Constructive characterisations of the must-preorder for asynchrony

Anonymous author
Anonymous affiliation

Abstract

De Nicola and Hennessy's MUST-preorder is a contextual refinement which states that a server q refines a server p if all clients satisfied by p are also satisfied by q. Owing to the universal quantification over clients, this definition does not yied a practical proof method for the MUST-preorder, and alternative characterisations are necessary to reason on it.

We present the first characterisations of the MUST-preorder that are constructive, supported by a mechanisation in Coq, and independent from any calculus: our results pertain to Selinger output-buffered agents with feedback. This is a class of Labelled Transition Systems that captures programs that communicate asynchronously via a shared unordered buffer, as in asynchronous CCS or the asynchronous π-calculus.

Our results are surprising: the behavioural characterisations devised for synchronous communication carry over as they stand to asynchronous communication, if servers are enhanced to act as forwarders, *i.e.* they can input any message as long as they store it back into the shared buffer. This suggests a technique to port standard characterisations from synchronous to asynchronous settings.

2012 ACM Subject Classification classification

Keywords and phrases Author: Please fill in \keywords macro

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

Code refactoring is a routine task to develop or update software, and it requires methods to ensure that a program p can be safely replaced by a program q. One way to address this issue is via refinement relations, *i.e.* preorders. For programming languages, the most well-known one is Morris *extensional* preorder [76, pag. 50], defined by letting $p \leq q$ if for all contexts C, whenever $C[p]$ reduces to a normal form N, then $C[q]$ also reduces to N.

Comparing servers. This paper studies a version of Morris preorder for nondeterministic asynchronous client-server systems. In this setting it is natural to reformulate the preorder by replacing reduction to normal forms (*i.e.* termination) with a suitable *liveness* property. Let $p \parallel r$ denote a *client-server system*, that is a parallel composition in which the identities of the server p and the client r are distinguished, and whose computations have the form $p \parallel r = p_0 \parallel r_0 \longrightarrow p_1 \parallel r_1 \longrightarrow p_2 \parallel r_2 \longrightarrow \ldots$, where each step represents either an internal computation of one of the two components, or an interaction between them. Interactions correspond to handshakes, where two components ready to perform matching input/output actions advance together. We express liveness by saying that p *must pass* r, denoted $p \,\textsc{must}\, r$, if in every maximal computation of $p \parallel r$ there exists a state $p_i \parallel r_i$ such that $\textsc{good}(r_i)$, where $\textsc{good}$ is a decidable predicate indicating that the client has reached a successful state. Servers are then compared according to their capacity to satisfy clients, *i.e.* via contexts of the form $[-] \parallel r$ and the predicate MUST. Morris preorder then becomes the MUST-preorder by De Nicola and Hennessy [44] : $p \mathrel{\underset{\sim}{\sqsubset}}_{\textsc{must}} q$ when $\forall r.\, p \,\textsc{must}\, r$ implies $q \,\textsc{must}\, r$.

Advantages. The MUST-preorder is by definition liveness preserving, because $p \,\textsc{must}\, r$ literally means that "in every execution something good must happen (on the client side)". Results on $\mathrel{\underset{\sim}{\sqsubset}}_{\textsc{must}}$ thus shed light on liveness-preserving program transformations.

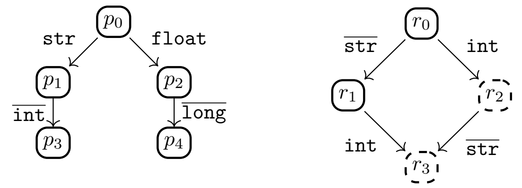


Figure 1 The behaviours of a server p_0 and of a client r_0.

The MUST-preorder is independent of any particular calculus, as its definition requires simply (1) a reduction semantics for the parallel composition $p \parallel r$, and (2) a predicate GOOD over programs. Hence $\sqsubseteq_{\text{MUST}}$ may relate servers written in different languages. For instance, servers written in OCAML may be compared to servers written in JAVA according to clients written in PYTHON, because all these languages communicate using the same basic protocols.

