangle] = \Delta''$$

for some $\mathscr{B}_2$. So $s[p_2] : \mathbf{T}_2$ and Δ_0 reduce until $sn\langle p_1!\,\widetilde{S} \vee F, \widetilde{p}', \widetilde{p}''\rangle$ is ready fire. Their reductions result Δ_0''. We now have $N \to^* N_1'$ and $\Gamma \vdash N_1' \rhd \Delta''$. Then it goes to the case below (i.e. Case 1.(I).(ii)) by replacing N with N_1' and Δ with Δ'' and $\Delta_0', s : \mathbf{q}_0$ with Δ_0''.

This case says that, it is possible that $sn\langle p_1!\,\widetilde{S} \vee F, \widetilde{p}', \widetilde{p}''\rangle$ is guarded by some receiving actions or some synchronisation points. For the former, the reasoning for the process associated with $s[p_2]$ goes to Case 1 (i.e. this case); for the latter, the reasoning for the process associated with $s[p_2]$ goes to Case 3. Note that $sn\langle p_1!\,\widetilde{S} \vee F, \widetilde{p}', \widetilde{p}''\rangle$ will not be guarded forever because Δ is coherent and will anyway reduce as long as Δ is not end-only, which implies that those receiving actions and synchronisation points will be reduced with Δ_0 and be gone at the end. Finally $sn\langle p_1!\,\widetilde{S} \vee F, \widetilde{p}', \widetilde{p}''\rangle$ will fire, which is the following case.

(ii) $\mathbf{T}_2 = \mathscr{B}_2[sn\langle p_1!\,\widetilde{S} \vee F, \widetilde{p}', \widetilde{p}''\rangle]$ for some $\mathscr{B}_2$ and, by [R-snd], [R-sndthw], [R-try], we have

$$\Delta = \Delta_0', s : \mathbf{q}_0, s[p_1] : \mathbf{T}_1, s[p_2] : \mathbf{T}_2 \rightharpoonup \Delta_0', s : \mathbf{q}_0 \cdot \langle p_2, p_1, \langle \widetilde{S} \rangle\rangle, s[p_1] : \mathbf{T}_1, s[p_2] : \mathscr{B}_2[\varepsilon] = \Delta'$$

Because $\Gamma \vdash N \rhd \Delta$, a process associated with $s[p_2]$ is well-typed in Δ. We have $\Gamma \vdash \tilde{v} : \widetilde{S}$. Therefore we have $\Delta \rightharpoonup \Delta'$ and $N \to N_1$ and $\Gamma \vdash N_1 \rhd \Delta'$, where $\Delta'(s) = \mathbf{q}_0 \cdot \langle p_2, p_1, \langle \widetilde{S} \rangle\rangle$, which implies $\Gamma \vdash \tilde{v} : \widetilde{S}$ and $\langle p_2, p_1, \langle \tilde{v} \rangle\rangle$ exists in N_1..

(II) $s[p_1]$ and $\Delta_0 = \Delta_0', s : \mathbf{q}_0$ can reduce. By rules defined in Figure 8 (reduction rules of session environments) and by Definition 29 (Permuation), the reduction comes from

$$\Delta = \Delta_0', s : \mathbf{q}_0, s[p_1] : \mathbf{T}_1, s[p_2] : \mathbf{T}_2 \rightharpoonup^* \Delta_0'', s : \mathbf{q}'', s[p_1] : \mathscr{B}_1[\varepsilon], s[p_2] : \mathbf{T}_2$$

where $\mathbf{q}_0 = \mathbf{q} \cdot \langle p_2, p_1, \langle \widetilde{S} \rangle\rangle \cdot \mathbf{q}''$ for some $\mathbf{q}''$. So we have $\Delta(s) = \mathbf{q} \cdot \langle p_2, p_1, \langle \widetilde{S} \rangle\rangle$. Therefore we have $\Gamma \vdash N = N_1 \rhd \Delta$ and $\Gamma \vdash \tilde{v} : \widetilde{S}$ and $\langle p_2, p_1, \langle \tilde{v} \rangle\rangle$ exists in N_1.

