Logical Methods in Computer Science
Volume 21, Issue 3, 2025, pp. 6:1–6:65
https://lmcs.episciences.org/

Submitted Jan. 23, 2024
Published Jul. 23, 2025

TIMEOUT ASYNCHRONOUS SESSION TYPES: SAFE ASYNCHRONOUS MIXED-CHOICE FOR TIMED INTERACTIONS

JONAH PEARS [a], LAURA BOCCHI [a], MAURIZIO MURGIA [b], AND ANDY KING [a]

[a] University of Kent, Canterbury, UK
e-mail address: j.pears@kent.ac.uk, l.bocchi@kent.ac.uk, a.m.king@kent.ac.uk

[b] Gran Sasso Science Institute, L'Aquila, Italy
e-mail address: maurizio.murgia@gssi.it

Abstract. Mixed-choice has long been barred from models of asynchronous communication since it compromises the decidability of key properties of communicating finite-state machines. Session types inherit this restriction, which precludes them from fully modelling timeouts – a core property of web and cloud services. To address this deficiency, we present (binary) Timeout Asynchronous Session Types (TOAST) as an extension to (binary) asynchronous timed session types, that permits mixed-choice. TOAST deploys timing constraints to regulate the use of mixed-choice so as to preserve communication safety. We provide a new behavioural semantics for TOAST which guarantees progress in the presence of mixed-choice. Building upon TOAST, we provide a calculus featuring process timers which is capable of modelling timeouts using a `receive-after` pattern, much like Erlang, and capture the correspondence with TOAST specifications via a type system for which we prove subject reduction.

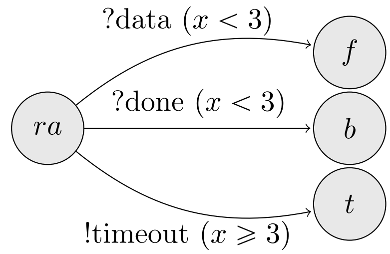
1. Introduction

Mixed-choice is an inherent feature of models of communications such as communicating finite-state machines (CFSM) [BZ83] where actions are classified as either send or receive. In this setting, a state of a machine is said to be mixed if there exist both a sending action and a receiving action from that state. When considering an asynchronous model of communication, deadlock freedom is undecidable in general [GMY84] but can be guaranteed in presence of three sufficient and decidable conditions: determinism, compatibility, and *absence* of mixed-states [GMY84, DY13]. Intuitively, determinism means that it is not possible, from a state, to reach two different states with the same kind of action, and compatibility requires

Key words and phrases: Session types · Mixed-choice · Timeouts · π-calculus.
This work is an extended version of [PBK23] that was accepted at COORDINATION 2023.
This work has been partially supported by EPSRC project EP/T014512/1 (STARDUST) and the BehAPI project funded by the EU H2020 RISE under the Marie Sklodowska-Curie action (No: 778233). We thank Simon Thompson for his insightful comments on an early version of this work.

```
receive
    {Q, data} -> foo();
    {Q, done} -> bar();
after 3000  ->
    Q ! {self(), timeout},
    handle_timeout()
end.
```

(A) Erlang snippet



(B) Mixed-state CFSM

Figure 1. An Erlang snippet and its mixed-state machine representation

that for each send action of one machine, the rest of the system can eventually perform a complementary receive action.

In the pursuit of deadlock freedom mixed-choice has been cast out, even though this curtails the descriptive capabilities of CFSM and its derivatives. Despite the rapid evolution of session types, even to the point of deployment in Java [HYH08], Python [NYH13, Ney13], Rust [LNY20], F# [NHYA18] and Go [CHJ+19], thus far mixed-choice has only been introduced into the synchronous binary setting [VCAM20]. In fact, the exclusion of mixed-choice pervades work on asynchronous communication which guarantee of deadlock freedom, both for communicating timed automata [BLY15, KY06] and session types [BCD+08, CHY08, HYC08, YV07]. Determinism and the absence of mixed-choice is baked into the very syntax of session types (the correspondence between session types and CFSM is explained in [DY13]).

Timed session types [BCP14, BMVY19, BYY14], which extend session types with time constraints, inherit the same syntactic restrictions of session types, and hence also rule out mixed-choice. This is unfortunate since in the timed setting, mixed-choice are a useful abstraction for timeouts. To illustrate, Figure 1 shows how an Erlang [CV16] `receive-after` statement (1A) can be rendered as a mixed-state CFSM (1B): if neither a `data` nor a `done` message are received within 3 time units from co-party `Q`, then a `timeout` message is sent.

Timeouts are important for handling failure and unexpected delays, for instance, the SMTP protocol stipulates: *"An SMTP client* must *provide a timeout mechanism"* [Kle08, Section 4.5.3.2]. Mixed-states would allow, for example, states where a server is waiting to *receive* a message from the client and, if nothing is received after a certain amount of time, *send* a notification that ends the session. Current variants of timed session types allow deadlines to be expressed but cannot, because of the absence of mixed-states, characterise (and verify) the behaviour that should follow a missed deadline, e.g., a restart or retry strategy. In this paper, we argue that time makes mixed-states more powerful (allowing timeouts to be expressed), while just adding sufficient synchronisation to ensure that mixed-states are safe in an asynchronous semantics (cannot produce deadlocks).

Contributions. This work makes three orthogonal contributions to the theory of (binary) session types, with a focus on improving their descriptive capabilities:

- We introduce Timeout Asynchronous Session Types (TOAST) to support timeouts. Inspired by asynchronous timed (binary) session types [BMVY19], TOAST shows how timing constraints provide an elegant solution for guaranteeing the safety of mixed-choice. Technically, we provide a semantics for TOAST and a well-formedness condition. We show that well-formedness is sufficient to guarantee progress for TOAST (which may, instead, get stuck in general).

- We provide a new process calculus whose functionality extends to support programming motifs such as the widely used `receive-after` pattern of Erlang for expressing timeouts.
- We introduce timers in our process calculus to structure the counterpart of a timeout, as well as time-sensitive conditional statements, where the selection of a branch may be determined by timers. Time-sensitive conditional statements provide processes with knowledge as to which branch should be followed e.g., in the case of timeout.
- We provide an informal discussion on the correspondence between TOAST and the aforementioned primitives of our new process calculus.
- We show a formal correspondence between TOAST and process calculus via a typing system, for which we establish Subject Reduction.

For simplicity, we focus on binary sessions.

Outline. In Section 2 we begin by introducing Timeout Asynchronous Session Types and present their syntax, semantics, and discuss mixed-choice and other motivations behind our well-formedness conditions. We end Section 2 by discussing our proof of progress for TOAST.

In Section 3 we present a calculus for processes with timeouts, which are designed to correspond to TOAST, and discuss their syntax, structure and reduction rules. Section 4 discusses the expressiveness of our types and processes, and provides some examples of the kinds of programs that TOAST is capable of modelling, and vice versa, how to express certain behaviour in TOAST and our process calculus.

We present our typing system in Section 5. We first introduce the typing judgements, then discuss each rule in detail and conclude with an example evaluation using our typing system. In Section 6 we present subject reduction for our typing system. We conclude our work in Sections 7 and 8, where we discuss related work and summarise our contribution.

2. Timeout Asynchronous Session Types (TOAST)

This section presents the syntax, semantics and formation rules for (binary) Timeout Asynchronous Session Types (TOAST), which are an extension of (binary) Asynchronous Timed Session Types [BMVY19] with a well-disciplined (and hence safe) form of mixed-choice.

Clocks & Constraints. We start with a few preliminary definitions borrowed from timed automata [AD94]. Let $\mathbb{X}$ be a finite set of clocks denoted x, y and z. A (clock) valuation ν is a map $\nu : \mathbb{X} \to \mathbb{R}_{\geqslant 0}$. The initial valuation is ν_0, where $\nu_0 = \{x \mapsto 0 \mid x \in \mathbb{X}\}$. Given a time offset $t \in \mathbb{R}_{\geqslant 0}$ and a valuation ν, $\nu + t = \{x \mapsto \nu(x) + t \mid x \in \mathbb{X}\}$. Given ν and $\lambda \subseteq \mathbb{X}$, $\nu\,[\lambda \mapsto 0] = \{0 \text{ if } (x \in \lambda) \text{ else } \nu(x) \mid x \in \mathbb{X}\}$. Observe $\nu\,[\varnothing \mapsto 0] = \nu$.

A clock constraint δ takes the following form:

$$\delta ::= \texttt{true} \mid x > n \mid x = n \mid x - y > n \mid x - y = n \mid \neg\delta \mid \delta_1 \wedge \delta_2 \qquad (\text{where } n \in \mathbb{Q}_{\geqslant 0}) \quad (2.1)$$

We write $\nu \models \delta$ to denote that the clock valuations in ν satisfy the constraints δ. Defined formally, $(\nu \models \delta) = (\forall x \in \texttt{cn}(\delta).\delta\,[{}^{\nu(x)}/_{x}])$ to say that, for all clocks in the *clock names*[1]of δ we substitute the clock name with the corresponding valuation in ν. E.g., $\nu[x \mapsto 3] \models (x > 2)$ holds since $(x > 2)\,[{}^{\nu(x)}/_{x}] = (3 > 2)$. We write $\downarrow \delta$ to denote the *weak past* of δ.

[1]Given in Definition A.3.

Definition 2.1 (Weak Past)**.** We write $\downarrow\delta$ (the past of δ) for a constraint δ' such that $\nu \models \delta'$, if and only if $\exists t$ such that $\nu + t \models \delta$.

By Definition 2.1, $\downarrow\delta$ effectively removes any lower-bounds in δ, preserving the upper-bounds. For example, $\downarrow(3 < x < 5) = (x < 5)$ and $\downarrow(x > 2) = (\mathtt{true})$. In practice $\downarrow\delta$ allows us to reason on constraints being satisfiable at some point in the future, since we only require that the clocks valuation does not exceed the upper-bound of the constraint.

2.1. **Syntax of TOAST.** The syntax of TOAST (or just *types*) is given in Eq (2.2). A type S is either a choice $\{\mathtt{C}_i\}_{i\in I}$, recursive definition $\mu\alpha.S$, call α, or termination type $\mathtt{end}$.

$$\begin{array}{lllllll} S & ::= & \{\mathtt{C}_i\}_{i\in I} \mid \mu\alpha.S \mid \alpha \mid \mathtt{end} & \quad & \mathtt{C} & ::= & \Box\, l\,\langle T\rangle\,(\delta,\lambda).S \\ T & ::= & \big(\delta,\ S\big) \mid \mathtt{Unit} \mid \mathtt{Nat} \mid \mathtt{Bool} \mid \mathtt{String} \mid \ldots & \quad & \Box & ::= & \,!\mid\, ? \end{array} \tag{2.2}$$

Type $\{\mathtt{C}_i\}_{i\in I}$ models a choice among options i ranging over a non-empty set I. Each option i is a selection/send action if $\Box =!$, or alternatively a branching/receive action if $\Box =?$. An option sends (resp. receives) a label l and a message of a specified data type T is delineated by $\langle\cdot\rangle$. The sending or receiving action of an option is guarded by a time constraint δ. After the action, the clocks within λ are reset to 0. Data types, ranged over by $T, T_i, \ldots$ can be higher-order types $\big(\delta,\ S\big)$ to model session delegation, $\mathtt{Unit}$ types, or *base types* (e.g., $\mathtt{Nat}$, $\mathtt{Bool}$, $\mathtt{String}$). We omit a $\mathtt{Unit}$ type when the payload is nothing. We discuss the structure of delegation types further in Section 2.3.1. Labels of the options in a choice are pairwise distinct. Recursion and termination types are as standard [BCD+08, HYC08, Vas12, YV07].

Remarks on the notation. One convention is to model the exchange of payloads as a separated action with respect to the communication of branching labels. In this paper we follow [BLY15, YZF21], and model them as unique actions. When irrelevant we omit the payload, yielding a notation closer to that of timed automata.

Feasibility & Junk Types. Unfortunately, when annotating session types with time constraints one may obtain protocols that are infeasible, as shown in Example 2.2. This is a known problem, which has been addressed by providing additional conditions or constraints on timed session types, for example compliance [BCM17], feasibility [BYY14], interaction enabling [BLY15], and well-formedness [BMVY19]. (We address feasibility in Section 2.3.1.)

Example 2.2 (Junk Types)**.** Consider type S (defined below) that describes first waiting 3 seconds, then receiving a, and finally a mixed-choice between sending b and receiving c.

$$S = ?a\,(x > 3, \varnothing)\,.\begin{Bmatrix} !b\,(y = 2, \varnothing)\,.\mathtt{end}, \\ ?c\,(2 < x < 5, \varnothing)\,.\mathtt{end} \end{Bmatrix}$$

Initially all clocks are 0. After a is received all clocks hold values greater than 3, since time passes evenly over all clocks. Therefore, b is never able to be sent since $(y > 3)$ and c is only *sometimes* able to be sent (i.e., if a is received when $(x < 5)$ which constraint $(x > 3)$ does nothing to guarantee). Types with unsatisfiable constraints are called *junk types* [BMVY19].

$$S' = ?a\,(x > 3, \{x\})\,.\begin{Bmatrix} !b\,(y = 2, \varnothing)\,.\mathtt{end}, \\ ?c\,(2 < x < 5, \varnothing)\,.\mathtt{end} \end{Bmatrix}$$

Above we show an amended S which now resets clock x after receiving a in order to guarantee that receiving c is always feasible. By further amending S' such that clock y is also reset after receiving a then sending b would also become feasible. △

2.2. **Semantics of TOAST.** We define the semantics using three layers: configurations ($\mathbf{s}$), configurations with queues $\mathtt{M}$ that model asynchronous interactions ($\mathbf{S}$), and systems that model the parallel composition of configurations with queues. The semantics, for all layers, are defined over the transition labels ℓ, formally defined along with $\mathtt{M}$, $\mathbf{s}$ and $\mathbf{S}$ in Eq (2.3).

$$\ell ::= \Box m \mid t \mid \tau \qquad m ::= l\,\langle T\rangle \qquad \mathtt{M} ::= \varnothing \mid m;\, \mathtt{M} \qquad \mathbf{s} ::= (\nu,\, S) \qquad \mathbf{S} ::= (\nu,\, S,\, \mathtt{M}) \quad (2.3)$$

where transition label ℓ is either an interaction ($\Box m$), time-step ($t \in \mathbb{R}_{\geqslant 0}$) or silent action ($\tau$), and m is a message and $\mathtt{M}$ is a FIFO message queue. Communication directions $\Box$ are the same as in Eq (2.2), where ! denotes a send/output action and ? a receive/input action.

2.2.1. *Configurations.* A configuration $\mathbf{s}$ is a pair $(\nu,\, S)$. The semantics for configurations are defined by a Labelled Transition System (LTS) over configurations, the labels in Eq (2.3) and the rules given in Eq (2.4). A transition $(\nu,\, S) \xrightarrow{t\ \Box m} (\nu',\, S')$ indicates $(\nu,\, S) \xrightarrow{t} \mathbf{s}'' \xrightarrow{\Box m} (\nu',\, S')$, where $\mathbf{s}''$ is some intermediate configuration. Changes from [BMVY19] are highlighted.

$$\frac{\nu \models \delta_j \quad m = l_j\,\langle T_j\rangle \quad j \in I}{(\nu,\, \{\Box_i\, l_i\,\langle T_i\rangle\,(\delta_i, \lambda_i).S_i\}_{i\in I}) \xrightarrow{\Box_j m} (\nu\,[\lambda_j \mapsto 0]\,,\, S_j)}\ [\mathtt{act}]$$

$$\frac{(\nu,\, S\,[{}^{\mu\alpha.S}/_{\alpha}]) \xrightarrow{\ell} (\nu',\, S')}{(\nu,\, \mu\alpha.S) \xrightarrow{\ell} (\nu',\, S')}\ [\mathtt{unfold}] \qquad (\nu,\, S) \xrightarrow{t} (\nu + t,\, S) \quad [\mathtt{tick}] \tag{2.4}$$

By rule [act] a configuration may perform one action $j \in I$, provided the corresponding δ_j is satisfied by the current valuation of clocks ν. The resulting configuration has all clocks in λ_j reset to 0. Rule [tick] describes time passing. Rule [unfold] unfolds recursive types.

Definition 2.3 (Future-enabled Configurations)**.** $(\nu,\, S)$ is *future-enabled*, written either as $(\nu,\, S) \overset{\Box}{\Rightarrow}$ or $(\nu,\, S)$ is $\mathtt{fe}$, iff $\exists t$ such that $(\nu,\, S) \xrightarrow{t\ \Box m}$ via the rules in Eq (2.4).

2.2.2. *Configurations with queues.* A configuration with queues $\mathbf{S}$ is a triple $(\nu,\, S,\, \mathtt{M})$, where $\mathtt{M}$ is a FIFO queue of messages which have been received but not yet processed. A queue takes the form $\mathtt{M} ::= \varnothing \mid m;\mathtt{M}$ and thus is either empty, or has a message at its head. The semantics of configurations with queues is defined by an LTS over the labels in Eq (2.3) and the rules in Eq (2.5). The transition $\mathbf{S} \xrightarrow{t\ \Box m} \mathbf{S}'$ is defined analogously to $(\nu,\, S) \xrightarrow{t\ \Box m} (\nu',\, S')$.

$$\frac{(\nu,\, S) \xrightarrow{!m} (\nu',\, S')}{(\nu,\, S,\, \mathtt{M}) \xrightarrow{!m} (\nu',\, S',\, \mathtt{M})}\ [\mathtt{send}] \qquad \frac{(\nu,\, S) \xrightarrow{?m} (\nu',\, S')}{(\nu,\, S,\, m;\, \mathtt{M}) \xrightarrow{\tau} (\nu',\, S',\, \mathtt{M})}\ [\mathtt{recv}]$$

$$(\nu,\, S,\, \mathtt{M}) \xrightarrow{?m} (\nu,\, S,\, \mathtt{M};\, m) \quad [\mathtt{que}]$$

$$\frac{\begin{array}{ll} (\nu,\, S) \xrightarrow{t} (\nu',\, S') & \text{(configuration)} \\ ((\nu,\, S) \text{ is } \mathtt{fe}) \implies ((\nu',\, S') \text{ is } \mathtt{fe}) & \text{(persistency)} \\ \forall t' < t : (\nu + t',\, S,\, \mathtt{M}) \xrightarrow{\tau}\!\!\!\!\!/\ \ & \text{(urgency)} \end{array}}{(\nu,\, S,\, \mathtt{M}) \xrightarrow{t} (\nu',\, S',\, \mathtt{M})}\ [\mathtt{time}] \tag{2.5}$$

Rule [send] emits a message m. Message reception is handled by two rules: rule [que] inserts a message at the back of $\mathtt{M}$, and rule [recv] removes a message from the front of $\mathtt{M}$.

Rule [time] is for time passing which is formulated in terms of a future-enabled configuration, given in Definition 2.3. The second condition in the premise of rule [time] (persistency) ensures the "latest-enabled" action is never missed by advancing the clocks. Notably, this differs from the corresponding behaviour in [BMVY19] which, in the presence of mixed-choice, is too restrictive. By not allowing a participant to pass its last sending action, the latest-enabled receiving actions can never be reached (i.e., miss sending to receive a timeout). The third condition (urgency) models an urgent semantics, ensuring messages are processed as they arrive. Urgency is critical for reasoning about progress. Later, in Example 2.11 we further discuss the changes made to rule [time] and illustrate why they are necessary.

2.2.3. *Systems.* Systems are the parallel composition of two (as we consider binary types) configurations with queues, written as $(\nu_1,\, S_1,\, \mathtt{M}_1) \mid (\nu_2,\, S_2,\, \mathtt{M}_2)$ or $\mathbf{S}_2 \mid \mathbf{S}_2$. The semantics of systems is defined by an LTS over the labels in Eq (2.3) and the transition rules in Eq (2.6).

$$\frac{\mathbf{S}_1 \xrightarrow{!m} \mathbf{S}_1' \quad \mathbf{S}_2 \xrightarrow{?m} \mathbf{S}_2'}{\mathbf{S}_1 \mid \mathbf{S}_2 \xrightarrow{\tau} \mathbf{S}_1' \mid \mathbf{S}_2'}\,[\texttt{com-l}] \qquad \frac{\mathbf{S}_1 \xrightarrow{\tau} \mathbf{S}_1'}{\mathbf{S}_1 \mid \mathbf{S}_2 \xrightarrow{\tau} \mathbf{S}_1' \mid \mathbf{S}_2}\,[\texttt{par-l}] \qquad \frac{\mathbf{S}_1 \xrightarrow{t} \mathbf{S}_1' \quad \mathbf{S}_2 \xrightarrow{t} \mathbf{S}_2'}{\mathbf{S}_1 \mid \mathbf{S}_2 \xrightarrow{t} \mathbf{S}_1' \mid \mathbf{S}_2'}\,[\texttt{wait}] \tag{2.6}$$

$$\frac{\mathbf{S}_1 \xrightarrow{?m} \mathbf{S}_1' \quad \mathbf{S}_2 \xrightarrow{!m} \mathbf{S}_2'}{\mathbf{S}_1 \mid \mathbf{S}_2 \xrightarrow{\tau} \mathbf{S}_1' \mid \mathbf{S}_2'}\,[\texttt{com-r}] \qquad \frac{\mathbf{S}_2 \xrightarrow{\tau} \mathbf{S}_2'}{\mathbf{S}_1 \mid \mathbf{S}_2 \xrightarrow{\tau} \mathbf{S}_1 \mid \mathbf{S}_2'}\,[\texttt{par-r}]$$

Rule [com-l] handles $\mathbf{S}_1$ asynchronously sending m to the queue of $\mathbf{S}_2$; rule [com-r] is symmetric. Rule [par-l] allows $\mathbf{S}_1$ to process the message at the head of $\mathtt{M}_1$ via [recv]; rule [par-r] is symmetric. By rule [wait] time passes consistently across systems.

Example 2.4 (Unsafe Mixed-choice)**.** The use of mixed-choice in asynchronous communications may result in infeasible protocols or, more concretely, systems (or types) that get stuck. A mixed-choice is considered *unsafe* if actions of different directions compete to be performed (i.e., they are both viable at the same point in time). Consider the system $(\nu_0,\, S_1,\, \varnothing) \mid (\nu_0,\, S_2,\, \varnothing)$, where S_1 and S_2 are *dual* and defined as follows:

$$S_1 = \begin{Bmatrix} ?a\,(x<5,\varnothing)\,.\texttt{end} \\ !b\,(x=0,\varnothing)\,.S_1' \end{Bmatrix} \qquad\qquad S_2 = \begin{Bmatrix} !a\,(y<5,\varnothing)\,.\texttt{end} \\ ?b\,(y=0,\varnothing)\,.S_2' \end{Bmatrix} \tag{2.7}$$

In the system $\mathbf{S}_1 \mid \mathbf{S}_2$, it is possible for both $(\nu_0,\, S_1,\, \varnothing) \xrightarrow{!b} \mathbf{S}_1'$ and $(\nu_0,\, S_2,\, \varnothing) \xrightarrow{!a} \mathbf{S}_2'$ to occur at the same time. The resulting system $(\nu_0,\, S_1',\, a) \mid (\nu_0,\, \texttt{end},\, b)$ is unable to receive either of message a or b, and $\mathbf{S}_1'$ may be stuck waiting for interactions from $\mathbf{S}_2'$ indefinitely. Even though a deadlock can be avoided if message a is a delegation and S_1' prescribes receiving a, since message b is never received by $\mathbf{S}_2$, then Eq (2.7) still prescribes unsafe behaviour. △

2.3. **Duality, Well-formedness, and Progress (of TOAST).**

In the untimed scenario, the composition of a well-formed binary type with its dual characterises a *protocol*, which specifies the "correct" set of interactions between a party and its co-party. The dual of a type, formally defined below, is obtained by swapping the directions (! or ?) of each interaction:

Definition 2.5 (Type Duality)**.** Given a type S, we define its *dual* type $\overline{S}$ as follows:

$$\overline{\texttt{end}} = \texttt{end} \qquad \overline{\alpha} = \alpha \qquad \overline{\mu\alpha.S} = \mu\alpha.\overline{S} \qquad \overline{?} = ! \qquad \overline{!} = ?$$

$$\overline{\left\{\Box_i\, l_i\, \langle T_i\rangle\, (\delta_i,\lambda_i).S_i\right\}_{i\in I}} = \left\{\overline{\Box}_i\, l_i\langle T_i\rangle(\delta_i,\lambda_i).\overline{S}_i\right\}_{i\in I}$$

Notice that the time constraints on actions are identical for a type and its dual. For example, $(\nu_0,\ !a\,(x = 3, \varnothing)\,.S') \mid (\nu_0,\ ?a\,(x = 3, \varnothing)\,.\overline{S'})$ is clearly a system with dual types and identical clock valuations. It is crucial to remember that both the clock constraints and valuations of each configuration are on their own local set of clocks, and that 'x' is not a clock shared by each. For this reason, in our examples we often use different clock names for dual types. E.g., $(\nu_0,\ !a\,(x = 3, \varnothing)\,.S') \mid (\nu_0,\ ?a\,(y = 3, \varnothing)\,.\overline{S'})$. This is purely a matter of presentation.

Later, in Remark 2.7 we shall discuss the reason for our 'simple' definition of duality in regard to recursive types, as opposed to [BP12, LM16].

2.3.1. *Well-formedness.* We require a notion of *well-formedness* so that we may later provide guarantees of progress for types with mixed-choice. The formation rules for types are given in Eq (2.8). Again, since we extend [BMVY19] any rules that differ are clearly highlighted. Types are evaluated against judgements of the form: $A; \delta \vdash S$ where A is an environment containing recursive variables and δ is a constraint over all clocks characterising the times in which state S can be reached. Formally $A ::= \varnothing \mid \alpha : \delta, A$.

$$\frac{\begin{array}{ll} \forall i \in I : A;\ \gamma_i \vdash S_i\ \wedge\ \delta_i\,[\lambda_i \mapsto 0] \models \gamma_i & \text{(feasibility)} \\ \forall i, j \in I : i \neq j\ \implies\ \delta_i \wedge \delta_j \models \texttt{false}\ \vee\ \square_i = \square_j & \text{(mixed-choice)} \\ \forall i \in I : T_i = \left(\delta',\ S'\right)\ \implies\ \varnothing;\ \gamma' \vdash S'\ \wedge\ \delta' \models \gamma' & \text{(delegation)} \end{array}}{A;\ \downarrow \bigvee_{i \in I} \delta_i \vdash \left\{\square_i\, l_i \langle T_i \rangle\, (\delta_i, \lambda_i).S_i\right\}_{i \in I}}\ [\texttt{choice}] \tag{2.8}$$

$$\frac{}{A;\ \texttt{true} \vdash \texttt{end}}\ [\texttt{end}] \qquad \frac{A, \alpha : \delta;\ \delta \vdash S}{A;\ \delta \vdash \mu\alpha.S}\ [\texttt{rec}] \qquad \frac{}{A, \alpha : \delta;\ \delta \vdash \alpha}\ [\texttt{var}]$$

Rule [choice] checks well-formedness of choices with three conditions: the first and third conditions are from the branching and delegation rules in [BMVY19], respectively, and the second condition is new and critical to ensure progress of mixed-choice, By the first condition (feasibility) a choice is well-formed with respect to the weakest past among all options ($\downarrow \bigvee_{i \in I} \delta_i$) given that each continuation S_i is well-formed with respect to an environment γ_i, which models the corresponding guard updated with resets λ_i. By abuse of notation we write $\delta[\lambda \mapsto 0] \models \gamma$ to say $\forall \nu : (\nu \models \delta[\lambda \mapsto 0]) \implies (\nu \models \gamma)$, where ($\implies$) is an implication. This ensures that in every choice there is at least one viable action and, for example, would rule out Example 2.2. The second condition (mixed-choice) requires all actions that can occur at the same time to have the same (send/receive) direction. This condition allows for types modelling timeouts (discussed later in Example 2.11) and rules out scenarios as the one in Example 2.4. The third condition (delegation) checks for well-formedness of each delegated session with respect to their corresponding initialisation constraint δ'. Similar to the (feasibility) premise, by abuse of notation we write $\delta' \models \gamma'$ to say $\forall \nu : (\nu \models \delta') \implies (\nu \models \gamma')$. Rule [end] ensures termination types are *always* well-formed. Rule [rec] associates variable α with invariant δ in A. Rule [var] ensures recursive calls are defined.

Definition 2.6 (Well-formed Configurations)**.** Given $(\nu,\ S)$, S is *well-formed* against ν if, $\exists \delta$ such that $\varnothing; \delta \vdash S$ and $\nu \models \delta$. A type S is *well-formed* if it is *well-formed* against ν_0.

The rules in Eq (2.8) check that in every reachable state (which includes every possible clock valuation) it is possible to perform the next action immediately or at some point in the future, unless the state is final. (This is formalised as the progress property in Definition 2.8.) By these rules, the type in Examples 2.2 and 2.4 would not be well-formed.

Remark 2.7 (Delegation of Higher-Order TOAST)**.** Recall Eq (2.2) in Section 2.1, where higher-order types for session delegation are defined as (δ, S). By condition (delegation) in the premise of rule [`choice`] in Eq (2.8), any interaction with a delegation payload $T = (\delta', S')$, delegation type S' must be *well-formed* by $\varnothing; \gamma' \vdash S'$ where environment γ' corresponds to δ' and the environment for recursive variables is empty. In short, a *well-formed* TOAST cannot delegate a recursive variable (α). Furthermore, higher-order types must be *flattened* (i.e., finitely represented) and TOAST is unable to express self-referential recursive higher-order types (e.g., $S' = \mu\alpha.!l\langle(\delta', S')\rangle(\delta, \varnothing).\alpha$). This contrasts with [BH16, BP12], which allow recursive variables to be delegated directly, and [LM16] which discusses self-referential recursive higher-order types. We feel this is orthogonal to the primary focus of this work.

2.3.2. *Progress of Types.* Well-formedness, together with the persistency and receive urgency featured in rule [`time`] of the semantics in Eq (2.5) ensures that the composition of a well-formed type S with its dual $\overline{S}$ enjoys progress. A system enjoys progress if its configurations with queues can continue communicating until reaching the end of the protocol; formally:

Definition 2.8 (Type Progress)**.** A configuration with queues $(\nu,\, S,\, \mathtt{M})$ is *final* if $S \overset{\mathtt{unfold}}{\equiv} \mathtt{end}$ and $\mathtt{M} = \varnothing$. A system $\mathbf{S}_1 \mid \mathbf{S}_2$ *satisfies progress* if, for all $\mathbf{S}'_1 \mid \mathbf{S}'_2$ reachable from $\mathbf{S}_1 \mid \mathbf{S}_2$:

(1) either $(\nu'_1,\, S'_1,\, \mathtt{M}'_1)$ and $(\nu'_2,\, S'_2,\, \mathtt{M}'_2)$ are *final*;

(2) or, there exists a $t \in \mathbb{R}_{\geqslant 0}$ such that $(\nu'_1,\, S'_1,\, \mathtt{M}'_1) \mid (\nu'_2,\, S'_2,\, \mathtt{M}'_2) \xrightarrow{t\ \tau}$.

We give the definition of $(\overset{\mathtt{unfold}}{\equiv})$ in Definition A.6. We write $S \overset{\mathtt{unfold}}{\equiv} \mathtt{end}$ if S is equivalent to `end`, up-to the unfolding of recursive types. (e.g., $\mu\alpha_1 \dots \mu\alpha_i.\mathtt{end} \overset{\mathtt{unfold}}{\equiv} \mathtt{end}$.)

Theorem 2.9 (Progress of Systems)**.** *If S is* well-formed *against ν_0, then:*

$$(\nu_0,\, S,\, \varnothing) \mid (\nu_0,\, \overline{S},\, \varnothing) \text{ satisfies progress.}$$

The main result of this section is that for a system composed of a *well-formed* S and its dual $(\nu_1,\, S_1,\, \mathtt{M}_1) \mid (\nu_2,\, \overline{S}_2,\, \mathtt{M}_2)$, any state reached is either *final*, or allows for further communication; i.e., the system *satisfies progress*. Progress is critical to ensure, via type-checking, that a protocol implementation does not reach deadlock and is free from communication mismatches. (We introduce such a system for type-checking using TOAST in Section 5.)

The main differences from [BMVY19] are not in the formulation of the theory (e.g., Definition 2.8 and the statement of Theorem 2.9 are basically unchanged) but in proving rule [`choice`] is sufficient to ensure progress of asynchronous mixed-choice. Additionally, the proof of progress in [BMVY19] relies on a notion of persistency to a participant does not reach a point where there are no viable actions, and does so by requiring time-steps do not pass beyond the participants latest-enabled receiving action. Due to mixed-choice, it is necessary to reformulate (and relax) this condition in the semantic rule [`time`] in Eq (2.5) to require only the latest-enabled action is not missed. (See Example 2.11 for a discussion).

2.3.3. *Compatibility.* The proof of Theorem 2.9 is in Appendix A, and proceeds by showing that a property of systems called *compatibility* is preserved by transitions; given formally in Definition 2.10. As typical for binary systems, our proof of progress builds upon a notion of *duality* between participants. Since communication is asynchronous, each party may break duality with their co-party. Compatibility allows the duality of participants to be broken, only if, doing so does not violate *communication safety.* More specifically, compatibility requires that any message that arrives in a message queue is expected and therefore, able to be received and that the resulting configurations are still *compatible*. In practice, compatibility allows each party to behave independently, regardless the state of the other party, while retaining the essence of *duality*, and guaranteeing that all messages will eventually be received.

Definition 2.10 (Compatibility)**.** We write $(\nu_1, S_1, \mathtt{M}_1) \perp (\nu_2, S_2, \mathtt{M}_2)$ iff $(\nu_1, S_1, \mathtt{M}_1)$ and $(\nu_2, S_2, \mathtt{M}_2)$ are *compatible*. Given $(\nu_1, S_1, \mathtt{M}_1)$ and $(\nu_2, S_2, \mathtt{M}_2)$, we define *compatibility* as the largest relation satisfying each of the following conditions:

(1) $(\mathtt{M}_1 = \varnothing)$ or $(\mathtt{M}_2 = \varnothing)$

(2) $(\mathtt{M}_1 = m; \mathtt{M}_1') \implies ((\nu_1, S_1) \xrightarrow{?m} (\nu_1', S_1')$ and $(\nu_1', S_1', \mathtt{M}_1') \perp (\nu_2, S_2, \mathtt{M}_2))$

(3) $(\mathtt{M}_2 = m; \mathtt{M}_2') \implies ((\nu_2, S_2) \xrightarrow{?m} (\nu_2', S_2')$ and $(\nu_1, S_1, \mathtt{M}_1) \perp (\nu_2', S_2', \mathtt{M}_2'))$

(4) $(\mathtt{M}_1 = \varnothing = \mathtt{M}_2) \implies (S_1 = \overline{S_2}$ and $\nu_1 = \nu_2)$

Intuitively, $\mathbf{S}_1$ and $\mathbf{S}_2$ are compatible, written $\mathbf{S}_1 \perp \mathbf{S}_2$, if: (1) at most one of their queues is non-empty (equivalent to a half duplex automaton), (2 – 3) a type is always able to process any message that arrives in its queue, and (4) if both queues are empty then $\mathbf{S}_1$ and $\mathbf{S}_2$ have dual types and same clock valuations.

Example 2.11 (Weak Persistency)**.** In language-based approaches to timed semantics [KY06] time actions are always possible, even if they bring the model into a stuck state by preventing available actions. Execution traces are then filtered as necessary, removing all 'bad' traces (defined on the basis of final states). In contrast, and to facilitate the reasoning on process behaviour, we adopt a process-based approach, that only allows for actions that characterise *intended* executions of the model (as in [BBM18, BMVY19]). Precisely, we build on the semantics of [BBM18] for asynchronous timed automata with mixed-choice, where time actions are possible only if they do not disable: (a) the latest-enabled sending action, and (b) the (latest-enabled) receiving action if the queue is not empty. This ensures that time actions preserve the viability of at least one action (*weak-persistency*). In our scenario, constraint (a) is too strict. Consider type S and its dual $\overline{S}$ below:

$$S = \begin{Bmatrix} !data\langle\mathtt{String}\rangle\,(x < 3, \varnothing)\,.S' \\ ?timeout\,(x \geqslant 4, \varnothing)\,.\mathtt{end} \end{Bmatrix} \qquad \overline{S} = \begin{Bmatrix} ?data\langle\mathtt{String}\rangle\,(y < 3, \varnothing)\,.\overline{S'} \\ !timeout\,(y \geqslant 4, \varnothing)\,.\mathtt{end} \end{Bmatrix}$$

According to (a), it would never be possible for S to take the *timeout* branch since a time action of $t \geqslant 3$ would disable the latest-enabled send (*data*). This is reasonable in a setting where the behaviour of such a mixed-choice has no guarantee of a timeout being received later, such as in [BBM18]. However, this is not the case in our setting as we can rely on duality to guarantee that $\overline{S}$ will send *timeout* in all executions where $y \geqslant 4$. Our new [`time`] rule – condition (persistency) – implements a more general constraint than (a), requiring that one latest-enabled (send or receive) action is preserved. Constraint (b) remains as condition (urgency) and, for instance, prevents $\overline{S}$ from sending *timeout* if *data* is waiting in the queue when $y < 3$. For example, our semantics permit the following sequence of transitions:

$$(\nu_0, S, \varnothing) \xrightarrow{t=2} (\nu[x \mapsto 2], S, \varnothing) \xrightarrow{!data\langle\mathtt{String}\rangle} (\nu[x \mapsto 2], S', \varnothing)$$

where *data* is sent after 2 time units. Alternatively, by our semantics, the following holds:

$$(\nu_0, S, \varnothing) \xrightarrow{t=4} (\nu[x \mapsto 4], S, \varnothing) \xrightarrow{?timeout} (\nu[x \mapsto 4], S, timeout; \varnothing) \stackrel{t'}{\not\rightarrow} \qquad (t' \in \mathbb{R}_{\geqslant 0})$$

where condition (urgency) ensures no time can pass since a message currently in the queue that can be received via rule [`recv`]: $(\nu[x \mapsto 4], S, timeout; \varnothing) \xrightarrow{\tau} (\nu[x \mapsto 4], \mathtt{end}, \varnothing)$. △

3. A Calculus for Processes with Timeouts

We present a new calculus for timed processes which extends existing timed session calculi [BLY15, BMVY19] with timeouts and time-sensitive conditional statements. Timeouts are defined on receive actions and may be immediately followed by sending actions, hence providing an instance of mixed-choice – which is normally not supported. Time-sensitive conditional statements (i.e., `if-then-else` with conditions on process timers) provide a natural counter-part to the timeout construct and enhance the expressiveness of the typing system in [BMVY19]. To better align processes with TOAST, send and select actions have been streamlined by each message consisting of both a label l and some message variable v, which is either data or a delegated session; the same holds for receive/branch actions.

Processes are defined by the grammar given in Eq (3.1). Participants are the *endpoints* of session and are denoted by p and q. Within a (binary) session, endpoints p and q communicate over channels pq and qp, where p sends to q over pq, and q sends to p over qp. We also use a and b as ad-hoc roles, with channels ab and ba, when discussing multiple binary sessions.

$$
\begin{array}{rllll}
P, Q ::= & \texttt{set}\,ⓧ.P & \mid\ (\nu pq)\ P & & \\
\mid & p \triangleleft l\,(v).P & \mid\ P \mid Q & & \\
\mid & p^{e} \triangleright \{l_i(v_i) : P_i\}_{i\in I} & \mid\ \mathtt{0} & & \\
\mid & p^{\diamond n} \triangleright \{l_i(v_i) : P_i\}_{i\in I}\ \texttt{after}\ Q & \mid\ pq : h & & \\
\mid & \texttt{if}\ c\ \texttt{then}\ P\ \texttt{else}\ Q & e ::= \diamond n \mid \infty & (n \in \mathbb{R}_{\geqslant 0}) & (3.1)\\
\mid & \texttt{delay}\,(d)\,.P & \diamond ::= \ < \mid \leqslant & & \\
\mid & \texttt{delay}\,(t)\,.P & c ::= ⓧ \diamond n \mid n \diamond ⓧ \mid ⓧ = n & & \\
\mid & \texttt{def}\ X(\vec{v};\vec{r}) = P\ \texttt{in}\ Q & d ::= \ t \diamond n \mid n \diamond t \ \mid \texttt{true} & (t \in \mathbb{R}_{\geqslant 0}) & \\
\mid & X\langle \vec{v};\vec{r}\rangle & h ::= \ \varnothing \mid h \cdot lv & &
\end{array}
$$

Processes are equipped with timers that can be set and reset during the process execution. Let $\mathbb{T}$ be the set of timers denoted by ⓧ, ⓨ and ⓩ. While clocks are used in types to express the time constraints of a protocol (i.e., type), timers are used in processes to acquire awareness of relative time-passing during execution (as timer concurrency primitives in languages like Go and Erlang). Process $\texttt{set}\,ⓧ.P$ creates a process timer ⓧ, initialises it to 0 and continues as P. If a process timer named ⓧ already exists it is reset to 0. Note, there is no correspondence between the clocks used in the constraints of types and process timers, regardless if they have matching names/labels.