Drawback. The definition of the MUST-preorder is *contextual*: proving that $p \sqsubseteq_{\text{MUST}} q$ requires analysing an *infinite* amount of clients, and so the definition of the preorder does not entail an effective proof method. A solution to this problem is to define an *alternative (semantic) characterisation* of the preorder $\sqsubseteq_{\text{MUST}}$, *i.e.* a preorder $\preccurlyeq_{alt}$ that coincides with $\sqsubseteq_{\text{MUST}}$ and does away with the universal quantification over clients (*i.e.* contexts). In *synchronous* settings, i.e. when both input and output actions are blocking, such alternative characterisations have been thoroughly investigated, typically via a behavioural approach.

Behavioural characterisations. Alternative preorders are usually defined in two steps:
- First, programs are associated with labelled transition systems (LTSs) like those in Figure 1, where transitions are labelled by input actions such as `str`, output actions such as $\overline{\texttt{str}}$, or the internal action τ while dotted nodes represent successful states, *i.e.* those satisfying the predicate GOOD. There, the server p_0 is ready to input either a string or a float. The client r_0, on the other hand, is ready to either output a string, or input an integer. The input `int` makes the client move to the successful state r_2, while the output $\overline{\texttt{str}}$ makes the client move to the state r_1, where it can still perform the input `int` to reach the successful state r_3. Output transitions enjoy a sort of commutativity property on which we will return later. Programs $p, q, r, \ldots$ are usually associated with their behaviours via inferences rules, which implicitly define a function $\text{LTS}(-)$ that, given a program p, returns the LTS whose root is p.
- Second, program behaviours, *i.e.* LTSs, are used to define the alternative preorders for $\sqsubseteq_{\text{MUST}}$ following one of two different approaches: MUST-sets or acceptance sets.

Alternative preorders for synchrony. Both approaches were originally proposed for the calculus CCS [75], where communication is synchronous. The first alternative preorder, which we denote by $\preccurlyeq_{\mathsf{MS}}$, was put forth by De Nicola [44], and it compares server behaviours according to their MUST-sets, *i.e.* the sets of actions that they may perform. The second alternative preorder, which we denote by $\preccurlyeq_{\mathsf{AS}}$, was put forth by Hennessy [55], and it compares the acceptance sets of servers, *i.e.* how servers can be moved out of their potentially deadlocked states. Both these preorders characterise $\sqsubseteq_{\text{MUST}}$ in the following sense:

$$\forall p, q \in \text{CCS}.\, p \sqsubseteq_{\text{MUST}} q \text{ iff } \text{LTS}(p) \preccurlyeq_{\mathsf{MS}} \text{LTS}(q) \tag{1}$$

$$\forall p, q \in \text{CCS}.\, p \sqsubseteq_{\text{MUST}} q \text{ iff } \text{LTS}(p) \preccurlyeq_{\mathsf{AS}} \text{LTS}(q) \tag{2}$$

Asynchrony. In distributed systems, however, communication is inherently asynchronous. For instance, the standard TCP transmission on the Internet is asynchronous. Actor languages like ELIXIR and ERLANG implement asynchrony via mailboxes, and both PYTHON and

$$\begin{tikzcd} p \arrow[r, "\overline{a}"] & p' \arrow[d, "\alpha"] \\ & q \end{tikzcd} \Rightarrow \begin{tikzcd} p \arrow[r, "\overline{a}"] \arrow[d, "\alpha"] & p' \arrow[d, "\alpha"] \\ p'' \arrow[r, "\overline{a}"] & q \end{tikzcd}$$

Output-commutativity

$$\begin{tikzcd} p \arrow[r, "\overline{a}"] & p' \arrow[d, "a"] \\ & q \end{tikzcd} \Rightarrow \begin{tikzcd} p \arrow[r, "\overline{a}"] \arrow[dr, "\tau"] & p' \arrow[d, "a"] \\ & q \end{tikzcd}$$

Feedback

$$\begin{tikzcd} p \arrow[r, "\overline{a}"] \arrow[d, "\overline{a}"] & p' \\ p'' & \end{tikzcd} \Rightarrow p' = p''$$