(III) If $\{s[p_1], s[p_2]\} \subseteq dom(\Delta)$ and $s[p_1] : \mathbf{T}_1, s[p_2] : \mathbf{T}_2, \Delta_0$ cannot reduce, then, since Δ is coherent which always can reduce, there exists $s'[p] \in dom(\Delta)$ for

some s', p, such that $s'[p] \neq s[p_1]$ and $s'[p] \neq s[p_1]$, will reduce. When all such $s'[p]$ have reduced until they cannot reduce anymore and result $\Delta \rightharpoonup^* \Delta''$, which is coherent and $N \rightharpoonup^* N'$ and $\Gamma \vdash N' \triangleright \Delta''$. Since Δ'' is not end-only (because $\Delta''(s[p_1]) \neq \mathsf{end}$), either $s[p_1]$ and the updated global queue or $s[p_2]$ should reduce. Then we go back to cases of (I) and (II) by replacing N with N' and replacing Δ with Δ''.

Case 2. $N = \mathscr{C}[\mathscr{E}[\mathsf{try}\{s[p_1] \otimes F'; P\}\mathsf{h}\{H\}]_{\mathbf{T}}]$ and $F' \subseteq F \neq \emptyset$.

Since $\Gamma \vdash N \triangleright \Delta$, we deduce that $\Delta(s[p_1]) = \mathscr{B}_1[try\{\mathscr{B}_1'[yield\langle F'\rangle], \mathbf{H}\}]$ for some $\mathscr{B}_1$ and $\mathscr{B}_1'$ and $\mathbf{H}$ such that $dom(\mathbf{H}) = dom(H)$.

By Definition 27, there exists a coherent $\mathbf{G}$ such that

$$\mathbf{G} = \mathscr{G}[p \to p' : \widetilde{S} \vee F] \text{ and } p_1 \in C(\alpha(Struct(\mathbf{G})), F)$$

and

$$Transform(\mathbf{G}, p) = \mathscr{B}[try\{\mathscr{B}'[sn\langle p'!\, \widetilde{S} \vee F, \widetilde{p}', \widetilde{p}''\rangle], \mathbf{H}'\}]$$

for some $\mathscr{B}$ and $\mathscr{B}'$ and $\mathbf{H}'$.

By [R-sndthw] and [R-try], either

(I) The failure notification $\langle p_1, f\rangle$ is sent by the participant associated with $s[p]$. Then $N = N_2$ or, by Theorem 1.(2), $\Delta \rightharpoonup^* \Delta'$ and $N \to^* N_2$ and $\Gamma \vdash N_2 \triangleright \Delta'$, where $\Delta' = \Delta'', \{s : \mathbf{q} \cdot \langle p_1, f\rangle \cdot \mathbf{q}'\}$, which implies the existence of $\langle p_1, f\rangle$ in N_2.

(II) The non-failure notification $\langle\langle p_1, F\rangle\rangle$ is sent by the participant associated with $s[p]$. Then $N = N_3$ or, by Theorem 1.(2), $\Delta \rightharpoonup^* \Delta'$ and $N \to^* N_3$ and $\Gamma \vdash N_3 \triangleright \Delta'$, where $\Delta' = \Delta'', \{s : \mathbf{q} \cdot \langle\langle p_1, F\rangle\rangle \cdot \mathbf{q}'\}$, which implies the existence of $\langle\langle p_1, F\rangle\rangle$ in N_3.

Case 3. $N = \mathscr{C}[\mathscr{E}[\mathsf{try}\{v\}\mathsf{h}\{H\}]_{\mathbf{T}}]$ and $f \in dom(H)$ and the process $H(f)$ is acting on channel $s[p_1]$, i.e. $act(\mathscr{E}[\mathsf{try}\{v\}\mathsf{h}\{H\}]_{\mathbf{T}}) = s[p_1]$. Since $\Gamma \vdash N \triangleright \Delta$ and Δ is coherent, by [T-try] we have $\Delta(s[p_1]) = \mathscr{B}[try\{\mathbf{T}'', \mathbf{H}\}]$ for some $\mathscr{B} \neq [\,] \dashrightarrow \mathscr{B}'$, and $\mathbf{H}$, $\mathbf{T}''$ such that $\Gamma \vdash \mathscr{E}[\mathsf{try}\{v\}\mathsf{h}\{H\}]_{\mathbf{T}} \triangleright s[p_1] : \mathscr{B}[try\{\mathbf{T}'', \mathbf{H}\}]$ and $dom(H) = dom(\mathbf{H})$, and $\Gamma \vdash \mathscr{E}[\varepsilon] \triangleright \mathscr{B}[\mathbf{T}'']$. Note that, $\mathbf{T}''$ may not be end.