Process $p \triangleleft l\,(v).P$ is the select/send process: it selects label l and sends payload v to endpoint pq, and continues as P. Its counterpart is the branch/receive process $p^{e} \triangleright \{l_i(v_i) : P_i\}_{i\in I}$. It receives one of the labels l_i, instantiates v_i with the received payload, and continues as P_i. Parameter e is a deadline for the receive action. It can be either $\diamond n$ or ∞. If e is ∞, then the receive action will block until a message is received, waiting potentially forever. If e is $\diamond n$, then n is the upper-bound of the receive action and $\diamond$ specifies whether the wait duration is exclusive ($<$) or inclusive ($\leqslant$). For example, setting e to ($\leqslant 3$) specifies waiting *up-to and including* 3 time units, while (< 3) specifies waiting *strictly less than* 3 time units. Setting e to ($\leqslant 0$) models a non-blocking receive action on branches that expect messages to be immediately available to receive. Process $p^{\diamond n} \triangleright \{l_i(v_i) : P_i\}_{i\in I}$ `after` Q is the timeout process, an extended version of the branch/receive process, where $\diamond n$ specifies the receive deadline, after which the timeout process Q gets triggered. We specify timeouts to be ($\diamond n$) to tactically ensure timeouts cannot be ∞, which would make Q dead code and would be equivalent to a branch/receive action without a timeout. For simplicity, in branch/receive and timeout processes: (i) we omit the brackets in the case of a single option (i.e., $|I| = 1$) and (ii) for options with no payloads we omit (v).

Process $\texttt{if}\ c\ \texttt{then}\ P\ \texttt{else}\ Q$ is a conditional statement. The condition c is a constraint on process timers, and is notably simpler than those constraints found in Eq (2.1) for types. Process $\texttt{delay}\,(d)\,.P$ models time passing where d models a non-deterministic delay. At runtime process $\texttt{delay}\,(d)\,.P$ is reduced to process $\texttt{delay}\,(t')\,.P$, for some $t' \models d\,[t'/t]$, where the non-deterministic duration d is resolved into a single duration t'.

Recursive processes are defined by a process variable X and parameters $\vec{v}$ and $\vec{r}$, containing *base type* values and session channels, respectively. As standard [BCD+08, BYY14, HYC08, Vas12, YV07] the process calculus allows parallel processes $P \mid Q$ and scoped processes $(\nu pq)\ P$ between endpoints p and q. The end process is $\mathtt{0}$. Endpoints communicate over pairs of channels pq and qp, each with their own unbounded FIFO buffers h.

3.1. **Well-formed Processes.** We assume that sessions within a process have already been instantiated, meaning that rather than relying on reduction rules to produce correct session instantiation, we rely on a syntactic well-formedness assumption, as in [BMVY19]. A *well-formed process* P consists of sessions of the form $(\nu pq)\ \big(P' \mid Q \mid qp : h \mid pq : h'\big)$ and can be checked syntactically by function $\texttt{wf}\,(P)$ given in Definition 3.1.

Definition 3.1 (Well-formed Process)**.** The function $\texttt{wf}\,(P)$ is defined inductively as:

$$\texttt{wf}\,(P) = \begin{cases} \texttt{true} & \text{if } P \in \Big\{\mathtt{0},\ X\langle\vec{v};\vec{r}\rangle,\ qp : h\Big\} \\ \texttt{wf}\,(P') \wedge (\texttt{fq}\,(P') = \{pq, qp\}) & \text{if } P = (\nu pq)\ P' \\ \texttt{wf}\,(P') \wedge (\texttt{fq}\,(P') = \varnothing) & \text{if } P \in \begin{Bmatrix} p \lhd l\,(v).P',\ \texttt{set}\,\textcircled{x}.P', \\ \texttt{delay}\,(d)\,.P',\ \texttt{delay}\,(t)\,.P' \end{Bmatrix} \\ \texttt{wf}\,(P') \wedge \texttt{wf}\,(Q) & \text{if } P \in \begin{Bmatrix} \texttt{if}\ c\ \texttt{then}\ P'\ \texttt{else}\ Q, \\ \texttt{def}\ X(\vec{v};\vec{r}) = P'\ \texttt{in}\ Q \end{Bmatrix} \\ \texttt{wf}\,(P') \wedge \texttt{wf}\,(Q) \wedge (\texttt{ft}\,(P') \cap \texttt{ft}\,(Q) = \varnothing) & \text{if } P = P' \mid Q \\ \bigwedge_{i\in I} \texttt{wf}\,(P_i) \wedge (\texttt{fq}\,(P_i) = \varnothing) & \text{if } P = p^e \rhd \Big\{l_i(v_i) : P_i\Big\}_{i\in I} \\ \texttt{wf}\,\Big(p^{\diamond n} \rhd \Big\{l_i(v_i) : P_i\Big\}_{i\in I}\Big) \wedge \texttt{wf}\,(Q) & \text{if } P = p^{\diamond n} \rhd \Big\{l_i(v_i) : P_i\Big\}_{i\in I}\ \texttt{after}\ Q \end{cases}$$

where function $\texttt{fq}\,(P)$ returns the set of queues present in process P that are *not* contained within the respective scope of their corresponding session, and function $\texttt{ft}\,(P)$ returns the set of process timers used in process P. (We relegate the definitions of $\texttt{fq}\,(P)$ and $\texttt{ft}\,(P)$ to the appendix, see Definitions B.1 and B.2.) By Definition 3.1, a *well-formed* process must: (a) not contain any *free queues* other than those belonging to ongoing (binary) sessions contained within *scoped processes* $(\nu pq)\ P$, which should each contain the two queues necessary for participants to exchange messages, and (b) not contain any parallel processes that make use of the *same* process timer, i.e., process timers must only be used by a single process.

Hereafter, we assume that a process is *well-formed*, unless stated otherwise.

3.2. **Process Reduction.** The semantics of processes are given in Figure 2, as a reduction relation on pairs of the form (θ, P) i.e., a processes P with an timer environment θ that maps timers to their values in the current state (formally given below in Definition 3.2).

$$\frac{P \equiv P' \quad (\theta,\, P') \longrightarrow (\theta',\, Q') \quad Q \equiv Q'}{(\theta, P) \longrightarrow (\theta', Q)}\ [\texttt{Str}] \quad \frac{(\theta,\, P) \rightharpoonup (\theta',\, P')}{(\theta,\, (\nu pq)\ P) \rightharpoonup (\theta',\, (\nu pq)\ P')}\ [\texttt{Scope}]$$

$$\frac{(\theta_1,\, P) \rightharpoonup (\theta_1',\, P')}{(\theta_1, \theta_2,\, P \mid Q) \rightharpoonup (\theta_1', \theta_2,\, P' \mid Q)}\ [\texttt{Par-L}] \quad \frac{(\theta_2,\, Q) \rightharpoonup (\theta_2',\, Q')}{(\theta_1, \theta_2,\, P \mid Q) \rightharpoonup (\theta_1, \theta_2',\, P \mid Q')}\ [\texttt{Par-R}] \tag{3.2}$$

$$(\theta,\, p \triangleleft l\,(v).P \mid pq : h) \rightharpoonup (\theta,\, P \mid pq : h \cdot lv) \quad [\texttt{Send}]$$

$$\frac{j \in I \quad l = l_j}{(\theta,\, p^e \triangleright \{l_i(v_i) : P_i\}_{i \in I} \mid qp : lv \cdot h) \rightharpoonup (\theta,\, P_j\,[{}^{v}/_{v_j}] \mid qp : h)}\ [\texttt{Recv}] \tag{3.3}$$

$$\frac{j \in I \quad l = l_j}{(\theta,\, p^{\diamond n} \triangleright \{l_i(v_i) : P_i\}_{i \in I}\ \texttt{after}\ Q \mid qp : lv \cdot h) \rightharpoonup (\theta,\, P_j\,[{}^{v}/_{v_j}] \mid qp : h)}\ [\texttt{Recv-T}]$$

$$\frac{(\theta,\, Q) \rightharpoonup (\theta',\, Q')}{(\theta,\, \texttt{def}\ X(\vec{v};\vec{r}) = P\ \texttt{in}\ Q) \rightharpoonup (\theta',\, \texttt{def}\ X(\vec{v};\vec{r}) = P\ \texttt{in}\ Q')}\ [\texttt{Def}] \tag{3.4}$$

$$(\theta,\, \texttt{def}\ X(\vec{v'};\vec{r'}) = P\ \texttt{in}\ X\langle \vec{v};\vec{r}\rangle \mid Q) \rightharpoonup (\theta,\, \texttt{def}\ X(\vec{v'};\vec{r'}) = P\ \texttt{in}\ P\,[{}^{\vec{v};\vec{r}}/_{\vec{v'};\vec{r'}}] \mid Q) \quad [\texttt{Call}]$$

$$\frac{\theta \models c}{(\theta,\, \texttt{if}\ c\ \texttt{then}\ P\ \texttt{else}\ Q) \rightharpoonup (\theta,\, P)}\ [\texttt{If-T}] \quad (\theta, ⓧ : n,\, \texttt{set}\,ⓧ.P) \rightharpoonup (\theta, ⓧ : 0,\, P)\ [\texttt{Reset}]$$

$$\frac{\theta \not\models c}{(\theta,\, \texttt{if}\ c\ \texttt{then}\ P\ \texttt{else}\ Q) \rightharpoonup (\theta,\, P)}\ [\texttt{If-F}] \quad (\theta, P) \rightsquigarrow (\theta + t,\, \Phi_t\,(P))\ [\texttt{Delay}] \tag{3.5}$$

$$\frac{t' \models d\,[{}^{t'}/_{t}]}{(\theta,\, \texttt{delay}\,(d)\,.P) \rightharpoonup (\theta,\, \texttt{delay}\,(t')\,.P)}\ [\texttt{Det}]$$

Figure 2. Rules of process reduction

Definition 3.2 (Timer Environment)**.** Recall $\mathbb{T}$ is the set of timers, ranged over by ⓧ, ⓨ and ⓩ. A *timer environment* θ is a linear map $\theta : \mathbb{T} \mapsto \mathbb{R}_{\geqslant 0}$, from timers to valuations, defined: $\theta ::= \varnothing \mid \theta,\ ⓧ : n$ where $n \in \mathbb{R}_{\geqslant 0}$. We define $\theta + t = \{ⓧ \mapsto \theta(ⓧ) + t \mid ⓧ \in \mathbb{T}\}$ so that time may pass over θ, and $\theta\,[ⓧ \mapsto 0]$ to be the map $\theta\,[ⓧ \mapsto 0]\,(ⓨ) =$ if $(ⓧ = ⓨ)$ 0 else $\theta(ⓨ)$. We write $\mathrm{dom}(\theta)$ for the domain of θ, and θ_1, θ_2 for $\theta_1 \cup \theta_2$ when $\mathrm{dom}(\theta_1) \cap \mathrm{dom}(\theta_2) = \varnothing$. We write $\theta \models c$ to denote that θ satisfies c. Defined formally, $(\theta \models c) = (\forall ⓧ \in \texttt{fn}\,(c)\,.c\,[{}^{\theta(ⓧ)}/_{ⓧ}])$.

3.2.1. *Reduction Rules.* The reduction relation is defined on two kinds of reduction: instantaneous communication actions ($\rightharpoonup$), and time-consuming actions ($\rightsquigarrow$). We write $\longrightarrow$ to denote a reduction that is either by ($\rightharpoonup$) or ($\rightsquigarrow$).

Rules [Str], [Scope], [Par-L] and [Par-R] are as standard [BCD+08, HYC08, Vas12, YV07]. The rule for structural congruence [Str] is the only rule that applies to both instantaneous and time-consuming actions. Structural equivalence follows as standard [HYC08, Mil99, Pie02, Vas12, YV07], with additions from [BYY14, BMVY19]. Formally given below:

Definition 3.3 (Structural Congruence)**.** We define $P \equiv Q$ as the smallest relation on processes:

$$P \mid \mathtt{0} \equiv P \qquad (P \mid Q) \equiv (Q \mid P) \qquad (P_1 \mid P_2) \mid Q \equiv (P_1 \mid Q) \mid P_2 \qquad P \mid Q \equiv Q \mid P \qquad (\nu pq)\ \mathtt{0} \equiv \mathtt{0}$$

$$\mathtt{delay}\,(0)\,.P \equiv P \qquad (\nu pq)\ (\nu ab)\ P \equiv (\nu ab)\ (\nu pq)\ P \qquad (\nu pq)\ P \mid Q \equiv (\nu pq)\ (P \mid Q) \quad \text{if } pq \notin \mathtt{fn}\,(Q)$$

$$(\nu pq)\ P \equiv (\nu qp)\ P \qquad (\mathtt{def}\ X(\vec{v};\vec{r}) = P\ \mathtt{in}\ P') \mid Q \equiv \mathtt{def}\ X(\vec{v};\vec{r}) = P\ \mathtt{in}\ (P' \mid Q) \quad \text{if } X\langle \vec{v'};\vec{r'}\rangle \notin \mathtt{fpv}\,(Q)$$

where $\mathtt{fn}\,(Q)$ and $\mathtt{fpv}\,(Q)$ return the set of free names and process variables in Q, respectively.

3.2.2. *Communication Rules, Eq (3.3).* Rules [Send] and [Recv] are standard [Mil99]. By rule [Send] a process inserts a message lv into the queue of the other party, while by rule [Recv] a process may remove an expected message from their queue and continue on the corresponding branch. Rule [Recv-T] is similar to rule [Recv] except, as we will discuss shortly in Section 3.3, it allows the timeout branch Q to be taken if the process does not receive a message within duration e. (We include rule [Recv-T] for consistency, but hereafter only refer to rule [Recv].)

3.2.3. *Recursion Rules, Eq (3.4).* Both rules [Def] and [Call] are unchanged from [BMVY19]. Rule [Def] defines a recursive process P and allows the recursive body Q to reduce. Rule [Call] handles recursive calls of predefined recursive processes and yields a process configuration consisting of the next iteration of the recursive process using the parameters provided by the process variable X; the parallel process Q is unaffected.

3.2.4. *Time-sensitive Rules, Eq (3.5).* Rule [Reset] resets a process timer $ⓧ$ in the timer environment θ to 0. Rule [If-T] selects branch P if time-sensitive condition c holds for θ. Rule [If-F] is symmetric, selecting Q if c does not hold for θ. Rule [Det] determines the exact duration t' of a non-deterministic runtime delay modelled by d. Rule [Delay] handles delay processes, so long as they are defined in $\Phi_t\,(P)$ which, in short, returns process P after t units of time have elapsed. (We formally define $\Phi_t\,(P)$ below in Definition 3.4.) Additionally, by rule [Delay] any delay on a process also causes the timers within θ to elapse accordingly.

3.3. **Time Passing.** The definition of $\Phi_t\,(P)$ is given in Definition 3.4. The first two cases in Definition 3.4 model the effect of time passing on branching and timeout processes. The third case is for time-consuming processes. The fourth case distributes time passing in parallel compositions and ensures that time passes for all parts of the system equally. The remaining cases define the processes where time is allowed to pass.

Definition 3.4 (Time Passing Function)**.** The time-passing function $\Phi_t\,(P)$ is a partial function only defined for the cases below:

$$\Phi_t\left(p^{\diamond n} \triangleright \{l_i(v_i) : P_i\}_{i\in I}\ \mathtt{after}\ Q\right) = \begin{cases} p^{\diamond n-t} \triangleright \left\{l_i(v_i) : P_i\right\}_{i\in I}\ \mathtt{after}\ Q & \text{if } (t \diamond n) \\ \Phi_{t-n}\,(Q) & \text{otherwise} \end{cases}$$

$$\Phi_t\left(p^{e} \triangleright \{l_i(v_i) : P_i\}_{i\in I}\right) = \begin{cases} p^{e} \triangleright \left\{l_i(v_i) : P_i\right\}_{i\in I} & \text{if } e = \infty \\ p^{n-t} \triangleright \left\{l_i(v_i) : P_i\right\}_{i\in I} & \text{if } e = \diamond n \text{ and } (t \diamond n) \end{cases}$$

$$\Phi_t\left(\mathtt{delay}\,(t')\,.P\right) = \begin{cases} \mathtt{delay}\,(t'-t)\,.P & \text{if } t' \geqslant t \\ \Phi_{t-t'}\,(P) & \text{otherwise} \end{cases}$$

$$\Phi_t\,(pq : h) = pq : h \qquad \Phi_t\,(P_1 \mid P_2) = \Phi_t\,(P_1) \mid \Phi_t\,(P_2) \quad \text{if } (\mathtt{Wait}\,(P_i) \cap \mathtt{NEQ}\,(P_j) = \varnothing) \text{ and } i \neq j \in \{1,2\}$$

$$\Phi_t\,(\mathtt{0}) = \mathtt{0} \qquad \Phi_t\,((\nu pq)\ P) = (\nu pq)\ \Phi_t\,(P) \qquad \Phi_t\,(\mathtt{def}\ X(\vec{v};\vec{r}) = P\ \mathtt{in}\ Q) = \mathtt{def}\ X(\vec{v};\vec{r}) = P\ \mathtt{in}\ \Phi_t\,(Q)$$

$$\mathtt{Wait}\,(P) = \begin{cases} \{p\} & \text{if } P \in \left\{p^{\diamond n} \triangleright \left\{l_i(v_i) : P_i\right\}_{i\in I} \; \mathtt{after}\; Q,\; p^{e} \triangleright \left\{l_i(v_i) : P_i\right\}_{i\in I}\right\} \\ \mathtt{Wait}\,(Q) \setminus \{p, q\} & \text{if } P = (\nu pq)\; Q \\ \mathtt{Wait}\,(Q) & \text{if } P = \mathtt{def}\; X(\vec{v};\vec{r}) = P' \;\mathtt{in}\; Q \\ \mathtt{Wait}\,(P') \cup \mathtt{Wait}\,(Q) & \text{if } P = P' \mid Q \\ \varnothing & \text{otherwise} \end{cases}$$

$$\mathtt{NEQ}\,(P) = \begin{cases} \{p\} & \text{if } P = qp : h \wedge h \neq \varnothing \\ \mathtt{NEQ}\,(Q) \setminus \{p, q\} & \text{if } P = (\nu pq)\; Q \\ \mathtt{NEQ}\,(Q) & \text{if } P = \mathtt{def}\; X(\vec{v};\vec{r}) = P' \;\mathtt{in}\; Q \\ \mathtt{NEQ}\,(P') \cup \mathtt{NEQ}\,(Q) & \text{if } P = P' \mid Q \\ \varnothing & \text{otherwise} \end{cases}$$

Figure 3. Definition of $\mathtt{Wait}\,(P)$ and $\mathtt{NEQ}\,(P)$.

The auxiliary functions $\mathtt{Wait}\,(P)$ and $\mathtt{NEQ}\,(P)$ are given in Figure 3. Together, they indicate which role (if any) is able to receive, and are used in Definition 3.4 to ensure *receive urgency*, by requiring that there are no processes that are both waiting to receive and have a non-empty queue. Informally, $\mathtt{Wait}\,(P)$ returns the set of channels on which P is waiting to receive a message, and $\mathtt{NEQ}\,(P)$ returns the set of endpoints with a non-empty inbound queue.

Example 3.5 (Time Passing Function)**.** In Definition 3.4 we define $\Phi_t\,(P)$ as a *partial function* in order to: (a) ensure that certain processes are always non-blocking and can never be arbitrarily delayed (e.g., send processes) and (b) make it easier to differentiate between a process that can not be delayed, and a process that can be delayed for 0. For example, consider the process: $P = \mathtt{delay}(5).P' \mid p \triangleleft \mathit{data}.P''$. Under the current definition of $\Phi_t\,(P)$, it is clear that the *data* must be sent prior to any delay occurring, and in this instance, the definition of $\Phi_t\,(P)$ in Definition 3.4 enforces that certain processes, such as send processes, should never be delayed under any circumstances. (The only exception being a formally defined delay process prefixed to a send process.) However, if $\Phi_t\,(P)$ were to be defined for all cases of P including send processes, the reduction of our example process P becomes unclear and may be indefinitely reduced via rule [Delay] for a delay of 0. Furthermore, by the current definition of structural congruence in Definition 3.3, sending *data* may be skipped altogether. Therefore, we tactically define $\Phi_t\,(P)$ to be only a partial function in order to distinctly group delayable and non-delayable processes. Consider the process below:

$$P = (\nu pq)\; \left(p^{\diamond n} \triangleright \left\{l_i(v_i) : P_i\right\}_{i\in I} \;\mathtt{after}\; Q \mid qp : \varnothing \mid Q'\right)$$

For a time-consuming action of t to occur on P it is required that $\Phi_t\,(P)$ is defined for all parallel components in P. Suppose $t = n{+}1$ so that we can observe the expiring of the timeout. The evaluation of $\Phi_t\left(p^{\diamond n} \triangleright \left\{l_i(v_i) : P_i\right\}_{i\in I} \;\mathtt{after}\; Q\right)$ results in the evaluation of $\Phi_1\,(Q)$. If Q were e.g., a sending process, then $\Phi_{t'}\,(Q)$ is not defined for any $t' > 0$. Otherwise, if $\Phi_1\,(Q)$ and $\Phi_t\,(Q')$ are defined, then t may pass and $\Phi_t\,(P) = (\nu pq)\; \left(\Phi_1\,(Q) \mid qp : \varnothing \mid \Phi_t\,(Q')\right)$. △

4. Expressiveness

In this section we reflect on the expressiveness of our mixed-choice extension, particularly in regard to [BMVY19], using examples to illustrate differences. Given the increase in expressiveness, we discuss how type-checking becomes more interesting with the inclusion of `receive-after` and give an intuition on the relationship between the TOAST and processes.

4.1. Missing deadlines.

The process corresponding to $?a\,(\texttt{true}, \varnothing)\,.S$ is merely $p^{\infty} \triangleright a : P$, which waits to receive a forever. By way of contrast, $?b\,(x < 3, \varnothing)\,.S'$, cannot receive when $x \geqslant 3$, requiring the process to take the form: $p^{<3} \triangleright b : P'$. More generally, if we consider: $\{?b\ (x < 3,\ \varnothing).S',\ ?c\ (3 < x < 5,\ \varnothing).S''\}$ when $(x \geqslant 3)$ then, amending the previous process, $p^{<3} \triangleright b : P'$ `after` $p^{<2} \triangleright c : P''$.

4.2. Ping-pong protocol.

The example in this section illustrates the usefulness of time-sensitive conditional statements. The *ping-pong* protocol consists of two participants exchanging messages between themselves on receipt of a message from the other [LNY22]. One interpretation of the protocol is the following:

$$\mu\alpha.\left\{\begin{array}{l} !\mathit{ping}(x \leqslant 3,\ \{x\}).\left\{\begin{array}{l} ?\mathit{ping}(x \leqslant 3,\ \{x\}).\alpha \\ ?\mathit{pong}(x > 3,\ \{x\}).\alpha \end{array}\right\} \\ \\ !\mathit{pong}(x > 3,\ \{x\}).\left\{\begin{array}{l} ?\mathit{ping}(x \leqslant 3,\ \{x\}).\alpha \\ ?\mathit{pong}(x > 3,\ \{x\}).\alpha \end{array}\right\} \end{array}\right\}$$

where each participant exchanges the role of sender, either sending *ping* early, or *pong* late. Without time-sensitive conditional statements, the setting in [BMVY19] only allows implementations where the choice between the '*ping*' and the '*pong*' branch are hard-coded. In presence of non-deterministic delays (e.g., $\texttt{delay}\,(t < 6)$), the hard-coded choice can only be for the latest branch to 'expire', and the highlighted fragment of the *ping-pong* protocol above could be naively implemented as follows (omitting Q for simplicity):

$$\texttt{def}\ X(p) = (\texttt{delay}\,(t < 6)\,.p \triangleleft \mathit{pong}.Q\,)\ \texttt{in}\ X\langle r\rangle$$

The choice of sending *ping* is *always* discarded as it may be unsatisfied in *some* executions. The calculus in this paper, thanks to the time-awareness deriving from a process timer $ⓨ$, allows us to *potentially* honour each branch, as follows:

$$\texttt{def}\ X(p) = (\texttt{set}\,ⓨ\,.\texttt{delay}\,(t < 6)\,.\texttt{if}\,\left(ⓨ \leqslant 3\right)\texttt{then}\ p \triangleleft \mathit{ping}.Q\ \texttt{else}\ p \triangleleft \mathit{pong}.Q'\,)\ \texttt{in}\ X\langle r\rangle$$

4.3. Mixed-choice Ping-pong protocol.

An alternative interpretation of the *ping-pong* protocol can result in an implementation with mixed-choice, as shown below:

$$\mu\alpha.\left\{\begin{array}{l} ?\mathit{ping}(x \leqslant 3,\ \{x\}).\left\{\begin{array}{l} !\mathit{pong}(x \leqslant 3,\ \{x\}).\alpha \\ ?\mathit{timeout}(x > 3,\ \varnothing).\mathit{end} \end{array}\right\} \\ \\ !\mathit{pong}(x > 3,\ \{x\}).\left\{\begin{array}{l} ?\mathit{ping}(x \leqslant 3,\ \{x\}).\alpha \\ !\mathit{timeout}(x > 3,\ \varnothing).\mathit{end} \end{array}\right\} \end{array}\right\}$$

where '*pings*' are responded by '*pongs*' and vice versa. Notice that if a timely *ping* is not received, a *pong* is sent instead, which if not responded to by a *ping*, triggers a timeout.

Similarly, once a *ping* has been received, a *pong* must be sent on time to avoid a timeout. Such a convoluted protocol can be fully implemented: $\texttt{def}\ X(p) = P\ \texttt{in}\ X\langle r\rangle$ where P:

$$P = p^{\leqslant 3} \triangleright \mathit{ping} : \texttt{set}\,ⓧ\,.\texttt{delay}\,(t \leqslant 4)\,.\texttt{if}\,(ⓧ \leqslant 3)\,\texttt{then}\ p \triangleleft \mathit{pong}.X\langle p\rangle\ \texttt{else}\ p^{\infty} \triangleright \mathit{timeout} : \mathtt{0}$$
$$\texttt{after}\ p \triangleleft \mathit{pong}.p^{\leqslant 3} \triangleright \mathit{ping} : X\langle p\rangle\ \texttt{after}\ p \triangleleft \mathit{timeout}.\mathtt{0}$$

4.4. **Message throttling.** A real-world application of the previous example is *message throttling*. The rationale behind message throttling is to cull unresponsive processes, which do not keep up with the message processing tempo set by the system. This avoids a server from becoming overwhelmed by a flood of incoming messages. In such a protocol, upon receiving a message, a participant is permitted a grace period to respond before receiving another. The grace period is specified as a number of unacknowledged messages, rather than a period of time. Below we present a fully parametric implementation of this behaviour, where m is the maximum number of unacknowledged messages before a timeout is issued.

$$\begin{aligned} S_0 &= \mu\alpha^0.!\mathit{msg}(x \geqslant 3, \{x\}).S_1 \\ S_i &= \mu\alpha^i.\left\{?\mathit{ack}(x < 3, \{x\}).\alpha^{i-1},\ !\mathit{msg}(x \geqslant 3, \{x\}).S_{i+1}\right\} \\ S_m &= \left\{?\mathit{ack}(x < 3, \{x\}).\alpha^{m-1},\ !\mathit{tout}(x \geqslant 3,\ \varnothing).\texttt{end}\right\} \end{aligned} \tag{4.1}$$

where $0 < i < m$. Below is the corresponding processes (again, $0 < i < m$):

$$\begin{aligned} P_0 &= \texttt{def}\ X_0(p) = \texttt{delay}\,(3)\,.p \triangleleft \mathit{msg}.P_1\ \texttt{in}\ X_0\langle p\rangle \\ P_i &= \texttt{def}\ X_i(p) = p^{<3} \triangleright \mathit{ack} : X_{i-1}\langle p\rangle\ \texttt{after}\ p \triangleleft \mathit{msg}.P_{i+1}\ \texttt{in}\ X_i\langle p\rangle \\ P_m &= p^{<3} \triangleright \mathit{ack} : X_{m-1}\langle p\rangle\ \texttt{after}\ p \triangleleft \mathit{tout}.\mathtt{0} \end{aligned} \tag{4.2}$$

We now concretize this example with the $m = 2$ instance for type S and process P:

$$S = \mu\alpha^0.!\mathit{msg}(x \geqslant 3, \{x\}).\mu\alpha^1.\left\{\begin{array}{l} ?\mathit{ack}(x < 3, \{x\}).\alpha^0, \\ !\mathit{msg}(x \geqslant 3, \{x\}).\left\{\begin{array}{l} ?\mathit{ack}(x < 3, \{x\}).\alpha^1, \\ !\mathit{tout}(x \geqslant 3, \varnothing).\texttt{end} \end{array}\right\} \end{array}\right\}$$

$$\begin{aligned} P = {} & \texttt{def}\ X_0(p) = \texttt{delay}\,(3)\,.p \triangleleft \mathit{msg}. \\ & \quad \texttt{def}\ X_1(p) = p^{<3} \triangleright \mathit{ack} : X_0\langle p\rangle \\ & \qquad \texttt{after}\ p \triangleleft \mathit{msg}.p^{<3} \triangleright \mathit{ack} : X_1\langle p\rangle\ \texttt{after}\ p \triangleleft \mathit{tout}.\mathtt{0} \\ & \quad \texttt{in}\ X_1\langle p\rangle \\ & \texttt{in}\ X_0\langle p\rangle \end{aligned}$$

The system shown in Figure 4 also illustrates the instance of $m = 2$. △

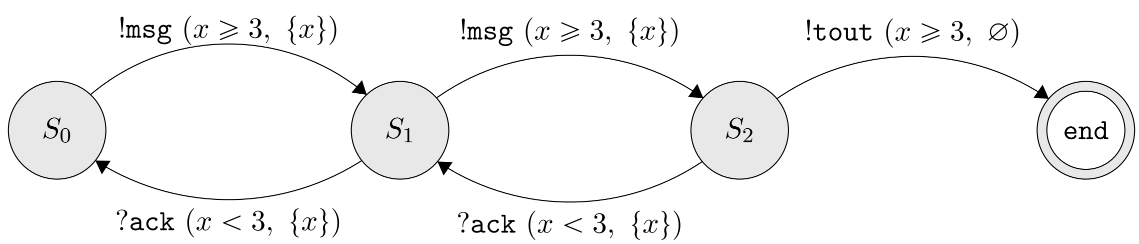


Figure 4. Message throttling protocol for $m = 2$.

5. Type Checking TOAST Processes

Processes are validated against types using judgements of the form: $\Gamma; \theta \vdash P \blacktriangleright \Delta$.

$$\begin{aligned} \Gamma &::= \varnothing \mid \Gamma,\ v : T \mid \Gamma,\ X : (\vec{T}; \boldsymbol{\theta}; \boldsymbol{\Delta}) && \text{(where } T \text{ is a } \textit{base-type}\text{)} \\ \theta &::= \varnothing \mid \theta,\ ⓧ : n && \text{(recall Definition 3.2, } n \in \mathbb{R}_{\geqslant 0}\text{)} \\ \Delta &::= \varnothing \mid \Delta,\ p : (\nu,\ S) \mid \Delta,\ qp : \mathtt{M} \end{aligned} \tag{5.1}$$

Variable Environments Γ map values v to data types T and process variables X to triples $X : (\vec{T}; \boldsymbol{\theta}; \boldsymbol{\Delta})$, where $\vec{T}$ is a vector of messages (consisting of labels and data types), and $\boldsymbol{\theta}$ and $\boldsymbol{\Delta}$ are sets of timer environments and session environments, respectively and both are possibly infinite. With respect to usual formulations of session types, we use the mapping θ of timers (denoted ⓧ, ⓨ and ⓩ) as a timer environment (given in Definition 3.2) to correctly type check time-sensitive conditional statements. As usual, *Session Environments* Δ map roles p and q to configurations and session endpoints qp and pq to queues $\mathtt{M}$. Similarly to the syntax of processes in Section 3, we may use a and b as ad-hoc roles, along with endpoints ab and ba, during discussions of more than one session.

5.1. Auxiliary definitions & notation.

We write $\mathrm{dom}(\Delta)$ for the domain of Δ, and Δ_1, Δ_2 for $\Delta_1 \cup \Delta_2$ when $\mathrm{dom}(\Delta_1) \cap \mathrm{dom}(\Delta_2) = \varnothing$. We define $\Delta(p) = (\nu,\ S)$ iff $p \in \Delta$ and $\exists \Delta'$ such that $\Delta = \Delta', p : (\nu,\ S)$, and define $\mathtt{val}(\nu, S) = \nu$ and $\mathtt{type}(\nu, S) = S$. Given a time offset $t \in \mathbb{R}_{\geqslant 0}$, we define:

- $\Delta + t = \{(\mathtt{val}(\Delta(p)) + t,\ \mathtt{type}(\Delta(p))) \mid p \in \mathrm{dom}(\Delta)\}$
- $\theta + t = \{ⓧ \mapsto \theta(ⓧ) + t \mid ⓧ \in \mathbb{T}\}$ (recall Definition 3.2)

We say that Δ is t-reading if for the duration t, there exist roles within Δ that are able to perform a receiving action. We say that Δ is *not* t-reading if there are no roles within Δ able to receive for the duration of t. Formally:

Definition 5.1 (t-reading Δ)**.** Δ is t-reading if, $\exists t' < t, p \in \mathrm{dom}(\Delta)$ such that $\Delta(p) = (\nu,\ S) \implies (\nu + t',\ S) \xrightarrow{?m}$. We write Δ is $\diamond t$-reading if, $\exists t' \diamond t, p \in \mathrm{dom}(\Delta)$ such that $\Delta(p) = (\nu,\ S) \implies (\nu + t',\ S) \xrightarrow{?m}$.

Definition 5.1 is useful to check against violations of *receive urgency*.[2] Finally, we extend the definition of well-formed configurations to session environments in the obvious way:

Definition 5.2 (Well-formed Δ)**.** Δ is *well-formed* if, for all $p \in \mathrm{dom}(\Delta)$ such that $\Delta = \Delta', p : (\nu,\ S) \implies (\nu,\ S)$ is *well-formed* by Definition 2.6.

5.2. Type Checking Rules.

The *type checking* rules are given in Figures 5 and 6. Figure 5 shows standard typing rules in Eq (5.2) and time-sensitive typing rules in Eq (5.3). Figure 6 shows rules pertaining to communication, with: sending processes in Eq (5.4), branching in Eq (5.5), receptions in Eq (5.6), and queues in Eq (5.7). We will now discuss each rule.

[2]See Example 5.4.

$$\Gamma;\theta\vdash \mathtt{0} \blacktriangleright \varnothing \quad [\texttt{End}] \qquad \Gamma;\theta\vdash pq:\varnothing \blacktriangleright pq:\varnothing \quad [\texttt{Empty}]$$

$$\frac{\Gamma;\theta\vdash P\blacktriangleright\Delta}{\Gamma;\theta\vdash P\blacktriangleright\Delta,\ p:(\nu,\ \mathtt{end})}\ [\texttt{Weak}] \qquad \frac{\Gamma;\theta_1\vdash P\blacktriangleright\Delta_1 \quad \Gamma;\theta_2\vdash Q\blacktriangleright\Delta_2}{\Gamma;\theta_1,\theta_2\vdash P\mid Q\blacktriangleright\Delta_1,\Delta_2}\ [\texttt{Par}]$$

$$\frac{\begin{array}{c}\Gamma;\theta\vdash P\blacktriangleright\Delta,\ p:(\nu_1,\ S_1),\ qp:\mathtt{M}_1,\ q:(\nu_2,\ S_2),\ pq:\mathtt{M}_2\\ (\nu_1,\ S_1,\ \mathtt{M}_1)\perp(\nu_2,\ S_2,\ \mathtt{M}_2) \qquad \forall i\in\{1,2\}.\,S_i \text{ well-formed against } \nu_i\end{array}}{\Gamma;\theta\vdash(\nu pq)\ P\blacktriangleright\Delta}\ [\texttt{Res}] \tag{5.2}$$

$$\frac{\begin{array}{c}\forall(\vec{\nu},\ \vec{S})\in\boldsymbol{\Delta},\theta'\in\boldsymbol{\theta}:\Gamma,\vec{v}:\vec{T},X:(\vec{T};\boldsymbol{\theta};\boldsymbol{\Delta});\theta'\vdash P\blacktriangleright\vec{r}:(\vec{\nu},\ \vec{S})\\ \Gamma,X:(\vec{T};\boldsymbol{\theta};\boldsymbol{\Delta});\theta\vdash Q\blacktriangleright\Delta\end{array}}{\Gamma;\theta\vdash\mathtt{def}\ X(\vec{v};\vec{r})=P\ \mathtt{in}\ Q\blacktriangleright\Delta}\ [\texttt{Rec}]$$

$$\frac{(\vec{\nu},\vec{S})\in\boldsymbol{\Delta}\quad \theta\in\boldsymbol{\theta}\quad \forall i:\Gamma\vdash\vec{v}_i:\vec{T}_i}{\Gamma,X:(\vec{T};\boldsymbol{\theta};\boldsymbol{\Delta});\theta\vdash X\langle\vec{v};\vec{r}\rangle\blacktriangleright\vec{r}:(\vec{\nu},\vec{S})}\ [\texttt{Var}]$$

$$\frac{\theta\models c\quad \Gamma;\theta\vdash P\blacktriangleright\Delta}{\Gamma;\theta\vdash\mathtt{if}\ c\ \mathtt{then}\ P\ \mathtt{else}\ Q\blacktriangleright\Delta}\ [\texttt{IfTrue}] \quad \frac{\Gamma;\theta,ⓧ:0\vdash P\blacktriangleright\Delta}{\Gamma;\theta,ⓧ:n\vdash\mathtt{set}\,ⓧ.P\blacktriangleright\Delta}\ [\texttt{Timer}]$$

$$\frac{\theta\not\models c\quad \Gamma;\theta\vdash Q\blacktriangleright\Delta}{\Gamma;\theta\vdash\mathtt{if}\ c\ \mathtt{then}\ P\ \mathtt{else}\ Q\blacktriangleright\Delta}\ [\texttt{IfFalse}] \quad \frac{\forall t\in d:\Gamma;\theta\vdash\mathtt{delay}\,(t)\,.P\blacktriangleright\Delta}{\Gamma;\theta\vdash\mathtt{delay}\,(d)\,.P\blacktriangleright\Delta}\ [\texttt{Del-}d] \tag{5.3}$$

$$\frac{\Gamma;\theta+t\vdash P\blacktriangleright\Delta+t\quad \Delta\text{ not }t\text{-reading}}{\Gamma;\theta\vdash\mathtt{delay}\,(t)\,.P\blacktriangleright\Delta}\ [\texttt{Del-}t]$$

Figure 5. Standard & time-sensitive type checking rules

5.2.1. *Standard Rules, Eq (5.2).* Terminated processes and empty channel buffers are typed using axioms [End] and [Empty], respectively. Rule [Weak] handles the type-checking of a session where an individual process has successfully reached termination, and where the rest of the session may be ongoing. Such an instance is to be expected in the asynchronous setting, where one party may perform a sequence of non-blocking sending actions and finish, leaving the other party to process and receive. Rule [Par] divides the session environment into two disjoint environments, Δ_1 and Δ_2, so that each session is carried by only one of P and Q. The timer environment is also split into two disjoint environments, θ_1 and θ_2, as processes would otherwise be capable of indirectly communicating or influencing the behaviour of other processes outside the interactions prescribed by the types. Rule [Res] ensures scoped processes are well-typed against binary sessions composed of *compatible* configurations (by Definition 2.10). Notably, the only change from [BMVY19] in this rule is the well-formedness requirement of S_i against ν_i (see Definition 2.6). In [BMVY19], well-formedness was naturally entailed by the premises of their rules type-checking sending/receiving processes. In order to tackle the complexity of checking mixed-choice (branching and selections), we have used multiple simpler rules that make it difficult to encapsulate such a requirement. Therefore, without loss of expressiveness, we have made the requirement explicit here.