Output-determinacy

$$\begin{tikzcd} p \arrow[r, "\overline{a}"] \arrow[d, "\tau"] & p' \\ p'' & \end{tikzcd} \Rightarrow \begin{tikzcd} p \arrow[r, "\overline{a}"] \arrow[d, "\tau"] & p' \arrow[d, "\tau"] \\ p'' \arrow[r, "\overline{a}"] & q \end{tikzcd} \text{ or } \begin{tikzcd} p \arrow[r, "\overline{a}"] \arrow[d, "\tau"] & p' \arrow[dl, "a"] \\ p'' & \end{tikzcd}$$

Output-tau

$$\begin{tikzcd} p \arrow[r, "\overline{a}"] \arrow[d, "\alpha"] & p' \\ p'' & \end{tikzcd} \Rightarrow \begin{tikzcd} p \arrow[r, "\overline{a}"] \arrow[d, "\alpha"] & p' \arrow[d, "\alpha"] \\ p'' \arrow[r, "\overline{a}"] & q \end{tikzcd}$$

where $\alpha \neq \overline{a}$ and $\alpha \neq \tau$

Output-confluence

$$\begin{tikzcd} & p' \arrow[d, "\overline{a}"] \\ p'' \arrow[r, "\overline{a}"] & p \end{tikzcd} \Rightarrow p' = p''$$

Backward-output-determinacy

Figure 2 First-order axioms for output-buffered agents with feedback as given by Selinger [92], extended with the Backward-output-determinacy axiom.

JavaScript offer developers the constructs async/wait, to return promises (of results) or wait for them. In this paper we model asynchrony via *output-buffered agents with feedback*, as introduced by Selinger [92]. These are LTSs obeying the axioms in Figure 2, where a denotes an input action, $\overline{a}$ denotes an output action, τ denotes the internal action, and α ranges over all these actions. For instance, the Output-commutativity axiom states that an output $\overline{a}$ can always be postponed: if $\overline{a}$ is followed by any action α, it can commute with it. In other terms, outputs are non-blocking,as illustrated by the LTS for r_0 in Figure 1.

Theoretical issues. The practical importance of asynchrony motivates a specific study of $\sqsubseteq_{\text{MUST}}$. Efforts in this direction have already been made, all of which focussed on process calculi [39, 24, 95, 57]. Note that the axioms in Figure 2 impose conditions only over outputs. This asymmetric treatment of inputs and outputs substantially complicates the proofs of completeness and soundness of the alternative characterisations of $\sqsubseteq_{\text{MUST}}$. To underline the subtleties due to asynchrony, we note that the completeness result for asynchronous CCS given by Castellani and Hennessy in [39], and subsequently extended to the π-calculus by Hennessy [57], is false (see Appendix H).

Contributions and paper structure. Our main contributions may be summarised as follows (where for each of them, we detail where it is presented):

- The first behavioural characterisations of the must-preorder (Theorem 17, Theorem 21), that are calculus independent, in that both our definitions and our proofs work directly on LTSs. Contrary to all the previous works on the topic, we show that the *standard* alternative preorders characterise the must-preorder also in Selinger asynchronous setting. To this end, it suffices to enrich the server semantics with *forwarding*, i.e. ensure that servers are ready to receive any input message, as long as they store it back in a global shared buffer. This idea, although we use it here in a slightly different form, was pioneered by Honda et al. [64]. In this paper we propose a construction that works on any LTS (Lemma 13) and we show the following counterparts of Equations (1) and (2), where OF denotes the LTSs of output-buffered agents with feedback, and FW is the function that

```
Class Sts (A: Type) := MkSts {
  sts_step: A → A → Prop;
  sts_stable: A → Prop; }.

Inductive ExtAct (A: Type) :=
| ActIn (a: A) | ActOut (a: A).

Inductive Act (A: Type) :=
| ActExt (ext: ExtAct A) | τ.

Class Label (L: Type) :=
MkLabel {
 label_eqdec: EqDecision L;
 label_countable: Countable L; }.

Class Lts (A L : Type) `{Label L} :=
MkLts {
 lts_step: A → Act L → A → Prop;
 lts_outputs: A → finite_set L;
 lts_performs: A → (Act L) → Prop; }.
```

Figure 3 Highlights of our Sts and Lts typeclasses.