We prove this case by contradiction.

(I) Assume $N \equiv N_2$, which implies that $\langle p_1, f\rangle$ and $f \in dom(h)$ exists in N_2.

Without loss of generality, assume $\langle p_1, f\rangle$ is outputed by p at some previous session environments, say Δ'', such that $\Delta'' \rightharpoonup \Delta$ and Δ'' is coherent. By Definition 27, there exists a coherent $\mathbf{G}$ such that

$$\mathbf{G} = \mathscr{F}[\mathbf{T}[g]\mathbf{H}[h]] \text{ and } g = \mathscr{G}[p \to p' : \widetilde{S} \vee F] \text{ and } p_1 \in C(\alpha(Struct(\mathbf{G})), F)$$

where $F \subseteq dom(h) = dom(H) = dom(\mathbf{H})$.

By Definition 18 (Transformation) and Definition 2 (Projection) and Definition 17 (Synchronisation), for $Transform(\mathbf{G}, p_1)$, we have

$$Transform(\mathbf{G}, p_1) = \mathscr{B}[try\{\mathscr{B}'[yield\langle F\rangle], \mathbf{H}] = \Delta''(s[p_1]) - \restriction(\Delta''(s), p_1)$$

for some $\mathscr{B}'$.

It implies that the possible behaviours of $\Delta''(s[p_1])$ after receiving all messages in $\Delta''(s)$ heading to her, from the next session environment after reduction of Δ'', she will be able to receive either a failure which can trigger $\mathbf{H}$ or a synchronisation point.

Since g may have more failure-raising interactions, assume after reductions from $\Delta'' \rightharpoonup^* \Delta'''$, we have

$$\Delta'''(s[p_1]) = \mathscr{B}[try\{\mathbf{T}', \mathbf{H}\}] = Transform(\mathscr{F}[\mathbf{T}[\varepsilon]\mathbf{H}[h]], p_1)$$

By rules defined in Figure 8 (reduction of session environments), it implies that no failures tagging on the failure-raising interactions appearing in g (i.e. including those in F) occur during the reductions from Δ'' to Δ'''.

Recall that we say $\Delta(s[p_1]) = \mathscr{B}[try\{\mathbf{T}'', \mathbf{H}\}]$.

(i) If $\mathbf{T}'' \ni \mathbf{T}'$ (see Definition 21) and $\mathbf{T}'' \neq \mathbf{T}'$, then $\mathscr{B}[try\{\mathbf{T}'', \mathbf{H}\}]$ types $\mathscr{E}[\mathsf{try}\{P\}\mathsf{h}\{H\}]_{\mathbf{T}}$ where $P \neq v$, which leads to a contradiction.

(ii) If $\mathbf{T}' = \mathbf{T}''$, then $\Delta''' = \Delta$.

(iii) If $\mathbf{T}' \ni \mathbf{T}''$ and $\mathbf{T}'' \neq \mathbf{T}'$, then $\Delta''' \rightharpoonup^* \Delta$.

(II) Assume $N \rightarrow^* N_2$, which implies that $\langle p_1, f\rangle$ and $f \in dom(h)$ is outputted from some N'' such that $N \rightharpoonup^* N'' \rightharpoonup N_2$.

Without loss of generality, assume $\langle p_1, f\rangle$ is outputed by p at some session environments, say Δ'', such that $\Delta \rightharpoonup^* \Delta'' \rightharpoonup \Delta'$ and Δ'' is coherent. With the reasoning as Case 3.(I), we know there exists a coherent $\mathbf{G}$ such that

$$\mathbf{G} = \mathscr{F}[\mathbf{T}[g]\mathbf{H}[h]] \text{ and } g = \mathscr{G}[p \to p' : \widetilde{S} \vee F] \text{ and } p_1 \in C(\alpha(Struct(\mathbf{G})), F)$$

where $F \subseteq dom(h) = dom(H) = dom(\mathbf{H})$ and

$$Transform(\mathbf{G}, p_1) = \mathscr{B}[try\{\mathscr{B}'[yield\langle F\rangle], \mathbf{H}] = \Delta''(s[p_1]) - \restriction(\Delta''(s), p_1)$$

However, $\mathscr{B}[try\{\mathscr{B}'[yield\langle F\rangle], \mathbf{H}]$ cannot type $\mathscr{E}[\mathsf{try}\{v\}\mathsf{h}\{H\}]_{\mathbf{T}}$ and any N' reduced from $\mathscr{E}[\mathsf{try}\{v\}\mathsf{h}\{H\}]_{\mathbf{T}} \rightharpoonup^* N'$, which leads to a contradiction.