Rules [Rec] and [Var] are essentially unchanged from [BMVY19], only now extended with process timers. Rule [Rec] handles recursion definitions by evaluating Q and collecting within the triple $(\vec{T}; \boldsymbol{\theta}; \boldsymbol{\Delta})$ all messages, timer and session environments present at any corresponding call. This triple is mapped to a process variable assigned to P, and added to Γ as the evaluation continues with Q. Rule [Var] determines if a call is well-typed by checking that the provided messages and channels correspond to those in the process variable.

5.2.2. *Time-sensitive Rules, Eq (5.3).* Rule [Timer] handles setting or resetting of timer ⓧ. We require ⓧ to be already in θ in the set case (also handled by this rule) to avoid race conditions on timers with concurrent sessions, which would compromise subject reduction. Inspection of the syntactic structure of a process is enough to know beforehand which timers will be used. Rule [Del-d] checks that P is well-typed against all solutions of d. Rule [Del-t] reflects time passing on θ and the session Δ. The right-hand side premise of 'Δ not t-reading' enforces *receive urgency* in processes by requiring that no participants in Δ are able to receive a message from their queue for the duration of t. (Inverse to Definition 5.1.) Rules [IfTrue] and [IfFalse] are for typing time-sensitive conditional statements. If the timers in θ satisfy the condition c, then by rule [IfTrue] the evaluation continues with P. Else, by the symmetric rule [IfFalse] the evaluation continues with Q.

5.2.3. *Sending Rules, Eq (5.4).* Rules [VSend] and [DSend] are for typing processes that send messages against a choice type, by finding a matching pair of labels in the process and type. Rule [VSend] applies for sending base-types, requiring that v is well-typed against T, and the continuation process P is well-typed against the continuation S, with any clock resets applied. Rule [DSend] is similar, but for delegating an ongoing session with role b to role p, and then removes role b from the session environment.

5.2.4. *Branching Rules, Eq (5.5).* Rule [Branch] checks a branching process against a choice type and is comprised of a single premise. The first condition of the premise requires that the type or the processes has more than one element, to ensure that type-checking is deterministic (if both are singleton sets, then either rule [DRecv] or [VRecv] in Eq (5.6) should be used). The second condition of the premise requires that each currently enabled receiving action in the type has a corresponding branch in the process (i.e., since all labels are unique, they have matching labels) which when paired together, are *well-typed*.

Rule [Timeout] checks that a receive-after process is well-typed against a choice type; decomposing the checking into two parts: the branch process, and the process Q; requiring that both are *well-typed*. The first line in the premise forms a judgement to be checked by rule [Branch], by discarding the 'timeout' part of the original process. The second line of the premise checks the process is still *well-typed* in the case where the timeout 'triggers' (i.e., time n passes). Put simply, each enabled action prescribed by the type must have a corresponding branch in the process that, when paired together, form a well-typed judgement.

5.2.5. *Reception Rules, Eq (5.6).* Rule [VRecv] is applied when evaluating individual base-type receptions. The first line of the premise checks that waiting the duration of n adheres to the timings specified by the type's constraints δ; also, this line requires that there are no other roles in Δ that have enabled receiving actions. The second line of the premise evaluates the proceeding process P against the continuation type S, given any possible delay that could occur $(t \diamond n)$. Rule [DRecv] is similar, but for evaluating individual *delegation* receptions, which are then added to the session environment.

$$\frac{\begin{array}{c}\exists i \in I : (l = l_i) \wedge (\nu \models \delta_i) \wedge (T_i \textit{ base-type}) \wedge (\Gamma \vdash v : T_i) \wedge \\ (\square_i = !) \wedge \Gamma; \theta \vdash P \blacktriangleright \Delta, p : (\nu \left[\lambda_i \mapsto 0\right], S_i)\end{array}}{\Gamma; \theta \vdash p \triangleleft l\,(v).P \blacktriangleright \Delta,\ p : (\nu,\ \left\{\square_i\, l_i \left\langle T_i \right\rangle (\delta_i, \lambda_i).S_i\right\}_{i \in I})}\ [\texttt{VSend}]$$

$$\frac{\begin{array}{c}\exists i \in I : (l = l_i) \wedge (\nu \models \delta_i) \wedge (T_i = \left(\delta',\ S'\right)) \wedge (\nu' \models \delta') \wedge \\ (\square_i = !) \wedge \Gamma; \theta \vdash P \blacktriangleright \Delta, p : (\nu \left[\lambda_i \mapsto 0\right], S_i)\end{array}}{\Gamma; \theta \vdash p \triangleleft l\,(b).P \blacktriangleright \Delta,\ p : (\nu,\ \left\{\square_i\, l_i \left\langle T_i \right\rangle (\delta_i, \lambda_i).S_i\right\}_{i \in I}),\ b : (\nu',\ S')}\ [\texttt{DSend}] \tag{5.4}$$

$$\frac{\begin{array}{c}\neg(|J| = |I| = 1) \quad \forall j \in J : (\nu \models \delta_j) \implies (\square_j = ?) \ \wedge\ \exists i \in I : (l_i = l_j) \ \wedge \\ \Gamma; \theta \vdash p^e \triangleright l_i(v_i) : P_i \blacktriangleright \Delta,\ p : (\nu,\ \square_j\, l_j \left\langle T_j \right\rangle (\delta_j, \lambda_j).S_j)\end{array}}{\Gamma; \theta \vdash p^e \triangleright \left\{l_i(v_i) : P_i\right\}_{i \in I} \blacktriangleright \Delta,\ p : (\nu,\ \left\{\square_j\, l_j \left\langle T_j \right\rangle (\delta_j, \lambda_j).S_j\right\}_{j \in J})}\ [\texttt{Branch}]$$

$$\frac{\begin{array}{c}\Gamma; \theta \vdash p^{\diamond n} \triangleright \left\{l_i(v_i) : P_i\right\}_{i \in I} \blacktriangleright \Delta,\ p : (\nu,\ \{\mathtt{C}_j\}_{j \in J}) \\ \Gamma; \theta + n \vdash Q \blacktriangleright \Delta + n,\ p : (\nu + n,\ \{\mathtt{C}_j\}_{j \in J})\end{array}}{\Gamma; \theta \vdash p^{\diamond n} \triangleright \left\{l_i(v_i) : P_i\right\}_{i \in I}\ \texttt{after}\ Q \blacktriangleright \Delta,\ p : (\nu,\ \{\mathtt{C}_j\}_{j \in J})}\ [\texttt{Timeout}] \tag{5.5}$$

$$\frac{\begin{array}{c}T \textit{ base-type} \quad \Delta \text{ not } e\text{-reading} \quad \forall t : (\nu + t \models \delta) \Longleftrightarrow (t \in e) \\ \forall t \in e : \Gamma, v : T; \theta + t \vdash P \blacktriangleright \Delta + t,\ p : (\nu + t \left[\lambda \mapsto 0\right],\ S)\end{array}}{\Gamma; \theta \vdash p^e \triangleright l(v) : P \blacktriangleright \Delta,\ p : (\nu,\ ?l\langle T \rangle\, (\delta, \lambda)\, .S)}\ [\texttt{VRecv}]$$

$$\frac{\begin{array}{c}T = \left(\delta',\ S'\right) \quad \nu' \models \delta' \quad \Delta \text{ not } e\text{-reading} \quad \forall t : (\nu + t \models \delta) \Longleftrightarrow (t \in e) \\ \forall t \in e : \Gamma; \theta + t \vdash P \blacktriangleright \Delta + t,\ p : (\nu + t \left[\lambda \mapsto 0\right],\ S),\ q : (\nu',\ S')\end{array}}{\Gamma; \theta \vdash p^e \triangleright l(q) : P \blacktriangleright \Delta,\ p : (\nu,\ ?l\langle T \rangle\, (\delta, \lambda)\, .S)}\ [\texttt{DRecv}] \tag{5.6}$$

$$\frac{T \textit{ base-type} \quad \Gamma \vdash v : T \quad \Gamma; \theta \vdash qp : h \blacktriangleright \Delta,\ qp : \mathtt{M}}{\Gamma; \theta \vdash qp : lv \cdot h \blacktriangleright \Delta,\ qp : l\left\langle T \right\rangle; \mathtt{M}}\ [\texttt{VQue}]$$

$$\frac{T = \left(\delta,\ S\right) \quad \nu \models \delta \quad \Gamma; \theta \vdash qp : h \blacktriangleright \Delta,\ qp : \mathtt{M}}{\Gamma; \theta \vdash qp : lq \cdot h \blacktriangleright \Delta,\ qp : l\left\langle T \right\rangle; \mathtt{M},\ q : (\nu,\ S)}\ [\texttt{DQue}] \tag{5.7}$$

Figure 6. Rules for type-checking communicating processes against TOAST

5.2.6. *Queue Rules, Eq (5.7).* Non-empty buffers and queues are typed by [`VQue`] or rule [`DQue`]. The former is for *base-type* payloads and the latter for delegated sessions.

5.3. **Type Checking Examples.** The handling of scope restriction, parallel composition, and recursion is similar to existing formulations. We give an example of the less obvious rules to handle timers and mixed-choice. Precisely, we show in Example 5.3 how to type check an implementation of a simplified (non-recursive version) the mixed choice *ping-pong* protocol featured in Section 4.3. Later, in Example 5.4 we discuss type-checking multiple binary sessions, and illustrate the issues that can arise from *incompatible interleavings*.

Example 5.3 (Type Checking Ping-pong)**.** Consider the protocol S below:

$$S = \left\{ \begin{array}{l} ?ping(x \leqslant 3,\ \{x\}).\left\{\begin{array}{l} !pong(x \leqslant 3,\ \{x\}).\mathtt{end}, \\ ?timeout(x > 3,\ \varnothing).\mathtt{end} \end{array}\right\}, \\ \\ !pong(x > 3,\ \{x\}).\left\{\begin{array}{l} ?ping(x \leqslant 3,\ \{x\}).\mathtt{end}, \\ !timeout(x > 3,\ \varnothing).\mathtt{end} \end{array}\right\} \end{array} \right\} \tag{5.8}$$

and its candidate implementations P:

$$\begin{array}{l} P = p^{\leqslant 3} \triangleright ping : \mathtt{set}\,ⓩ.\mathtt{delay}\,(t \leqslant 4)\,.\mathtt{if}\,(ⓩ \leqslant 3)\,\mathtt{then}\ p \triangleleft pong.\mathtt{0} \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathtt{else}\ p^{\infty} \triangleright timeout : \mathtt{0} \\ \quad \mathtt{after\ delay}\,(0.1)\,.p \triangleleft pong.p^{\leqslant 3} \triangleright ping : \mathtt{0} \\ \qquad\qquad\qquad\qquad\qquad\qquad \mathtt{after\ delay}\,(0.1)\,.p \triangleleft timeout.\mathtt{0} \end{array} \tag{5.9}$$

We discuss how one can use our type system to check the following: $\varnothing; ⓩ : 0 \vdash P \blacktriangleright\ p : (\nu_0, S)$. From here, one can apply rule [Timeout], which decomposes the judgement into two: one for branching process in Eq (5.10), and one for the timeout continuation Q in Eq (5.11).

$$\varnothing; ⓩ : 0 \vdash p^{\leqslant 3} \triangleright ping : \mathtt{set}\,ⓩ.\mathtt{delay}\,(t \leqslant 4)\,.\mathtt{if}\,(ⓩ \leqslant 3) \ldots \blacktriangleright\ p : (\nu_0, S) \tag{5.10}$$

$$\varnothing; ⓩ : 3 \vdash \mathtt{delay}\,(0.1)\,.p \triangleleft pong.R \blacktriangleright\ p : (\nu_3[x \mapsto 0], S) \tag{5.11}$$

where $R = p^{\leqslant 3} \triangleright ping : \mathtt{0}\ \mathtt{after\ delay}\,(0.1)\,.p \triangleleft timeout.\mathtt{0}$. In Eq (5.11), note that the values of process timer ⓩ and virtual clock x have incremented by the duration of the timeout, i.e., 3. Following up Eq (5.10), one can apply rule [Branch], which checks that all viable actions in S are receiving actions, and that each is correctly implemented in the process being typed. Clearly this holds, since the only viable receive action in S is $?ping\,(x \leqslant 3, \{x\})$, which matches the only branch $(ping : \ldots)$ in the process. We next apply rule [VRecv]:

$$\varnothing; ⓩ : t' \vdash \mathtt{set}\,ⓩ.\mathtt{delay}\,(t \leqslant 4)\,.\mathtt{If}\,(ⓩ \leqslant 3) \ldots \blacktriangleright\ p : (\nu_{t'}[x \mapsto 0],\ S') \qquad (t' \leqslant 3) \tag{5.12}$$

with $n \leqslant 3$ and $S' = \{!pong(x \leqslant 3,\ \{x\}).end, ?timeout(x > 3,\ \varnothing).end\}$. Since clock x has been reset to 0, and is x the only clock used by our type S, the set of clock valuations is back to ν_0. Application of [Set] then resets timer ⓩ to 0. Next, [Del-d] decomposes Eq (5.12) into a set of judgements, one for each solution t'' of $d = (t \leqslant 4)$; which by [Del-t] yields:

$$\varnothing; ⓩ : t' + t'' \vdash \mathtt{if}\ (ⓩ \leqslant 3)\mathtt{then}\ p \triangleleft pong.\mathtt{0}\ \mathtt{else}\ p^{\infty} \triangleright timeout : \mathtt{0} \blacktriangleright\ p : (\nu_{t'+t''}[x \mapsto t''],\ S')$$

Notably, if t'' is smaller than or equal to 3 ($t'' \leqslant 3$) we can apply rule [IfTrue]; e.g., $t'' = 3$:

$$\varnothing; ⓩ : 3 \vdash p \triangleleft pong.\mathtt{0} \blacktriangleright\ p : (\nu_{t'+3}[x \mapsto 3],\ S') \tag{5.13}$$

and if t'' is greater than 3 (e.g., $t'' = 4$) we can apply rule [IfFalse]:

$$\varnothing; ⓩ : 4 \vdash p^{\infty} \triangleright timeout : \mathtt{0} \blacktriangleright\ p : (\nu_{t'+4}[x \mapsto 4],\ S') \tag{5.14}$$

To proceed the evaluation of Eq (5.13), observe that the process is selecting a label *pong*. By rule [VSend], we only need to check that S' has a sending branch with label *pong* viable when $(x = 3)$, which is indeed the case (i.e., $!pong(x \leqslant 3,\ \{x\}).end$). Similarly, looking at Eq (5.14), we need to use the rule branching to isolate all (one in this case) the receive actions viable at the current time. The remaining of both derivations is straightforward.

The case for the '`after`' process in Eq (5.11) is similar, except it starts with a shifted time in the timers and virtual clocks. Note, we insert '`delay` $(0.1) \ldots$.' immediately upon entering the '`after`' to compensate for the inclusive upper-bound of $(\leqslant 3)$.

We show the full derivation tree in Appendix C. △

Example 5.4 (Incompatible Interleavings)**.** Consider the following session environments:

$$\begin{array}{rcl}\Delta_{ab} & = & a:(\nu_1,\, S_1),\;\; b:(\nu_2,\, S_2),\;\; ba:\mathtt{M}_1,\;\; ab:\mathtt{M}_2 \\ \Delta_{pq} & = & p:(\nu_3,\, S_3),\;\; q:(\nu_4,\, S_4),\;\; qp:\mathtt{M}_3,\;\; pq:\mathtt{M}_4\end{array}$$

where both Δ_{ab} and Δ_{pq} are *well-formed* and composed together, under the session environment $\Delta = (\Delta_{ab}, \Delta_{pq})$. Both p and b are roles performed by the same participant in two distinct sessions. All clocks within Δ are 0. Below are the definitions of S_2 and S_3:

$$S_2 = \begin{Bmatrix} ?request\langle\mathtt{String}\rangle\,(x < 5, \{x\})\,.S_2' \\ !timeout\,(x \geqslant 5, \{x\})\,.S_2'' \end{Bmatrix} \qquad S_3 = ?message\,(y > 3, \varnothing)\,.S_3'$$

Consider the following implementation P, where role b first waits 5 time units for a *request* before receiving *message* as role p.

$$\begin{array}{c} P = ab^{<5} \triangleright request(\mathtt{req\_str}) : qp^{\infty} \triangleright message : P' \\ \mathtt{after}\; ba \triangleleft timeout.qp^{\infty} \triangleright message : P'' \end{array}$$

Assume *message* arrives in the queue qp at time 3.5 and *request* arrives at time 4. Process P will receive *request* at time 4 while *message* is still pending in the queue, breaking *receive urgency*. In fact, P is not 4-reading since there exists a role p in Δ that may be able to receive before 4 time units. Definition 5.1 is utilised by the *type checking* rules in Figures 5 and 6 to ensure such implementations are *not well-typed*. In this case, one may observe that there exists no well-typed implementation of S_2 and S_3 that satisfies receive urgency, hence the roles are fundamentally incompatible and unable to be interleaved. △

6. Subject Reduction

In this section we establish subject reduction for our typing system, namely we establish a relation between well-typed processes and their types that is preserved by reduction.

As usual, subject reduction is given for closed systems (Corollary 6.5) with Γ and Δ empty. The proof relies on two lemmas that establish subject reduction for open systems, for time-consuming steps (Theorem 6.3) and instantaneous steps (Theorem 6.4). We give an overview in this section and relegate the full proofs to Appendix B. Crucially, by rule [Res] in Figure 5 Eq (5.2), in any typing judgement of an open system the sessions in Δ are both *well-formed* and comprised of *compatible* configurations, enabling us to utilise progress of TOAST Theorem 2.9. Such sessions we say are *fully-balanced*. The notion of balanced session, inspired by the one in [CDCSY17] is given in Definition 6.2.

Definition 6.1 (Balanced Δ)**.** Let Bal be the set containing all session environments Δ, such that if $\Delta \in \text{Bal}$, then Δ adheres to the following:

(1) $\Delta = \Delta', p:(\nu,\, S), qp:m;\mathtt{M} \implies (\nu,\, S) \xrightarrow{?m} (\nu',\, S')$ and $\Delta', p:(\nu',\, S'), qp:\mathtt{M} \in \text{Bal}$.
(2) $\Delta = \Delta', p:(\nu_1,\, S_1), qp:\mathtt{M}_1, q:(\nu_2,\, S_2) \implies \exists \mathtt{M}_2 : (\nu_1,\, S_1,\, \mathtt{M}_1) \perp (\nu_2,\, S_2,\, \mathtt{M}_2)$.
(3) $\Delta = \Delta', p:(\nu_1,\, S_1), qp:\mathtt{M}_1, q:(\nu_2,\, S_2), pq:\mathtt{M}_2 \implies (\nu_1,\, S_1,\, \mathtt{M}_1) \perp (\nu_2,\, S_2,\, \mathtt{M}_2)$.

Definition 6.2 (Fully-Balanced Δ)**.** A *balanced* Δ is said to be *fully-balanced* if:

(1) $\Delta = \Delta', p:(\nu_1,\, S_1) \implies \exists \Delta'', q, \nu_2, S_2, \mathtt{M}_1, \mathtt{M}_2 : \Delta' = \Delta'', qp:\mathtt{M}_1, q:(\nu_2,\, S_2), pq:\mathtt{M}_2$.
(2) $\Delta = \Delta', qp:\mathtt{M}_1 \implies \exists \Delta'', \nu_1, S_1, \nu_2, S_2, \mathtt{M}_2 : \Delta' = \Delta'', p:(\nu_1,\, S_1), q:(\nu_2,\, S_2), pq:\mathtt{M}_2$.

Theorem 6.3 (Time Step)**.** *Let* Δ *be* fully-balanced *and* well-formed*. If* $\Gamma; \theta \vdash P \blacktriangleright \Delta$ *and* $\Phi_t(P)$ *is defined, then* $\Gamma; \theta + t \vdash \Phi_t(P) \blacktriangleright \Delta + t$ *and* $\Delta + t$ *is* fully-balanced *and* well-formed*.*

By Theorem 6.3 we prove that, given a *well-typed* process P, for any value of t such that $\Phi_t(P)$ is defined by Definition 3.4, t passing evenly over the session environment preserves both *well-formedness* and *balancedness*. (Similar to the preservation of *well-formedness* and *compatibility* for system configurations in Lemmas A.28 and A.30 of Theorem 2.9.) Clearly, by inspection of the *type checking* rules in Figures 5 and 6, such processes would require several divisions of the initial session environment (and map of process timers) before each individual process could be evaluated. Therefore, we rely on the notion of *balancedness* (Definition 6.1). Naturally, a *fully-balanced* session must also be *balanced*, and this allows us to reason on specific *balanced* components of a larger *fully-balanced* session environment.

Theorem 6.4 (Action Step)**.** *Let* Δ *be* balanced *and* well-formed*. If* $\Gamma; \theta \vdash P \blacktriangleright \Delta$ *and* $(\theta, P) \rightharpoonup (\theta', P')$ *then* $\exists \Delta' : \Delta \longrightarrow^* \Delta'$ *and* $\Gamma; \theta' \vdash P' \blacktriangleright \Delta'$ *and* Δ' *is* balanced *and* well-formed*.*

By Theorem 6.4 we prove, among other things, that any sending or receiving action performed by a *well-typed* process corresponds to an action prescribed by the type of the corresponding configuration, at the correct time. The best illustration of this is perhaps Case 2 in Theorem 6.4, which shows the case of reduction via rule [`Recv`] in Figure 2 Eq (3.4), and that the reduction preserves *well-typedness*. We relegate the rules for session environment reduction to Figure 7 in Appendix B, along with the full proofs of Theorems 6.3 and 6.4.

Corollary 6.5 (Subject Reduction)**.** *If* $\varnothing; \theta \vdash P \blacktriangleright \varnothing$ *and* $(\theta, P) \rightarrow (\theta', P')$ *then* $\varnothing; \theta' \vdash P' \blacktriangleright \varnothing$*.*

Proof. Since $\Delta = \varnothing$, then by Definition 6.2, Δ is *fully-balanced*. We proceed by induction on the nature of the reduction ($\longrightarrow$):

Case 1. If $\longrightarrow = \rightsquigarrow$ then the thesis follows Theorem 6.3.

Case 2. If $\longrightarrow = \rightharpoonup$ then the thesis follows Theorem 6.4. □

As a safety property of our typing system, subject reduction guarantees that any action performed by a well-typed process is as prescribed by the types; i.e., is timely and correct.

7. Related Work

This paper is an extended version of [PBK23] featuring numerous improvements and three substantial additions to the existing work: (1) the proofs of progress for TOAST given in Appendix A; (2) the type-checking rules in Section 5; and, (3) the subject reduction result in Section 6 together with its proof in Appendix B. We have also made several changes to the syntax of processes in Section 3 and the reduction rules in Section 3.2 to better accommodate the new type-checking rules and our focus on subject reduction result. These changes are largely syntactic, and are equivalent to those in the original. For convenience, we report below in detail about the changes to the syntax of processes with respect to the short version of this paper. Most notably, we have simplified (without loss of generality) the definition of e from being a linear expression over process timers and numeric constants, to either $\diamond n$, ∞ or $\leqslant 0$, as defined in Eq (3.1). For readability, we have streamlined the syntax of branching processes with and without timeout. We have removed `error` processes, as our focus is not that of showing error-freedom.

Asynchronous Timed Session Types. Since our work builds on Asynchronous Timed Session Types [BMVY19], we begin by highlighting the differences between the two. Our work adds expressiveness to the types in [BMVY19] by allowing mixed-choice. Our processes are also extended. The processes in [BMVY19] allow a simplified variant of a timeout where a receive action is annotated with a deadline, akin to our branching primitive $p^e \triangleright \{l_i(v_i) : P_i\}_{i \in I}$. The added expressiveness of mixed-choice in TOAST allow us to introduce a process to execute upon timeout to express, for instance, retry strategies, which are not expressible in [BMVY19]. In addition to timeouts we have allowed our processes to model timers. By using timing constraints over timers as conditional statements it becomes straightforward to derive the corresponding "*dual*" of a given timeout processes.

Similarly to [BMVY19], type-checking using our typing system is decidable under the assumption that channel and recursion variables are annotated with typing information. The rules in our typing system with infinitely many premises, such as '$\forall t \in d$' in the premise of rule [`Del`-d] in Eq (5.3), can be accounted for symbolically, such as by exploiting zones and Difference Bound Matricies (DBMs) [BY03] or Satisfiability Modulo Theory (SMT).

In this version of our work we do not feature the `error` process, which is featured in [BMVY19] and [PBK23]. More similarly to the earlier work of Timed MPST [BYY14], instead of letting a process reaching an error state (`error`) when a time constraint is violated, we do not allow time to pass. (See Definition 3.4.) This choice allows us to manage the complexity added by mixed-choice and separate the concerns of subject reduction, safety and progress. Normally, safety and progress come as consequences of subject reduction. Conversely, in a timed scenario with error states, subject reduction depends on progress (i.e., if a process gets stuck some time constraint may be violated, yielding an error state reached, hence compromising subject reduction). In [BMVY19] this circular dependency was overcome by requiring a progress property called receive-liveness, which is defined on the untimed counterpart of a timed process, which can be checked by well-established techniques [DdY07, BCD$^+$08]. Unfortunately, not only do techniques such as [DdY07] not apply to mixed-choice, but they are hardly attainable in this setting: removing time from a mixed-choice is likely to introduce progress violations that would otherwise not be there in the presence of time. Therefore, removing `error` processes became critical in order to break this circular dependency and establish subject reduction. As a future work, we would like to establish a progress property on the basis of subject reduction.

Runtime Monitoring. Recent work by [PBH24] presented a work-in-progress toolchain for generating Erlang stub programs from a protocol specification notation derived from the theory of TOAST presented in this paper, with the aim of bridging the gap between protocol specifications and program implementations. The toolchain generates correct-by-construction Erlang stub programs which provide a bare-bones scaffolding structure for a user to extend with functionality (i.e., what to send and how to handle receptions).

In addition to generating Erlang stub programs, [PBH24] provide a generic runtime monitoring program for Erlang, which is provided as a precautionary measure to ensure that user of the toolchain does not unintentionally introduce unspecified behaviour when extending the Erlang stubs with functionality. The monitoring program can be setup to monitor the communication of any Erlang process against a given protocol specification, and can be configured to either: (a) *verify* the process communicates as specified, or (b) try to *enforce* the specified behaviour (e.g., by delaying the sending/receiving of messages).

Other work on timeouts. Outside of session types, a similar construct to timeout processes appears in [BLTV22], which builds upon the Temporal Process Language. Similarly to our work, [BLTV22] extends the receiving branch processes in [BMVY19] in a manner based on the `receive-after` pattern in Erlang. However [BLTV22] does not provide a typing discipline and rather focuses on characterising reliability properties in a mailbox-based communication setting. The "*periodic recursion*" in Rate-Based Session Types [ICHZ23], features a similar construct where a timeout-recursive loop runs indefinitely, at a steady and fixed rate. Instead of timing constraints on individual actions, periods are imposed on recursive loops, specifying the rate at which each iteration must adhere to, and optional deadlines which specify a fixed upper-bound.

Session types. The (untimed) Multiparty, Asynchronous and Generalised π-calculus [BD23] (MAGπ) features the option for a unique "timeout branch" in the syntax of both branching types and processes. Notably, "timeout branches" in [BD23] are non-deterministic, and may trigger regardless of another branch being viable. Additionally, the semantics go as far as to facilitate errors, such as by dropping unreliable messages from a queue before they can be received. In our work, we use timeouts to model safe mixed-choice, structured in such a way that communication mismatches cannot occur. Our work focuses on the timeliness of messages being sent and received, and to this end our timeouts ensure only one participant can send at any time. By comparison, the timeouts in [BD23] can also model mixed-choice, but since they are unstructured and non-deterministic, communication mismatches may occur and as such, the type/process must be designed to handle these failures. Our processes feature time-sensitive conditional statements in our `if-then-else` processes to model the counterpart of a timeout, while [BD23] instead staggers timeout processes in an alternating pattern. While this is effective in their context, it does limit the descriptive capabilities to mixed-choice with only two distinct regions of sending and receiving actions. In comparison, our work allows mixed-choice composed of any number of sending and receiving actions interleaved together in the same state.

Fault-Tolerant MPST [PNW22, PNW23] allows a branching type/process to specify a default value, to be used in the case when nothing is received, abstracting from the decision procedure on when to trigger this case. (This approach relies on an external failure detector to inform the process to take the default value.) While different from a timeout, this does allow a process that would otherwise be stuck waiting forever, to instead make progress. Our timeout process and timers closer align with programming primitives like timeouts and timers in e.g., Erlang (with its `receive-after` pattern and timers) and Go.

Mixed Sessions [VCAM20] introduces mixed-choice in untimed *synchronous* session types. They allow for mixed-choice with a single communication primitive, similar to our own types. Unlike our work, choices in [VCAM20] are non-deterministic and labels in a choice do not have to be unique. Additionally, options in a choice are either linear or *unrestricted.* This approach leads to patterns such as the producer/consumer pattern being elegantly represented by a single type, whereas in our system several would be needed. Effectively, the unrestricted action embeds into one of the options action within a choice, a recursive call that returns back to the choice. This is once again similar to the timeout-recursion behaviour found in [BLTV22, ICHZ23], except that [VCAM20] is *untimed.* Recent work by [PY24] present a typing system for mixed-choice in *synchronous* multiparty sessions and explores the expressiveness offered by mixed-choice by evaluating and comparing various other works (with and without mixed-choice).

Further afield, coordination structures have been proposed that overlap with mixed-choice, for example, fork and join [DY12], which permit messages within a fork (and its corresponding join) to be sent or received in any order; reminiscent of mixed-choice. Affine sessions [LNY22, MV18] support exception handling by enabling an end-point to perform a subset of the interactions specified by their type, but there is no consideration of time, hence timeouts. Before session types gained traction, timed processes [BY07] were proposed for realising timeouts, but lack any notion of counterpart.

Timed semantics. Our work is based on the timed semantics of Communicating Timed Automata [KY06] (CTA). CTA is the combined product of Timed Automata [AD94] and CFSM [BZ83], yielding a timed asynchronous semantics. Our time-passing semantics of [`time`] in Eq (2.5) is based upon the *urgent semantics* [BBM18], to enforce *receive urgency* and ensure that the *latest enabled* action is not missed [BBM18]. Our semantics for time-passing over configurations differs from [BMVY19], which requires that sending actions are never missed. In our setting a configuration featuring a mixed-choice, such as $(\nu_0, \{!a\,(x < 3, \varnothing)\,.S, ?b\,(x \geqslant 3, \varnothing)\,.S'\})$ would be unable to reach the latest-enabled receiving action b, since the preceding sending action a would *always* happen instead, effectively rendering b redundant. However, in this work we only require that the latest-enabled action is never missed, allowing instances such as a to be passed. This provides implementations of TOAST protocols the affordance of not featuring every sending action, and allows them to draw upon the full range of interactions described by the given TOAST protocol.

The work of [BCM17] presents (binary) Timed Session Types in the synchronous setting. Instead of relying on a notion of *duality*, [BCM17] presents a means of determining if two parties are *compliant*, given their sets of constraints. This allows the timing constraints on actions to be constructed without considering the dual, and therefore, avoiding such issues as those discussed in Example 2.2. E.g., if one party expects to receive a some time after 3 time units, but has some proceeding action c that will not be enabled after a total of 5 time units, then the party can still be guaranteed to be free from deadlocks if there is some *compliant co-party* that always sends before 4 time units; as in Example 2.2.

8. Conclusion

We have shown how timing constraints provide an intuitive way of integrating mixed-choice into asynchronous (binary) session types. There are many conceivable ways to realise mixed-choice using programming primitives. However, our integration with time, embodied in TOAST, offers new capabilities for modelling timeouts which sit at the heart of protocols and are a widely-used idiom in programming practice. To provide a bridge to programming languages, we provide a timed session calculus enriched with a `receive-after` pattern and process timers, providing the means to implement a timeout process and its dual.

Taken altogether, we have lifted a long-standing restriction on asynchronous session types by allowing for safe mixed-choice, through the judicious application of timing constraints. In this extended version of [PBK23], we have established a means of type-checking our processes with TOAST. Additionally, we have shown that processes that are well-typed by our type system adhere to subject reduction (Corollary 6.5) which means that any step the system takes maintains the shape of a well-typed system. Subject reduction guarantees that when an action happens, its sender, receiver, label, payload, and timing all conform to the types.

Future Work. There are two primary focuses for future work: (1) an extension to the multiparty setting; and (2) a progress property for our typing system. Progress was included in previous work on timed session types [BMVY19]. This result is not easily transferable to our work since we deal with mixed-choice. In fact, the result in [BMVY19] builds upon existing techniques to ensure global progress [DdY07] on an untimed version of their timed processes. In essence, [BMVY19] uses an *untimed* version of a process to check for deadlocks in the interaction structures of the process, using the checks described in [DdY07]. We are unable to take this approach since: (a) the work in [DdY07] does not account for mixed-choice, and (b) the untimed counterpart of a *safe* timed mixed-choice may not enjoy progress since the timing constraints that make it *safe* would be removed. Proving progress in this setting requires explicit reasoning on the interactions structures. Therefore, we will have to take a different approach to in order to provide a stronger property for our typing system.

Additionally, in future work we would like to continue the development of the theory, such as to the multiparty setting, and in turn, enable work such as the toolchain presented by [PBH24] to have a broader potential for application to real world systems. As part of this growing toolchain, we aim to include an implementation of our typing system.

References

[AD94] Rajeev Alur and David L. Dill. A Theory of Timed Automata. *TCS*, 126:183–235, 1994. doi:10.1016/0304-3975(94)90010-8.

[BBM18] Massimo Bartoletti, Laura Bocchi, and Maurizio Murgia. Progress-Preserving Refinements of CTA. In *CONCUR*, volume 118 of *LIPIcs*, pages 40:1–40:19. Schloss Dagstuhl, 2018. doi:10.4230/LIPIcs.CONCUR.2018.40.

[BCD+08] Lorenzo Bettini, Mario Coppo, Loris D'Antoni, Marco De Luca, Mariangiola Dezani-Ciancaglini, and Nobuko Yoshida. Global Progress in Dynamically Interleaved Multiparty Sessions. In *CONCUR*, volume 5201 of *LNCS*, pages 418–433. Springer, 2008. doi:10.1007/978-3-540-85361-9_33.

[BCM17] Massimo Bartoletti, Tiziana Cimoli, and Maurizio Murgia. Timed Session Types. *Logical Methods in Computer Science*, 13(4), 2017. arXiv:1710.05388, doi:10.23638/LMCS-13(4:25)2017.

[BCP14] Massimo Bartoletti, Tiziana Cimoli, and G. Michele Pinna. A note on two notions of compliance. *EPTCS*, 166:86–93, 2014. arXiv:1410.7472, doi:10.4204/EPTCS.166.9.

[BD23] Matthew Alan Le Brun and Ornela Dardha. MAGπ: Types for Failure-Prone Communication. In *Programming Languages and Systems*, volume 13990 of *LNCS*, pages 363–391. Springer, 2023. doi:10.1007/978-3-031-30044-8_14.

[BH16] Giovanni Bernardi and Matthew Hennessy. Using higher-order contracts to model session types. *Logical Methods in Computer Science*, 12, 2016. URL: https://lmcs.episciences.org/1642, doi:10.2168/LMCS-12(2:10)2016.

[BLTV22] Laura Bocchi, Julien Lange, Simon Thompson, and A. Laura Voinea. A Model of Actors and Grey Failures. In *COORDINATION*, volume 13271 of *LNCS*, pages 140–158. Springer, 2022. doi:10.1007/978-3-031-08143-9_9.

[BLY15] Laura Bocchi, Julien Lange, and Nobuko Yoshida. Meeting Deadlines Together. In *CONCUR*, volume 42 of *LIPIcs*, pages 283–296, 2015. doi:10.4230/LIPIcs.CONCUR.2015.283.

[BMVY19] Laura Bocchi, Maurizio Murgia, Vasco Thudichum Vasconcelos, and Nobuko Yoshida. Asynchronous timed session types: from duality to time-sensitive processes. In *ESOP*, volume 11423 of *LNCS*, pages 583–610. Springer, 2019. doi:10.1007/978-3-030-17184-1_21.

[BP12] Viviana Bono and Luca Padovani. Typing Copyless Message Passing. *Logical Methods in Computer Science*, 8, 2012. doi:10.2168/LMCS-8(1:17)2012.

[BY03] Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In *Lectures on Concurrency and Petri Nets*, volume 3098 of *LNCS*, pages 87–124. Springer, 2003. doi:10.1007/978-3-540-27755-2_3.

[BY07] Martin Berger and Nobuko Yoshida. Timed, Distributed, Probabilistic, Typed Processes. In *APLAS*, volume 4807 of *LNCS*, pages 158–174. Springer, 2007. doi:10.1007/978-3-540-76637-7_11.

[BYY14] Laura Bocchi, Weizhen Yang, and Nobuko Yoshida. Timed Multiparty Session Types. In *CONCUR*, volume 8704 of *LNCS*, pages 419–434. Springer, 2014. doi:10.1007/978-3-662-44584-6_29.

[BZ83] Daniel Brand and Pitro Zafiropulo. On Communicating Finite-State Machines. *Journal of the ACM*, 30(2):323–342, 1983. doi:10.1145/322374.322380.

[CDCSY17] Tzu-chun Chen, Mariangiola Dezani-Ciancaglini, Alceste Scalas, and Nobuko Yoshida. On the Preciseness of Subtyping in Session Types. *Logical Methods in Computer Science*, 13(2), 2017. doi:10.23638/LMCS-13(2:12)2017.

[CHJ+19] David Castro, Raymond Hu, Sung-Shik Jongmans, Nicholas Ng, and Nobuko Yoshida. Distributed programming using role-parametric session types in go: statically-typed endpoint APIs for dynamically-instantiated communication structures. In *POPL*, volume 3, pages 1–30. ACM, 2019. doi:10.1145/3290342.

[CHY08] Marco Carbone, Kohei Honda, and Nobuko Yoshida. Structured Interactional Exceptions in Session Types. In *CONCUR*, volume 5201 of *LNCS*, pages 402–417. Springer Berlin Heidelberg, 2008. doi:10.1007/978-3-540-85361-9_32.

[CV16] Francesco Cesarini and Steve Vinoski. *Designing for Scalability with Erlang/OTP*. O'Reilly Media, Inc., 2016.

[DdY07] Mariangiola Dezani-Ciancaglini, Ugo de'Liguoro, and Nobuko Yoshida. On Progress for Structured Communications. In *TGC*, volume 4912 of *LNCS*, pages 257–275. Springer, 2007. doi:10.1007/978-3-540-78663-4_18.

[DY12] Pierre-Malo Deniélou and Nobuko Yoshida. Multiparty Session Types Meet Communicating Automata. In *ESOP*, volume 7966 of *LNCS*, pages 194–213. Springer, 2012. doi:10.1007/978-3-642-28869-2_10.

[DY13] Pierre-Malo Deniélou and Nobuko Yoshida. Multiparty Compatibility in Communicating Automata: Characterisation and Synthesis of Global Session Types. In *ICALP*, volume 7966 of *LNCS*, pages 174–186. Springer, 2013. doi:10.1007/978-3-642-39212-2_18.

[GMY84] M.G. Gouda, E.G. Manning, and Y.T. Yu. On the progress of communication between two finite state machines. *Information and Control*, 63(3):200–216, 1984. doi:10.1016/S0019-9958(84)80014-5.

[HYC08] Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In *POPL*, pages 273–284. ACM, 2008. doi:10.1145/1328438.1328472.

[HYH08] Raymond Hu, Nobuko Yoshida, and Kohei Honda. Session-Based Distributed Programming in Java. In *ECOOP*, volume 5142 of *LNCS*, pages 516–541. Springer, 2008. doi:10.1007/978-3-540-70592-5_22.

[ICHZ23] Grant Iraci, Cheng-En Chuan, Raymond Hu, and Lukasz Ziarek. Validating IoT Devices with Rate-Based Session Types. *Proc. ACM Program. Lang.*, 7(OOPSLA):1589–1617, 2023. doi:10.1145/3622854.

[Kle08] John Klensin. SMTP, Request for Comments: 5321, 2008. Section 4.5.3.2.7: Server Timeout. URL: https://datatracker.ietf.org/doc/html/rfc5321.

[KY06] Pavel Krcál and Wang Yi. Communicating Timed Automata: The More Synchronous, the More Difficult to Verify. In *CAV*, volume 4144 of *LNCS*, pages 249–262. Springer, 2006. doi:10.1007/11817963_24.

[LM16] Sam Lindley and J. Garrett Morris. Talking bananas: structural recursion for session types. In *ICFP*, pages 434–447. ACM, 2016. doi:10.1145/2951913.2951921.