enhances them with forwarding:

$$\forall p, q \in \text{OF}.\, p \sqsubseteq_{\text{MUST}} q \text{ iff } \text{FW}(p) \preccurlyeq_{\mathsf{MS}} \text{FW}(q) \tag{a}$$

$$\forall p, q \in \text{OF}.\, p \sqsubseteq_{\text{MUST}} q \text{ iff } \text{FW}(p) \preccurlyeq_{\mathsf{AS}} \text{FW}(q) \tag{b}$$

Quite surprisingly, the alternative preorders $\preccurlyeq_{\mathsf{AS}}$ and $\preccurlyeq_{\mathsf{MS}}$ need not be changed. We present these results in Section 3. Selinger axioms are fundamental to prove completeness, which we discuss in Appendix C.

- The first constructive account of the MUST-preorder. We show that if the MUST and termination predicates are defined *intensionally* (in the sense of Brede and Herbelin [29]), then $\sqsubseteq_{\text{MUST}}$ can be characterised constructively. The original definitions of MUST and termination given by De Nicola [44], though, are *extensional*. Showing that intensional and extensional definitions are logically equivalent is a known problem, discussed for instance by Coquand [43] and Brede and Herbelin [29]. We follow their approach and adapt the bar-induction principle to our setting to prove the desired equivalences. Our treatment shows that Kőnig's lemma, which is a mainstay in the literature on the MUST-preorder, is actually unnecessary: the bar-induction principle suffices for our purposes[1]. Since Rahli et al. [83] have shown bar-induction to be compatible with constructive type theory, we argue that our development is entirely constructive. Due to space constraints, we explain the principle of bar-induction and how to adapt it to our usage in Appendix A, while in this extended abstract we merely employ the principle.
- The first mechanisation of the theory of MUST-preorder in a fully nondeterministic setting, which consists of around 8000 lines of Coq. In Appendix I we gather the Coq versions of all the definitions and the results used in the main body of the paper.

In Section 5, we discuss the impact of the above contributions, as well as related and future work. In Section 2, we recall the necessary background definitions and illustrate them with a few examples.

[1] In fact even its version for finite branching trees, i.e. the fan theorem, suffices in the current treatment.

$$[\text{S-SRV}]\ \frac{p \xrightarrow{\tau} p'}{p \mathbin{[\![} r \longrightarrow p' \mathbin{[\![} r} \qquad [\text{S-CLT}]\ \frac{r \xrightarrow{\tau} r'}{p \mathbin{[\![} r \longrightarrow p \mathbin{[\![} r'} \qquad [\text{S-COM}]\ \frac{p \xrightarrow{\mu} p' \quad r \xrightarrow{\overline{\mu}} r'}{p \mathbin{[\![} r \longrightarrow p' \mathbin{[\![} r'}$$

Figure 4 The STS of server-client systems.

2 Preliminaries

We model individual programs such as servers p and clients r as LTSs obeying Selinger axioms, while client-server systems $p \mathbin{[\![} r$ are modelled as state transition systems with a reduction semantics. We now formally define this two-level semantics.

Labelled transition systems. A *labelled transition system* (LTS) is a triple $\mathcal{L} = \langle A, L, \longrightarrow \rangle$ where A is the set of states, L is the set of labels and $\longrightarrow \subseteq A \times L \times A$ is the transition relation. When modelling programs as LTSs, we use transition labels to represent program actions. The set of labels in Selinger LTSs has the same structure as the set of actions in Milner's calculus CCS: one assumes a set of names $\mathcal{N}$, denoting input actions and ranged over by a, b, c, a complementary set of conames $\overline{\mathcal{N}}$, denoting output actions and ranged over by $\overline{a}, \overline{b}, \overline{c}$, and an *invisible* action τ, representing internal computation. The set of all actions, ranged over by α, β, γ, is given by $\mathsf{Act}_\tau \stackrel{\mathsf{def}}{=} \mathcal{N} \uplus \overline{\mathcal{N}} \uplus \{\tau\}$. We use μ, μ' to range over the set of visible actions $\mathcal{N} \uplus \overline{\mathcal{N}}$, and we extend the complementation function $\overline{\cdot}$ to this set by letting $\overline{\overline{a}} \stackrel{\mathsf{def}}{=} a$. In the following, we will always assume $L = \mathsf{Act}_\tau$. Once the LTS is fixed, we write $p \xrightarrow{\alpha} p'$ to mean that $(p, \alpha, p') \in \longrightarrow$ and $p \xrightarrow{\alpha}$ to mean $\exists p'. p \xrightarrow{\alpha} p'$.