Therefore, there does not exist failure notification $\langle p_1, f\rangle$ such that $f \in dom(\mathbf{H}) = dom(h) = dom(H)$ in Δ, which implies that $\langle p_1, f\rangle$ does not exist in N_2. Then we conclude that $N \not\equiv N_2$ and $N \not\rightarrow^* N_2$.

□

Property of Progress

Theorem 2 (Progress) $\Gamma \vdash N \triangleright \Delta$ with Δ coherent and $N \to N'$ imply that N' is communication safe or $N' = \mathbf{0} \parallel s : \varepsilon$.

Proof. Suppose $\Gamma \vdash N' \triangleright \Delta'$. Δ' is coherent by Theorem 3.

1. If $\Delta'(s) = \mathbf{M} \cdot \mathbf{q} \neq \varepsilon$.

   (a) If $\mathbf{M} = \langle p, p', \langle \widetilde{S} \rangle \rangle^F$, then by [R-snd] and [R-sndthw], it should be sent by $s[p]$ in some pre-step environment, say a coherent Δ'', such that $\Delta \rightharpoonup^* \Delta'' \rightharpoonup^* \Delta'$. Without loss of generality, assume $\Delta''(s[p]) = \mathscr{B}[sn\langle p'! \widetilde{S} \vee F, \widetilde{p}', \widetilde{p}'' \rangle]$.

   Since Δ'' is coherent, by Lemma 1 and Definition 27 (Coherence), there exists a coherent $\mathbf{G}$ which is able to guide $s[p]$ and the one interacting with it, i.e. $s[p']$. By the shape of $\Delta''(s[p])$ and Definition 2 (Projection), we have

   $$\mathbf{G} = \mathscr{G}[p \to p' : \widetilde{S} \vee F]$$

   $$Transform(\mathbf{G}, p') = \mathscr{L}[rn\langle p? \widetilde{S} \vee F \rangle] = \Delta''(s[p']) - \upharpoonright (\Delta''(s), p')$$

   for some $\mathscr{G}$ and $\mathscr{L}$. The last equivalent equation means that, after $\Delta''(s[p'])$ absorbs all messages in $\Delta''(s)$ for her, she will be able to receive $\mathbf{M}$. It also implies that, as $\Delta \rightharpoonup^* \Delta''$ and $N \to^* N''$ and $\Gamma \vdash N'' \triangleright \Delta''$, we have $N'' = \mathscr{C}[\mathscr{E}[s[p']?(p, (\tilde{x})^F).P]_{\mathbf{T}}]$. Therefore, as $N \rightharpoonup^* N'' \rightharpoonup N'$, we have that N' is communication safe by Corollary 1.(1).

   (b) If $\mathbf{M} = \langle\langle p', F \rangle\rangle$ or $\mathbf{M} = \langle p', f \rangle$, then by [R-snd] and [R-sndthw], it should be sent by some $s[p]$ in some pre-step environment, say a coherent Δ'', such that $\Delta \rightharpoonup^* \Delta'' \rightharpoonup^* \Delta'$. Without loss of generality, assume $\Delta''(s[p]) = try\{\mathscr{B}[sn\langle p''! \widetilde{S} \vee F, \widetilde{p}', \widetilde{p}'' \rangle], \mathbf{H}\}$ for some $\mathscr{B}$ and $\mathbf{H}$.

   Since Δ'' is coherent, by Lemma 1 and Definition 27 (Coherence), there exists a coherent $\mathbf{G}$ which is able to guide $s[p]$ and those interacting with it, including $s[p'']$ and those $\{s[p]\}_{p \in \widetilde{p}'}$ and $\{s[p]\}_{p \in \widetilde{p}''}$, in Δ''. By the shape of $\Delta''(s[p])$ and Definition 2 (Projection), we have

   $$\mathbf{G} = \mathscr{G}[\mathbf{T}[\mathscr{G}'[p \to p'' : \widetilde{S} \vee F]]\mathbf{H}[h]] \text{ and } p' \in C(\alpha(Struct(\mathbf{G})), F)$$

   for some $\mathscr{G}, \mathscr{G}'$ and h.