[LNY20] Nicolas Lagaillardie, Rumyana Neykova, and Nobuko Yoshida. Implementing Multiparty Session Types in Rust. In *COORDINATION*, pages 127–136. Springer, 2020. doi:10.1007/978-3-030-50029-0_8.

[LNY22] Nicolas Lagaillardie, Rumyana Neykova, and Nobuko Yoshida. Stay Safe under Panic: Affine Rust Programming with Multiparty Session Types. In *ECOOP*, LIPIcs. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2022. arXiv:2204.13464, doi:10.48550/arXiv.2204.13464.

[Mil99] Robin Milner. *Communicating and mobile systems - the Pi-calculus*. Cambridge University Press, 1999.

[MV18] Dimitris Mostrous and Vasco T. Vasconcelos. Affine Sessions. *Logical Methods in Computer Science*, 14(4), 2018. arXiv:1809.02781, doi:10.23638/LMCS-14(4:14)2018.

[Ney13] Rumyana Neykova. Session Types Go Dynamic or How to Verify Your Python Conversations. *EPTCS*, 137:95–102, 2013. doi:10.4204/EPTCS.137.8.

[NHYA18] Rumyana Neykova, Raymond Hu, Noboku Yoshida, and Fahd Abdeljallal. A session type provider: compile-time API generation of distributed protocols with refinements in F#. In *CC*, pages 128–138. ACM, 2018. doi:10.1145/3178372.3179495.

[NYH13] Rumyana Neykova, Nobuko Yoshida, and Raymond Hu. SPY: Local Verification of Global Protocols. In *RV*, volume 8174 of *LNCS*, pages 358–363. Springer, 2013. doi:10.1007/978-3-642-40787-1_25.

[PBH24] Jonah Pears, Laura Bocchi, and Raymond Hu. Erlang on TOAST: Generating Erlang Stubs with Inline TOAST Monitors. In *Erlang Workshop*, pages 33–44. ACM, 2024. doi:10.1145/3677995.3678192.

[PBK23] Jonah Pears, Laura Bocchi, and Andy King. Safe asynchronous mixed-choice for timed interactions. In *COORDINATION*, volume 13908 of *LNCS*, pages 214–231. Springer, 2023. doi:10.1007/978-3-031-35361-1_12.

[Pie02] Benjamin C. Pierce. *Types and Programming Languages*. MIT Press, 2002.

[PNW22] Kirstin Peters, Uwe Nestmann, and Christoph Wagner. Fault-Tolerant Multiparty Session. In *Formal Techniques for Distributed Objects, Components, and Systems*, pages 93–113. Springer, 2022. doi:10.1007/978-3-031-08679-3_7.

[PNW23] Kirstin Peters, Uwe Nestmann, and Christoph Wagner. FTMPST: Fault-Tolerant Multiparty Session Types. volume 19, 2023. doi:10.46298/lmcs-19(4:14)2023.

[PY24] Kirstin Peters and Nobuko Yoshida. Separation and encodability in mixed choice multiparty sessions. Symposium on Logic in Computer Science. ACM, 2024. doi:10.1145/3661814.3662085.

[Vas12] Vasco T. Vasconcelos. Fundamentals of session types. *Information and Control*, 217:52–70, 2012. doi:10.1016/J.IC.2012.05.002.

[VCAM20] Vasco T. Vasconcelos, Filipe Casal, Bernardo Almeida, and Andreia Mordido. Mixed Sessions. In *ESOP*, volume 12075 of *LNCS*, pages 715–742. Springer, 2020. doi:10.1007/978-3-030-44914-8_26.

[YV07] Nobuko Yoshida and Vasco T. Vasconcelos. Language Primitives and Type Discipline for Structured Communication-Based Programming Revisited: Two Systems for Higher-Order Session Communication. *Electronic Notes in Theoretical Computer Science*, 171(4):73–93, 2007. doi:10.1016/j.entcs.2007.02.056.

[YZF21] Nobuko Yoshida, Fangyi Zhou, and Francisco Ferreira. Communicating Finite State Machines and an Extensible Toolchain for Multiparty Session Types. In *Fundamentals of Computation Theory*, volume 12867 of *LNCS*, pages 18–35. Springer, 2021. doi:10.1007/978-3-030-86593-1_2.

Appendix A. Proof of Type Progress

The result of Theorem 2.9 follows similar reasoning to [BMVY19]. While Theorem 2.9 states progress for *initial system configurations*, progress is also guaranteed for systems composed of *well-formed* and *compatible* configurations by Lemma A.33. Our proof shows that our *well-formedness* rules in Eq (2.8) ensure that the timing-constraints on actions are *feasible*, and structured such that any mixed-choice are *safe*. We show that our LTS in Eqs (2.4 to 2.6) preserve both *well-formedness* and compatibility of our configurations.

A.1. Auxiliary Definitions & Assumptions.

Definition A.1 (Latest-enabled Configurations)**.** $(\nu,\, S)$ is *latest-enabled* if, $\exists \square, \square'$ such that $\square \neq \square'$ and $(\nu,\, S) \overset{\square}{\Rightarrow}$ and $(\nu,\, S) \overset{\square'}{\not\Rightarrow}$.

Definition A.2 (Live Configurations)**.** $(\nu,\, S)$ is *live* if, $S = \mathtt{end}$ or $(\nu,\, S)$ is *future-enabled*.

Definition A.3 (Clock Names)**.** We define $\texttt{cn}(\delta)$, the *clock names* of a constraint δ, as:

$$\texttt{cn}(\delta) = \begin{cases} \{x\} & \text{if } \delta \in \left\{x > n, x = n\right\} \\ \{x, y\} & \text{if } \delta \in \left\{x - y > n, x - y = n\right\} \\ \texttt{cn}(\delta') & \text{if } \delta = \neg\delta' \\ \texttt{cn}(\delta_1) \cup \texttt{cn}(\delta_2) & \text{if } \delta = \delta_1 \wedge \delta_2 \\ \varnothing & \text{if } \delta = \texttt{true} \end{cases}$$

Definition A.4 (Free Names)**.** We define $\texttt{fn}(S)$, the *free names* of a type S, as:[3]

$$\texttt{fn}(S) = \begin{cases} \{\alpha\} & \text{if } S = \alpha \\ \texttt{fn}(S') \backslash \{\alpha\} & \text{if } S = \mu\alpha.S' \\ \bigcup_{i\in I} \texttt{fn}(S_i) & \text{if } S = \left\{\Box_i\, l_i \langle T_i \rangle (\delta_i, \lambda_i).S_i\right\}_{i\in I} \\ \varnothing & \text{if } S = \texttt{end} \end{cases}$$

Definition A.5 (Capture Avoiding Substitution)**.** Substitution $[{}^{S'}\!/\!_{\alpha}]$ is *capture avoiding* for S iff:

$$\begin{array}{lll} S = \left\{\Box_i\, l_i \langle T_i \rangle (\delta_i, \lambda_i).S_i\right\}_{i\in I} & \Longrightarrow & [{}^{S'}\!/\!_{\alpha}] \text{ is capture avoiding for } S_j,\ \forall j \in I \\ S = \mu\beta.S'' \wedge \alpha \neq \beta & \Longrightarrow & \beta \notin \texttt{fn}(S') \text{ and } [{}^{S'}\!/\!_{\alpha}] \text{ is capture avoiding for} S'' \end{array}$$

With an abuse of notation, we say that $S\,[{}^{S'}\!/\!_{\alpha}]$ is capture avoiding when $[{}^{S'}\!/\!_{\alpha}]$ is capture avoiding for S.

Definition A.6 (Unfold Equivalence)**.** We define $(\overset{\text{unfold}}{\equiv})$ as the smallest relation such that:

$$S \overset{\text{unfold}}{\equiv} S \qquad S \overset{\text{unfold}}{\equiv} S' \implies S' \overset{\text{unfold}}{\equiv} S \qquad S \overset{\text{unfold}}{\equiv} S' \wedge S' \overset{\text{unfold}}{\equiv} S'' \implies S \overset{\text{unfold}}{\equiv} S''$$

$$S\,[{}^{\mu\alpha.S}\!/\!_{\alpha}] \text{ is capture avoiding} \implies \mu\alpha.S \overset{\text{unfold}}{\equiv} S\,[{}^{\mu\alpha.S}\!/\!_{\alpha}] \qquad S \overset{\text{unfold}}{\equiv} S' \implies \mu\alpha.S \overset{\text{unfold}}{\equiv} \mu\alpha.S'$$

$$S \overset{\text{unfold}}{\equiv} S' \implies \left\{\Box l \langle T \rangle (\delta, \lambda).S,\, \Box'' l'' \langle T'' \rangle (\delta'', \lambda'').S''\right\} \overset{\text{unfold}}{\equiv} \left\{\Box l \langle T \rangle (\delta, \lambda).S',\, \Box'' l'' \langle T'' \rangle (\delta'', \lambda'').S''\right\}$$

Assumption A.7. If $A, \alpha : \delta'; \delta \vdash S$, then $\alpha \notin \text{dom}(A)$.

A.2. **Well-formed Types.**

Lemma A.8. *If $\alpha \notin \texttt{fn}(S)$ and $\alpha \notin$ dom(A), then $A, \alpha : \delta'; \delta \vdash S \iff A; \delta \vdash S$.*

Proof. Since $\alpha \notin \text{dom}(A)$ it follows that recursive call α has not yet been defined. Additionally, since $\alpha \notin \texttt{fn}(S)$, then it follows Definition A.4 that if $\mu\alpha.S'$ appears at any point within S, then α *may* appear at some point within S'. For each case of ($\iff$):

Case 1. $A, \alpha : \delta'; \delta \vdash S \Longrightarrow A; \delta \vdash S$. If $S = \mu\beta.S'$, then by rule [rec]:

$$\frac{A, \alpha : \delta', \beta : \delta; \delta \vdash S'}{A, \alpha : \delta'; \delta \vdash \mu\beta.S'}\ [\texttt{rec}]$$

Notice the environment $A, \alpha : \delta', \beta : \delta$ in the premise of rule [rec] (above). If $\beta = \alpha$ then, the environment would contain duplicate variables. Therefore, $\beta \neq \alpha$ or β must be renamed to some $\beta' \notin \text{dom}(A, \alpha : \delta')$ and $S = \mu\beta'.S'\,[{}^{\beta'}\!/\!_{\beta}]$. Similarly, if $S = \beta$ then, since $\alpha \notin \texttt{fn}(\beta)$, it must be that $\beta \neq \alpha$. In both of these cases, the presence of $\alpha : \delta$ is immediately inconsequential, and will remain so for the remainder of the

[3]As in [Pie02].

evaluation. Clearly, if $S = \texttt{end}$ then the same holds. If $S = \{\Box_i\, l_i\, \langle T_i \rangle\, (\delta_i, \lambda_i).S_i\}_{i \in I}$ then by Definition A.4, it must also hold for all $i \in I$. In summary, since $\alpha \notin \mathrm{dom}(A)$ and $\alpha \notin \texttt{fn}\,(S)$, it follows that α must correspond to a recursive variable in a different unfolding (either outside S, or a future unfolding within S), and that it has been renamed from the corresponding recursive variable in the current unfolding.

Case 2. $A; \delta \vdash S \Longrightarrow A, \alpha : \delta'; \delta \vdash S$. Following Case 1, the presence of $\alpha : \delta$ is inconsequential since α corresponds to some already unfolded recursive call outside S, or a some future unfolding within S. In both cases, the α has been renamed from the corresponding recursive variable in the current unfolding. ◻

Lemma A.9 (Substitution Lemma)**.** *Let* $A; \delta' \vdash S'$*. If* $\alpha \notin dom(A)$ *and* $S\,[S'/\alpha]$ *is* capture avoiding*, then* $A; \delta \vdash S\,[S'/\alpha] \iff A, \alpha : \delta'; \delta \vdash S$ *holds.*

Proof. We proceed by induction on the structure of S:

Case 1. If $S = \mu\beta.S''$ then, for each case of ($\iff$):

i. $A; \delta \vdash \mu\beta.S''\,[S'/\alpha] \Longrightarrow A, \alpha : \delta'; \delta \vdash \mu\beta.S''$.

a. If $\beta = \alpha$, then $\mu\beta.S''\,[S'/\alpha] = \mu\beta.S''\,[S'/\beta] = \mu\beta.S''$. Therefore:

$$A; \delta \vdash \mu\beta.S''$$

Since $\alpha = \beta$ and $\alpha \notin \mathrm{dom}(A)$, by Lemma A.9 we have the thesis:

$$A, \alpha : \delta'; \delta \vdash \mu\beta.S''$$

b. If $\beta \neq \alpha$, then by rule [rec]:

$$\frac{A, \beta : \delta; \delta \vdash S''\,[S'/\alpha]}{A; \delta \vdash \mu\beta.S''\,[S'/\alpha]}\ [\texttt{rec}]$$

By Definition A.5, $\beta \notin \texttt{fn}\,(S')$ and $S''\,[S'/\alpha]$ is *capture avoiding*. By Lemma A.8: $A, \beta : \delta; \delta'; \vdash S' \iff A; \delta' \vdash S'$. By the induction hypothesis:

$$A, \alpha : \delta', \beta : \delta; \delta \vdash S'' \iff A, \beta : \delta; \delta; \vdash S''\,[S'/\alpha]$$

We obtain our thesis following the conclusion of rule [rec] (below), using the former case (above) as our premise:

$$\frac{A, \alpha : \delta', \beta : \delta; \delta \vdash S''}{A, \alpha : \delta'; \delta \vdash \mu\beta.S''}\ [\texttt{rec}]$$

ii. $A, \alpha : \delta'; \delta \vdash \mu\beta.S'' \Longrightarrow A; \delta \vdash \mu\beta.S''\,[S'/\alpha]$. By rule [rec]:

$$\frac{A, \alpha : \delta', \beta : \delta; \delta \vdash S''}{A, \alpha : \delta'; \delta \vdash \mu\beta.S''}\ [\texttt{rec}]$$

It cannot be that $\beta = \alpha$, as otherwise the environment $A, \alpha : \delta', \beta : \delta$ would contain a duplicate variable (as in the premise of rule [rec], above). Therefore, it must be that $\beta \neq \alpha$. Since $\mu\beta.S''\,[S'/\alpha]$ is *capture avoiding*, by Definition A.5, $\beta \notin \texttt{fn}\,(S')$ and $S''\,[S'/\alpha]$ is also *capture avoiding*. By Lemma A.8: $A; \delta' \vdash S' \iff A, \beta : \delta; \delta' \vdash S'$. By the induction hypothesis:

$$A, \alpha : \delta', \beta : \delta; \delta \vdash S'' \iff A, \beta : \delta; \delta \vdash S''\,[S'/\alpha]$$

We obtain our thesis following the conclusion of rule [rec] (below), using the latter case (above) as the premise:

$$\frac{A, \beta : \delta; \delta \vdash S'' \left[{}^{S'}\!/\alpha\right]}{A; \delta \vdash \mu\beta.S'' \left[{}^{S'}\!/\alpha\right]} \; [\texttt{rec}]$$

Case 2. If $S = \beta$ then, for each case of ($\iff$):

i. $A; \delta \vdash \beta \left[{}^{S'}\!/\alpha\right] \implies A, \alpha : \delta'; \delta \vdash \beta$.

a. If $\beta = \alpha$, then $\beta \left[{}^{S'}\!/\alpha\right] = S'$ and $\delta' = \delta$. Since $\alpha \notin \mathrm{dom}(A)$, it follows:

$$\frac{}{A', \beta : \delta; \delta \vdash \beta} \; [\texttt{var}]$$

b. If $\beta \neq \alpha$, then $\beta \left[{}^{S'}\!/\alpha\right] = \beta$ and $A = A', \beta : \delta$ and by rule [var]:

$$\frac{}{A', \beta : \delta; \delta \vdash \beta} \; [\texttt{var}]$$

By rule [var] (and up-to reordering of variables):

$$\frac{}{A', \beta : \delta, \alpha : \delta'; \delta \vdash \beta} \; [\texttt{var}]$$

ii. $A, \alpha : \delta'; \delta \vdash \beta \implies A; \delta \vdash \beta \left[{}^{S'}\!/\alpha\right]$.

a. If $\beta = \alpha$, then $\delta' = \delta$ and $\beta \left[{}^{S'}\!/\alpha\right] = S'$. The thesis coincides with the hypothesis:

$$A; \delta' \vdash S'$$

b. If $\beta \neq \alpha$, then $\beta \left[{}^{S'}\!/\alpha\right] = \beta$ and $A = A', \beta : \delta$ and by rule [var]:

$$\frac{}{A', \alpha : \delta', \beta : \delta; \delta \vdash \beta} \; [\texttt{var}]$$

The thesis follows immediately by rule [var]:

$$\frac{}{A', \beta : \delta; \delta \vdash \beta} \; [\texttt{var}]$$

Case 3. If $S = \left\{\Box_i\, l_i \langle T_i \rangle (\delta_i, \lambda_i).S_i\right\}_{i \in I}$ then, for each case of ($\iff$):

i. $A; \delta \vdash \left\{\Box_i\, l_i \langle T_i \rangle (\delta_i, \lambda_i).S_i\right\}_{i \in I} \left[{}^{S'}\!/\alpha\right] \implies A, \alpha : \delta'; \delta \vdash \left\{\Box_i\, l_i \langle T_i \rangle (\delta_i, \lambda_i).S_i\right\}_{i \in I}$.

By Definition A.5, for all $i \in I$, $S_i \left[{}^{S'}\!/\alpha\right]$ is *capture avoiding.* By rule [choice]:

$$\frac{\begin{array}{ll} \forall i \in I : A;\; \gamma_i \vdash S_i \left[{}^{S'}\!/\alpha\right] \;\wedge\; \delta_i \left[\lambda_i \mapsto 0\right] \models \gamma_i & \text{(feasibility)} \\ \forall i, j \in I : i \neq j \implies \delta_i \wedge \delta_j \models \texttt{false} \;\vee\; \Box_i = \Box_j & \text{(mixed-choice)} \\ \forall i \in I : T_i = \left(\delta''',\; S'''\right) \implies \varnothing;\; \gamma' \vdash S''' \;\wedge\; \delta''' \models \gamma' & \text{(delegation)} \end{array}}{A;\; \downarrow \bigvee_{i \in I} \delta_i \vdash \left\{\Box_i\, l_i \langle T_i \rangle (\delta_i, \lambda_i).S_i\right\}_{i \in I} \left[{}^{S'}\!/\alpha\right]} \; [\texttt{choice}] \tag{A.1}$$

where $\delta = \downarrow \bigvee_{i \in I} \delta_i$. By the induction hypothesis:

$$A; \gamma_i \vdash S_i \left[{}^{S'}\!/\alpha\right] \iff A, \alpha : \delta'; \gamma_i \vdash S_i \qquad \text{where } \delta_i \left[\lambda_i \mapsto 0\right] \models \gamma_i$$

Therefore, we obtain our thesis following the conclusion of rule [choice] (below), using the latter case (above) as the (feasibility) premise:

$$\frac{\begin{array}{ll} \forall i \in I : A, \alpha : \delta';\ \gamma_i \vdash S_i \ \wedge\ \delta_i\,[\lambda_i \mapsto 0] \models \gamma_i & \text{(feasibility)} \\ \forall i,j \in I : i \neq j \implies \delta_i \wedge \delta_j \models \texttt{false} \ \vee\ \Box_i = \Box_j & \text{(mixed-choice)} \\ \forall i \in I : T_i = \big(\delta''',\ S'''\big) \implies \varnothing;\ \gamma' \vdash S''' \ \wedge\ \delta''' \models \gamma' & \text{(delegation)} \end{array}}{A, \alpha : \delta';\ \downarrow \bigvee_{i\in I} \delta_i \vdash \big\{\Box_i\, l_i\, \langle T_i\rangle\, (\delta_i, \lambda_i).S_i\big\}_{i\in I}}\ [\texttt{choice}] \tag{A.2}$$

ii. $A, \alpha : \delta'; \delta \vdash \big\{\Box_i\, l_i\, \langle T_i\rangle\, (\delta_i, \lambda_i).S_i\big\}_{i\in I} \implies A; \delta \vdash \big\{\Box_i\, l_i\, \langle T_i\rangle\, (\delta_i, \lambda_i).S_i\big\}_{i\in I}\, [{}^{S'}\!/\alpha]$.

By rule [choice] as in Eq (A.2), $\delta =\downarrow \bigvee_{i\in I} \delta_i$. By the (feasibility) premise of rule [choice], for all $i \in I$, $A, \alpha : \delta'; \gamma_i \vdash S_i$ and $\delta_i\,[\lambda_i \mapsto 0] \models \gamma_i$. Therefore, by the induction hypothesis:

$$A, \alpha : \delta; \gamma_i \vdash S_i \iff A; \gamma_i \vdash S_i\,[{}^{S'}\!/\alpha] \qquad \text{where } \delta_i\,[\lambda_i \mapsto 0] \models \gamma_i$$

Our thesis follows the conclusion of rule [choice] in Eq (A.1), where we use the latter case (above) as the (feasibility) premise.

Case 4. If $S = \texttt{end}$ then, by rule [end]:

$$\frac{}{A; \texttt{true} \vdash \texttt{end}}\ [\texttt{end}]$$

where $\delta = \texttt{true}$. By rule [end], end is *well-formed* against any δ and any A. It follows that $\texttt{end}\,[{}^{S'}\!/\alpha] = \texttt{end}$. Therefore, the hypothesis holds for $S = \texttt{end}$.

We have shown the hypothesis to hold for any S. The interesting cases are when $S = \mu\beta.S''$ or $S = \beta$, which vary depending on $\alpha = \beta$ or $\alpha \neq \beta$. Since $\alpha \notin \text{dom}(A)$, we are dealing with either: (1) an unfolding of a yet-to-be defined recursion α, or (2) an future unfolding of a renamed recursion. By our thesis, we prove that unfolding and renaming recursive types does not interfere with the judgements of *well-formedness* found in the rules in Eq (2.8). □

A.3. **Well-formed Configurations.**

Lemma A.10. *(ν, end) is always* well-formed.

Proof. By Definition 2.6, $\exists\delta$ such that $\nu \models \delta$ and $\varnothing; \delta \vdash \texttt{end}$. By rule [end] in Eq (2.8), $\delta = \texttt{true}$. Therefore, our thesis follows $\nu \models \texttt{true}$. □

Lemma A.11. *If (ν, S) is* well-formed*, then $S \neq \alpha$.*

Proof. Let us consider by contradiction that $S = \alpha$. By Definition 2.6, $\exists\delta$ such that $\nu \models \delta$ and $\varnothing; \delta \vdash \alpha$. The only rule applicable to α is [var]:

$$\frac{}{A, \alpha : \delta; \delta \vdash \alpha}\ [\texttt{var}]$$

Yet, $\varnothing \neq A, \alpha : \delta$, and so $S = \alpha$ contradicts with Definition 2.6 and we obtain our thesis. □

Lemma A.12. *If $S \overset{\texttt{unfold}}{\equiv} S'$, then $A; \delta \vdash S \implies A; \delta \vdash S'$.*

Proof. We proceed by induction on the last equation of Definition A.6 used to establish $S \overset{\texttt{unfold}}{\equiv} S'$.

Case 1. If $S = \mu\alpha.S_0$ with $S_0\,[\mu\alpha.S_0/\alpha]$ capture avoiding, then $S' = S_0\,[\mu\alpha.S_0/\alpha]$. The only rule applicable to $A;\delta \vdash \mu\alpha.S_0$ is rule [rec]:

$$\frac{A, \alpha : \delta; \delta \vdash S_0}{A; \delta \vdash \mu\alpha.S_0}\ [\texttt{rec}]$$

Note that α cannot be in $\mathrm{dom}(A)$. Since $S_0\,[\mu\alpha.S_0/\alpha]$ is capture avoiding, by Lemma A.9:

$$A, \alpha : \delta; \delta \vdash S_0 \iff A; \delta \vdash S_0\,[\mu\alpha.S_0/\alpha]$$

The thesis follows the conclusion of rule [rec], using the latter case (above) as premise:

$$\frac{A, \alpha : \delta; \delta \vdash S_0\,[\mu\alpha.S_0/\alpha]}{A; \delta \vdash \mu\alpha.S_0\,[\mu\alpha.S_0/\alpha]}\ [\texttt{rec}]$$

Case 2. If $S = \mu\alpha S_0$, then $S' = \mu\alpha S_0'$ with $S_0 \overset{\texttt{unfold}}{\equiv} S_0'$. The only rule applicable to $A;\delta \vdash \mu\alpha.S_0$ is rule [rec]:

$$\frac{A, \alpha : \delta; \delta \vdash S_0}{A; \delta \vdash \mu\alpha.S_0}\ [\texttt{rec}]$$

By the induction hypothesis applied to the premise of the above: $A, \alpha : \delta; \delta \vdash S_0'$. Therefore, we obtain our thesis following the conclusion of rule [rec]:

$$\frac{A, \alpha : \delta; \delta \vdash S_0'}{A; \delta \vdash \mu\alpha.S_0'}\ [\texttt{rec}]$$

Case 3. If $S = \{\Box' l' \langle T' \rangle(\delta', \lambda').S_0'\}_{i\in I}$ then $S' = \{\Box' l' \langle T' \rangle(\delta', \lambda').S_0''\}_{i\in I}$ with $S_0' \overset{\texttt{unfold}}{\equiv} S_0''$. (For brevity and clarity we only show for $|I| = 1$.) The only rule applicable to $A;\delta \vdash \{\Box' l' \langle T' \rangle(\delta', \lambda').S_0'\}_{i\in I}$ is rule [choice]:

$$\frac{\begin{array}{ll} A;\ \gamma \vdash S_0' \ \wedge\ \delta'\,[\lambda \mapsto 0]' \models \gamma & \text{(feasibility)} \\ T' = (\delta'',\ S'') \implies \varnothing;\ \gamma' \vdash S'' \ \wedge\ \delta'' \models \gamma' & \text{(delegation)} \end{array}}{A;\ \downarrow \bigvee_{i\in I} \delta_i \vdash \{\Box' l' \langle T' \rangle(\delta', \lambda').S_0'\}_{i\in I}}\ [\texttt{choice}]$$

where $\delta = \downarrow \bigvee_{i\in I} \delta_i = \downarrow \delta'$. (Since $|I| = 1$, we omit the (mixed-choice) premise of rule [choice].) By the induction hypothesis, applied to the (feasibility) premise above:

$$A;\ \gamma \vdash S_0'' \ \wedge\ \delta'\,[\lambda \mapsto 0]' \models \gamma$$

We obtain our thesis following the conclusion of rule [choice], when the above is used as the (feasibility) premise:

$$\frac{\begin{array}{ll} A;\ \gamma \vdash S_0'' \ \wedge\ \delta'\,[\lambda \mapsto 0]' \models \gamma & \text{(feasibility)} \\ T' = (\delta'',\ S'') \implies \varnothing;\ \gamma' \vdash S'' \ \wedge\ \delta'' \models \gamma' & \text{(delegation)} \end{array}}{A;\ \downarrow \bigvee_{i\in I} \delta_i \vdash \{\Box' l' \langle T' \rangle(\delta', \lambda').S_0''\}_{i\in I}}\ [\texttt{choice}]$$

□

Lemma A.13. *If* $S \overset{\texttt{unfold}}{\equiv} S'$*, then* $(\nu,\ S)$ *is* well-formed $\implies$ $(\nu,\ S')$ *is* well-formed*.*

Proof. By Definition 2.6, $\exists\delta$ such that $\nu \models \delta$ and $\varnothing;\delta \vdash S$. The thesis follows immediately from Lemma A.12, since $\varnothing;\delta \vdash S'$. □

Lemma A.14. *If* $(\nu,\ S)$ *is* well-formed*, then* $S \overset{\texttt{unfold}}{\equiv} \texttt{end}$ *or* $S \overset{\texttt{unfold}}{\equiv} \{\Box_i\, l_i \langle T_i \rangle\, (\delta_i, \lambda_i).S_i\}_{i\in I}$*.*

Proof. Since $(\nu,\, S)$ is *well-formed*, by Definition 2.6 $\exists\delta$ such that $\nu \models \delta$ and $\varnothing; \delta \vdash S$. We proceed by induction on the derivation, analysing the structure of S. By Definition A.6, the only interesting case is $S = \mu\alpha.S''$ and $S \overset{\texttt{unfold}}{\equiv} S'\,[{}^{\mu\alpha.S'}/_{\alpha}]$. By rule [rec]:

$$\frac{\alpha : \delta; \delta \vdash S'}{\varnothing; \delta \vdash \mu\alpha.S'}\ [\texttt{rec}]$$

By Lemma A.9, $\alpha : \delta; \delta \vdash S' \iff \varnothing; \delta \vdash S'\,[{}^{\mu\alpha.S'}/_{\alpha}]$. Therefore, it holds that $(\nu,\, S'\,[{}^{\mu\alpha.S'}/_{\alpha}])$ is *well-formed*, and so, we obtain our thesis by the induction hypothesis:

$$(\nu,\, S'\,[{}^{\mu\alpha.S'}/_{\alpha}]) \text{ is } \textit{well-formed} \implies (\nu,\, S'') \text{ is } \textit{well-formed}$$

where $S' \overset{\texttt{unfold}}{\equiv} S''$. Since types are *contractive*, we rule out the possibility of $S' = \mu\alpha_1 \ldots \mu\alpha_n.\alpha_i$, for some $i \leqslant n$ where $1 < n \in \mathbb{N}$. ◻

A.4. **Live Configurations.**

Lemma A.15. *If* $(\nu,\, S)$ *is* well-formed*, then* $(\nu,\, S)$ *is* live*.*

Lemma A.11. By Definition 2.6, $\exists\delta$ such that $\nu \models \delta$ and $\varnothing; \delta \vdash S$. We proceed by induction on the derivation of $\varnothing; \delta \vdash S$ by the *well-formedness* rules in Eq (2.8):

Case 1. If rule [choice], then $S = \left\{\Box_i\, l_i \langle T_i\rangle\, (\delta_i, \lambda_i).S_i\right\}_{i\in I}$ and $\delta = \downarrow \bigvee_{i\in I} \delta_i$. By Definition 2.1, $\exists t$ such that $\nu{+}t \models \bigvee_{i\in I} \delta_i$. It follows Definition 2.3 that $(\nu,\, S)$ is *future-enabled*. Following Definition A.2 we obtain our thesis.

Case 2. If rule [rec], then $S = \mu\alpha.S'$. By Lemma A.13, since $(\nu,\, \mu\alpha.S')$ is *well-formed*, if $S \overset{\texttt{unfold}}{\equiv} S''$ then $(\nu,\, S'')$ is *well-formed*. By Definition A.6, $\mu\alpha.S' \overset{\texttt{unfold}}{\equiv} S'\,[{}^{\mu\alpha.S'}/_{\alpha}] = S''$. Therefore, since $(\nu,\, S'\,[{}^{\mu\alpha.S'}/_{\alpha}])$ is *well-formed*, the thesis follows by induction hypothesis on $(\nu,\, S'\,[{}^{\mu\alpha.S'}/_{\alpha}])$ being *live*.

Case 3. If rule [end], then $S = \texttt{end}$. By Definition A.2, $(\nu,\, \texttt{end})$ is *live*.

Case 4. Rule [var] is not applicable following Lemma A.11, as $S \neq \alpha$. ◻

Lemma A.16. *If* $(\nu + t,\, S)$ *is* well-formed*, then* $(\nu,\, S)$ *is* well-formed*.*

Proof. By Lemma A.14, $S \overset{\texttt{unfold}}{\equiv} \texttt{end}$ or $S \overset{\texttt{unfold}}{\equiv} \left\{\Box_i\, l_i \langle T_i\rangle\, (\delta_i, \lambda_i).S_i\right\}_{i\in I}$. By Lemma A.13, together with rules [end] and [choice] we have that $\varnothing; \delta \vdash S$ where $\delta = \texttt{true}$ or $\delta = \downarrow \bigvee_{i\in I} \delta_i$. Obviously, $\nu \models \texttt{true}$. The fact $\nu \models \downarrow \bigvee_{i\in I} \delta_i$ instead follows from the definition of $\downarrow$. ◻

Lemma A.17. *If* $\varnothing; \delta \vdash S$ *and* $(\nu,\, S)$ *is* future-enabled*, then* $(\nu,\, S)$ *is* well-formed*.*

Definition 2.3 and Lemma A.17. By Definition 2.3, $\exists t$ s.t. $(\nu,\, S) \xrightarrow{t} (\nu + t,\, S) \xrightarrow{\Box m} (\nu',\, S')$. We proceed by induction on the last rule applied for the transition $(\nu + t,\, S) \xrightarrow{\Box m}$ of those in Eq (2.4):

Case 1. If rule [act], then $S = \left\{\Box_i\, l_i \langle T_i\rangle\, (\delta_i, \lambda_i).S_i\right\}_{i\in I}$ and:

$$\frac{\nu'' \models \delta_j \quad m = l_j \langle T_j\rangle \quad j \in I}{(\nu'',\, \left\{\Box_i\, l_i \langle T_i\rangle\, (\delta_i, \lambda_i).S_i\right\}_{i\in I}) \xrightarrow{\Box_j m} (\nu''\,[\lambda_j \mapsto 0]\,,\, S_j)}\ [\texttt{act}]$$

where $\square = \square_j$ and $\nu'' = \nu + t$. Since $\varnothing; \delta \vdash \{\square_i\, l_i\, \langle T_i \rangle\, (\delta_i, \lambda_i).S_i\}_{i \in I}$ by hypothesis, by rule [choice] we conclude:

$$A;\ \downarrow \bigvee_{i \in I} \delta_i \vdash \{\square_i\, l_i\, \langle T_i \rangle\, (\delta_i, \lambda_i).S_i\}_{i \in I}$$

where $A = \varnothing$ and $\delta = \downarrow \bigvee_{i \in I} \delta_i$. By Definition 2.6, it remains to show that $\nu \models \downarrow \bigvee_{i \in I} \delta_i$. First we show that $\nu + t \models \downarrow \bigvee_{i \in I} \delta_i$. By the premise of rule [act], $\nu'' \models \delta_j$ (where $\nu'' = \nu + t$). Since $\delta_j \models \downarrow \bigvee_{i \in I} \delta_i$, it follows that $\nu + t \models \downarrow \bigvee_{i \in I} \delta_i$ and therefore, $(\nu + t,\ \{\square_i\, l_i\, \langle T_i \rangle\, (\delta_i, \lambda_i).S_i\}_{i \in I})$ is *well-formed*. By the conclusion of rule [choice], the set of constraints $\delta = \downarrow \bigvee_{i \in I} \delta_i$ only require a minimum of *one weak past* constraint to be satisfied for the entire set constraint to be satisfied. Therefore, given that we know $\nu + t \models \downarrow \bigvee_{i \in I} \delta_i$, it follows that $\nu \models \downarrow \bigvee_{i \in I} \delta_i$ must also hold, and we obtain our thesis.

Case 2. If rule [unfold], then $S = \mu\alpha.S''$ and:

$$\frac{(\nu + t,\ S''\,[{}^{\mu\alpha.S''}/\alpha]) \xrightarrow{\ell} (\nu',\ S')}{(\nu + t,\ \mu\alpha.S'') \xrightarrow{\ell} (\nu',\ S')}\ [\texttt{unfold}]$$

By the induction hypothesis we have that $(\nu + t,\ S''\,[{}^{\mu\alpha.S''}/\alpha])$ is *well-formed*. Then by Lemma A.13 we have that $(\nu + t,\ \mu\alpha.S'')$ is *well-formed*. Finally, by Lemma A.16, $(\nu,\ \mu\alpha.S'')$ is *well-formed*, as required. □

Lemma A.18. *If* $(\nu,\ S) \xrightarrow{\square m}$, *then* $(\nu,\ \overline{S}) \xrightarrow{\overline{\square} m}$.

Proof. We proceed by induction on the last rule applied for the transition $(\nu,\ S) \xrightarrow{\square m}$ of those in Eq (2.4):

Case 1. If rule [act], then $S = \{\square_i\, l_i\, \langle T_i \rangle\, (\delta_i, \lambda_i).S_i\}_{i \in I}$ and:

$$\frac{\nu \models \delta_j \quad m = l_j\, \langle T_j \rangle \quad j \in I}{(\nu,\ \{\square_i\, l_i\, \langle T_i \rangle\, (\delta_i, \lambda_i).S_i\}_{i \in I}) \xrightarrow{\square_j m} (\nu\, [\lambda_j \mapsto 0]\,,\ S_j)}\ [\texttt{act}]$$

where $\square = \square_j$. By Definition 2.5, $\overline{S} = \{\overline{\square}_i\, l_i \langle T_i \rangle (\delta_i, \lambda_i).\overline{S}_i\}_{i \in I}$. Therefore, as the preconditions (δ_i) in S and $\overline{S}$ are identical, it follows that the premise of rule [act] shown above is equally applicable to $(\nu,\ \overline{S})$. With the only difference being the interactions having opposite directions (sending or receiving), we obtain our thesis.

Case 2. If rule [unfold], then $S = \mu\alpha.S''$ and:

$$\frac{(\nu,\ S''\,[{}^{\mu\alpha.S''}/\alpha]) \xrightarrow{\ell} (\nu',\ S')}{(\nu,\ \mu\alpha.S'') \xrightarrow{\ell} (\nu',\ S')}\ [\texttt{unfold}]$$

where $\ell = \square m$. By Definition 2.5, $\overline{S} = \mu\alpha.\overline{S''}$. The thesis follows by induction:

$$(\nu,\ S''\,[{}^{\mu\alpha.S''}/\alpha]) \xrightarrow{\square m} \quad \Longrightarrow \quad (\nu,\ \overline{S''}\,[{}^{\mu\alpha.\overline{S''}}/\alpha]) \xrightarrow{\overline{\square} m}$$ □

A.5. Configuration Transitions.

Lemma A.19. *Let* $(\nu,\, S)$ *be* well-formed*. Claims A.20 and A.21 both hold.*

Claim A.20. $(\nu,\, S) \xrightarrow{t} (\nu',\, S') \implies \nu' = \nu + t \wedge S \overset{\texttt{unfold}}{\equiv} S'$

Claim A.21. $(\nu,\, S) \xrightarrow{\Box m} (\nu',\, S') \implies \exists \delta \text{ s.t. } \varnothing; \delta \vdash S \wedge \nu \models \delta$
$\wedge\ \exists \delta' \text{ s.t. } \delta' = \delta\,[\lambda \mapsto 0] \wedge \nu' = \nu\,[\lambda \mapsto 0]$
$\wedge\ \exists \gamma \text{ s.t. } \nu' \models \delta' \models \gamma \wedge \varnothing; \gamma \vdash S'$

Proof. We proceed by addressing each claim in turn:

Claim A.20: We proceed by induction on the derivation of the transition $(\nu,\, S) \xrightarrow{t} (\nu',\, S')$ analysing the last rule applied of those given in Eq (2.4):

Case 1. If rule [tick], then the thesis holds as $\nu' = \nu + t$ and $S = S'$.

Case 2. If rule [unfold], then $S = \mu\alpha.S''$ and:

$$\frac{(\nu,\, S''\,[{}^{\mu\alpha.S''}/\alpha]) \xrightarrow{\ell} (\nu',\, S')}{(\nu,\, \mu\alpha.S'') \xrightarrow{\ell} (\nu',\, S')}\ [\texttt{unfold}]$$

where $\ell = t$. By Definition A.6, $\mu\alpha.S'' \overset{\texttt{unfold}}{\equiv} S''\,[{}^{\mu\alpha.S''}/\alpha]$. By Lemma A.13, $(\nu,\, S''\,[{}^{\mu\alpha.S''}/\alpha])$ is *well-formed*. By the induction hypothesis, $S''\,[{}^{\mu\alpha.S''}/\alpha] \overset{\texttt{unfold}}{\equiv} S'$. By the transitivity equation in Definition A.6, $\mu\alpha.S'' \overset{\texttt{unfold}}{\equiv} S'$, as required.