We use $\mathcal{L}$ to range over LTSs. To reason simultaneously on different LTSs, we will use the symbols $\mathcal{L}_A$ and $\mathcal{L}_B$ to denote respectively the LTSs $\langle A, L, \longrightarrow_A \rangle$ and $\langle B, L, \longrightarrow_B \rangle$.

In our mechanisation LTSs are borne out by the typeclass `Lts` in Figure 3. The states of the LTS have type A, labels have type L, and `lts_step` is the characteristic function of the transition relation, which we assume to be decidable. We let $O(p) = \{\overline{a} \in \overline{\mathcal{N}} \mid p \xrightarrow{\overline{a}}\}$ and $I(p) = \{a \in \mathcal{N} \mid p \xrightarrow{a}\}$ be respectively the set of outputs and the set of inputs of state p. We assume that the set $O(p)$ is finite for any p. In our mechanisation, the set $O(p)$ is rendered by the function `lts_outputs`, and we shall also use a function `lts_performs` that lets us decide whether a state can perform a transition labelled by a given action.

Client-server systems. A *client-server* system (or *system*, for short) is a pair $p \mathbin{[\![} r$ in which p is deemed to be the server of client r. In general, every system $p \mathbin{[\![} r$ is the root of a *state transition system* (STS), $\langle S, \longrightarrow \rangle$, where S is the set of states and $\longrightarrow$ is the reduction relation. For the sake of simplicity[2]we derive the reduction relation from the LTS semantics of servers and clients as specified by the rules in Figure 4. In our mechanisation (Figure 3), `sts_step` is the characteristic function of the reduction relation $\longrightarrow$, and `sts_stable` is the function that states whether a state can reduce or not. Both functions are assumed decidable.

▶ **Definition 1** (Computation). *Given an STS $\langle S, \longrightarrow \rangle$ and a state $s_0 \in S$, a* computation *of s_0 is a finite or infinite reduction sequence*[3]*,* i.e. *a partial function η from $\mathbb{N}$ to S whose domain is downward-closed, such that $s_0 = \eta(0)$ and for each $n \in dom(\eta) \setminus \{0\}$, $\eta(n-1) \longrightarrow \eta(n)$.* ◼

[2] In general the reduction semantics and the LTS of a calculus are defined independently, and connected via the Harmony lemma ([87], Lemma 1.4.15 page 51). We have a mechanised proof of it.
[3] Which is defined as a coinductive type in our Coq development.

A computation η is *infinite* if $dom(\eta) = \mathbb{N}$. A computation η is *maximal* if either it is infinite or it cannot be extended, *i.e.* $\eta(n_{max}) \not\longrightarrow$ where $n_{max} = max(dom(\eta))$. To formally define the MUST-preorder, we assume a decidable predicate GOOD over clients. A computation η of $s_0 = p_0 \parallel r_0$ is *successful* if there exists $n \in \mathbb{N}$ such that $\textsc{good}(\mathsf{snd}(\eta(n)))$. We assume the predicate GOOD to be preserved by output actions. To the best of our knowledge, this assumption is true in all the papers on testing theory for asynchronous calculi that rely on ad-hoc actions such as ω or ✓ to signal success. In Appendix F we show that this assumption holds for the language ACCS extended with the process 1. Moreover, when considering an equivalence on programs $\simeq$ that is compatible with transitions, in the sense of Figure 5, we assume the predicate GOOD to be preserved also by this equivalence. These assumptions are met by the frameworks in [39, 24, 57].