   By Definition 18

   $$Transform(\mathbf{G}, p') = try\{\mathscr{L}[yield\langle F \rangle], \mathbf{H}'\} = \Delta''(s[p']) - \upharpoonright (\Delta''(s), p')$$

   for some $\mathscr{L}$ and H' such that $f \in dom(\mathbf{H}')$ It means, after $\Delta''(s[p'])$ absorbs all messages in $\Delta''(s)$ for her, $\Delta''(s[p'])$ is yielding to $\langle\langle p', F \rangle\rangle$ or $\langle p', f \rangle$. It

also implies that, as $\Delta \rightharpoonup^* \Delta''$ and $N \to^* N''$ and $\Gamma \vdash N'' \triangleright \Delta''$, we have $N'' = \mathscr{C}[\mathscr{E}[\mathsf{try}\{s[p'] \otimes F'; P\}\mathsf{h}\{H\}]_{\mathbf{T}}]$ and $F' \subseteq F$. Therefore N' is communication safe by Corollary 1. (2).

2. If $s : \varepsilon$ but $N' \neq \mathbf{0}$. By the fact that $\Gamma \vdash N' \triangleright \Delta'$ and by the typing rules defined in Figure 5, Δ' is not end-only and, so that by Lemma 3, Δ' will reduce by rules defined in Figure 8:

Case (a) If the reduction comes from [R-try-end], then by Corollary 1.(3), N' is communication safe.

Case (b) If the reduction comes from a sending action in Δ', with the similar analysis above (Case 1 in this proof), there must exist a corresponding receiving action in Δ', which implies that N' is communication safe by Corollary 1 (1,2).

Case (c) If the reduction comes from [R-ε], say $s[p] : \varepsilon \dashrightarrow \mathbf{T}$, then, if $\mathbf{T} \not\equiv_{type} \mathsf{end}$, by Definition 1, $\mathbf{T}$ must starts from some action. If the action is in Case 2.(a) or Case 2.(b), then N' is communication safe; if the action is for receiving, then immediately N' is communication safe.

If $\mathbf{T} \equiv_{type} \mathsf{end}$, the process associated with $s[p]$ is $\mathbf{0}$ and the process itself cannot reduce. Then we need to look for other $s[p] \in dom(\Delta')$ which can be reduced, and go back to Case 1 and 2 for those reducible $s[p]$.

3. If $\Delta'(s) = \varepsilon$ and $N' = \mathbf{0}$. Then Δ' is end-only, and by the typing rules defined in Figure 8, Δ' will not reduce anymore, so that N' will not reduce anymore.

□

C Optional Appendix: Encoding The Syntax of [3]

Since [3] is the most related work which formalises exception handling for multiple interactional processes by global escape mechanism extended from session types, and other related works have the same try-handle syntax as theirs, we here encode the grammar, except parallel composition, of global types defined in [3] to show that our grammar of protocol types is sufficient to express the frameworks of previous works for failure-handling activities. We do not consider parallel composition, session interleaving or multi-threads at endpoints, because we focus on failure-handling. Thus we do not have a syntax for them in this version.

Let γ be the grammar of interaction types and G be grammar of global types defined in Section 2.1, [3]:

$$\gamma ::= \varepsilon \mid p \to q : c\langle \widetilde{S} \rangle \mid p \to q : c\{l_j : \gamma_j\}_{j\in J} \mid \gamma\,;\gamma' \mid \{\{\tilde{c}, \gamma, \gamma'\}\} \mid \mu t.\gamma \mid t$$
$$G ::= \gamma\,;\mathsf{end} \mid \mathsf{end}$$

We show that our protocol types can encode the global types proposed in [3]. We first define $tag\langle g,f\rangle$ as a function with two parameters: the first is our interaction type, the second is a failure. We use $tag\langle g,f\rangle$ to tag f at interactions in g. Then we define $[\![G]\!]$ as a function encoding G to protocol types $\mathbf{G}$.