Claim A.21: We proceed by induction on the derivation of the transition $(\nu,\, S) \xrightarrow{\Box m} (\nu',\, S')$, analysing the last rule applied of those given in Eq (2.4):

Case 1. If rule [act], then $S = \left\{\Box_i\, l_i \langle T_i\rangle\,(\delta_i, \lambda_i).S_i\right\}_{i\in I}$ and:

$$\frac{\nu \models \delta_j \quad m = l_j \langle T_j\rangle \quad j \in I}{(\nu,\, \left\{\Box_i\, l_i \langle T_i\rangle\,(\delta_i, \lambda_i).S_i\right\}_{i\in I}) \xrightarrow{\Box_j m} (\nu\,[\lambda_j \mapsto 0]\,,\, S_j)}\ [\texttt{act}]$$

where $\nu' = \nu\,[\lambda_j \mapsto 0]$, $S' = S_j$ and $\delta = \delta_j$. Since $(\nu,\, S)$ is *well-formed*, by Definition 2.6, $\exists\delta$ such that $\nu \models \delta$ and $\varnothing; \delta \vdash S$. By rule [choice]:

$$\frac{\begin{array}{ll} \forall i \in I : A;\ \gamma_i \vdash S_i \ \wedge\ \delta_i\,[\lambda_i \mapsto 0] \models \gamma_i & \text{(feasibility)} \\ \forall i, j \in I : i \neq j \implies \delta_i \wedge \delta_j \models \texttt{false} \vee \Box_i = \Box_j & \text{(mixed-choice)} \\ \forall i \in I : T_i = \left(\delta'',\, S''\right) \implies A;\ \gamma' \vdash S'' \ \wedge\ \delta'' \models \gamma' & \text{(delegation)} \end{array}}{A;\ \downarrow \bigvee_{i\in I} \delta_i \vdash \left\{\Box_i\, l_i \langle T_i\rangle\,(\delta_i, \lambda_i).S_i\right\}_{i\in I}}\ [\texttt{choice}]$$

where $A = \varnothing$. As in Case 1 of Lemma A.17, the set of constraints $\downarrow \bigvee_{i\in I} \delta_i$ requires as minimum the *weak past* of only *one* set of constraints (δ_j) to be satisfied for the entire set of constraints to be satisfied; hence $\delta_j \models \downarrow \bigvee_{i\in I} \delta_i$. It remains to show: $\nu\,[\lambda_j \mapsto 0] \models \delta_j\,[\lambda_j \mapsto 0] \models \gamma$ and $\varnothing; \gamma \vdash S_j$. Clearly, by the (feasibility) premise of rule [choice] it holds that $\delta_j\,[\lambda_j \mapsto 0] \models \gamma$ and $\varnothing; \gamma \vdash S_j$ (as $A = \varnothing$). By the premise of rule [act], it holds that $\nu \models \delta_j$, and therefore it follows that $\nu\,[\lambda_j \mapsto 0] \models \delta_j\,[\lambda_j \mapsto 0]$.

Case 2. If rule [`unfold`], then $S = \mu\alpha.S''$ and:

$$\frac{(\nu,\, S''\,[{}^{\mu\alpha.S''}/_{\alpha}]) \xrightarrow{\ell} (\nu',\, S')}{(\nu,\, \mu\alpha.S'') \xrightarrow{\ell} (\nu',\, S')}\ [\texttt{unfold}]$$

where $\ell = \Box m$. The rest follows Case 2 of Claim A.20 (above). □

Lemma A.22. *Let* $(\nu,\, S)$ *be* well-formed*. Claims A.23 to A.26 all hold.*

Claim A.23. $(\nu,\, S,\, \mathtt{M}) \xrightarrow{!m} (\nu',\, S',\, \mathtt{M}') \implies \mathtt{M}' = \mathtt{M} \wedge (\nu,\, S) \xrightarrow{!m} (\nu',\, S')$

Claim A.24. $(\nu,\, S,\, \mathtt{M}) \xrightarrow{\tau} (\nu',\, S',\, \mathtt{M}') \implies \exists m \text{ s.t. } \mathtt{M} = m; \mathtt{M}' \wedge (\nu,\, S) \xrightarrow{?m} (\nu',\, S')$

Claim A.25. $(\nu,\, S,\, \mathtt{M}) \xrightarrow{?m} (\nu',\, S',\, \mathtt{M}') \implies S' = S \wedge \mathtt{M}' = \mathtt{M}; m \wedge \nu' = \nu$

Claim A.26. $(\nu,\, S,\, \mathtt{M}) \xrightarrow{t} (\nu',\, S',\, \mathtt{M}') \implies S' \overset{\texttt{unfold}}{\equiv} S \wedge \mathtt{M}' = \mathtt{M} \wedge \nu' = \nu + t$

Proof. We proceed addressing each claim in turn, using the rules in Eq (2.5):

Claim A.23: By rule [`send`] the claim holds. By the premise $(\nu,\, S) \xrightarrow{!m} (\nu',\, S')$.

Claim A.24: By rule [`recv`] the claim holds. By the premise $(\nu,\, S) \xrightarrow{?m} (\nu',\, S')$.

Claim A.25: By rule [`que`] the claim holds.

Claim A.26: By rule [`time`] the claim holds. By the premise $\nu' = \nu + t$ via rule [`tick`]. (See Claim A.20 of Lemma A.19.) □

Lemma A.27. *Let* $(\nu,\, S)$ *be* well-formed*. If* $(\nu,\, S) \xrightarrow{\Box m}$ *and* $(\nu,\, S) \xrightarrow{\Box' m'}$ *then* $\Box = \Box'$*.*

Proof. We proceed by induction on the derivation of the transition $(\nu,\, S) \xrightarrow{\Box m}$, analysing the last rule applied of those in Eq (2.4). We only show the case of rule [`act`], as the only other applicable case (rule [`unfold`]) follows by induction hypothesis as in Claims A.20 and A.21 of Lemma A.19. Therefore, if rule [`act`], then $S = \left\{\Box_i\, l_i \langle T_i \rangle (\delta_i, \lambda_i).S_i\right\}_{i \in I}$ and:

$$\frac{\nu \models \delta_j \quad m = l_j \langle T_j \rangle \quad j \in I}{(\nu,\, \left\{\Box_i\, l_i \langle T_i \rangle (\delta_i, \lambda_i).S_i\right\}_{i \in I}) \xrightarrow{\Box_j m} (\nu\,[\lambda_j \mapsto 0]\,,\, S_j)}\ [\texttt{act}] \tag{A.3}$$

where $\Box = \Box_j$ and $m = l_j \langle T_j \rangle$. Since $(\nu,\, S)$ is *well-formed*, by Definition 2.6, $\exists \delta$ such that $\nu \models \delta$ and $\varnothing; \delta \vdash \left\{\Box_i\, l_i \langle T_i \rangle (\delta_i, \lambda_i).S_i\right\}_{i \in I}$. The only possible rule is [`choice`]:

$$\frac{\begin{array}{ll} \forall i \in I : A;\ \gamma_i \vdash S_i \ \wedge\ \delta_i\,[\lambda_i \mapsto 0] \models \gamma_i & \text{(feasibility)} \\ \forall i, j \in I : i \neq j \implies \delta_i \wedge \delta_j \models \texttt{false} \ \vee\ \Box_i = \Box_j & \text{(mixed-choice)} \\ \forall i \in I : T_i = \left(\delta'',\, S''\right) \implies \varnothing;\ \gamma' \vdash S'' \ \wedge\ \delta'' \models \gamma' & \text{(delegation)} \end{array}}{A;\ \downarrow \bigvee_{i \in I} \delta_i \vdash \left\{\Box_i\, l_i \langle T_i \rangle (\delta_i, \lambda_i).S_i\right\}_{i \in I}}\ [\texttt{choice}]$$

As in Lemma A.17, it follows that $\delta_j \models \downarrow \bigvee_{i \in I} \delta_i$. Since $(\nu,\, S) \xrightarrow{\Box' m'}$, we proceed by induction on the derivation of the transition, analysing the last rule applied. Again, we omit the case by rule [`unfold`]. By rule [`act`], it follows similarly to Eq (A.3): for some $k \in I$, $\Box' = \Box_k$, $m = l_k \langle T_k \rangle$ and $\nu \models \delta_k$. In the case where $j = k$, then the thesis coincides with the hypothesis. Otherwise, if $j \neq k$, then by the (mixed-choice) premise of rule [`choice`], $\delta_j \wedge \delta_k \models \texttt{false}$ or $\Box_j = \Box_k$. Since $\nu \models \delta_j$ and $\nu \models \delta_k$, it follows that $\delta_j \wedge \delta_k \not\models \texttt{false}$. Therefore, it must be that $\Box_j = \Box_k$ and we obtain our thesis. □

A.6. **System Transition Preservations.**

Lemma A.28. *Let* (ν_1, S_1) *and* (ν_2, S_2) *be* well-formed*, and* $(\nu_1, S_1, \mathtt{M}_1) \perp (\nu_2, S_2, \mathtt{M}_2)$*. If* $\mathbf{S}_1 \mid \mathbf{S}_2 \rightarrow (\nu_1', S_1', \mathtt{M}_1') \mid (\nu_2', S_2', \mathtt{M}_2')$*, then* (ν_1', S_1') *and* (ν_2', S_2') *are* well-formed.

Proof. We proceed by cases on the rule used in the derivation of the transition:

$$(\nu_1, S_1, \mathtt{M}_1) \mid (\nu_2, S_2, \mathtt{M}_2) \rightarrow (\nu_1', S_1', \mathtt{M}_1') \mid (\nu_2', S_2', \mathtt{M}_2')$$

analysing the last rule applied of those given in Eq (2.6):

Case 1. If rule [wait], then:

$$\frac{(\nu_1, S_1, \mathtt{M}_1) \xrightarrow{t} (\nu_1', S_1', \mathtt{M}_1') \quad (\nu_2, S_2, \mathtt{M}_2) \xrightarrow{t} (\nu_2', S_2', \mathtt{M}_2')}{(\nu_1, S_1, \mathtt{M}_1) \mid (\nu_2, S_2, \mathtt{M}_2) \xrightarrow{t} (\nu_1', S_1', \mathtt{M}_1') \mid (\nu_2', S_2', \mathtt{M}_2')} \; [\mathtt{wait}]$$

Hereafter, we only show the case for $(\nu_1, S_1, \mathtt{M}_1)$, as the transition for $(\nu_2, S_2, \mathtt{M}_2)$ is analogous. We proceed by inner induction on the derivation of the transition $(\nu_1, S_1, \mathtt{M}_1) \xrightarrow{t} (\nu_1', S_1', \mathtt{M}_1')$ analysing the last rule applied. By rule [time]:

$$\frac{\begin{array}{ll} (\nu_1, S_1) \xrightarrow{t} (\nu_1', S_1') & \text{(configuration)} \\ (\nu_1, S_1) \text{ is } \mathtt{fe} \implies (\nu_1', S_1') \text{ is } \mathtt{fe} & \text{(persistency)} \\ \forall t' < t : (\nu_1 + t', S_1, \mathtt{M}_1) \stackrel{\tau}{\not\rightarrow} & \text{(urgency)} \end{array}}{(\nu_1, S_1, \mathtt{M}_1) \xrightarrow{t} (\nu_1', S_1', \mathtt{M}_1)} \; [\mathtt{time}]$$

By inner induction on the derivation of the transition $(\nu_1, S_1) \xrightarrow{t} (\nu_1', S_1')$ analysing the last rule applied of those in Eq (2.4):

i. If rule [tick], then by Claim A.20 of Lemma A.19, $\nu_1' = \nu_1 + t$, $\mathtt{M}_1' = \mathtt{M}_1$ and $S_1' \stackrel{\mathtt{unfold}}{\equiv} S_1$. If $t = 0$ then $\nu_1 = \nu_1 + t$ and therefore, the thesis coincides with the hypothesis. By the (urgency) premise of rule [time], t must be valued such that no message can be received from the queue $\mathtt{M}_1$ via rule [recv]. If $t = 0$, then $\nexists t' < t$, and the (urgency) premise of rule [time] always holds, even if $(\nu_2, S_2, \mathtt{M}_2) \xrightarrow{\tau}$. However, since $\nu_1' = \nu_1$, this is the same as no transition occurring via rule [time]. Otherwise, $t > 0$. Since (ν_1, S_1) is *well-formed*, it follows Lemma A.31, (ν_1, S_1) is also *future-enabled*. Since (ν_1, S_1) is *future-enabled*, it follows the (persistency) premise of rule [time] that (ν_1', S_1') is also *future-enabled*. By Lemma A.17, $(\nu_1 + t, S_1')$ is *well-formed*.

ii. If rule [unfold], then $S_1 = \mu\alpha.S_1''$ and:

$$\frac{(\nu_1, S_1'' [{}^{\mu\alpha.S_1''}/_{\alpha}]) \xrightarrow{\ell} (\nu_1', S_1')}{(\nu_1, \mu\alpha.S_1'') \xrightarrow{\ell} (\nu_1', S_1')} \; [\mathtt{unfold}]$$

where $\ell = t$. Since $(\nu_1, \mu\alpha.S_1'')$ is *well-formed*, $\exists\delta$ such that $\nu_1 \models \delta$ and $\varnothing; \delta \vdash \mu\alpha.S_1''$. By rule [rec]:

$$\frac{\alpha : \delta; \delta \vdash S_1'' [{}^{\mu\alpha.S_1''}/_{\alpha}]}{\varnothing; \delta \vdash \mu\alpha.S_1''} \; [\mathtt{rec}]$$

By Lemma A.9: $\alpha : \delta; \delta \vdash S_1'' [{}^{\mu\alpha.S_1''}/_{\alpha}] \iff \varnothing; \delta \vdash S_1''$. By Definition A.6, it follows $\mu\alpha.S_1'' \stackrel{\mathtt{unfold}}{\equiv} S_1'' [{}^{\mu\alpha.S_1''}/_{\alpha}]$. By Lemma A.12, since $\varnothing; \delta \vdash S_1''$, then $\varnothing; \delta \vdash \mu\alpha.S_1''$.

By Lemma A.13, if $(\nu_1,\, S_1''\,[{}^{\mu\alpha.S_1''}/\alpha])$ is *well-formed*, then $(\nu_1,\, \mu\alpha.S_1'')$ is *well-formed*. The thesis holds by the induction hypothesis. (As in Case 2 in Lemma A.17.)

Case 2. If rule [com-l], then:

$$\frac{(\nu_1,\, S_1,\, \mathtt{M}_1) \xrightarrow{!m} (\nu_1',\, S_1',\, \mathtt{M}_1') \quad (\nu_2,\, S_2,\, \mathtt{M}_2) \xrightarrow{?m} (\nu_2',\, S_2',\, \mathtt{M}_2')}{(\nu_1,\, S_1,\, \mathtt{M}_1) \mid (\nu_2,\, S_2,\, \mathtt{M}_2) \xrightarrow{\tau} (\nu_1',\, S_1',\, \mathtt{M}_1') \mid (\nu_2',\, S_2',\, \mathtt{M}_2')}\ [\mathtt{com\text{-}l}]$$

First, we proceed by cases on the derivation of the transition $(\nu_2,\, S_2,\, \mathtt{M}_2) \xrightarrow{?m} (\nu_2',\, S_2',\, \mathtt{M}_2')$ analysing the last rule applied. By rule [que]:

$$(\nu_2,\, S_2,\, \mathtt{M}_2) \xrightarrow{?m} (\nu_2,\, S_2,\, \mathtt{M}_2; m) \quad [\mathtt{que}]$$

where, as in Claim A.25 of Lemma A.22, $\nu_2' = \nu_2$, $S_2' = S_2$ and $\mathtt{M}_2' = \mathtt{M}_2; m$. Therefore, it follows that $(\nu_2',\, S_2')$ is *well-formed*. Next, by cases on the derivation of the transition $(\nu_1,\, S_1,\, \mathtt{M}_1) \xrightarrow{!m} (\nu_1',\, S_1',\, \mathtt{M}_1')$ analysing the last rule applied. By rule [send]:

$$\frac{(\nu_1,\, S_1) \xrightarrow{!m} (\nu_1',\, S_1')}{(\nu_1,\, S_1,\, \mathtt{M}_1) \xrightarrow{!m} (\nu_1',\, S_1',\, \mathtt{M}_1')}\ [\mathtt{send}]$$

where, as in Claim A.23 of Lemma A.22, $\mathtt{M}_1' = \mathtt{M}_1$ and $(\nu_1,\, S_1) \xrightarrow{!m} (\nu_1',\, S_1')$. As in Claims A.20 and A.21 of Lemma A.19, we proceed to only show the case of rule [act] (as the only other applicable rule is rule [unfold], which follows by induction). Therefore, $S_1 = \{\Box_i\, l_i \langle T_i\rangle\, (\delta_i, \lambda_i).S_i\}_{i\in I}$ and for some $j \in I$, $\nu_1 \models \delta_j$, $m = l_j\,\langle T_j\rangle$, $! = \Box_j$, $\nu_1' = \nu_1\,[\lambda_j \mapsto 0]$ and $S_1' = S_j$. It remains to show that $(\nu_1\,[\lambda_j \mapsto 0]\,,\, S_j)$ is *well-formed*. The thesis follows Claim A.21 of Lemma A.19, since $\nu_1\,[\lambda_j \mapsto 0] \models \delta_j\,[\lambda_j \mapsto 0] \models \gamma_j$ and $\varnothing; \gamma_j \vdash S_j$, for some $j \in I$. By Definition 2.6, $(\nu_1\,[\lambda_j \mapsto 0]\,,\, S_j)$ is *well-formed*.

Case 3. If rule [par-l], then:

$$\frac{(\nu_1,\, S_1,\, \mathtt{M}_1) \xrightarrow{\tau} (\nu_1',\, S_1',\, \mathtt{M}_1')}{(\nu_1,\, S_1,\, \mathtt{M}_1) \mid (\nu_2,\, S_2,\, \mathtt{M}_2) \xrightarrow{\tau} (\nu_1',\, S_1',\, \mathtt{M}_1') \mid (\nu_2,\, S_2,\, \mathtt{M}_2)}\ [\mathtt{par\text{-}l}]$$

where, $\nu_2' = \nu_2$, $S_2' = S_2$ and $\mathtt{M}_2' = \mathtt{M}_2$. It holds that $(\nu_2',\, S_2')$ remains *well-formed*. We proceed by inner induction on the derivation of the transition $(\nu_1,\, S_1,\, \mathtt{M}_1) \xrightarrow{\tau} (\nu_1',\, S_1',\, \mathtt{M}_1')$ analysing the last rule applied of those in Eq (2.5). By rule [recv]:

$$\frac{(\nu_1,\, S_1) \xrightarrow{?m} (\nu_1',\, S_1')}{(\nu_1,\, S_1,\, \mathtt{M}_1) \xrightarrow{\tau} (\nu_1',\, S_1',\, \mathtt{M}_1')}\ [\mathtt{recv}]$$

As in Case 2, the thesis follows Claim A.21 of Lemma A.19 (except $? = \Box_j$).

Case 4. Rules [com-r] and [par-r] are analogous to Cases 2 and 3 respectively. □

Lemma A.29. *Let* $(\nu_1,\, S_1)$ *and* $(\nu_2,\, S_2)$ *be* well-formed*, and* $(\nu_1,\, S_1,\, \mathtt{M}_1) \perp (\nu_2,\, S_2,\, \mathtt{M}_2)$*. If* $(\nu_1,\, S_1,\, \mathtt{M}_1) \mid (\nu_2,\, S_2,\, \mathtt{M}_2) \xrightarrow{t} (\nu_1',\, S_1',\, \mathtt{M}_1') \mid (\nu_2',\, S_2',\, \mathtt{M}_2')$ *and* $t > 0$*, then* $\mathtt{M}_1 = \varnothing = \mathtt{M}_2$*.*

Proof. We proceed by cases on the derivation of the transition $(\nu_1,\, S_1,\, \mathtt{M}_1) \mid (\nu_2,\, S_2,\, \mathtt{M}_2) \xrightarrow{t}$ analysing the last rule applied of those in Eq (2.6). By rule [wait]:

$$\frac{(\nu_1,\, S_1,\, \mathtt{M}_1) \xrightarrow{t} (\nu_1',\, S_1',\, \mathtt{M}_1') \quad (\nu_2,\, S_2,\, \mathtt{M}_2) \xrightarrow{t} (\nu_2',\, S_2',\, \mathtt{M}_2')}{(\nu_1,\, S_1,\, \mathtt{M}_1) \mid (\nu_2,\, S_2,\, \mathtt{M}_2) \xrightarrow{t} (\nu_1',\, S_1',\, \mathtt{M}_1') \mid (\nu_2',\, S_2',\, \mathtt{M}_2')}\ [\mathtt{wait}]$$

where, as in Claim A.26 of Lemma A.22, $\mathtt{M}_1' = \mathtt{M}_1$ and $\mathtt{M}_2' = \mathtt{M}_2$. Hereafter, we only show $(\nu_1,\, S_1,\, \mathtt{M}_1)$ as $(\nu_2,\, S_2,\, \mathtt{M}_2)$ is analogous. By cases on the derivation of the transition $(\nu_1,\, S_1,\, \mathtt{M}_1) \xrightarrow{t} (\nu_1',\, S_1',\, \mathtt{M}_1')$ analysing the last rule applied. By rule [time]:

$$\frac{\begin{array}{ll} (\nu_1,\, S_1) \xrightarrow{t} (\nu_1',\, S_1') & \text{(configuration)} \\ (\nu_1,\, S_1) \text{ is } \mathtt{fe} \implies (\nu_1',\, S_1') \text{ is } \mathtt{fe} & \text{(persistency)} \\ \forall t' < t : (\nu_1 + t',\, S_1,\, \mathtt{M}_1) \xrightarrow{\tau}\!\!\!\!\!/\ \ & \text{(urgency)} \end{array}}{(\nu_1,\, S_1,\, \mathtt{M}_1) \xrightarrow{t} (\nu_1',\, S_1',\, \mathtt{M}_1)}\ [\mathtt{time}]$$

where, as in Claim A.26 of Lemma A.22, $\nu_1' = \nu_1 + t$ and $S_1' \overset{\mathtt{unfold}}{\equiv} S_1$. Suppose by contradiction that one of the queues $\mathtt{M}_1$ were *non-empty*, and $\mathtt{M}_1 = m;\mathtt{M}_1''$. Since $(\nu_1,\, S_1,\, m;\mathtt{M}_1) \perp (\nu_2,\, S_2,\, \mathtt{M}_2)$, by Condition (2) in Definition 2.10:

$$\mathtt{M}_1 = m;\mathtt{M}_1' \implies (\nu_1,\, S_1) \xrightarrow{?m} (\nu_1',\, S_1') \wedge (\nu_1',\, S_1',\, \mathtt{M}_1') \perp (\nu_2,\, S_2,\, \mathtt{M}_2)$$

Such a transition is only viable by rule [recv]. By Claim A.24 of Lemma A.22, it follows that $(\nu_1,\, S_1) \xrightarrow{?m} (\nu_1',\, S_1')$ is made by the premise of rule [recv]. However, by the (urgency) premise of rule [time], $\forall t' < t$ no transitions by rule [recv] are viable – which contradicts with Condition (2) in Definition 2.10. Therefore, it cannot be that either queue is *non-empty* for a transition t, where $t > 0$. For such a transition, both queues must be *empty*. □

Lemma A.30. *Let* $(\nu_1,\, S_1)$ *and* $(\nu_2,\, S_2)$ *be* well-formed*, and* $(\nu_1,\, S_1,\, \mathtt{M}_1) \perp (\nu_2,\, S_2,\, \mathtt{M}_2)$*. If* $(\nu_1,\, S_1,\, \mathtt{M}_1) \mid (\nu_2,\, S_2,\, \mathtt{M}_2) \to (\nu_1',\, S_1',\, \mathtt{M}_1') \mid (\nu_2',\, S_2',\, \mathtt{M}_2')$*, then* $(\nu_1',\, S_1',\, \mathtt{M}_1') \perp (\nu_2',\, S_2',\, \mathtt{M}_2')$*.*

Proof. We proceed by cases on the derivation of the transition:

$$(\nu_1,\, S_1,\, \mathtt{M}_1) \mid (\nu_2,\, S_2,\, \mathtt{M}_2) \to (\nu_1',\, S_1',\, \mathtt{M}_1') \mid (\nu_2',\, S_2',\, \mathtt{M}_2')$$

analysing the last rule applied of those given in Eq (2.6):

Case 1. If rule [wait], then:

$$\frac{(\nu_1,\, S_1,\, \mathtt{M}_1) \xrightarrow{t} (\nu_1',\, S_1',\, \mathtt{M}_1') \quad (\nu_2,\, S_2,\, \mathtt{M}_2) \xrightarrow{t} (\nu_2',\, S_2',\, \mathtt{M}_2')}{(\nu_1,\, S_1,\, \mathtt{M}_1) \mid (\nu_2,\, S_2,\, \mathtt{M}_2) \xrightarrow{t} (\nu_1',\, S_1',\, \mathtt{M}_1') \mid (\nu_2',\, S_2',\, \mathtt{M}_2')}\ [\mathtt{wait}]$$

By Claim A.26 of Lemma A.22, it follows that $\nu_1' = \nu_1 + t$, $S_1' \overset{\mathtt{unfold}}{\equiv} S_1$ and $\mathtt{M}_1' = \mathtt{M}_1$ (and similarly for ν_2', S_2' and $\mathtt{M}_2'$). If $t = 0$ then $\nu_1 = \nu_1 + t$ (and $\nu_2 = \nu_2 + t$) and therefore, the thesis coincides with the hypothesis. Else $t > 0$, and by Lemma A.29, $\mathtt{M}_1 = \varnothing = \mathtt{M}_2$. By Condition (4) of Definition 2.10: $\mathtt{M}_1 = \varnothing = \mathtt{M}_2 \implies S_1 = \overline{S_2} \ \wedge\ \nu_1 = \nu_2$. Since $\nu_1 + t = \nu_2 + t$, it follows that $(\nu_1 + t,\, S_1',\, \mathtt{M}_1) \perp (\nu_2 + t,\, S_2',\, \mathtt{M}_2)$.

Case 2. If rule [com-l], then:

$$\frac{(\nu_1,\, S_1,\, \mathtt{M}_1) \xrightarrow{!m} (\nu_1',\, S_1',\, \mathtt{M}_1') \quad (\nu_2,\, S_2,\, \mathtt{M}_2) \xrightarrow{?m} (\nu_2',\, S_2',\, \mathtt{M}_2')}{(\nu_1,\, S_1,\, \mathtt{M}_1) \mid (\nu_2,\, S_2,\, \mathtt{M}_2) \xrightarrow{\tau} (\nu_1',\, S_1',\, \mathtt{M}_1') \mid (\nu_2',\, S_2',\, \mathtt{M}_2')}\ [\mathtt{com\text{-}l}]$$

First, by induction on the derivation of the transition $(\nu_2,\, S_2,\, \mathtt{M}_2) \xrightarrow{?m} (\nu_2',\, S_2',\, \mathtt{M}_2')$ analysing the last rule applied of those in Eq (2.5). By rule [que]:

$$(\nu_2,\, S_2,\, \mathtt{M}_2) \xrightarrow{?m} (\nu_2,\, S_2,\, \mathtt{M}_2;\, m) \quad [\mathtt{que}]$$

where $\nu_2' = \nu_2$, $S_2' = S_2$ and $\mathtt{M}_2' = \mathtt{M}_2; m$. Next, by induction on the derivation of the transition $(\nu_1,\, S_1,\, \mathtt{M}_1) \xrightarrow{!m} (\nu_1',\, S_1',\, \mathtt{M}_1')$ analysing the last rule applied. By rule [send]:

$$\frac{(\nu_1,\, S_1) \xrightarrow{!m} (\nu_1',\, S_1')}{(\nu_1,\, S_1,\, \mathtt{M}_1) \xrightarrow{!m} (\nu_1',\, S_1',\, \mathtt{M}_1)} \; [\mathtt{send}]$$

where $\mathtt{M}_1' = \mathtt{M}_1$. By inner induction on the derivation of the transition $(\nu_1,\, S_1) \xrightarrow{!m} (\nu_1',\, S_1')$ analysing the last rule applied of those in Eq (2.4). We omit the case of rule [unfold], since following Lemma A.14, $S_1 \overset{\mathtt{unfold}}{\equiv} \{\Box_i\, l_i\, \langle T_i \rangle\, (\delta_i, \lambda_i).S_i\}_{i\in I}$, and by Lemma A.12, $(\nu_1,\, \{\Box_i\, l_i\, \langle T_i \rangle\, (\delta_i, \lambda_i).S_i\}_{i\in I})$ is *well-formed*. Therefore, by rule [act]:

$$\frac{\nu_1 \models \delta_j \quad m = l_j\, \langle T_j \rangle \quad j \in I}{(\nu_1,\, \{\Box_i\, l_i\, \langle T_i \rangle\, (\delta_i, \lambda_i).S_i\}_{i\in I}) \xrightarrow{\Box_j m} (\nu_1\, [\lambda_j \mapsto 0]\, ,\, S_j)} \; [\mathtt{act}]$$

where $\nu_1' = \nu_1\, [\lambda_j \mapsto 0]$, $S_1' = S_j$ and $m = l_j\, \langle T_j \rangle$, for some $j \in I$ such that $\nu_1 \models \delta_j$. If $\mathtt{M}_1$ were *non-empty* (i.e.: $\mathtt{M}_1 = m'; \mathtt{M}_1''$), then by Condition (2) of Definition 2.10, it must be that $(\nu_1,\, S_1) \xrightarrow{?m'}$. However, by Lemma A.27, this is plainly not the case since $(\nu_1,\, S_1) \xrightarrow{!m}$. Therefore, $\mathtt{M}_1 = \varnothing$. We proceed by case analysis on the contents of $\mathtt{M}_2$:

i. If $\mathtt{M}_2 = \varnothing$, then by Condition (4) of Definition 2.10, $S_1 = \overline{S_2}$ and $\nu_1 = \nu_2$. Therefore, by the induction hypothesis:

$$\begin{aligned}(\nu_1,\, \{\Box_i\, l_i\, \langle T_i \rangle\, (\delta_i, \lambda_i).S_i\}_{i\in I},\, \varnothing) \mid (\nu_2,\, \{\overline{\Box}_i\, l_i\langle T_i \rangle(\delta_i, \lambda_i).\overline{S}_i\}_{i\in I},\, \varnothing) \xrightarrow{\tau}& \\ (\nu_1\, [\lambda_j \mapsto 0]\, ,\, S_j,\, \varnothing) \mid (\nu_2,\, \{\overline{\Box}_i\, l_i\langle T_i \rangle(\delta_i, \lambda_i).\overline{S}_i\}_{i\in I},\, m)&\end{aligned}$$

It remains to show that: $(\nu_1\, [\lambda_j \mapsto 0]\, ,\, S_j,\, \varnothing) \perp (\nu_2,\, \{\overline{\Box}_i\, l_i\langle T_i \rangle(\delta_i, \lambda_i).\overline{S}_i\}_{i\in I},\, m)$. Since $\mathtt{M}_2' = m; \mathtt{M}_2''$, then by Condition (3) of Definition 2.10:

$$(\nu_2,\, \{\overline{\Box}_i\, l_i\langle T_i \rangle(\delta_i, \lambda_i).\overline{S}_i\}_{i\in I}) \xrightarrow{?m} (\nu_2'',\, S_2'') \text{ and } (\nu_1\, [\lambda_j \mapsto 0]\, ,\, S_j,\, \varnothing) \perp (\nu_2'',\, S_2'',\, \mathtt{M}_2'') \tag{A.4}$$

The thesis follows Lemma A.18, since $(\nu_1,\, S_1) \xrightarrow{!m}$ and $\nu_1 = \nu_2$, then $(\nu_2,\, \overline{S_2}) \xrightarrow{?m}$.

ii. If $\mathtt{M}_2 = m'; \mathtt{M}_2''$, then $\mathtt{M}_2' = m'; \mathtt{M}_2''; m$ and therefore, by Condition (3) of Definition 2.10, it follows Eq (A.4). The thesis follows by the induction hypothesis:

$$(\nu_1,\, \{\Box_i\, l_i\, \langle T_i \rangle\, (\delta_i, \lambda_i).S_i\}_{i\in I},\, \varnothing) \mid (\nu_2,\, S_2,\, \mathtt{M}_2) \xrightarrow{\tau} (\nu_1\, [\lambda_j \mapsto 0]\, ,\, S_j,\, \varnothing) \mid (\nu_2,\, S_2,\, m'; \mathtt{M}_2''; m)$$

If $\mathtt{M}_2'' = \varnothing$, then it follows Case 2.i. Otherwise, it follows this case.

Case 3. If rule [par-l], then by Claim A.24 of Lemma A.22 it follows that $\mathtt{M}_1 = m; \mathtt{M}_1'$, and by Condition (2) in Definition 2.10, we obtain our thesis.

Case 4. If rule [com-r] or rule [par-r], then it follows Cases 2 and 3 respectively. □

A.7. **System Progress.**

Lemma A.31. *Let* (ν_1, S_1) *and* (ν_2, S_2) *be* well-formed*. If* $(\nu_1, S_1, \mathtt{M}_1) \perp (\nu_2, S_2, \mathtt{M}_2)$*, then* $(\nu_1, S_1, \mathtt{M}_1)$ *and* $(\nu_2, S_2, \mathtt{M}_2)$ *are* final*, or* $\exists t : (\nu_1, S_1, \mathtt{M}_1) \mid (\nu_2, S_2, \mathtt{M}_2) \xrightarrow{t\ \tau}$.

Proof. Since both (ν_1, S_1) and (ν_2, S_2) are *well-formed*, by Lemma A.15, both (ν_1, S_1) and (ν_2, S_2) are *live*. By Definition 2.8, $(\nu_1, S_1, \mathtt{M}_1)$ is *final* if $S_1 \overset{\mathtt{unfold}}{\equiv} \mathtt{end}$ (and similarly for S_2). By Definition A.2 if $S_1 \overset{\mathtt{unfold}}{\not\equiv} \mathtt{end}$, then (ν_1, S_1) is *future-enabled* (and similarly for (ν_2, S_2)). By Definition 2.3, $\exists t'$ such that $(\nu_1, S_1) \xrightarrow{t'\ \Box m}$ (and similarly for (ν_2, S_2)). Therefore, we have obtained our thesis as it holds that if, $(\nu_1, S_1, \mathtt{M}_1)$ is *not* final the by Definition A.2, it *must* be *future-enabled* by Definition 2.3, which coincides with the hypothesis. ☐

Lemma A.32. *If* (ν_1, S_1) *and* (ν_2, S_2) *are* well-formed*, and* $(\nu_1, S_1, \mathtt{M}_1) \perp (\nu_2, S_2, \mathtt{M}_2)$ *and* $(\nu_1, S_1, \mathtt{M}_1) \mid (\nu_2, S_2, \mathtt{M}_2) \longrightarrow^* (\nu_1', S_1', \mathtt{M}_1') \mid (\nu_2', S_2', \mathtt{M}_2')$*, then* $(\nu_1', S_1', \mathtt{M}_1') \perp (\nu_2', S_2', \mathtt{M}_2')$ *and* (ν_1', S_1') *and* (ν_2', S_2') *are* well-formed.

Proof. We proceed by induction on the length of $(\longrightarrow^*)$. The base case is trivial, as $(\nu_1, S_1, \mathtt{M}_1) \mid (\nu_2, S_2, \mathtt{M}_2) \longrightarrow^0 (\nu_1, S_1, \mathtt{M}_1) \mid (\nu_2, S_2, \mathtt{M}_2)$ and hence the thesis coincides with the hypothesis. For the induction case, suppose:

$$(\nu_1, S_1, \mathtt{M}_1) \mid (\nu_2, S_2, \mathtt{M}_2) \longrightarrow^n (\nu_1'', S_1'', \mathtt{M}_1'') \mid (\nu_2'', S_2'', \mathtt{M}_2'') \longrightarrow (\nu_1', S_1', \mathtt{M}_1') \mid (\nu_2', S_2', \mathtt{M}_2')$$

By the induction hypothesis: $(\nu_1'', S_1'', \mathtt{M}_1'') \perp (\nu_2'', S_2'', \mathtt{M}_2'')$, and both (ν_1'', S_1'') and (ν_2'', S_2'') are *well-formed*. It remains for us to show that both (ν_1', S_1') and (ν_2', S_2') are *well-formed*, and $(\nu_1', S_1', \mathtt{M}_1') \perp (\nu_2', S_2', \mathtt{M}_2')$. By Lemmas A.28 and A.30, we obtain our thesis. ☐

Lemma A.33. *The following holds:*

$$\forall \nu, S : (\nu, S) \textit{ is } \text{well-formed} \implies (\nu, S, \varnothing) \mid (\nu, \overline{S}, \varnothing) \textit{ satisfies progress.}$$

Proof. Since (ν, S) is *well-formed*, by Definition 2.5 $(\nu, \overline{S})$ is *well-formed*. By Definition 2.10, $(\nu, S, \varnothing) \perp (\nu, \overline{S}, \varnothing)$. If $S = \mathtt{end}$ then $\overline{S} = \mathtt{end}$ and therefore, by Definition 2.8, the thesis coincides with the hypothesis. Otherwise, $(\nu, S, \varnothing) \mid (\nu, \overline{S}, \varnothing)$ *satisfies progress* if, for all $\mathbf{S}' \mid \mathbf{S}''$ reachable from $(\nu, S, \varnothing) \mid (\nu, \overline{S}, \varnothing)$ then, either $\mathbf{S}'$ and $\mathbf{S}''$ are both *final*, or $\exists t$ such that: $\mathbf{S}' \mid \mathbf{S}'' \xrightarrow{t\ \tau}$. By Lemma A.32, any transition made by $(\nu, S, \varnothing) \mid (\nu, \overline{S}, \varnothing)$ will preserve both *well-formedness* and *compatible*. By Lemma A.31, we obtain our thesis. ☐

Theorem 2.9 (Progress of Systems)**.** *If* S *is* well-formed *against* ν_0*, then:*

$$(\nu_0, S, \varnothing) \mid (\nu_0, \overline{S}, \varnothing) \text{ satisfies progress.}$$

Proof. The thesis follows immediately from Lemma A.33. ☐

As typical for binary systems, our proof of progress builds upon a notion of *duality* between participants. Since communication is asynchronous, each party may break duality with their co-party. Compatibility allows the duality of participants to be broken, only if, doing so does not violate *communication safety*. More specifically, compatibility requires that any message received by a queue is expected (i.e., able to be received), and that the resulting configurations are still *compatible*. In practice, using compatibility allows each party to behave and progress independently, regardless the state of the other party, while retaining the essence of *duality*, and guaranteeing that all messages will eventually be received.

Appendix B. Proof of Subject Reduction

B.1. Auxiliary Definitions, Figures & Assumptions.