Tagging f on g

$tag\langle \varepsilon, f\rangle = tag\langle \mathsf{end}, f\rangle = \varepsilon \quad tag\langle (p_1 \to p_2 : \widetilde{S} \vee F), f\rangle = p_1 \to p_2 : \widetilde{S} \vee F \cup \{f\}$

$tag\langle g_1\,;g_2,f\rangle = tag\langle g_1,f\rangle; tag\langle g_2,f\rangle \quad tag\langle \mathbf{T}[g]\mathbf{H}[h],f\rangle = \mathbf{T}[tag\langle g,f\rangle]\mathbf{H}[h]$

$tag\langle t,f\rangle = t \quad tag\langle \mu t.g,f\rangle = \mu t.tag\langle g,f\rangle$

Encoding G to $\mathbf{G}$

$\langle\!|\gamma\,;\mathsf{end}|\!\rangle = [\![\gamma]\!];\mathsf{end} \qquad \langle\!|\mathsf{end}|\!\rangle = \varepsilon;\mathsf{end}$

Encoding γ to g

$[\![\varepsilon]\!] = \varepsilon \quad [\![p_1 \to p_2 : c\langle \widetilde{S}\rangle]\!] = p_1 \to p_2 : \widetilde{S} \vee \emptyset$

$[\![p_1 \to p_2 : c\{l_j : \gamma_j\}_{j\in\{1..n\}}]\!] = \mathbf{T}[p_1 \to p_2 : l_1, ...l_n]\mathbf{H}[l_1 : [\![\gamma_1]\!], ..., l_n : [\![\gamma_n]\!]]$

$[\![\gamma_1\,;\gamma_2]\!] = [\![\gamma]\!];[\![\gamma']\!]$

$[\![\{\{\tilde{c}, \gamma_1, \gamma_2\}\}]\!] = \mathbf{T}[tag\langle [\![\gamma_1]\!], f\rangle]\mathbf{H}[f : [\![\gamma_2]\!]] \quad f \text{ fresh}$

$[\![t]\!] = t \quad [\![\mu t.\gamma]\!] = \mu t.[\![\gamma]\!]$

Example C1. Recall the example, *the global type of the Client-Server-Bank*, in [3], Figure 2:

$$\{\{(1,2,3,4), \gamma, \varepsilon\}\} \text{ where}$$
$$\gamma = \mathsf{C} \to \mathsf{S} : 1\langle \mathsf{str}\rangle; \mathsf{S} \to \mathsf{C} : 2\langle \mathsf{str}\rangle; \mathsf{C} \to \mathsf{B} : 3\langle \mathsf{str}\rangle;$$
$$\{\{(3,4), \mathsf{B} \to \mathsf{C} : 4\{\mathsf{OK} : \varepsilon, \mathsf{NEM} : \gamma\}, \gamma'\}\};$$
$$\mathsf{C} \to \mathsf{S} : 1\langle \mathsf{Money}\rangle; \mathsf{S} \to \mathsf{C} : 2\langle \mathsf{Data}\rangle$$

$1,2,3,4$ are the identities of communication channels. E.g. $\mathsf{C} \to \mathsf{S} : 1\langle \mathsf{str} \rangle$ uses channel number 1 to transmit message in type str from C to S. $\{\{(1,2,3,4), \gamma, \varepsilon\}\}$ specifies that channels 1,2,3,4 are used in γ, and when there is no failure, the behaviour defined in γ is performed; once a failure occurs at either 1,2,3, or 4, all interactions at 1,2,3, and 4 (all participants' activities) are abort (i.e. ε).

We show that our protocol types can encode this example:

$$\begin{array}{l}\mathbf{T}[\mathsf{C} \to \mathsf{S} : \mathsf{str} \vee f; \mathsf{S} \to \mathsf{C} : \mathsf{str} \vee f; \mathsf{C} \to \mathsf{B} : \mathsf{str} \vee f; \\ \quad \mathbf{T}[\mathsf{B} \to \mathsf{C} : \mathsf{OK}, \mathsf{NEM}, f']\mathbf{H}[\mathsf{OK} : \varepsilon,\ \mathsf{NEM} : [\![\gamma]\!], f' : [\![\gamma]\!]]; \\ \mathsf{C} \to \mathsf{S} : \mathsf{Money} \vee f; \mathsf{S} \to \mathsf{C} : \mathsf{Data} \vee f]\mathbf{H}[f : \varepsilon]; \mathsf{end}\end{array}$$