Definition B.1 (Free-queues of a Process)**.** The function $\texttt{fq}\,(P)$ returns the set of *free-queues* in a given process P, defined inductively below:

$$\texttt{fq}\,(P) = \begin{cases} \{pq\} & \text{if } P = pq : h \\ \texttt{fq}\,(P')\setminus\{pq, qp\} & \text{if } P = (\nu pq)\ P' \\ \texttt{fq}\,(P') & \text{if } P \in \left\{\begin{matrix} p \triangleleft l\,(v)P',\ \texttt{def}\ X(\vec{v};\vec{r}) = P\ \texttt{in}\ P', \\ \texttt{set}\,ⓧ.P',\ \texttt{delay}\,(d)\,.P',\ \texttt{delay}\,(t)\,.P' \end{matrix}\right\} \\ \bigcup_{i\in I} \texttt{fq}\,(P_i) & \text{if } P = p^e \triangleright \left\{l_i(v_i) : P_i\right\}_{i\in I} \\ \texttt{fq}\,(P') \cup \texttt{fq}\,(Q) & \text{if } P \in \left\{P' \mid Q,\ \texttt{if}\ c\ \texttt{then}\ P'\ \texttt{else}\ Q\right\} \\ \texttt{fq}\left(p^{\diamond n} \triangleright \left\{l_i(v_i) : P_i\right\}_{i\in I}\right) \cup \texttt{fq}\,(Q) & \text{if } P = p^{\diamond n} \triangleright \left\{l_i(v_i) : P_i\right\}_{i\in I}\ \texttt{after}\ Q \\ \varnothing & \text{if } P = \texttt{0},\ X\langle\vec{v};\vec{r}\rangle \end{cases}$$

Definition B.2 (Free-timers of a Process)**.** The function $\texttt{ft}\,(P)$ returns the set of *free timers* in a given process P, defined inductively below:

$$\texttt{ft}\,(P) = \begin{cases} \{ⓧ\} \cup \texttt{ft}\,(P') & \text{if } P = \texttt{set}\,ⓧ.P' \\ \{ⓧ\} \cup \texttt{ft}\,(P') \cup \texttt{ft}\,(Q) & \text{if } P = \texttt{if}\ c\ \texttt{then}\ P'\ \texttt{else}\ Q \text{ and } ⓧ \in c \\ \texttt{ft}\,(P') & \text{if } P \in \left\{\begin{matrix} p \triangleleft l\,(v)P',\ \texttt{def}\ X(\vec{v};\vec{r}) = P\ \texttt{in}\ P', \\ \texttt{delay}\,(d)\,.P',\ \texttt{delay}\,(t)\,.P',\ (\nu pq)\ P' \end{matrix}\right\} \\ \bigcup_{i\in I} \texttt{ft}\,(P_i) & \text{if } P = p^e \triangleright \left\{l_i(v_i) : P_i\right\}_{i\in I} \\ \texttt{ft}\,(P') \cup \texttt{ft}\,(Q) & \text{if } P = P' \mid Q \\ \texttt{ft}\left(p^{\diamond n} \triangleright \left\{l_i(v_i) : P_i\right\}_{i\in I}\right) \cup \texttt{ft}\,(Q) & \text{if } P = p^{\diamond n} \triangleright \left\{l_i(v_i) : P_i\right\}_{i\in I}\ \texttt{after}\ Q \\ \varnothing & \text{if } P = \texttt{0},\ X\langle\vec{v};\vec{r}\rangle,\ pq : h \end{cases}$$

Definition B.3 (Free-names of a Process)**.** The function $\texttt{fn}\,(P)$ returns the set of *free-names* in a given process P, defined inductively below:

$$\texttt{fn}\,(P) = \begin{cases} \{pq\} \cup \texttt{fn}\,(P') & \text{if } P = (\nu pq)\ P' \\ \texttt{fn}\,(P') & \text{if } P \in \left\{\begin{matrix} p \triangleleft l\,(v)P',\ \texttt{def}\ X(\vec{v};\vec{r}) = P\ \texttt{in}\ P', \\ \texttt{set}\,ⓧ.P',\ \texttt{delay}\,(d)\,.P',\ \texttt{delay}\,(t)\,.P' \end{matrix}\right\} \\ \bigcup_{i\in I} \texttt{fn}\,(P_i) & \text{if } P = p^e \triangleright \left\{l_i(v_i) : P_i\right\}_{i\in I} \\ \texttt{fn}\,(P') \cup \texttt{fn}\,(Q) & \text{if } P \in \left\{P' \mid Q,\ \texttt{if}\ c\ \texttt{then}\ P'\ \texttt{else}\ Q\right\} \\ \texttt{fn}\left(p^{\diamond n} \triangleright \left\{l_i(v_i) : P_i\right\}_{i\in I}\right) \cup \texttt{fn}\,(Q) & \text{if } P = p^{\diamond n} \triangleright \left\{l_i(v_i) : P_i\right\}_{i\in I}\ \texttt{after}\ Q \\ \varnothing & \text{if } P = \texttt{0},\ X\langle\vec{v};\vec{r}\rangle,\ pq : h \end{cases}$$

Definition B.4 (Free process variables of a Process)**.** The function $\mathtt{fpv}\,(P)$ returns the set of *free process variables* in a given process P, defined inductively below:

$$\mathtt{fpv}\,(P) = \begin{cases} \{X\} & \text{if } P = X\langle\vec{v};\vec{r}\rangle \\ \{X\} \cup \mathtt{fpv}\,(P') & \text{if } P = \mathtt{def}\ X(\vec{v};\vec{r}) = P\ \mathtt{in}\ P' \\ \mathtt{fpv}\,(P') & \text{if } P \in \begin{Bmatrix} p \lhd l\,(v)P',\ (\nu pq)\ P',\ \mathtt{set}\,ⓧ.P', \\ \mathtt{delay}\,(d)\,.P',\ \mathtt{delay}\,(t)\,.P' \end{Bmatrix} \\ \bigcup_{i\in I} \mathtt{fpv}\,(P_i) & \text{if } P = p^e \rhd \left\{l_i(v_i) : P_i\right\}_{i\in I} \\ \mathtt{fpv}\,(P') \cup \mathtt{fpv}\,(Q) & \text{if } P \in \left\{P' \mid Q,\ \mathtt{if}\ c\ \mathtt{then}\ P'\ \mathtt{else}\ Q\right\} \\ \mathtt{fpv}\left(p^{\diamond n} \rhd \left\{l_i(v_i) : P_i\right\}_{i\in I}\right) \cup \mathtt{fpv}\,(Q) & \text{if } P = p^{\diamond n} \rhd \left\{l_i(v_i) : P_i\right\}_{i\in I}\ \mathtt{after}\ Q \\ \varnothing & \text{if } P = \mathtt{0},\ pq : h \end{cases}$$

B.1.1. *Session Environments.*

Definition B.5 (Live Δ)**.** Δ is *live* if, for all $p \in \mathrm{dom}(\Delta)$ such that $\Delta = \Delta', p : (\nu,\, S) \implies (\nu,\, S)$ is *live* by Definition A.2.

Definition B.6 (Delayable Δ)**.** A session environment Δ is *delayable* if:

$$\forall qp \in \mathrm{dom}(\Delta) : \Delta(qp) \neq \varnothing \implies p \notin \mathrm{dom}(\Delta)$$

B.1.2. *Session Environment Reduction.* See Figure 7.

$$\frac{\Delta_1 \longrightarrow \Delta_1'}{\Delta_1,\ \Delta_2 \longrightarrow \Delta_1',\ \Delta_2}\,[\Delta\text{-}\mathtt{L}] \qquad \frac{(\nu,\, S,\, l\,\langle T\rangle\,;\mathtt{M}) \xrightarrow{\tau} (\nu',\, S',\, \mathtt{M})}{p : (\nu,\, S),\ qp : l\,\langle T\rangle\,;\mathtt{M} \longrightarrow p : (\nu',\, S'),\ qp : \mathtt{M}}\,[\Delta\text{-}\mathtt{Recv}]$$

$$\frac{\Delta_2 \longrightarrow \Delta_2'}{\Delta_1,\ \Delta_2 \longrightarrow \Delta_1,\ \Delta_2'}\,[\Delta\text{-}\mathtt{R}] \qquad \frac{(\nu,\, S) \xrightarrow{!m} (\nu',\, S')}{p : (\nu,\, S),\ pq : \mathtt{M}, \longrightarrow p : (\nu',\, S'),\ pq : \mathtt{M}; m}\,[\Delta\text{-}\mathtt{Send}]$$

FIGURE 7. Reduction for Session Environments

B.2. **Delayable Processes.**

Lemma B.7. *If* $\Phi_t\,(P)$ *is defined, then* $\mathtt{Wait}\,(P) \cap \mathtt{NEQ}\,(P) = \varnothing$.

Proof. The hypothesis holds by the definition of $\Phi_t\,(P)$ in Definition 3.4. □

B.3. **Well-formed Δ.**

Lemma B.8. *Let* Δ *be* well-formed*. If* Δ *is* balanced *and* $\Delta \longrightarrow \Delta'$*, then* Δ' *is* balanced*. Furthermore, if* Δ *is* fully-balanced*, then* Δ' *is* fully-balanced.

Proof. We proceed by induction on the derivation of the reduction $\Delta \longrightarrow \Delta'$, analysing the last rule applied of those given in Figure 7:

Case 1. If rule [Δ-`Send`], then $\Delta = p : (\nu,\, S), pq : \mathtt{M}$ and:

$$\frac{(\nu,\, S) \xrightarrow{!l\langle T\rangle} (\nu',\, S')}{p : (\nu,\, S), pq : \mathtt{M} \;\longrightarrow\; p : (\nu',\, S'), pq : \mathtt{M}; l\,\langle T\rangle}\ [\Delta\text{-}\mathtt{Send}]$$

We have that $p : (\nu',\, S'), pq : \mathtt{M}; l\,\langle T\rangle$ is *balanced*, as every condition of Definition 6.1 holds trivially. Regarding the furthermore part, it holds trivially as Δ is not *fully-balanced*.

Case 2. If rule [Δ-`Recv`], then $\Delta = p : (\nu,\, S), qp : l\,\langle T\rangle; \mathtt{M}$ and:

$$\frac{(\nu,\, S,\, l\,\langle T\rangle; \mathtt{M}) \xrightarrow{?l\langle T\rangle} (\nu',\, S',\, \mathtt{M}')}{p : (\nu,\, S), qp : l\,\langle T\rangle; \mathtt{M} \;\longrightarrow\; p : (\nu',\, S'), qp : \mathtt{M}}\ [\Delta\text{-}\mathtt{Recv}]$$

By rule [`recv`] of Eq (2.5):

$$\frac{(\nu,\, S) \xrightarrow{?m} (\nu',\, S')}{(\nu,\, S,\, m;\, \mathtt{M}) \xrightarrow{\tau} (\nu',\, S',\, \mathtt{M})}\ [\mathtt{recv}]$$

By the premise of the above, thanks to Case (1) of Definition 6.1, it follows that $(\nu',\, S',\, \mathtt{M})$ is *balanced*, as required. The furthermore case holds trivially.

Case 3. If rule [Δ-`L`] then $\Delta = \Delta_1,\ \Delta_2$ and $\Delta' = \Delta_1',\ \Delta_2$ and:

$$\frac{\Delta_1 \;\longrightarrow\; \Delta_1'}{\Delta_1,\ \Delta_2 \;\longrightarrow\; \Delta_1',\ \Delta_2}\ [\Delta\text{-}\mathtt{L}]$$

It is easy to see that Δ_1 and Δ_2 are both balanced. By the induction hypothesis also Δ_1' is balanced. Notice that the composition of balanced environments is not necessarily balanced. So we need to show that all the items in Definition 6.1 hold for Δ_1'. We show only for Case (1) of Definition 6.1, as the other cases are similar. So, suppose: $\Delta' = \Delta'', p : (\nu,\, S), qp : m; \mathtt{M}$, then we have to show: $(\nu,\, S) \xrightarrow{?m} (\nu',\, S')$ and $\Delta', p : (\nu',\, S'), qp : \mathtt{M} \in \mathrm{Bal}$. The interesting cases are:

- $\Delta_1'(p) = (\nu,\, S)$, $\Delta_2(qp) = m; \mathtt{M}$ and $\Delta_1(p) \neq \Delta_1'(p)$. We argue that this case is impossible. So suppose, by contradiction, it is not the case. Then, since queue qp did not change during the transition, it must be $\Delta_1(p) = (\nu_0, S_0)$ for some ν_0, S_0 such that: $(\nu_0, S_0) \xrightarrow{!m'} (\nu,\, S)$ and $\Delta_1(pq) = \mathtt{M}'$ and $\Delta_1'(pq) = \mathtt{M}'; m'$. By Lemma A.27, $(\nu_0, S_0) \overset{?m}{\not\longrightarrow}$ and hence Δ was not balanced in the first place: contradiction.
- $\Delta_2(p) = (\nu,\, S)$, $\Delta_1'(qp) = m; \mathtt{M}$ and $\Delta_1(qp) \neq \Delta_1'(qp)$. Then $\mathtt{M} = \mathtt{M}'; m'$ and $\Delta_1(q) = (\nu_0, S_0)$, $\Delta_1'(q) = (\nu_0', S_0')$ and $\Delta_1(qp) = m; \mathtt{M}'$ with: $(\nu_0, S_0) \xrightarrow{!m'} (\nu_0', S_0')$. By Case (2) of Definition 6.1, for some $\mathtt{M}_0$: $(\nu, S, \mathtt{M}) \perp (\nu_0, S_0, \mathtt{M}_0)$. by rule [`com-r`] in Eq (2.6):

$$(\nu, S, m; \mathtt{M}') \mid (\nu_0, S_0, \mathtt{M}_0) \xrightarrow{\tau} (\nu, S, m; \mathtt{M}) \mid (\nu_0', S_0', \mathtt{M}_0)$$

By Lemma A.30:

$$(\nu, S, m; \mathtt{M}) \perp (\nu_0, S_0, \mathtt{M}_0)$$

Hence $(\nu, S) \xrightarrow{?m} (\nu', S')$ for some (ν', S') such that $(\nu', S', \mathtt{M}) \perp (\nu'_0, S'_0, \mathtt{M}_0)$. It remains to show that $\Delta', p : (\nu',\ S'), qp : \mathtt{M}$ is *balanced* by Definition 6.1. For Cases (2) and (3) of Definition 6.1 it surely holds. Regarding Case (1) of Definition 6.1, it follows by a simple induction on the length of $\mathtt{M}$. □

B.4. **Well-typed Processes.**

Lemma B.9 (Inversion Lemma)**.** *Let* Δ *be* well-formed*. Claims B.10 to B.24 each hold.*

Claim B.10. If $\Gamma; \theta \vdash P \mid Q \blacktriangleright \Delta$, then $\Delta = (\Delta_1, \Delta_2)$ and $\theta = (\theta_1, \theta_2)$ and $\Gamma; \theta_1 \vdash P \blacktriangleright \Delta_1$ and $\Gamma; \theta_2 \vdash Q \blacktriangleright \Delta_2$.

Claim B.11. If $\Gamma; \theta \vdash (\nu pq)\ P \blacktriangleright \Delta$, then $\Gamma; \theta \vdash P \blacktriangleright \Delta, p : (\nu_1,\ S_1), qp : \mathtt{M}_2, q : (\nu_2,\ S_2), pq : \mathtt{M}_2$ and $(\nu_1,\ S_1,\ \mathtt{M}_1) \perp (\nu_2,\ S_2,\ \mathtt{M}_2)$ and both $(\nu_1,\ S_1)$ and $(\nu_2,\ S_2)$ are *well-formed*.

Claim B.12. If $\Gamma; \theta \vdash \mathtt{def}\ X(\vec{v}; \vec{r}) = P\ \mathtt{in}\ Q \blacktriangleright \Delta$, then $\Gamma, X : (\vec{T}; \boldsymbol{\theta}; \boldsymbol{\Delta}); \theta \vdash Q \blacktriangleright \Delta$ and:

$$\forall (\vec{\nu},\ \vec{S}) \in \boldsymbol{\Delta}, \theta' \in \boldsymbol{\theta} : \quad \Gamma, \vec{v} : \vec{T}, X : (\vec{T}; \boldsymbol{\theta}; \boldsymbol{\Delta}); \theta' \vdash P \blacktriangleright \vec{r} : (\vec{\nu},\ \vec{S})$$

Claim B.13. If $\Gamma, X : (\vec{T}; \boldsymbol{\theta}; \boldsymbol{\Delta}); \theta \vdash \mathtt{def}\ X(\vec{v}; \vec{r}) = P\ \mathtt{in}\ P' \mid Q \blacktriangleright \Delta$, then $\Delta = (\Delta_1, \Delta_2)$, $\theta = (\theta_1, \theta_2)$ and:

- $\Gamma, X : (\vec{T}; \boldsymbol{\theta}; \boldsymbol{\Delta}); \theta_1 \vdash \mathtt{def}\ X(\vec{v}; \vec{r}) = P\ \mathtt{in}\ P' \blacktriangleright \Delta_1$.
- $\Gamma, X : (\vec{T}; \boldsymbol{\theta}; \boldsymbol{\Delta}); \theta_2 \vdash \mathtt{def}\ X(\vec{v}; \vec{r}) = P\ \mathtt{in}\ Q \blacktriangleright \Delta_2$.

Claim B.14. If $\Gamma; \theta \vdash X\langle \vec{v}; \vec{r} \rangle \blacktriangleright \Delta$, then $\Gamma = \Gamma',\ X : (\vec{T}; \boldsymbol{\theta}; \boldsymbol{\Delta})$ and:

$$\forall i : \Gamma' \vdash \vec{v}_i : \vec{T}_i \quad \text{and} \quad \theta \in \boldsymbol{\theta} \quad \text{and} \quad \Delta = \vec{r} : (\vec{\nu}, \vec{S}) \in \boldsymbol{\Delta}$$

Claim B.15. If $\Gamma; \theta \vdash p \triangleleft l\,(v).P \blacktriangleright \Delta$, then $\Delta = \Delta', p : (\nu,\ \{\Box_i\, l_i \langle T_i \rangle (\delta_i, \lambda_i).S_i\}_{i \in I})$ and $\mathtt{NEQ}\,(\Delta) = \varnothing$ and $\exists j \in I$ such that $\nu \models \delta_j$ and:

- T_j *base-type* $\implies \Gamma; \theta \vdash P \blacktriangleright \Delta', p : (\nu\,[\lambda_j \mapsto 0]\,,\ S_j)$ and $\Gamma \vdash v : T_j$.
- $T_j = (\delta', S') \implies \Delta' = \Delta'', v : (\nu',\ S')$ and $\nu' \models \delta'$ and $\Gamma; \theta \vdash P \blacktriangleright \Delta'', p : (\nu\,[\lambda_j \mapsto 0]\,,\ S_j)$.

Claim B.16. If $\Gamma; \theta \vdash p^e \triangleright l(v) : P \blacktriangleright \Delta$, then $\Delta = \Delta', p : (\nu,\ ?l\langle T \rangle\,(\delta, \lambda)\,.S)$ and Δ' not e-reading and:

- $\forall t$ such that $\nu + t \models \delta \iff t \in e$
- $\forall t \in e$:
  - $T = (\delta',\ S') \implies \Gamma; \theta + t \vdash P \blacktriangleright \Delta' + t, p : (\nu + t\,[\lambda \mapsto 0]\,,\ S), v : (\nu',\ S')$ and $\nu' \models \delta'$.
  - T *base-type* $\implies \Gamma, v : T; \theta + t \vdash P \blacktriangleright \Delta' + t, p : (\nu + t\,[\lambda \mapsto 0]\,,\ S)$.

Claim B.17. If $\Gamma; \theta \vdash p^e \triangleright \{l_i(v_i) : P_i\}_{i \in I} \blacktriangleright \Delta$, then $\Delta = \Delta', p : (\nu,\ \{\Box_j\, l_j \langle T_j \rangle (\delta_j, \lambda_j).S_j\}_{j \in J})$ and $\neg(|J| = |I| = 1)$ and, $\forall j \in J$ such that $\nu \models \delta_j$ implies:

$$\Box_j = ? \quad \text{and} \quad \Big(\exists i \in I : \Gamma; \theta \vdash p^e \triangleright l_i(v_i) : P_i \blacktriangleright \Delta', p : (\nu,\ \Box_j\, l_j \langle T_j \rangle (\delta_j, \lambda_j).S_j) \text{ and } l_i = l_j\Big)$$

Claim B.18. If $\Gamma; \theta \vdash p^{\diamond n} \triangleright \{l_i(v_i) : P_i\}_{i \in I}\ \mathtt{after}\ Q \blacktriangleright \Delta$, then $\Delta = \Delta', p : (\nu,\ \{\mathtt{C}_j\}_{j \in J})$ and:

- $\Gamma; \theta \vdash p^{\diamond n} \triangleright \{l_i(v_i) : P_i\}_{i \in I} \blacktriangleright \Delta',\ p : (\nu,\ \{\mathtt{C}_j\}_{j \in J})$.

- $\Gamma; \theta + n \vdash Q \blacktriangleright \Delta' + n,\ p : (\nu + n,\ \{\mathtt{C}_j\}_{j \in J})$.

Claim B.19. If $\Gamma; \theta \vdash \mathtt{set}\,ⓧ.P \blacktriangleright \Delta$, then $\theta = \theta', ⓧ : t$ and $\Gamma; \theta', ⓧ : 0 \vdash P \blacktriangleright \Delta$.

Claim B.20. If $\Gamma; \theta \vdash \mathtt{delay}\,(d)\,.P \blacktriangleright \Delta$, then $\forall t \in d : \Gamma; \theta \vdash \mathtt{delay}\,(t)\,.P \blacktriangleright \Delta$.

Claim B.21. If $\Gamma; \theta \vdash \mathtt{delay}\,(t)\,.P \blacktriangleright \Delta$, then Δ not t-reading and $\Gamma; \theta + t \vdash P \blacktriangleright \Delta + t$.

Claim B.22. If $\Gamma; \theta \vdash \mathtt{if}\,(c)\,\mathtt{then}\ P\ \mathtt{else}\ Q \blacktriangleright \Delta$, then:
- $\theta \models c \implies \Gamma; \theta \vdash P \blacktriangleright \Delta$,
- $\theta \not\models c \implies \Gamma; \theta \vdash Q \blacktriangleright \Delta$.

Claim B.23. If $\Gamma; \theta \vdash qp : \varnothing \blacktriangleright \Delta$, then $\Delta = \Delta', qp : \varnothing$.

Claim B.24. If $\Gamma; \theta \vdash qp : lv \cdot h \blacktriangleright \Delta$, then $\Delta = \Delta', qp : l\,\langle T \rangle\,; \mathtt{M}$ and $\mathtt{NEQ}\,(\Delta') = \varnothing$ and:
- $T = \left(\delta',\ S'\right) \implies \Delta' = \Delta'', q : (\nu',\ S')$ and $\Gamma; \theta \vdash qp : h \blacktriangleright \Delta'', qp : \mathtt{M}$ and $\nu' \models \delta'$.
- T *base-type* $\implies \Gamma \vdash v : T$ and $\Gamma; \theta \vdash qp : h \blacktriangleright \Delta', qp : \mathtt{M}$.

Proof. As standard, it follows by induction on each derivation, using the *type checking* rules given in Figures 5 and 6. □

B.4.1. *Well-typed Enabled Actions.*

Lemma B.25. *Let* $\Delta, p : (\nu,\ S)$ *be* well-formed. *If* $\Gamma; \theta \vdash P \blacktriangleright \Delta, p : (\nu,\ S)$ *and* $(\nu,\ S) \xrightarrow{?m}$ *then* $p \in \mathtt{Wait}\,(P)$.

Proof. By induction on the derivation of the transition $(\nu,\ S) \xrightarrow{?m}$, analysing the last rule applied of those given in Eq (2.4). The only applicable rules are [act] and [unfold]. As in Lemma A.27, we only show rule [act]:

$$\frac{\nu \models \delta_j \quad m = l_j\,\langle T_j \rangle \quad j \in I}{(\nu,\ \left\{\Box_i\, l_i\,\langle T_i \rangle\,(\delta_i, \lambda_i).S_i\right\}_{i \in I}) \xrightarrow{\Box_j m} (\nu\,[\lambda_j \mapsto 0]\,,\ S_j)}\ [\mathtt{act}]$$

where $S = \left\{\Box_i\, l_i\,\langle T_i \rangle\,(\delta_i, \lambda_i).S_i\right\}_{i \in I}$. By inspecting Figure 3 for when $p \in \mathtt{Wait}\,(P)$, it remains for us to show that P is structured as follows:

$$P \in \left\{\ p^e \triangleright \left\{l_j(v_j) : P_j\right\}_{j \in J},\ \ p^{\diamond n} \triangleright \left\{l_j(v_j) : P_j\right\}_{j \in J}\ \mathtt{after}\ Q\ \right\}$$

We proceed by induction on the derivation of $\Gamma; \theta \vdash P \blacktriangleright \Delta, p : (\nu,\ S)$, analysing the last rule applied of those given in Figures 5 and 6:

Case 1. If rule [Branch], then $P = p^e \triangleright \left\{l_j(v_j) : P_j\right\}_{j \in J}$ and we obtain our thesis.

Case 2. If rule [Timeout], then $P = p^{\diamond n} \triangleright \left\{l_j(v_j) : P_j\right\}_{j \in J}$ after Q.

Case 3. If rule [VRecv], then $P = p^e \triangleright l(v) : P'$. (The same holds for rule [DRecv].)

Case 4. If rule [VSend], then $P = p \triangleleft l\,(v).P'$. Clearly $p \notin \mathtt{Wait}\,(P)$, and it remains for us to prove that this case is not applicable. By rule [VSend]:

$$\frac{\begin{array}{c}\exists j \in I : (l = l_j) \wedge (\nu \models \delta_j) \wedge (T_j\ \textit{base-type}) \wedge (\Gamma \vdash v : T_j) \wedge \\ (\Box_j = !) \wedge \Gamma; \theta \vdash P \blacktriangleright \Delta,\ p : (\nu\,[\lambda_j \mapsto 0]\,,\ S_j)\end{array}}{\Gamma; \theta \vdash p \triangleleft l\,(v).P' \blacktriangleright \Delta, p : (\nu,\ \left\{\Box_i\, l_i\,\langle T_i \rangle\,(\delta_i, \lambda_i).S_i\right\}_{i \in I})}\ [\mathtt{VSend}]$$

By Lemma A.27, since $(\nu,\, S)$ is *well-formed*, it cannot be that both $(\nu,\, S) \xrightarrow{?m}$ and $(\nu,\, S) \xrightarrow{!m'}$, where $m' = l\,\langle T\rangle$. Therefore, this case is not applicable since it cannot be that a *sending* process P is *well-typed* against p, which has an enabled *receiving* action. (The same holds for rule [`DSend`].) □

B.4.2. *Type-checking Processes.*

Lemma B.26. *Let* Δ *be* well-formed. *If* $\Gamma; \theta \vdash P \blacktriangleright \Delta$ *and* $P \equiv Q$*, then* $\Gamma; \theta \vdash Q \blacktriangleright \Delta$.

Proof. As standard [BMVY19, HYC08, YV07], with the additions in Section 3.2.1. □

B.4.3. *Well-typed Substitutions.*

Lemma B.27 (Substitution)**.** *Let* Δ *be* well-formed. *Claims B.28 and B.29 both hold.*

Claim B.28. If $\Gamma; \theta \vdash P \blacktriangleright \Delta, p : (\nu,\, S)$ and $b \notin \mathrm{dom}(\Delta)$, then $\Gamma; \theta \vdash P\,[{}^{b}/_{p}] \blacktriangleright \Delta, b : (\nu,\, S)$.

Claim B.29. If $\Gamma_1, m : T; \theta \vdash P \blacktriangleright \Delta$ and $\Gamma_2 \vdash v : T$ and $\mathrm{dom}(\Gamma_1) \cap \mathrm{dom}(\Gamma_2) = \varnothing$, then $\Gamma_1, \Gamma_2; \theta \vdash P\,[{}^{v}/_{m}] \blacktriangleright \Delta$.

Proof. Claims B.28 and B.29 both hold as standard [BCD+08, HYC08, Vas12, YV07]. □

B.4.4. *Type-checking Messages.*

Lemma B.30 (Well-typed Messages)**.** *Let* $\Gamma; \theta \vdash qp : h \blacktriangleright \Delta, qp : \mathtt{M}$ *and* $\Delta, qp : \mathtt{M}$ *be* well-formed. *Claims B.31 and B.32 both hold, for all* l.

Claim B.31. If $\Gamma \vdash v : T$, then $\Gamma; \theta \vdash qp : h \cdot lv \blacktriangleright \Delta, qp : \mathtt{M}; l\,\langle T\rangle$.

Claim B.32. If $T = \big(\delta,\ S\big)$ and $b \neq \mathrm{dom}(\Delta)$ and $\exists \nu$ such that $\nu \models \delta$, then:

$$\Gamma; \theta \vdash qp : h \cdot lb \blacktriangleright \Delta, qp : \mathtt{M}; l\,\langle T\rangle, b : (\nu,\, S)$$

Proof. We proceed by addressing each claim in turn:

Claim B.31: Holds following the conclusion of rule [`VQue`] in Figure 6 Eq (5.7), where we use the given assumptions as premise:

$$\frac{T \textit{ base-type} \quad \Gamma \vdash v : T \quad \Gamma; \theta \vdash qp : h \blacktriangleright \Delta,\ qp : \mathtt{M}}{\Gamma; \theta \vdash qp : lv \cdot h \blacktriangleright \Delta,\ qp : l\,\langle T\rangle; \mathtt{M}}\ [\mathtt{VQue}]$$

Claim B.32: Holds following the conclusion of rule [`DQue`] in Figure 6 Eq (5.7), similar to the case of Claim B.31 (above). □

B.4.5. *Type-checking Message Handling.*

Lemma B.33. *Let* $\Delta, qp : l\langle T\rangle; \mathtt{M}$ *be* well-formed. *If* $\Gamma; \theta \vdash P \blacktriangleright \Delta, qp : l\langle T\rangle; \mathtt{M}$*, then* $p \in \mathtt{NEQ}(P)$*.*

Proof. By induction on the typing derivation, the key rules are [VQue] are [DQue], which are structured such that: $\Gamma; \theta \vdash qp : lv \cdot h \blacktriangleright \Delta,\ qp : m; \mathtt{M}$ where $m = l\langle T\rangle$. The thesis follows Figure 3, as $p \in \mathtt{NEQ}(qp : lv \cdot h)$ holds. □

Lemma B.34. *Let* Δ *be* well-formed. *If* $\Gamma; \theta \vdash P \blacktriangleright \Delta$ *and* $p \in \mathtt{NEQ}(P)$*, then* $\exists \Delta'$ *such that* $\Delta = \Delta',\ qp : l\langle T\rangle; \mathtt{M}$*.*

Proof. By the definition of $\mathtt{NEQ}(P)$ in Figure 3, it must be that $P \equiv qp : lv \cdot h \mid Q$, and by Claim B.24 of Lemma B.9, we obtain our thesis. □

Lemma B.35. *Let* Δ *be* well-formed. *If* $\Gamma; \theta \vdash P \blacktriangleright \Delta$ *and* $\mathtt{NEQ}(P) = \varnothing$*, then* $\mathtt{NEQ}(\Delta) = \varnothing$*.*

Proof. Since $\mathtt{NEQ}(P) = \varnothing$, it follows that $\forall qp : h$ such that $P \equiv qp : h \mid Q$, then $h = \varnothing$. By Claim B.10 in Lemma B.9, $\Delta = \Delta_1, \Delta_2$ and, $\Gamma; \theta \vdash qp : h \blacktriangleright \Delta_1$ and $\Gamma; \theta \vdash Q \blacktriangleright \Delta_2$. By Claim B.23 in Lemma B.9, it follows that $\Delta_1 = \Delta_1', qp : \mathtt{M}$ and $\mathtt{M} = \varnothing$. We obtain our thesis: given that all message queues in P are empty, if P is *well-typed* against Δ then, there exists corresponding queues in Δ that are all also empty. □

B.4.6. *Delayable Session Environments.*

Lemma B.36. *Let* Δ *be* well-formed. *If* Δ *is* balanced *and* delayable *and* $\Delta + t$ *is* well-formed*, then* $\Delta + t$ *is* balanced.

Proof. We show that $\Delta + t$ satisfies all conditions of Definition 6.1:

Case 1. Since Δ is *delayable*, by Definition B.6, all queues in the domain of Δ are *empty*. Therefore, the case where $\Delta = \Delta', p : (\nu, S), qp : m; \mathtt{M}$, conflicts with the hypothesis, and is not applicable.

Case 2. If $\Delta = \Delta', p : (\nu_1, S_1), qp : \mathtt{M}_1, q : (\nu_2, S_2), pq : \mathtt{M}_2$, then $(\nu_1, S_1, \mathtt{M}_1) \perp (\nu_2, S_2, \mathtt{M}_2)$. Since (ν_1, S_1) and (ν_2, S_2) are *well-formed*, then by Lemmas A.28 and A.30, $(\nu_1 + t, S_1)$ and $(\nu_2 + t, S_2)$ are *well-formed*, and $(\nu_1 + t, S_1, \mathtt{M}) \perp (\nu_2 + t, S_2, \mathtt{M}_2)$. Therefore, $\Delta + t$ is *balanced*.

Case 3. If $\Delta = \Delta', p : (\nu_1, S_1), qp : \mathtt{M}_1, q : (\nu_2, S_2)$, then $\exists \mathtt{M}_2 : (\nu_1, S_1, \mathtt{M}_1) \perp (\nu_2, S_2, \mathtt{M}_2)$ and the thesis follows Case 2. □

Lemma B.37. *If* Δ *is* balanced, well-formed *and not t-reading, then* Δ *is* delayable.

Proof. If we suppose by contradiction that Δ is *not delayable*, then by Definition B.6, $\exists qp, m, \mathtt{M}$ such that $qp \in \mathrm{dom}(\Delta)$ and $\Delta(qp) = m; \mathtt{M}$ and $\exists p \in \mathrm{dom}(\Delta)$. Since Δ is *balanced*, for some $\Delta(p) = (\nu', S')$ it follows Case (1) of Definition 6.1:

$$\Delta = \Delta', p : (\nu', S'), qp : m; \mathtt{M} \implies (\nu', S') \xrightarrow{?m} (\nu'', S'') \quad \text{and} \quad \Delta', p : (\nu'', S''), qp : \mathtt{M} \in \mathrm{Bal}$$

However, this contradicts with Δ not t-reading: by Definition 5.1, there are no roles $q \in \mathrm{dom}(\Delta)$ such that $\Delta(q) = (\nu''', S''')$ and $(\nu''' + t', S''') \xrightarrow{?m'}$, for all $t' < t$. Therefore, Δ must be *delayable*. □

Lemma B.38. *Let* Δ *be* well-formed. *If* $\Gamma; \theta \vdash P \blacktriangleright \Delta$ *and* $\Phi_t(P)$ *is* defined *and* Δ *is* balanced*, then* Δ *is* delayable.

Proof. Since $\Phi_t(P)$ is *defined*, we assume that $t > 0$. We proceed by induction on the typing derivation of P:

Case 1. If $P = p^e \triangleright \{l_i(v_i) : P_i\}_{i \in I} \mid qp : h$ then by Claim B.10 of Lemma B.9, $\theta = (\theta_1, \theta_2)$, $\Delta = (\Delta_1, \Delta_2)$ and:

$$\Gamma; \theta_1 \vdash p^e \triangleright \{l_i(v_i) : P_i\}_{i \in I} \blacktriangleright \Delta_1 \qquad \Gamma; \theta_2 \vdash qp : h \blacktriangleright \Delta_2$$

By Claim B.17 of Lemma B.9, $\Delta_1 = \Delta_1', p : (\nu_1,\, S_1)$ where $S_1 = \{\square_j\, l_j \langle T_j \rangle (\delta_j, \lambda_j).S_j\}_{j \in J}$ and $\neg(|J| = |I| = 1)$. It also follows that $\forall j \in J$ such that $\nu_1 \models \delta_j$, then $\square_j = ?$ and:

$$\exists i \in I : l_i = l_j \quad \text{and} \quad \Gamma; \theta_1 \vdash p^e \triangleright l_i(v_i) : P_i \blacktriangleright \Delta_1', p : (\nu_1,\, \square_j\, l_j \langle T_j \rangle (\delta_j, \lambda_j).S_j)$$

By Claim B.16 of Lemma B.9, Δ_1' is not e-reading, and $\forall t : \nu_1 + t \models \delta_j \iff t \in e$. Since $\nu_1 \models \delta_j$ and $\square_j = ?$, by Claim A.21 of Lemma A.19, by rule $[\texttt{act}]$ in Eq (2.4):

$$(\nu_1,\, \square_j\, l_j \langle T_j \rangle (\delta_j, \lambda_j).S_j) \xrightarrow{?l_j \langle T_j \rangle} (\nu_1 [\lambda_j \mapsto 0],\, S_j)$$

By Lemma B.25, $p \in \texttt{Wait}(P)$. By Definition 3.4, $\texttt{Wait}(P) \cap \texttt{NEQ}(P) = \varnothing$. Therefore, $p \notin \texttt{NEQ}(P)$ and, since $p \notin \texttt{NEQ}(P)$, then $qp : h = \varnothing$. By Claim B.23 of Lemma B.9, $\Delta_2 = \Delta_2', qp : \texttt{M}_1$ and since $h = \varnothing$ then by Lemma B.35, $\texttt{M}_1 = \varnothing$. It follows that Δ_2 is *delayable*, since there cannot be any other queues present in Δ_2' given that $\Gamma; \theta_2 \vdash qp : h \blacktriangleright \Delta_2', qp : \texttt{M}_1$. It remains to show that Δ_1 is *delayable* in order to obtain our thesis and prove that Δ is *delayable*. Since Δ is *balanced*, it follows Δ_1' is also *balanced*, Since Δ_1' is *balanced*, *well-formed* and not e-reading, the thesis follows Lemma B.37.

Case 2. If $P = p^{\diamond n} \triangleright \{l_i(v_i) : P_i\}_{i \in I}$ `after` $Q \mid qp : h$ then by Claim B.10 of Lemma B.9, $\theta = (\theta_1, \theta_2)$, $\Delta = (\Delta_1, \Delta_2)$ and:

$$\Gamma; \theta_1 \vdash p^{\diamond n} \triangleright \{l_i(v_i) : P_i\}_{i \in I} \texttt{ after } Q \blacktriangleright \Delta_1 \qquad \Gamma; \theta_2 \vdash qp : h \blacktriangleright \Delta_2$$

By Claim B.18 of Lemma B.9, $\Delta_1 = \Delta_1', p : (\nu_1,\, S_1)$ where $S_1 = \{\texttt{C}_j\}_{j \in J}$ and:

$$\Gamma; \theta_1 \vdash p^{\diamond n} \triangleright \{l_i(v_i) : P_i\}_{i \in I} \blacktriangleright \Delta_1', p : (\nu_1,\, \{\texttt{C}_j\}_{j \in J}) \tag{B.1}$$

$$\Gamma; \theta_1 + n \vdash Q \blacktriangleright \Delta_1 + n', p : (\nu_1 + n,\, \{\texttt{C}_j\}_{j \in J}) \tag{B.2}$$

By Definition 3.4, if $t \diamond n$ then the thesis follows by the induction hypothesis on Eq (B.1). (See Case 1.) Otherwise, $\neg(t \diamond n)$ and $\Phi_t\left(p^{\diamond n} \triangleright \{l_i(v_i) : P_i\}_{i \in I} \texttt{ after } Q\right) = \Phi_{t-n}(Q)$, and the thesis follows by induction on Eq (B.2).

Case 3. If $P = (\nu pq)\ P'$ then by Claim B.11 of Lemma B.9:

$$\Gamma; \theta \vdash P' \blacktriangleright \Delta,\ p : (\nu_1,\, S_1),\ qp : \texttt{M}_1,\ q : (\nu_2,\, S_2),\ pq : \texttt{M}_2 \tag{B.3}$$

where $(\nu_1,\, S_1,\, \texttt{M}_1) \perp (\nu_2,\, S_2,\, \texttt{M}_2)$ and both $(\nu_1,\, S_1)$ and $(\nu_2,\, S_2)$ are *well-formed*. By Definition 6.2, $\Delta, p : (\nu_1,\, S_1), qp : \texttt{M}_1, q : (\nu_2,\, S_2), pq : \texttt{M}_2$ is *fully-balanced*, and therefore the thesis follows by the induction hypothesis of Eq (B.3).

Case 4. If $P = \texttt{delay}(t').P'$ then by Claim B.21 of Lemma B.9, Δ is not t'-reading and $\Gamma; \theta + t' \vdash P' \Delta + t'$. By Definition 3.4:

i. If $t \leqslant t'$, then $\Phi_t(\texttt{delay}(t').P') = \texttt{delay}(t' - t).P'$. Since Δ is not t'-reading, it follows Δ is not t-reading also. The thesis follows Lemma B.37.

ii. If $t > t'$, then $\Phi_t(\mathtt{delay}(t').P') = \Phi_{t-t'}(P')$ and the thesis follows by the induction hypothesis on: $\Gamma; \theta + t' \vdash \Phi_{t-t'}(P') \blacktriangleright \Delta + t'$.

We only show the key cases of P. The other cases of P follow Definition 3.4. □

Lemma B.39. *If* $\Gamma; \theta \vdash P \blacktriangleright \Delta$ *then* Δ *is* future-enabled.

Proof. A simple induction on the typing derivation. □

Lemma B.40. *If* Δ *is* well-formed *and* $\Delta + t$ *is* live*, then* $\Delta + t$ *is* well-formed.

Proof. Suppose $\Delta + t(p) = (\nu,\ S)$ and $S \neq \mathtt{end}$. It must be $\nu = \nu' + t$ with $\Delta(p) = (\nu',\ S)$ and hence $(\nu',\ S)$ is *well-formed*. By Lemma A.16 we have that $(\nu' + t,\ S)$ is *well-formed*, and by Lemma A.15 it is *live*, as required. □

B.5. **Reduction Steps.**

B.5.1. *Time-passing Steps.*

Lemma B.41. *Let* Δ *be* well-formed. *If* $\Gamma; \theta \vdash P \blacktriangleright \Delta$, $\Phi_t(P)$ *is defined and* $\mathtt{NEQ}(P) = \varnothing$, *then* $\Delta + t$ *is* well-formed *and* $\Gamma; \theta + t \vdash \Phi_t(P) \blacktriangleright \Delta + t$.

Proof. We proceed by induction on the derivation of $\Gamma; \theta \vdash P \blacktriangleright \Delta$ analysing the last rule applied of the *type checking* rules given in Figures 5 and 6:

Case 1. If rule [VRecv], then $P = p^e \triangleright l(v) : P'$ and following Figure 6 Eq (5.6):

$$\frac{T\ \textit{base-type} \quad \Delta' \text{ not } e\text{-reading} \quad \forall t' : \nu + t' \models \delta \Longleftrightarrow t' \in e \qquad \forall t' \in e : \Gamma, v : T; \theta + t' \vdash P \blacktriangleright \Delta' + t',\ p : (\nu + t'[\lambda \mapsto 0],\ S)}{\Gamma; \theta \vdash p^e \triangleright l(v) : P \blacktriangleright \Delta',\ p : (\nu,\ ?l\langle T\rangle(\delta, \lambda).S)}\ [\mathtt{VRecv}]$$

where by Claim B.16 of Lemma B.9, $\Delta = \Delta', p : (\nu,\ ?l\langle T\rangle(\delta, \lambda).S)$. Since $\Phi_t(P)$ is defined, by Definition 3.4, it must be that $t \in e$. (If $e = \infty$ then this always holds.) Coinciding with the first premise of rule [VRecv], it follows that $\nu + t \models \delta$; additionally, by the second premise it follows that the time-step preserves *well-typedness*:

$$\forall t' \in e : \Gamma, v : T; \theta + t' \vdash P' \blacktriangleright \Delta' + t',\ p : (\nu + t'[\lambda \mapsto 0],\ S)$$

It remains to show that $\Delta + t$ is *well-formed*. By Lemma B.39 $\Delta + t$ is *live*. By Lemma B.40 it is *well-formed*.

Case 2. If rule [DRecv], then we obtain our thesis analogously to Case 1.

Case 3. If rule [Branch], then $P = p^e \triangleright \{l_i(v_i) : P_i\}_{i \in I}$ and following Figure 6 Eq (5.5):

$$\frac{\neg(|J| = |I| = 1) \quad \forall j \in J : \nu \models \delta_j \implies \Box_j = ? \ \wedge\ \exists i \in I : l_i = l_j \ \wedge \qquad \Gamma; \theta \vdash p^e \triangleright l_i(v_i) : P_i \blacktriangleright \Delta',\ p : (\nu,\ \Box_j\, l_j\, \langle T_j\rangle(\delta_j, \lambda_j).S_j)}{\Gamma; \theta \vdash p^e \triangleright \{l_i(v_i) : P_i\}_{i \in I} \blacktriangleright \Delta',\ p : (\nu,\ \{\Box_j\, l_j\, \langle T_j\rangle(\delta_j, \lambda_j).S_j\}_{j \in J})}\ [\mathtt{Branch}]$$

where by Claim B.17 of Lemma B.9, $\Delta = \Delta', p : (\nu,\ \{\Box_j\, l_j\, \langle T_j\rangle(\delta_j, \lambda_j).S_j\}_{j \in J})$. Since $\Phi_t(P)$ is defined, by Definition 3.4, it must be that $t \in e$. (If $e = \infty$ then this always holds.) By the premise of rule [Branch], each enabled action j in S is receiving, and has a corresponding branch i in P where $l_i = l_j$ and:

$$\Gamma; \theta \vdash p^e \triangleright l_i(v_i) : P_i \blacktriangleright \Delta', p : (\nu,\ \Box_j\, l_j\, \langle T_j\rangle(\delta_j, \lambda_j).S_j)$$

Therefore, the hypothesis holds by induction. (See Cases 1 and 2.)

Case 4. If rule [Timeout], then $p^{\diamond n} \triangleright \left\{l_i(v_i) : P_i\right\}_{i\in I}$ after Q and following Figure 6 Eq (5.5):

$$\frac{\Gamma;\theta \vdash p^{\diamond n} \triangleright \left\{l_i(v_i) : P_i\right\}_{i\in I} \blacktriangleright \Delta',\ p : (\nu,\ \{\mathtt{C}_j\}_{j\in J}) \quad \Gamma;\theta + n \vdash Q \blacktriangleright \Delta' + n,\ p : (\nu + n,\ \{\mathtt{C}_j\}_{j\in J})}{\Gamma;\theta \vdash p^{\diamond n} \triangleright \left\{l_i(v_i) : P_i\right\}_{i\in I}\ \mathtt{after}\ Q \blacktriangleright \Delta',\ p : (\nu,\ \{\mathtt{C}_j\}_{j\in J})}\ [\mathtt{Timeout}]$$

where by Claim B.18 of Lemma B.9, $\Delta = \Delta', p : (\nu,\ \{\mathtt{C}_j\}_{j\in J})$. Since $\Phi_t(P)$ is defined by Definition 3.4, we show for each case of t:

i. If $t \diamond n$, then $\Phi_t(P) = p^{\diamond n-t} \triangleright \left\{l_i(v_i) : P_i\right\}_{i\in I}$ after Q. Therefore, by the induction hypothesis:

$$\frac{\Gamma;\theta + t \vdash p^{\diamond n+t} \triangleright \left\{l_i(v_i) : P_i\right\}_{i\in I} \blacktriangleright \Delta' + t,\ p : (\nu + t,\ \{\mathtt{C}_j\}_{j\in J}) \quad \Gamma;\theta + n \vdash Q \blacktriangleright \Delta' + t + n - t,\ p : (\nu + t + n - t,\ \{\mathtt{C}_j\}_{j\in J})}{\Gamma;\theta + t \vdash p^{\diamond n-t} \triangleright \left\{l_i(v_i) : P_i\right\}_{i\in I}\ \mathtt{after}\ Q \blacktriangleright \Delta' + t,\ p : (\nu + t,\ \{\mathtt{C}_j\}_{j\in J})}\ [\mathtt{Timeout}]$$

The first premise of rule [Timeout] holds by induction on the derivation. (See Case 3.) The second premise of rule [Timeout] coincides with the hypothesis, as $t + n - t = n$.

ii. If $\neg(t \diamond n)$, then $\Phi_t(P) = \Phi_{t-n}(Q)$. The thesis follows the induction hypothesis:

$$\Gamma;\theta + t \vdash \Phi_{t-n}(Q) \blacktriangleright \Delta + t$$

Case 5. If rule [Del-t], then $P = \mathtt{delay}(t').P'$ and:

i. If $t \leqslant t'$, then $\Phi_t(P) = \mathtt{delay}(t'-t).P'$ and following Figure 5 Eq (5.3):

$$\frac{\Gamma;\theta + t' \vdash P' \blacktriangleright \Delta + t' \quad \Delta \text{ not } t'\text{-reading}}{\Gamma;\theta \vdash \mathtt{delay}(t').P' \blacktriangleright \Delta}\ [\mathtt{Del}\text{-}t]$$

Since Δ is not t'-reading, by Definition 5.1, it follows that Δ is also not t-reading. Therefore, we obtain our thesis: $\Gamma;\theta + t \vdash \mathtt{delay}(t'-t).P' \blacktriangleright \Delta + t$.

ii. If $t > t'$, then $\Phi_t(P) = \Phi_{t-t'}(P')$. The thesis follows the induction hypothesis:

$$\Gamma;\theta + t \vdash \Phi_{t-t'}\left(P'\right) \blacktriangleright \Delta + t$$

Case 6. If rule [Par], then $P = P' \mid Q$ and following Figure 5 Eq (5.2):

$$\frac{\Gamma;\theta_1 \vdash P' \blacktriangleright \Delta_1 \quad \Gamma;\theta_2 \vdash Q \blacktriangleright \Delta_2}{\Gamma;\theta \vdash P' \mid Q \blacktriangleright (\Delta_1,\Delta_2)}\ [\mathtt{Par}]$$

where $\theta = (\theta_1,\theta_2)$ and $\Delta = (\Delta_1,\Delta_2)$. Since Δ is *well-formed*, it follows both Δ_1 and Δ_2 are also *well-formed*. By the induction hypothesis:

$$\frac{\Gamma;\theta_1 + t \vdash \Phi_t(P') \blacktriangleright \Delta_1 + t \quad \Gamma;\theta_2 + t \vdash \Phi_t(Q) \blacktriangleright \Delta_2 + t}{\Gamma;(\theta_1 + t,\theta_2 + t) \vdash \Phi_t(P' \mid Q) \blacktriangleright (\Delta_1,\Delta_2) + t}\ [\mathtt{Par}]$$

where $\Delta + t = (\Delta_1,\Delta_2) + t = (\Delta_1 + t,\Delta_2 + t)$. (See other cases for P' and Q.)

Case 7. If rule [Res], then $P = (\nu pq)\ P'$ and it follows Figure 5 Eq (5.2):

$$\frac{\Gamma;\theta \vdash P' \blacktriangleright \Delta,\ p : (\nu_1,\ S_1),\ qp : \mathtt{M}_1,\ q : (\nu_2,\ S_2),\ pq : \mathtt{M}_2 \quad (\nu_1,\ S_1,\ \mathtt{M}_1) \perp (\nu_2,\ S_2,\ \mathtt{M}_2) \qquad \forall i \in \{1,2\}.\ S_i \text{ well-formed against } \nu_i}{\Gamma;\theta \vdash (\nu pq)\ P' \blacktriangleright \Delta}\ [\mathtt{Res}]$$

By the second premise of rule [Res], $(\nu_1,\ S_1,\ \mathtt{M}_1) \perp (\nu_2,\ S_2,\ \mathtt{M}_2)$ and both $(\nu_1,\ S_1)$ and $(\nu_2,\ S_2)$ are *well-formed*. Since $\mathtt{NEQ}\,(P) = \varnothing$ then by Lemma B.35, $\mathtt{NEQ}\,(\Delta) = \varnothing$ and therefore, it follows that $\mathtt{M}_1 = \varnothing = \mathtt{M}_2$. By Condition (4) of Definition 2.10, since $\mathtt{M}_1 = \varnothing = \mathtt{M}_2$, then $S_1 = \overline{S_2}$ and $\nu_1 = \nu_2$. Hereafter, we only show for p, as the proof is similar for q, and . By the premise of rule [Res], p is *well-formed*. By Lemma A.15, p is *live*. By Definition A.2, either $S_1 = \mathtt{end}$, or p is *future-enabled*.

i. If $S_1 = \mathtt{end}$, then by Lemma A.10 we obtain our thesis as $\mathtt{end}$ is always *well-formed*.

ii. Otherwise, by Lemma A.31 p is *future-enabled*. By Definition 2.3, $\exists t'$ such that $(\nu_1,\ S_1) \xrightarrow{t'\ \Box m}$. It follows that $\forall t'' \leqslant t'$ $(\nu + t'',\ S)$ is also *future-enabled*, since $(\nu + t'',\ S) \xrightarrow{t'-t''\ \Box m}$. Therefore, $(\nu_1 + t',\ S_1)$ is *well-formed*. (The same holds for $(\nu_2,\ S_2)$.) It follows that $(\nu_1 + t',\ S_1,\ \mathtt{M}_1) \perp (\nu_2 + t',\ S_2,\ \mathtt{M}_2)$.

It remains for us to show that $t \leqslant t'$ for the hypothesis to hold. We proceed by case analysis on the structure of $\Box$:

a. If $\Box = ?$, then by Lemma B.25, $p \in \mathtt{Wait}\,(P)$. By Figure 3, P must be structured $P = P' \mid Q$ where $P' \equiv p^{\diamond n} \triangleright \{l_i(v_i) : P_i\}_{i\in I}$ $\mathtt{after}$ Q. (We only show this case as it can be applied for the other cases, as discussed in Section 3.2.1.) By Claim B.18 of Lemma B.9, $S_1 = \{\Box_i\, l_i\, \langle T_i\rangle\,(\delta_i, \lambda_i).S_i\}_{i\in I}$. The thesis follows Case 4.

b. If $\Box = !$, then since $S_1 = \overline{S_2}$, by Lemma A.18, $(\nu_2 + t',\ \overline{S_2}) \xrightarrow{?m}$. (See Case 7.ii.a.) In the case that P is structured $P = P' \mid Q$ where $P' \equiv p \triangleleft l\,(v).P''$, then by Definition 3.4, $\Phi_t\,(P)$ is not defined, as sending actions cannot be delayed.

Case 8. If rule [Rec], then $P = \mathtt{def}\ X(\vec{v};\vec{r}) = P'\ \mathtt{in}\ Q$ and:

$$\Phi_t\,(P) = \mathtt{def}\ X(\vec{v};\vec{r}) = P'\ \mathtt{in}\ \Phi_t\,(Q)$$

and, following Figure 5 Eq (5.2):

$$\frac{\forall(\vec{\nu},\ \vec{S}) \in \boldsymbol{\Delta}, \theta' \in \boldsymbol{\theta} : \Gamma, \vec{v} : \vec{T}, X : (\vec{T};\boldsymbol{\theta};\boldsymbol{\Delta});\theta' \vdash P' \blacktriangleright \vec{r} : (\vec{\nu},\ \vec{S}) \quad \Gamma, X : (\vec{T};\boldsymbol{\theta};\boldsymbol{\Delta});\theta \vdash Q \blacktriangleright \Delta}{\Gamma;\theta \vdash \mathtt{def}\ X(\vec{v};\vec{r}) = P'\ \mathtt{in}\ Q \blacktriangleright \Delta}\ [\mathtt{Rec}]$$

By the induction hypothesis: $\Gamma, X : (\vec{T};\boldsymbol{\theta};\boldsymbol{\Delta});\theta + t \vdash \Phi_t\,(Q) \blacktriangleright \Delta + t$ where $\Delta + t$ is *well-formed*. The thesis then follows by rule [Rec]:

$$\frac{\forall(\vec{\nu},\ \vec{S}) \in \boldsymbol{\Delta}, \theta' \in \boldsymbol{\theta} : \Gamma, \vec{v} : \vec{T}, X : (\vec{T};\boldsymbol{\theta};\boldsymbol{\Delta});\theta' \vdash P' \blacktriangleright \vec{r} : (\vec{\nu},\ \vec{S}) \quad \Gamma, X : (\vec{T};\boldsymbol{\theta};\boldsymbol{\Delta});\theta + t \vdash \Phi_t\,(Q) \blacktriangleright \Delta + t}{\Gamma;\theta + t \vdash \Phi_t\,(P) \blacktriangleright \Delta + t}\ [\mathtt{Rec}]$$

□

Theorem 6.3 (Time Step)**.** *Let* Δ *be* fully-balanced *and* well-formed. *If* $\Gamma; \theta \vdash P \blacktriangleright \Delta$ *and* $\Phi_t(P)$ *is defined, then* $\Gamma; \theta + t \vdash \Phi_t(P) \blacktriangleright \Delta + t$ *and* $\Delta + t$ *is* fully-balanced *and* well-formed.

Proof. We proceed by induction on the derivation of $\Gamma; \theta \vdash P \blacktriangleright \Delta$, analysing the last rule applied of the *type checking* rules given in Figures 5 and 6. Since Δ is *fully-balanced*, only rules [Res] and [Par] are applicable:

Case 1. If rule [Res], then $P = (\nu pq)\ P'$ and following Figure 5 Eq (5.2):

$$\frac{\begin{array}{c}\Gamma; \theta \vdash P' \blacktriangleright \Delta,\ p : (\nu_1,\ S_1),\ qp : \mathtt{M}_1,\ q : (\nu_2,\ S_2),\ pq : \mathtt{M}_2 \\ (\nu_1,\ S_1,\ \mathtt{M}_1) \perp (\nu_2,\ S_2,\ \mathtt{M}_2) \qquad \forall i \in \{1,2\}.\, S_i \text{ well-formed against } \nu_i\end{array}}{\Gamma; \theta \vdash (\nu pq)\ P' \blacktriangleright \Delta}\ [\mathtt{Res}]$$

Let $\Delta' = \Delta, p : (\nu_1,\ S_1), qp : \mathtt{M}_1, q : (\nu_2,\ S_2), pq : \mathtt{M}_2$. By the second premise of rule [Res], it follows that Δ' is *well-formed*. By Definition 6.2, it follows that Δ' is *fully-balanced*. By Definition 3.4, $\Phi_t(P) = (\nu pq)\ \Phi_t(P')$. Therefore, by the induction hypothesis:

$$\frac{\begin{array}{c}\Gamma; \theta + t \vdash \Phi_t(P') \blacktriangleright \Delta + t,\ p : (\nu_1 + t,\ S_1),\ qp : \mathtt{M}_1,\ q : (\nu_2 + t,\ S_2),\ pq : \mathtt{M}_2 \\ (\nu_1,\ S_1,\ \mathtt{M}_1) \perp (\nu_2,\ S_2,\ \mathtt{M}_2) \qquad \forall i \in \{1,2\}.\, S_i \text{ well-formed against } \nu_i\end{array}}{\Gamma; \theta + t \vdash \Phi_t((\nu pq)\ P') \blacktriangleright \Delta + t}\ [\mathtt{Res}]$$

where $\Delta' + t$ is *well-formed* and *fully-balanced*. The second premise of rule [Res] holds:

- Since $\Delta' + t$ is *fully-balanced*, by Definition 6.2, $(\nu_1 + t,\ S_1,\ \mathtt{M}_1) \perp (\nu_2 + t,\ S_2,\ \mathtt{M}_2)$.
- The latter coincides with Definition 2.6 and holds since $\Delta' + t$ is *well-formed*.

The first premise holds by induction. It remains for us to show that $\Delta + t$ is *well-formed* and *fully-balanced*. Since $\mathrm{dom}(\Delta + t) \subseteq \mathrm{dom}(\Delta' + t)$, it follows that $\Delta + t$ is *well-formed* and *fully-balanced*, and we have obtained our thesis.

Case 2. If rule [Par], then $P = P' \mid Q$ and it follows Figure 5 Eq (5.2):

$$\frac{\Gamma; \theta_1 \vdash P' \blacktriangleright \Delta_1 \qquad \Gamma; \theta_2 \vdash Q \blacktriangleright \Delta_2}{\Gamma; \theta \vdash P' \mid Q \blacktriangleright (\Delta_1, \Delta_2)}\ [\mathtt{Par}]$$

where $\theta = (\theta_1, \theta_2)$ and $\Delta = (\Delta_1, \Delta_2)$. It follows that both Δ_1 and Δ_2 are *well-formed*. By Definition 6.1, both Δ_1 and Δ_2 are *balanced*. (Their *fully-balancedness* does not matter.) By Lemma B.38, both Δ_1 and Δ_2 are *delayable*. By Lemma B.36, both Δ_1 and Δ_2 are *balanced*. By Definition 3.4, $\Phi_t(P) = \Phi_t(P') \mid \Phi_t(Q)$. By the induction hypothesis:

$$\frac{\Gamma; \theta_1 + t \vdash \Phi_t(P') \blacktriangleright \Delta_1 + t \qquad \Gamma; \theta_2 + t \vdash \Phi_t(Q) \blacktriangleright \Delta_2 + t}{\Gamma; (\theta_1 + t, \theta_2 + t) \vdash \Phi_t(P' \mid Q) \blacktriangleright (\Delta_1, \Delta_2) + t}\ [\mathtt{Par}]$$

where $\theta + t = (\theta_1 + t, \theta_2 + t)$ and $\Delta + t = (\Delta_1, \Delta_2) + t = (\Delta_1 + t, \Delta_2 + t)$. By Case 6 in Lemma B.41, $\Delta + t$ is *well-formed*. It remains for us to prove that $\Delta + t$ is *fully-balanced*. Since Δ is *fully-balanced*, by Case (2) of Definition 6.2, it follows that $\forall p$ such that $p \in \mathrm{dom}(\Delta)$ and $\Delta(p) = (\nu_1,\ S_1)$, then $\exists \Delta', q, \nu_2, S_2, \mathtt{M}_1, \mathtt{M}_2$ such that $\Delta = \Delta', p : (\nu_1,\ S_1), qp : \mathtt{M}_1, q : (\nu_2,\ S_2), pq : \mathtt{M}_2$ is *balanced*. (By Case (2) of Definition 6.2, the same analogously holds $\forall qp$.) By Definition 6.1, $(\nu_1,\ S_1,\ \mathtt{M}_1) \perp (\nu_2,\ S_2,\ \mathtt{M}_2)$. By Condition (4) of Definition 2.10, if $\mathtt{M}_1 = \varnothing = \mathtt{M}_2$, then $S_1 = \overline{S_2}$ and $\nu_1 = \nu_2$. In such a case, it follows that $\Delta + t = \Delta' + t, p : (\nu_1 + t,\ S_1), qp : \mathtt{M}_1, q : (\nu_2 + t,\ S_2), pq : \mathtt{M}_2$ and $\Delta + t$ is *fully-balanced*. We now show that it must be that $\mathtt{M}_1 = \varnothing = \mathtt{M}_2$. Suppose by contradiction, that $\mathtt{M}_1 = m;\mathtt{M}'_1$. By Definition 6.1, if $\Delta, p : (\nu_1,\, S_1), qp : m;\mathtt{M}'_1$ then $(\nu_1,\, S_1) \xrightarrow{?m} (\nu'_1,\, S'_1)$ and $\Delta, p : (\nu'_1,\, S'_1), qp : \mathtt{M}'_1$ is *balanced*. Since $(\nu_1,\, S_1) \xrightarrow{?m}$, by Lemma B.25, $p \in \mathtt{Wait}\,(P)$. By Lemma B.33, $p \in \mathtt{NEQ}\,(P)$. However, since $\Phi_t\,(P)$ is defined, by Lemma B.7, $\mathtt{Wait}\,(P) \cap \mathtt{NEQ}\,(P) = \varnothing$, which is a contradiction. Therefore, all queues must be empty and $\Delta + t$ is *fully-balanced*. ☐

B.5.2. *Action Steps.*

Theorem 6.4 (Action Step)**.** *Let* Δ *be* balanced *and* well-formed. *If* $\Gamma;\theta \vdash P \blacktriangleright \Delta$ *and* $(\theta, P) \rightharpoonup (\theta', P')$ *then* $\exists \Delta' : \Delta \longrightarrow^* \Delta'$ *and* $\Gamma;\theta' \vdash P' \blacktriangleright \Delta'$ *and* Δ' *is* balanced *and* well-formed.

Proof. We proceed by induction on the derivation of the reduction $(\theta, P) \rightharpoonup (\theta', P')$ analysing the last rule applied of those given in Figure 2:

Case 1. If rule [Send], then $P = p \lhd l\,(v).P'' \mid pq : h$ and it follows Figure 2 Eq (3.3):

$$(\theta,\, p \lhd l\,(v).P'' \mid pq : h) \;\rightharpoonup\; (\theta,\, P'' \mid pq : h \cdot lv) \quad [\mathtt{Send}]$$

where $P' = P'' \mid pq : h \cdot lv$ and $\theta' = \theta$. By Claim B.10 of Lemma B.9, $\theta = (\theta_1, \theta_2)$. $\Delta = (\Delta_1, \Delta_2)$ and:

$$\Gamma;\theta_1 \vdash p \lhd l\,(v).P'' \blacktriangleright \Delta_1 \qquad \Gamma;\theta_2 \vdash pq : h \blacktriangleright \Delta_2 \tag{B.4}$$

By Claim B.23 of Lemma B.9, $\Delta_2 = \Delta'_2, pq : \mathtt{M}$ and if $h = \varnothing$ then $\mathtt{M} = \varnothing$. Conversely, if $h \neq \varnothing$ then, by Claim B.24 of Lemma B.9, $\mathtt{M} \neq \varnothing$. By Claim B.15 of Lemma B.9, $\Delta_1 = \Delta'_1, p : (\nu,\, \{\Box_i\, l_i\,\langle T_i\rangle\,(\delta_i, \lambda_i).S_i\}_{i\in I})$ and $\exists j \in I$ s.t. $l = l_j$, $! = \Box_j$, $\nu \models \delta_j$ and:

i. If T_j is *base-type*, then $\Gamma \vdash v : T_j$ and $\Gamma;\theta_1 \vdash P'' \blacktriangleright \Delta'_1, p : (\nu\,[\lambda_j \mapsto 0]\,,\, S_j)$. To obtain our thesis we must prove $\exists \Delta' : \Delta \longrightarrow^* \Delta'$, and Δ' is *well-formed*, *balanced*, and: $\Gamma;(\theta_1,\theta_2) \vdash P'' \mid pq : h \cdot lv \blacktriangleright \Delta'$. By Claim B.10 of Lemma B.9, $\Delta' = (\Delta''_1, \Delta''_2)$ and:

$$\Gamma;\theta_1 \vdash P'' \blacktriangleright \Delta''_1 \qquad \Gamma;\theta_2 \vdash pq : h \cdot lv \blacktriangleright \Delta''_2 \tag{B.5}$$

We proceed to show the structure of Δ' necessary to obtain our thesis. By inner induction on the derivation of each in Eq (B.4), analysing the last rule applied.

Firstly:

$$\frac{\begin{array}{c}\exists j \in I : (l = l_j) \wedge (\nu \models \delta_j) \wedge (T_j \textit{ base-type}) \wedge (\Gamma \vdash v : T_j) \wedge \\ (\Box_j = !) \wedge \Gamma;\theta \vdash P'' \blacktriangleright \Delta'_1,\; p : (\nu\,[\lambda_j \mapsto 0]\,,\, S_j)\end{array}}{\Gamma;\theta_1 \vdash p \lhd l\,(v).P'' \blacktriangleright \Delta'_1,\; p : (\nu,\, \{\Box_i\, l_i\,\langle T_i\rangle\,(\delta_i, \lambda_i).S_i\}_{i\in I})}\;[\mathtt{VSend}]$$

Therefore, it must be that $\Delta''_1 = \Delta'_1, p : (\nu\,[\lambda_j \mapsto 0]\,,\, S_j)$. Since $\Gamma \vdash v : T_j$ then by Claim B.31 of Lemma B.30 $\Gamma;\theta_2 \vdash pq : h \cdot lv \blacktriangleright \Delta'_2,\; pq : \mathtt{M}; l_j\,\langle T_j\rangle$ and $l = l_j$, which coincides with the latter of Eq (B.5). Therefore, $\Delta''_2 = \Delta'_2, pq : \mathtt{M}; l_j\,\langle T_j\rangle$.

To summarise:

- $\Delta = \Delta'_1,\; p : (\nu,\, \{\Box_i\, l_i\,\langle T_i\rangle\,(\delta_i, \lambda_i).S_i\}_{i\in I}),\; \Delta'_2,\; pq : \mathtt{M}$
- $\Delta' = \Delta'_1,\; p : (\nu\,[\lambda_j \mapsto 0]\,,\, S_j),\; \Delta'_2,\; pq : \mathtt{M}; l_j\,\langle T_j\rangle \qquad (j \in I)$

Such a reduction is possible via rule [Δ-Send] in Figure 7:

$$\frac{(\nu,\, \{\Box_i\, l_i\,\langle T_i\rangle\,(\delta_i, \lambda_i).S_i\}_{i\in I}) \xrightarrow{!m} (\nu',\, S')}{\Delta'',\; p : (\nu,\, \{\Box_i\, l_i\,\langle T_i\rangle\,(\delta_i, \lambda_i).S_i\}_{i\in I})\; pq : \mathtt{M} \;\longrightarrow\; \Delta'',\; p : (\nu',\, S')\; pq : \mathtt{M}; m}\;[\Delta\text{-}\mathtt{Send}]$$

where $m = l_j \langle T_j \rangle$ and $\Delta'' = (\Delta_1', \Delta_2')$. By Case 2 in Lemma A.28, it follows that, for some $j \in I$, $\nu \models \delta_j$, $\nu' = \nu[\lambda_j \mapsto 0]$ and $S' = S_j$. Additionally, it follows that p being *well-formed* indicates that the constraints on actions are structured to preserve *well-formedness* across such transitions. Therefore, it follows that $\Delta' = \Delta'',\ p : (\nu[\lambda_j \mapsto 0], S_j)\ pq : \mathtt{M}; m$ is *well-formed*. By Lemma B.8, Δ' is *balanced*.

Finally, by rule [Par] (where $l = l_j$):

$$\frac{\Gamma;\theta_1 \vdash P'' \blacktriangleright \Delta_1',\ p : (\nu[\lambda_j \mapsto 0], S_j) \quad \Gamma;\theta_2 \vdash pq : h \cdot lv \blacktriangleright \Delta_2',\ pq : \mathtt{M}; l_j \langle T_j \rangle}{\Gamma;(\theta_1,\theta_2) \vdash P'' \mid pq : h \cdot lv \blacktriangleright \Delta_1',\ p : (\nu[\lambda_j \mapsto 0], S_j),\ \Delta_2',\ pq : \mathtt{M}; l_j \langle T_j \rangle}\ [\mathtt{Par}]$$

ii. If $T_j = (\delta',\ S')$, then $\Delta_1'' = \Delta_1''', b : (\nu',\ S')$ and, $\nu' \models \delta'$, $v = b$ and:

$$\Gamma;\theta_1 \vdash P'' \blacktriangleright \Delta_1''',\ p : (\nu[\lambda_j \mapsto 0], S_j)$$

The rest follows Case 1.i.

In Case 1.i, we have shown that the process of a configuration yielded by a reduction via rule [Send] of Figure 2 Eq (3.3), is *well-typed* against a session environment Δ', which itself is reachable via a reduction via the rules in Figure 7, and that the *well-formedness* and *balancedness* of Δ' is preserved. Case 1.ii follows similarly, for delegations.

Case 2. If rule [Recv], then $P = p^e \triangleright \{l_i(v_i) : P_i\}_{i \in I} \mid qp : lv \cdot h$ and by Figure 2 Eq (3.3):

$$\frac{j \in I \quad l = l_j}{(\theta,\ p^e \triangleright \{l_i(v_i) : P_i\}_{i \in I} \mid qp : lv \cdot h) \rightharpoonup (\theta,\ P_j[{}^{v}/_{v_j}] \mid qp : h)}\ [\mathtt{Recv}]$$

where $P' = P_j[{}^{v}/_{v_j}] \mid qp : h$ and $\theta' = \theta$. By Claim B.10 of Lemma B.9, $\theta = (\theta_1, \theta_2)$ and $\Delta = (\Delta_1, \Delta_2)$ and:

$$\Gamma;\theta_1 \vdash p^e \triangleright \{l_i(v_i) : P_i\}_{i \in I} \blacktriangleright \Delta_1 \qquad \Gamma;\theta_2 \vdash qp : lv \cdot h \blacktriangleright \Delta_2 \tag{B.6}$$

By Claim B.24 of Lemma B.9, $\Delta_2 = \Delta_2', pq : l\langle T \rangle; \mathtt{M}$. By Claim B.16 of Lemma B.9, $\Delta_1 = \Delta_1', p : (\nu,\ \{\Box_k\, l_k \langle T_k \rangle (\delta_k, \lambda_k).S_k\}_{k \in K})$ and $\forall k \in K$ such that $\nu \models \delta_k$, then $\Box_k = ?$ and $\exists i \in I$ such that:

$$\Gamma;\theta_1 \vdash p^e \triangleright l_i(v_i) : P_i \blacktriangleright \Delta_1',\ p : (\nu,\ ?l\langle T \rangle(\delta,\lambda).S)$$

where $?l\langle T \rangle(\delta,\lambda).S = \Box_k l_k \langle T_k \rangle (\delta_k, \lambda_k).S_k$. Hereafter we write $\Box_k l_k \langle T_k \rangle (\delta_k, \lambda_k).S_k$. It follows that in Eq (B.6), $lv = l_i v_i$. We proceed by inner induction on the derivation, analysing the last rule applied of those in Figure 6 Eq (5.6):

i. If rule [VRecv], then:

$$\frac{\begin{array}{c} T_k \textit{ base-type} \quad \Delta_1' \text{ not } e\text{-reading} \quad \forall t : \nu + t \models \delta_k \Longleftrightarrow t \in e \\ \forall t \in e : \Gamma, v_i : T_k; \theta_1 + t \vdash P_i \blacktriangleright \Delta_1' + t,\ p : (\nu + t[\lambda_k \mapsto 0], S_k) \end{array}}{\Gamma;\theta_1 \vdash p^e \triangleright l_i(v_i) : P_i \blacktriangleright \Delta_1',\ p : (\nu,\ \Box_k l_k \langle T_k \rangle (\delta_k, \lambda_k).S_k)}\ [\mathtt{VRecv}] \tag{B.7}$$

where $l_k = l_i$ (implicitly). It remains to show that $\exists \Delta'$ such that $\Delta \longrightarrow^* \Delta'$, and Δ' is *well-formed*, *balanced*, and: $\Gamma;(\theta_1,\theta_2) \vdash P_j[{}^{v}/_{v_j}] \mid qp : h \blacktriangleright \Delta'$. By Claim B.10 of Lemma B.9, $\theta' = (\theta_1', \theta_2')$ and $\Delta' = (\Delta_1'', \Delta_2'')$ and:

$$\Gamma;\theta_1' \vdash P_j[{}^{v}/_{v_j}] \blacktriangleright \Delta_1'' \qquad \Gamma;\theta_2 \vdash qp : h \blacktriangleright \Delta_2''$$

Clearly, following Eq (B.7), $\Delta_1'' = \Delta_1', p : (\nu[\lambda_k \mapsto 0], S_k)$. By Claims B.23 and B.24 of Lemma B.9, $\Delta_2'' = \Delta_2', qp : \mathtt{M}$ and, if $h = \varnothing$ then $\mathtt{M} = \varnothing$. (Simiarly, if $h \neq \varnothing$ then

$\mathtt{M} \neq \varnothing$.) Therefore, it must be that $\Delta' = \Delta'_1,\ p : (\nu\,[\lambda_k \mapsto 0]\,,\, S_k),\ \Delta'_2,\ qp : \mathtt{M}$ and, it follows $\Delta \longrightarrow^* \Delta'$ is possible via rule [Δ-Recv] in Figure 7:

$$\frac{(\nu,\, \Box_k\, l_k\,\langle T_k\rangle\,(\delta_k, \lambda_k).S_k,\, l\,\langle T\rangle\,;\mathtt{M}) \xrightarrow{\tau} (\nu',\, S',\, \mathtt{M})}{\Delta'',\ p : (\nu,\, \Box_k l_k\langle T_k\rangle\,(\delta_k, \lambda_k)\,.S_k),\ qp : l\,\langle T\rangle\,;\mathtt{M} \longrightarrow \Delta'',\ p : (\nu',\, S'),\ qp : \mathtt{M}}\ [\Delta\text{-}\mathtt{Recv}]$$

where $\Delta'' = (\Delta'_1, \Delta'_2)$. By Case 3 in Lemma A.28, $l\,\langle T\rangle = l_k\,\langle T_k\rangle$ and, the *well-formedness* rules in Eq (2.8) are enough to guarantee that *well-formedness* is preserved across transitions via rules in Eqs (2.4 to 2.6). Therefore, Δ' is *well-formed*. By Lemma B.8, Δ' is also *balanced*. By induction on the latter derivation in Eq (B.6), analysing the last rule applied. Since T_k is *base-type*, by rule [VQue]:

$$\frac{T_k\ \textit{base-type} \quad \Gamma \vdash v_i : T_k \quad \Gamma; \theta_2 \vdash qp : h \blacktriangleright \Delta'_2,\ qp : \mathtt{M}}{\Gamma; \theta_2 \vdash qp : l_i v_i \cdot h \blacktriangleright \Delta'_2,\ qp : l_k\,\langle T_k\rangle\,;\mathtt{M}}\ [\mathtt{VQue}] \tag{B.8}$$

It remains to show that $\Gamma; (\theta_1, \theta_2) \vdash P_j\,[{}^{v}/{}_{v_j}] \mid qp : h \blacktriangleright \Delta'$. Since $l_k = l_i$ and $lv = l_i v_i$, then by reformulation of Eq (B.6):

$$\Gamma; \theta_1 \vdash p^e \rhd l_i(v_i) : P_i \blacktriangleright \Delta'_1,\ p : (\nu,\, \Box_k l_k\langle T_k\rangle\,(\delta_k, \lambda_k)\,.S_k)$$

$$\Gamma; \theta_2 \vdash qp : l_i v_i \cdot h \blacktriangleright \Delta'_2,\ qp : l_k\,\langle T_k\rangle\,;\mathtt{M}$$

The induction of the reformulated derivations (above) follow the same as Eq (B.7) and Eq (B.8) respectively. Therefore, we conclude by rule [Par]:

$$\frac{\Gamma; \theta_1 \vdash P_j\,[{}^{v}/{}_{v_j}] \blacktriangleright \Delta'_1,\ p : (\nu,\, p : (\nu\,[\lambda_k \mapsto 0]\,,\, S_k)), \qquad \Gamma; \theta_2 \vdash qp : h \blacktriangleright \Delta'_2,\ qp : h}{\Gamma; (\theta_1, \theta_2) \vdash P_j\,[{}^{v}/{}_{v_j}] \mid qp : h \blacktriangleright \Delta'_1,\ p : (\nu,\, p : (\nu\,[\lambda_k \mapsto 0]\,,\, S_k)),\ \Delta'_2,\ qp : h}\ [\mathtt{Par}]$$

ii. If rule [DRecv], then it follows similarly to Case 2.i.

By Case 2.i, a *well-typed* receiving process may reduce and remain *well-typed* against a session environment resulting from a corresponding reduction via the rules in Figure 7.

Case 3. If rule [Recv-T], then $P = p^{\diamond n} \rhd \{l_i(v_i) : P_i\}_{i \in I}$ `after` $Q \mid qp : lv \cdot h$ and it follows Figure 2 Eq (3.3):

$$\frac{j \in I \quad l = l_j}{(\theta,\, p^{\diamond n} \rhd \{l_i(v_i) : P_i\}_{i \in I}\ \mathtt{after}\ Q \mid qp : lv \cdot h) \rightharpoonup (\theta,\, P_j\,[{}^{v}/{}_{v_j}] \mid qp : h)}\ [\mathtt{Recv\text{-}T}]$$

where $P' = P_j\,[{}^{v}/{}_{v_j}] \mid qp : h$. The rest follows Case 2.

Case 4. If rule [Set], then $P = \mathtt{set}\,ⓧ.P'$ and following Figure 2 Eq (3.5):

$$(\theta,\, \mathtt{set}\,ⓧ.P') \rightharpoonup (\theta\,[ⓧ \mapsto 0],\, P')\ \ [\mathtt{Set}]$$

where $\theta' = \theta\,[ⓧ \mapsto 0]$. By Claim B.19 in Lemma B.9, it follows that $\Delta = \Delta'$. Therefore, the thesis coincides with the hypothesis.

Case 5. If rule [Det], then $P = \mathtt{delay}\,(d)\,.P''$ and following Figure 2 Eq (3.5):

$$\frac{t \models d\,[{}^{t}/{}_{t'}]}{(\theta,\, \mathtt{delay}\,(d)\,.P'') \rightharpoonup (\theta,\, \mathtt{delay}\,(t)\,.P'')}\ [\mathtt{Det}]$$

where $P' = \mathtt{delay}\,(t)\,.P''$. By Claim B.20 in Lemma B.9, it follows that $\Delta = \Delta'$. Therefore, the thesis coincides with the hypothesis.

Case 6. If rule [If-T], then $P = \texttt{if}\ c\ \texttt{then}\ P''\ \texttt{else}\ Q$ and following Figure 2 Eq (3.5):

$$\frac{\theta \models c}{(\theta,\, \texttt{if}\ c\ \texttt{then}\ P''\ \texttt{else}\ Q) \rightharpoonup (\theta,\, P'')}\ [\texttt{If-T}]$$

where $P' = P''$. By Claim B.22 in Lemma B.9, it follows that $\Delta = \Delta'$. Therefore, the thesis coincides with the hypothesis.

Case 7. If rule [If-F], then it follows similarly to Case 6, except $\theta \not\models c$ and $P' = Q$.

Case 8. If rule [Par-L], then $P = P'' \mid Q$ and following Figure 2 Eq (3.2):

$$\frac{(\theta_1,\, P'') \rightharpoonup (\theta_1',\, P''')}{(\theta_1, \theta_2,\, P'' \mid Q) \rightharpoonup (\theta_1', \theta_2,\, P''' \mid Q)}\ [\texttt{Par-L}]$$

where $\theta' = (\theta_1', \theta_2)$ and $P' = P''' \mid Q$. By Claim B.10 of Lemma B.9, $\theta = (\theta_1, \theta_2)$ and $\Delta = (\Delta_1, \Delta_2)$ and:

$$\Gamma; \theta_1 \vdash P'' \blacktriangleright \Delta_1 \qquad \Gamma; \theta_2 \vdash Q \blacktriangleright \Delta_2$$

We must show $\exists \Delta'$ such that $\Delta \longrightarrow^* \Delta'$ and $\Gamma; \theta' \vdash P''' \mid Q \blacktriangleright \Delta'$. Again, by Claim B.10 of Lemma B.9, $\theta' = (\theta_1', \theta_2)$ and $\Delta' = (\Delta_1', \Delta_2')$ and:

$$\Gamma; \theta_1' \vdash P''' \blacktriangleright \Delta_1' \qquad \Gamma; \theta_2' \vdash Q \blacktriangleright \Delta_2'$$

Clearly, $\theta_2' = \theta_2$ and $\Delta_2' = \Delta_2$. By rule [Δ-L] in Figure 7:

$$\frac{\Delta_1 \longrightarrow \Delta_1'}{\Delta_1,\ \Delta_2 \longrightarrow \Delta_1',\ \Delta_2}\ [\Delta\texttt{-L}]$$

The thesis follows by the induction hypothesis: since $\Gamma; \theta \vdash P'' \blacktriangleright \Delta_1$, if $(\theta_1,\, P'') \rightharpoonup (\theta_1',\, P''')$ then $\exists \Delta_1'$ such that $\Delta_1 \longrightarrow^* \Delta_1'$ and $\Gamma; \theta_1' \vdash P''' \blacktriangleright \Delta_1'$ and Δ_1' is *balanced* and *well-formed*.

Case 9. If rule [Par-R], then it follows similarly to Case 8.

Case 10. If rule [Scope], then $P = (\nu pq)\ P''$ and following Figure 2 Eq (3.2):

$$\frac{(\theta,\, P'') \rightharpoonup (\theta',\, P''')}{(\theta,\, (\nu pq)\ P'') \rightharpoonup (\theta',\, (\nu pq)\ P''')}\ [\texttt{Scope}]$$

where $P' = (\nu pq)\ P'''$. By Claim B.11 in Lemma B.9, it follows that $\Delta = \Delta'$, as the scope restriction ensures that the inner session (corresponding to pq) is independent of the rest of the session. Therefore, the thesis coincides with the hypothesis.

Case 11. If rule [Def], then $P = \texttt{def}\ X(\vec{v}; \vec{r}) = P''\ \texttt{in}\ Q$ and following Figure 2 Eq (3.4):

$$\frac{(\theta,\, Q) \rightharpoonup (\theta',\, Q')}{(\theta,\, \texttt{def}\ X(\vec{v}; \vec{r}) = P''\ \texttt{in}\ Q) \rightharpoonup (\theta',\, \texttt{def}\ X(\vec{v}; \vec{r}) = P''\ \texttt{in}\ Q')}\ [\texttt{Def}]$$

where $P' = \texttt{def}\ X(\vec{v}; \vec{r}) = P''\ \texttt{in}\ Q'$. By Claim B.12 in Lemma B.9, it follows that:

$$\Gamma, X : (\vec{T}; \boldsymbol{\theta}; \boldsymbol{\Delta}); \theta \vdash Q \blacktriangleright \Delta$$

The thesis follows by the induction hypothesis: since $\Gamma, X : (\vec{T}; \boldsymbol{\theta}; \boldsymbol{\Delta}); \theta \vdash Q \blacktriangleright \Delta$, if $(\theta,\, Q) \rightharpoonup (\theta',\, Q')$ then $\exists \Delta'$ such that $\Delta \longrightarrow^* \Delta'$ and $\Gamma, X : (\vec{T}; \boldsymbol{\theta}; \boldsymbol{\Delta}); \theta' \vdash Q' \blacktriangleright \Delta'$ and Δ' is *balanced* and *well-formed*.

Case 12. If rule [Call], then $P = \texttt{def}\ X(\vec{v'};\vec{r'}) = P''\ \texttt{in}\ X\langle\vec{v};\vec{r}\rangle \mid Q$ and following Figure 2 Eq (3.4):

$$(\theta, \texttt{def}\ X(\vec{v'};\vec{r'}) = P''\ \texttt{in}\ X\langle\vec{v};\vec{r}\rangle \mid Q) \rightharpoonup (\theta, \texttt{def}\ X(\vec{v'};\vec{r'}) = P''\ \texttt{in}\ P''\left[\vec{v};\vec{r}/\vec{v'};\vec{r'}\right] \mid Q) \quad [\texttt{Call}]$$

where $P' = \texttt{def}\ X(\vec{v'};\vec{r'}) = P''\ \texttt{in}\ P''\left[\vec{v};\vec{r}/\vec{v'};\vec{r'}\right] \mid Q$ and $\theta' = \theta$. By Claims B.12 and B.13 of Lemma B.9: $\theta = (\theta_1, \theta_2)$, $\Delta = (\Delta_1, \Delta_2)$ and:

$$\forall(\vec{\nu},\ \vec{S}) \in \boldsymbol{\Delta}, \theta' \in \boldsymbol{\theta} : \quad \Gamma, \vec{v} : \vec{T}, X : (\vec{T}; \boldsymbol{\theta}; \boldsymbol{\Delta}); \theta'' \vdash P'' \blacktriangleright \vec{r'} : (\vec{\nu},\ \vec{S}) \tag{B.9}$$

$$\Gamma, \vec{v} : \vec{T}, X : (\vec{T}; \boldsymbol{\theta}; \boldsymbol{\Delta}); \theta_1 \vdash X\langle\vec{v};\vec{r}\rangle \blacktriangleright \Delta_1 \qquad \Gamma, \vec{v} : \vec{T}, X : (\vec{T}; \boldsymbol{\theta}; \boldsymbol{\Delta}); \theta_2 \vdash Q \blacktriangleright \Delta_2 \tag{B.10}$$

By Claim B.14 of Lemma B.9 on Eq (B.10) we have that $\Delta_1 = \vec{r} : (\vec{\nu}, \vec{S})$ and $(\vec{\nu}, \vec{S}) \in \boldsymbol{\Delta}$. We must show $\exists \Delta'$ such that $\Delta \longrightarrow^* \Delta'$ and:

$$\Gamma; \theta' \vdash \texttt{def}\ X(\vec{v'};\vec{r'}) = P''\ \texttt{in}\ P''\left[\vec{v};\vec{r}/\vec{v'};\vec{r'}\right] \mid Q \blacktriangleright \Delta'$$

By Lemma B.27 on Eq (B.9): By Lemma B.27 on Eq (B.9):

$$\forall(\vec{\nu},\ \vec{S}) \in \boldsymbol{\Delta}, \theta' \in \boldsymbol{\theta} : \quad \Gamma, \vec{v'} : \vec{T}, X : (\vec{T}; \boldsymbol{\theta}; \boldsymbol{\Delta}); \theta'' \vdash P''\left[\vec{v};\vec{r}/\vec{v'};\vec{r'}\right] \blacktriangleright \vec{r} : (\vec{\nu},\ \vec{S})$$

By rule [Par]:

$$\frac{\Gamma, \vec{v} : \vec{T}, X : (\vec{T}; \boldsymbol{\theta}; \boldsymbol{\Delta}); \theta_1 \vdash P''\left[\vec{v};\vec{r}/\vec{v'};\vec{r'}\right] \blacktriangleright \Delta_1 \qquad \Gamma, \vec{v} : \vec{T}, X : (\vec{T}; \boldsymbol{\theta}; \boldsymbol{\Delta}); \theta_2 \vdash Q \blacktriangleright \Delta_2}{\Gamma, \vec{v} : \vec{T}, X : (\vec{T}; \boldsymbol{\theta}; \boldsymbol{\Delta}); \theta_1, \theta_2 \vdash P''\left[\vec{v};\vec{r}/\vec{v'};\vec{r'}\right] \mid Q \blacktriangleright \Delta_1, \Delta_2}\ [\texttt{Par}]$$

By rule [Rec]:

$$\frac{\forall(\vec{\nu},\ \vec{S}) \in \boldsymbol{\Delta}, \theta' \in \boldsymbol{\theta} : \Gamma, \vec{v} : \vec{T}, X : (\vec{T}; \boldsymbol{\theta}; \boldsymbol{\Delta}); \theta'' \vdash P'' \blacktriangleright \vec{r} : (\vec{\nu},\ \vec{S}) \qquad \Gamma, \vec{v} : \vec{T}, X : (\vec{T}; \boldsymbol{\theta}; \boldsymbol{\Delta}); \theta \vdash P''\left[\vec{v};\vec{r}/\vec{v'};\vec{r'}\right] \mid Q \blacktriangleright \Delta}{\Gamma; \theta \vdash \texttt{def}\ X(\vec{v'};\vec{r'}) = P''\ \texttt{in}\ P''\left[\vec{v};\vec{r}/\vec{v'};\vec{r'}\right] \mid Q \blacktriangleright \Delta}\ [\texttt{Rec}]$$

The thesis then holds with $\Delta = \Delta'$.

In the cases above we have shown, given a process P that is *well-typed* against a session environment Δ that is *balanced* and *well-formed*, any *instantaneous* reduction made by the process yields a process P' that is *well-typed* against some session environment Δ' reachable from Δ using the rules in Figure 7. □

B.5.3. *Subject Reduction.*

Corollary 6.5 (Subject Reduction). *If $\varnothing; \theta \vdash P \blacktriangleright \varnothing$ and $(\theta, P) \to (\theta', P')$ then $\varnothing; \theta' \vdash P' \blacktriangleright \varnothing$.*

Proof. Since $\Delta = \varnothing$, then by Definition 6.2, Δ is *fully-balanced*. We proceed by induction on the nature of the reduction ($\longrightarrow$):

Case 1. If $\longrightarrow = \rightsquigarrow$ then the thesis follows Theorem 6.3.

Case 2. If $\longrightarrow = \rightharpoonup$ then the thesis follows Theorem 6.4. □

Appendix C. Example Derivation: Ping-Pong (Example 5.3)

Recall Eqs (5.8) and (5.9) from Example 5.3:

$$S = \begin{Bmatrix} ?ping(x \leqslant 3,\ \{x\}).\begin{Bmatrix} !pong(x \leqslant 3,\ \{x\}).\mathtt{end}, \\ ?timeout(x > 3,\ \varnothing).\mathtt{end} \end{Bmatrix}, \\ \\ !pong(x > 3,\ \{x\}).\begin{Bmatrix} ?ping(x \leqslant 3,\ \{x\}).\mathtt{end}, \\ !timeout(x > 3,\ \varnothing).\mathtt{end} \end{Bmatrix} \end{Bmatrix}$$

$$\begin{aligned} P = {} & p^{\leqslant 3} \triangleright ping : \mathtt{set}\,ⓩ.\mathtt{delay}\,(t \leqslant 4)\,.\mathtt{if}\,(ⓩ \leqslant 3)\,\mathtt{then}\ p \triangleleft pong.\mathtt{0} \\ & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathtt{else}\ p^{\infty} \triangleright timeout : \mathtt{0} \\ & \mathtt{after\ delay}\,(0.1)\,.p \triangleleft pong.p^{\leqslant 3} \triangleright ping : \mathtt{0} \\ & \qquad\qquad\qquad\qquad\qquad \mathtt{after\ delay}\,(0.1)\,.p \triangleleft timeout.\mathtt{0} \end{aligned}$$

Below is the initial typing judgement:

$$\begin{array}{l} \varnothing;\ \theta \vdash p^{\leqslant 3} \triangleright ping : \\ \qquad \mathtt{set}\,ⓩ.\mathtt{delay}\,(t \leqslant 4)\,. \\ \qquad \mathtt{if}\,(ⓩ \leqslant 3) \\ \qquad\ \mathtt{then}\ p \triangleleft pong.\mathtt{0} \\ \qquad\ \mathtt{else}\ p^{\infty} \triangleright timeout : \mathtt{0} \\ \qquad \mathtt{after\ delay}\,(0.1)\,. \\ \qquad p \triangleleft pong. \\ \qquad p^{\leqslant 3} \triangleright ping : \mathtt{0} \\ \qquad \mathtt{after\ delay}\,(0.1)\,. \\ \qquad\ p \triangleleft timeout.\mathtt{0} \end{array} \blacktriangleright\ p : \left(\nu_0,\ \begin{Bmatrix} ?ping(x \leqslant 3,\ \{x\}). \\ \quad \begin{Bmatrix} !pong(x \leqslant 3,\ \{x\}).\mathtt{end}, \\ ?timeout(x > 3,\ \varnothing).\mathtt{end} \end{Bmatrix}, \\ !pong(x > 3,\ \{x\}). \\ \quad \begin{Bmatrix} ?ping(x \leqslant 3,\ \{x\}).\mathtt{end}, \\ !timeout(x > 3,\ \varnothing).\mathtt{end} \end{Bmatrix} \end{Bmatrix}\right)$$

The evaluation begins in Eq (C.1) on the next page.

$$
\dfrac{
\dfrac{\text{(See Eq (C.2))}}{
\begin{array}{l}\varnothing;\ \theta_0 \vdash p^{\leqslant 3} \triangleright ping : \\ \qquad \mathtt{set}\ \textcircled{z}.\mathtt{delay}\,(t \leqslant 4)\,. \\ \qquad \mathtt{if}\ (\textcircled{z} \leqslant 3) \\ \qquad\ \mathtt{then}\ p \triangleleft pong.\mathtt{0} \\ \qquad\ \mathtt{else}\ p^{\infty} \triangleright timeout : \mathtt{0}\end{array}
\blacktriangleright p : \left(\nu_0,\ \left\{\begin{array}{l} ?ping(x \leqslant 3,\ \{x\}). \\ \quad \left\{\begin{array}{l} !pong(x\leqslant 3,\ \{x\}).\mathtt{end}, \\ ?timeout(x>3,\ \varnothing).\mathtt{end}\end{array}\right\}, \\ !pong(x>3,\ \{x\}). \\ \quad \left\{\begin{array}{l} ?ping(x\leqslant 3,\ \{x\}).\mathtt{end}, \\ !timeout(x>3,\ \varnothing).\mathtt{end}\end{array}\right\}\end{array}\right\}\right)
}\,[\mathtt{Branch}]
\\[2ex]
\dfrac{\text{(See Eq (C.5))}}{
\begin{array}{l}\varnothing;\ \theta_0 + 3 \vdash p \triangleleft pong. \\ \qquad\quad p^{\leqslant 3} \triangleright ping : \mathtt{0} \\ \qquad\quad \mathtt{after}\ p \triangleleft timeout.\mathtt{0}\end{array}
\blacktriangleright p : \left(\nu_0 + 3,\ \left\{\begin{array}{l} ?ping(x \leqslant 3,\ \{x\}). \\ \quad \left\{\begin{array}{l} !pong(x\leqslant 3,\ \{x\}).\mathtt{end}, \\ ?timeout(x>3,\ \varnothing).\mathtt{end}\end{array}\right\}, \\ !pong(x>3,\ \{x\}). \\ \quad \left\{\begin{array}{l} ?ping(x\leqslant 3,\ \{x\}).\mathtt{end}, \\ !timeout(x>3,\ \varnothing).\mathtt{end}\end{array}\right\}\end{array}\right\}\right)
}\,[\mathtt{VSend}]
}{
\begin{array}{l}\varnothing;\ \theta_0 \vdash p^{\leqslant 3} \triangleright ping : \\ \qquad \mathtt{set}\ \textcircled{z}.\mathtt{delay}\,(t \leqslant 4)\,. \\ \qquad \mathtt{if}\ (\textcircled{z} \leqslant 3) \\ \qquad\ \mathtt{then}\ p \triangleleft pong.\mathtt{0} \\ \qquad\ \mathtt{else}\ p^{\infty} \triangleright timeout : \mathtt{0} \\ \quad \mathtt{after}\ \mathtt{delay}\,(0.1)\,. \\ \qquad p \triangleleft pong. \\ \qquad p^{\leqslant 3} \triangleright ping : \mathtt{0} \\ \qquad \mathtt{after}\ \mathtt{delay}\,(0.1)\,.p \triangleleft timeout.\mathtt{0}\end{array}
\blacktriangleright p : \left(\nu_0,\ \left\{\begin{array}{l} ?ping(x \leqslant 3,\ \{x\}). \\ \quad \left\{\begin{array}{l} !pong(x\leqslant 3,\ \{x\}).\mathtt{end}, \\ ?timeout(x>3,\ \varnothing).\mathtt{end}\end{array}\right\}, \\ !pong(x>3,\ \{x\}). \\ \quad \left\{\begin{array}{l} ?ping(x\leqslant 3,\ \{x\}).\mathtt{end}, \\ !timeout(x>3,\ \varnothing).\mathtt{end}\end{array}\right\}\end{array}\right\}\right)
}\,[\mathtt{Timeout}]
\tag{C.1}
$$

$$
\dfrac{
\dfrac{
\begin{array}{l|l}
(t = 0) & \dfrac{\text{(See Eq (C.3))}}{\begin{array}{l}\varnothing;\ \theta_0 + 0 \vdash \mathtt{set}\ \textcircled{z}.\mathtt{delay}\,(t \leqslant 4)\,. \\ \qquad\qquad \mathtt{if}\ (\textcircled{z} \leqslant 3) \\ \qquad\qquad\ \mathtt{then}\ p \triangleleft pong.\mathtt{0} \\ \qquad\qquad\ \mathtt{else}\ p^{\infty} \triangleright timeout : \mathtt{0}\end{array} \blacktriangleright p : \left((\nu_0[x \mapsto 0]) + 0,\ \left\{\begin{array}{l} !pong(x\leqslant 3,\ \{x\}).\mathtt{end}, \\ ?timeout(x>3,\ \varnothing).\mathtt{end}\end{array}\right\}\right)}\,[\mathtt{Timer}] \\
\vdots & \\
(t = 3) & \dfrac{\text{(See Eq (C.4))}}{\begin{array}{l}\varnothing;\ \theta_0 + 3 \vdash \mathtt{set}\ \textcircled{z}.\mathtt{delay}\,(t \leqslant 4)\,. \\ \qquad\qquad \mathtt{if}\ (\textcircled{z} \leqslant 3) \\ \qquad\qquad\ \mathtt{then}\ p \triangleleft pong.\mathtt{0} \\ \qquad\qquad\ \mathtt{else}\ p^{\infty} \triangleright timeout : \mathtt{0}\end{array} \blacktriangleright p : \left((\nu_0[x \mapsto 0]) + 3,\ \left\{\begin{array}{l} !pong(x\leqslant 3,\ \{x\}).\mathtt{end}, \\ ?timeout(x>3,\ \varnothing).\mathtt{end}\end{array}\right\}\right)}\,[\mathtt{Timer}]
\end{array}
\qquad \forall t : \nu_0 + t \models (x \leqslant 3) \Longleftrightarrow t \in (\leqslant 3)
}{
\begin{array}{l}\varnothing;\ \theta_0 \vdash p^{\leqslant 3} \triangleright ping : \\ \qquad \mathtt{set}\ \textcircled{z}.\mathtt{delay}\,(t \leqslant 4)\,. \\ \qquad \mathtt{if}\ (\textcircled{z} \leqslant 3) \\ \qquad\ \mathtt{then}\ p \triangleleft pong.\mathtt{0} \\ \qquad\ \mathtt{else}\ p^{\infty} \triangleright timeout : \mathtt{0}\end{array}
\blacktriangleright p : \left(\nu_0,\ ?ping(x \leqslant 3,\ \{x\}). \left\{\begin{array}{l} !pong(x\leqslant 3,\ \{x\}).\mathtt{end}, \\ ?timeout(x>3,\ \varnothing).\mathtt{end}\end{array}\right\}\right)
}\,[\mathtt{VRecv}]
\qquad \neg(1 = 2 = 1)
}{
\begin{array}{l}\varnothing;\ \theta_0 \vdash p^{\leqslant 3} \triangleright ping : \\ \qquad \mathtt{set}\ \textcircled{z}.\mathtt{delay}\,(t \leqslant 4)\,. \\ \qquad \mathtt{if}\ (\textcircled{z} \leqslant 3) \\ \qquad\ \mathtt{then}\ p \triangleleft pong.\mathtt{0} \\ \qquad\ \mathtt{else}\ p^{\infty} \triangleright timeout : \mathtt{0}\end{array}
\blacktriangleright p : \left(\nu_0,\ \left\{\begin{array}{l} ?ping(x \leqslant 3,\ \{x\}). \\ \quad \left\{\begin{array}{l} !pong(x\leqslant 3,\ \{x\}).\mathtt{end}, \\ ?timeout(x>3,\ \varnothing).\mathtt{end}\end{array}\right\}, \\ !pong(x>3,\ \{x\}). \\ \quad \left\{\begin{array}{l} ?ping(x\leqslant 3,\ \{x\}).\mathtt{end}, \\ !timeout(x>3,\ \varnothing).\mathtt{end}\end{array}\right\}\end{array}\right\}\right)
}\,[\mathtt{Branch}]
\tag{C.2}
$$

$$
\begin{array}{l|l}
 & \dfrac{\dfrac{\varnothing;\ \theta'_0, \textcircled{z} : 0 \vdash \mathtt{0} \blacktriangleright\ \varnothing \quad [\texttt{End}]}{\varnothing;\ \theta'_0, \textcircled{z} : 0 \vdash \mathtt{0} \blacktriangleright\ p : (\nu_0[x \mapsto 0],\ \mathtt{end})}\ [\texttt{Weak}]}{} \\[2ex]
 & \dfrac{\nu_0[x \mapsto 0] \models (x \leqslant 3)}{\varnothing;\ \theta'_0, \textcircled{z} : 0 \vdash p \lhd pong.\mathtt{0} \blacktriangleright\ p : \left(\nu_0[x \mapsto 0],\ \left\{\begin{array}{l} !pong(x \leqslant 3,\ \{x\}).\mathtt{end}, \\ ?timeout(x > 3,\ \varnothing).\mathtt{end} \end{array}\right\}\right)}\ [\texttt{VSend}] \\[2ex]
(t = 0) & \dfrac{\dfrac{\theta'_0, \textcircled{z} : 0 \models (\textcircled{z} \leqslant 3)}{\varnothing;\ (\theta'_0, \textcircled{z} : 0) + 0 \vdash \begin{array}{l}\mathtt{if}\ (\textcircled{z} \leqslant 3) \\ \mathtt{then}\ p \lhd pong.\mathtt{0} \\ \mathtt{else}\ p^{\infty} \rhd timeout : \mathtt{0}\end{array} \blacktriangleright\ p : \left((\nu_0[x \mapsto 0]) + 0,\ \left\{\begin{array}{l} !pong(x \leqslant 3,\ \{x\}).\mathtt{end}, \\ ?timeout(x > 3,\ \varnothing).\mathtt{end} \end{array}\right\}\right)}\ [\texttt{IfTrue}]}{\varnothing;\ \theta'_0, \textcircled{z} : 0 \vdash \begin{array}{l}\mathtt{delay}\,(0)\,. \\ \mathtt{if}\ (\textcircled{z} \leqslant 3) \\ \mathtt{then}\ p \lhd pong.\mathtt{0} \\ \mathtt{else}\ p^{\infty} \rhd timeout : \mathtt{0}\end{array} \blacktriangleright\ p : \left(\nu_0[x \mapsto 0],\ \left\{\begin{array}{l} !pong(x \leqslant 3,\ \{x\}).\mathtt{end}, \\ ?timeout(x > 3,\ \varnothing).\mathtt{end} \end{array}\right\}\right)}\ [\texttt{Del-}t] \\[2ex]
\vdots & \\[2ex]
 & \begin{array}{l|l}
(t' = 0) & \dfrac{\varnothing;\ \theta'_4, \textcircled{z} : 4 \vdash \mathtt{0} \blacktriangleright\ \varnothing \quad [\texttt{End}]}{\varnothing;\ (\theta'_4, \textcircled{z} : 4) + 0 \vdash \mathtt{0} \blacktriangleright\ p : ((\nu_4[x \mapsto 4]) + 0,\ \mathtt{end})}\ [\texttt{Weak}] \\
\vdots & \\
(t' = \infty) & \dfrac{\varnothing;\ \theta'_\infty, \textcircled{z} : \infty \vdash \mathtt{0} \blacktriangleright\ \varnothing \quad [\texttt{End}]}{\varnothing;\ (\theta'_4, \textcircled{z} : 4) + \infty \vdash \mathtt{0} \blacktriangleright\ p : ((\nu_4[x \mapsto 4]) + \infty,\ \mathtt{end})}\ [\texttt{Weak}]
\end{array} \\[2ex]
 & \dfrac{\dfrac{\mathtt{None}\ base\text{-}type \quad \forall t' : \nu_4[x \mapsto 4] + t' \models (x > 3) \iff t' \in \infty}{\varnothing;\ \theta'_4, \textcircled{z} : 4 \vdash p^{\infty} \rhd timeout : \mathtt{0} \blacktriangleright\ p : (\nu_4[x \mapsto 4],\ ?timeout(x > 3,\ \varnothing).\mathtt{end})}\ [\texttt{VRecv}] \quad \neg(1 = 2 = 1)}{\varnothing;\ \theta'_4, \textcircled{z} : 4 \vdash p^{\infty} \rhd timeout : \mathtt{0} \blacktriangleright\ p : \left(\nu_4[x \mapsto 4],\ \left\{\begin{array}{l} !pong(x \leqslant 3,\ \{x\}).\mathtt{end}, \\ ?timeout(x > 3,\ \varnothing).\mathtt{end} \end{array}\right\}\right)}\ [\texttt{Branch}] \\[2ex]
(t = 4) & \dfrac{\dfrac{\theta'_4, \textcircled{z} : 4 \not\models (\textcircled{z} \leqslant 3)}{\varnothing;\ (\theta'_0, \textcircled{z} : 0) + 4 \vdash \begin{array}{l}\mathtt{if}\ (\textcircled{z} \leqslant 3) \\ \mathtt{then}\ p \lhd pong.\mathtt{0} \\ \mathtt{else}\ p^{\infty} \rhd timeout : \mathtt{0}\end{array} \blacktriangleright\ p : \left((\nu_0[x \mapsto 0]) + 4,\ \left\{\begin{array}{l} !pong(x \leqslant 3,\ \{x\}).\mathtt{end}, \\ ?timeout(x > 3,\ \varnothing).\mathtt{end} \end{array}\right\}\right)}\ [\texttt{IfFalse}]}{\varnothing;\ \theta'_0, \textcircled{z} : 0 \vdash \begin{array}{l}\mathtt{delay}\,(4)\,. \\ \mathtt{if}\ (\textcircled{z} \leqslant 3) \\ \mathtt{then}\ p \lhd pong.\mathtt{0} \\ \mathtt{else}\ p^{\infty} \rhd timeout : \mathtt{0}\end{array} \blacktriangleright\ p : \left(\nu_0[x \mapsto 0],\ \left\{\begin{array}{l} !pong(x \leqslant 3,\ \{x\}).\mathtt{end}, \\ ?timeout(x > 3,\ \varnothing).\mathtt{end} \end{array}\right\}\right)}\ [\texttt{Del-}t]
\end{array}
$$

$$
\dfrac{\dfrac{\cdots}{\varnothing;\ \theta'_0, \textcircled{z} : 0 \vdash \begin{array}{l}\mathtt{delay}\,(t \leqslant 4)\,. \\ \mathtt{if}\ (\textcircled{z} \leqslant 3) \\ \mathtt{then}\ p \lhd pong.\mathtt{0} \\ \mathtt{else}\ p^{\infty} \rhd timeout : \mathtt{0}\end{array} \blacktriangleright\ p : \left(\nu_0[x \mapsto 0],\ \left\{\begin{array}{l} !pong(x \leqslant 3,\ \{x\}).\mathtt{end}, \\ ?timeout(x > 3,\ \varnothing).\mathtt{end} \end{array}\right\}\right)}\ [\texttt{Del-}d]}{\varnothing;\ \theta'_0, \textcircled{z} : 0 \vdash \begin{array}{l}\mathtt{set}\,\textcircled{z}.\mathtt{delay}\,(t \leqslant 4)\,. \\ \mathtt{if}\ (\textcircled{z} \leqslant 3) \\ \mathtt{then}\ p \lhd pong.\mathtt{0} \\ \mathtt{else}\ p^{\infty} \rhd timeout : \mathtt{0}\end{array} \blacktriangleright\ p : \left(\nu_0[x \mapsto 0],\ \left\{\begin{array}{l} !pong(x \leqslant 3,\ \{x\}).\mathtt{end}, \\ ?timeout(x > 3,\ \varnothing).\mathtt{end} \end{array}\right\}\right)}\ [\texttt{Timer}] \qquad (C.3)
$$

$$\dfrac{\emptyset;\ \theta'_3, \textcircled{z}:3 \vdash \mathtt{0} \blacktriangleright\ \emptyset \quad [\mathtt{End}]}{\emptyset;\ \theta'_3,\textcircled{z}:3\vdash \mathtt{0}\blacktriangleright\ p:\big(\nu_3[x\mapsto 0],\ \mathtt{end}\big)}\ [\mathtt{Weak}]$$

$$\dfrac{\nu_3[x\mapsto 3]\models (x\leqslant 3)}{\emptyset;\ \theta'_3,\textcircled{z}:3\vdash p\triangleleft pong.\mathtt{0}\blacktriangleright\ p:\left(\nu_3[x\mapsto 3],\ \begin{Bmatrix}!pong(x\leqslant 3,\ \{x\}).\mathtt{end},\\ ?timeout(x>3,\ \emptyset).\mathtt{end}\end{Bmatrix}\right)}\ [\mathtt{VSend}]$$

$$\dfrac{\dfrac{\theta'_3,\textcircled{z}:3\models(\textcircled{z}\leqslant 3)}{\emptyset;\ (\theta'_3,\textcircled{z}:3)+0\vdash \begin{array}{l}\mathtt{if}\,(\textcircled{z}\leqslant 3)\\ \ \mathtt{then}\ p\triangleleft pong.\mathtt{0}\\ \ \mathtt{else}\ p^{\infty}\triangleright timeout:\mathtt{0}\end{array}\blacktriangleright\ p:\left((\nu_3[x\mapsto 3])+0,\ \begin{Bmatrix}!pong(x\leqslant 3,\ \{x\}).\mathtt{end},\\ ?timeout(x>3,\ \emptyset).\mathtt{end}\end{Bmatrix}\right)}\ [\mathtt{IfTrue}]}{\emptyset;\ \theta'_3,\textcircled{z}:3\vdash \begin{array}{l}\mathtt{delay}\,(0)\,.\\ \mathtt{if}\,(\textcircled{z}\leqslant 3)\\ \ \mathtt{then}\ p\triangleleft pong.\mathtt{0}\\ \ \mathtt{else}\ p^{\infty}\triangleright timeout:\mathtt{0}\end{array}\blacktriangleright\ p:\left(\nu_3[x\mapsto 3],\ \begin{Bmatrix}!pong(x\leqslant 3,\ \{x\}).\mathtt{end},\\ ?timeout(x>3,\ \emptyset).\mathtt{end}\end{Bmatrix}\right)}\ [\mathtt{Del}\text{-}t] \quad (t=0)$$

$$\vdots$$

$$(t'=0)\quad \dfrac{\emptyset;\ \theta'_7,\textcircled{z}:7\vdash\mathtt{0}\blacktriangleright\ \emptyset\quad[\mathtt{End}]}{\emptyset;\ (\theta'_7,\textcircled{z}:7)+0\vdash\mathtt{0}\blacktriangleright\ p:\big((\nu_7[x\mapsto 7])+0,\ \mathtt{end}\big)}\ [\mathtt{Weak}]$$

$$\vdots$$

$$(t'=\infty)\quad \dfrac{\emptyset;\ \theta'_\infty,\textcircled{z}:\infty\vdash\mathtt{0}\blacktriangleright\ \emptyset\quad[\mathtt{End}]}{\emptyset;\ (\theta'_7,\textcircled{z}:7)+\infty\vdash\mathtt{0}\blacktriangleright\ p:\big((\nu_7[x\mapsto 7])+\infty,\ \mathtt{end}\big)}\ [\mathtt{Weak}]$$

$$\dfrac{\dfrac{\mathtt{None}\ \ base\text{-}type\quad \forall t':\nu_7[x\mapsto 7]+t'\models(x>3)\Longleftrightarrow t'\in\infty}{\emptyset;\ \theta'_7,\textcircled{z}:7\vdash p^{\infty}\triangleright timeout:\mathtt{0}\blacktriangleright\ p:\big(\nu_7[x\mapsto 7],\ ?timeout(x>3,\ \emptyset).\mathtt{end}\big)}\ [\mathtt{VRecv}]\qquad \neg(1=2=1)}{\emptyset;\ \theta'_7,\textcircled{z}:7\vdash p^{\infty}\triangleright timeout:\mathtt{0}\blacktriangleright\ p:\left(\nu_7[x\mapsto 7],\ \begin{Bmatrix}!pong(x\leqslant 3,\ \{x\}).\mathtt{end},\\ ?timeout(x>3,\ \emptyset).\mathtt{end}\end{Bmatrix}\right)}\ [\mathtt{Branch}]$$

$$\dfrac{\dfrac{\theta'_7,\textcircled{z}:7\not\models(\textcircled{z}\leqslant 3)}{\emptyset;\ (\theta'_3,\textcircled{z}:3)+4\vdash \begin{array}{l}\mathtt{if}\,(\textcircled{z}\leqslant 3)\\ \ \mathtt{then}\ p\triangleleft pong.\mathtt{0}\\ \ \mathtt{else}\ p^{\infty}\triangleright timeout:\mathtt{0}\end{array}\blacktriangleright\ p:\left((\nu_3[x\mapsto 3])+4,\ \begin{Bmatrix}!pong(x\leqslant 3,\ \{x\}).\mathtt{end},\\ ?timeout(x>3,\ \emptyset).\mathtt{end}\end{Bmatrix}\right)}\ [\mathtt{IfFalse}]}{\emptyset;\ \theta'_3,\textcircled{z}:3\vdash \begin{array}{l}\mathtt{delay}\,(4)\,.\\ \mathtt{if}\,(\textcircled{z}\leqslant 3)\\ \ \mathtt{then}\ p\triangleleft pong.\mathtt{0}\\ \ \mathtt{else}\ p^{\infty}\triangleright timeout:\mathtt{0}\end{array}\blacktriangleright\ p:\left(\nu_3[x\mapsto 3],\ \begin{Bmatrix}!pong(x\leqslant 3,\ \{x\}).\mathtt{end},\\ ?timeout(x>3,\ \emptyset).\mathtt{end}\end{Bmatrix}\right)}\ [\mathtt{Del}\text{-}t] \quad (t=4)$$

$$\dfrac{\dfrac{(t=0)\ \cdots\ (t=4)}{\emptyset;\ \theta'_3,\textcircled{z}:0\vdash \begin{array}{l}\mathtt{delay}\,(t\leqslant 4)\,.\\ \mathtt{if}\,(\textcircled{z}\leqslant 3)\\ \ \mathtt{then}\ p\triangleleft pong.\mathtt{0}\\ \ \mathtt{else}\ p^{\infty}\triangleright timeout:\mathtt{0}\end{array}\blacktriangleright\ p:\left(\nu_3[x\mapsto 3],\ \begin{Bmatrix}!pong(x\leqslant 3,\ \{x\}).\mathtt{end},\\ ?timeout(x>3,\ \emptyset).\mathtt{end}\end{Bmatrix}\right)}\ [\mathtt{Del}\text{-}d]}{\emptyset;\ \theta'_3,\textcircled{z}:3\vdash \begin{array}{l}\mathtt{set}\,\textcircled{z}.\mathtt{delay}\,(t\leqslant 4)\,.\\ \mathtt{if}\,(\textcircled{z}\leqslant 3)\\ \ \mathtt{then}\ p\triangleleft pong.\mathtt{0}\\ \ \mathtt{else}\ p^{\infty}\triangleright timeout:\mathtt{0}\end{array}\blacktriangleright\ p:\left(\nu_3[x\mapsto 3],\ \begin{Bmatrix}!pong(x\leqslant 3,\ \{x\}).\mathtt{end},\\ ?timeout(x>3,\ \emptyset).\mathtt{end}\end{Bmatrix}\right)}\ [\mathtt{Timer}]$$

(C.4)

$$
\dfrac{
\dfrac{
\begin{array}{c|c}
(t=0) & \dfrac{\varnothing;\ \theta_{3.1} \vdash \mathtt{0} \blacktriangleright\ \varnothing \quad [\mathtt{End}]}{\varnothing;\ \theta_{3.1} \vdash \mathtt{0} \blacktriangleright\ p : \big((\nu_{3.1}[x \mapsto 0]) + 0,\ \mathtt{end}\big)}\ [\mathtt{Weak}] \\
\vdots & \\
(t=3) & \dfrac{\varnothing;\ \theta_{3.1} \vdash \mathtt{0} \blacktriangleright\ \varnothing \quad [\mathtt{End}]}{\varnothing;\ \theta_{3.1} \vdash \mathtt{0} \blacktriangleright\ p : \big((\nu_{3.1}[x \mapsto 0]) + 3[x \mapsto 0],\ \mathtt{end}\big)}\ [\mathtt{Weak}]
\end{array}
}{\varnothing;\ \theta_{3.1} \vdash p^{\leqslant 3} \triangleright ping : \mathtt{0} \blacktriangleright\ p : \big(\nu_{3.1}[x \mapsto 0],\ ?ping(x \leqslant 3,\ \{x\}).\mathtt{end}\big)}\ [\mathtt{VRecv}]
}{\varnothing;\ \theta_{3.1} \vdash p^{\leqslant 3} \triangleright ping : \mathtt{0} \blacktriangleright\ p : \left(\nu_{3.1}[x \mapsto 0],\ \begin{Bmatrix} ?ping(x \leqslant 3,\ \{x\}).\mathtt{end}, \\ !timeout(x > 3,\ \varnothing).\mathtt{end} \end{Bmatrix}\right)}\ [\mathtt{Branch}]
$$

$$
\dfrac{
\dfrac{\varnothing;\ \theta_{6.2} \vdash \mathtt{0} \blacktriangleright\ \varnothing \quad [\mathtt{End}]}{\varnothing;\ \theta_{6.2} \vdash \mathtt{0} \blacktriangleright\ p : \big(\nu_{6.2}[x \mapsto 3.1],\ \mathtt{end}\big)}\ [\mathtt{Weak}]
}{\varnothing;\ \theta_{6.1} + 0.1 \vdash p \triangleleft timeout.\mathtt{0} \blacktriangleright\ p : \left((\nu_{6.1}[x \mapsto 3]) + 0.1,\ \begin{Bmatrix} ?ping(x \leqslant 3,\ \{x\}).\mathtt{end}, \\ !timeout(x > 3,\ \varnothing).\mathtt{end} \end{Bmatrix}\right)}\ [\mathtt{VSend}]
$$

$$
\dfrac{
\dfrac{
\dfrac{\nu_{6.2}[x \mapsto 3.1] \models (x > 3)}{\begin{array}{l} \varnothing;\ \theta_{3.1} + 3 \vdash \mathtt{delay}\,(0.1)\,. \\ \qquad p \triangleleft timeout.\mathtt{0} \end{array} \blacktriangleright\ p : \left((\nu_{3.1}[x \mapsto 0]) + 3,\ \begin{Bmatrix} ?ping(x \leqslant 3,\ \{x\}).\mathtt{end}, \\ !timeout(x > 3,\ \varnothing).\mathtt{end} \end{Bmatrix}\right)}\ [\mathtt{Del}\text{-}t]
}{\begin{array}{l} \varnothing;\ \theta_{3.1} \vdash p^{\leqslant 3} \triangleright ping : \mathtt{0} \\ \qquad \mathtt{after\ delay}\,(0.1)\,. \\ \qquad p \triangleleft timeout.\mathtt{0} \end{array} \blacktriangleright\ p : \left(\nu_{3.1}[x \mapsto 0],\ \begin{Bmatrix} ?ping(x \leqslant 3,\ \{x\}).\mathtt{end}, \\ !timeout(x > 3,\ \varnothing).\mathtt{end} \end{Bmatrix}\right)}\ [\mathtt{Timeout}]
}{\begin{array}{l} \varnothing;\ \theta_3 + 0.1 \vdash p \triangleleft pong. \\ \qquad p^{\leqslant 3} \triangleright ping : \mathtt{0} \\ \qquad \mathtt{after\ delay}\,(0.1)\,.p \triangleleft timeout.\mathtt{0} \end{array} \blacktriangleright\ p : \left(\nu_3 + 0.1,\ !pong(x > 3,\ \{x\}).\begin{Bmatrix} ?ping(x \leqslant 3,\ \{x\}).\mathtt{end}, \\ !timeout(x > 3,\ \varnothing).\mathtt{end} \end{Bmatrix}\right)}\ [\mathtt{VSend}]
$$

$$
\dfrac{\nu_{3.1} \models (x > 3)}{\begin{array}{l} \varnothing;\ \theta_3 \vdash \mathtt{delay}\,(0.1)\,. \\ \qquad p \triangleleft pong. \\ \qquad p^{\leqslant 3} \triangleright ping : \mathtt{0} \\ \qquad \mathtt{after\ delay}\,(0.1)\,.p \triangleleft timeout.\mathtt{0} \end{array} \blacktriangleright\ p : \left(\nu_3,\ !pong(x > 3,\ \{x\}).\begin{Bmatrix} ?ping(x \leqslant 3,\ \{x\}).\mathtt{end}, \\ !timeout(x > 3,\ \varnothing).\mathtt{end} \end{Bmatrix}\right)}\ [\mathtt{Del}\text{-}t] \tag{C.5}
$$