▶ **Definition 2** (Client satisfaction). *We write p MUST r if every maximal computation of $p \parallel r$ is successful.* ■

▶ **Definition 3** (MUST-preorder). *We let $p \sqsubseteq_{\scriptscriptstyle MUST} q$ whenever for every client r we have that p MUST r implies q MUST r.* ■

▶ **Example 4.** Consider the system $p_0 \parallel r_0$, where p_0 and r_0 are the server and client given in Figure 1. The unique maximal computation of this system is $p_0 \parallel r_0 \longrightarrow p_1 \parallel r_1 \longrightarrow p_3 \parallel r_3$. This computation is successful since it leads the client to the GOOD state r_3. Hence, client r_0 is satisfied by server p_0. Since OUTPUT-COMMUTATIVITY implies a lack of causality between the output $\overline{\texttt{str}}$ and the input `int` in the client, it is the order between the input `str` and the output $\overline{\texttt{int}}$ in the server that guides the order of client-server interactions. ◀

A closer look at Selinger axioms. Let us now discuss the axioms in Figure 2. The OUTPUT-COMMUTATIVITY axiom expresses the non-blocking behaviour of outputs: an output cannot be a cause of any subsequent transition, since it can also be executed after it, leading to the same resulting state. Hence, outputs are concurrent with any subsequent transition. The FEEDBACK axiom says that an output followed by a complementary input can also synchronise with it to produce a τ-transition. These first two axioms specify properties of outputs that are followed by another transition. Instead, the following three axioms, OUTPUT-CONFLUENCE, OUTPUT-DETERMINACY and OUTPUT-TAU, specify properties of outputs that are co-initial with another transition[4]. The OUTPUT-DETERMINACY and OUTPUT-TAU axioms apply to the case where the co-initial transition is an identical output or a τ-transition respectively, while the OUTPUT-CONFLUENCE axiom applies to the other cases. When taken in conjunction, these three axioms state that outputs cannot be in conflict with any co-initial transition, except when this is a τ-transition: in this case, the OUTPUT-TAU axiom allows for a confluent nondeterminism between the τ-transition on one side and the output followed by the complementary input on the other side.

We now explain the novel BACKWARD-OUTPUT-DETERMINACY axiom. It is the dual of OUTPUT-DETERMINACY, as it states that also backward transitions with identical outputs lead to the same state. The intuition is that if two programs arrive at the same state by removing the same message from the mailbox, then they must coincide. This axiom need not be assumed in [92] because it can be derived from Selinger axioms when modelling a calculus like ACCS equipped with a parallel composition operator $\parallel$ (see Lemma 86 in Appendix F). We use the BACKWARD-OUTPUT-DETERMINACY axiom only to prove a technical property of clients (Lemma 53) that is used to prove our completeness result.

[4] Two transitions are co-initial if they stem from the same state.

$$\begin{tikzcd} p \arrow[d, dotted, no head, "\simeq"'] & \\ q \arrow[r, "\alpha"] & q' \end{tikzcd} \quad \Rightarrow \quad \begin{tikzcd} p \arrow[r, "\alpha"] \arrow[d, dotted, no head, "\simeq"'] & p' \arrow[d, dotted, no head, "\simeq"] \\ q \arrow[r, "\alpha"] & q' \end{tikzcd}$$

Figure 5 Axiom stating that equivalence $\simeq$ is compatible with a transition relation.

Calculi. A number of asynchronous calculi [64, 25, 39, 61, 79, 88] have an LTS that enjoys the axioms in Figure 2, at least up to some structural equivalence $\equiv$. The reason is that these calculi syntactically enforce outputs to have no continuation, *i.e.* outputs can only be composed in parallel with other processes.[5]. For example, Selinger [92] shows that the axioms of Figure 2 hold for the LTS of the calculus `ACCS` (the asynchronous variant of CCS[6]) modulo bisimulation, and in Lemma 89 we prove this for the LTS of `ACCS` modulo $\equiv$:

▶ **Lemma 5.** *We have that* $\langle \mathit{ACCS}_{\equiv}, L, \longrightarrow_{\equiv}\rangle \in OF$.

To streamline reasoning modulo some (structural) equivalence we introduce the typeclass `LstEq`, whose instances are LTSs equipped with an equivalence $\simeq$ that satisfies the property in Figure 5. Defining output-buffered agents with feedback using `LtsEq` does not entail any loss of generality, because the equivalence $\simeq$ can be instantiated using the identity over the states A. Further details can be found in Appendix F.1.

When convenient we denote LTSs using the following minimal syntax for `ACCS`:

$$p, q, r ::= \overline{a} \mid g \mid p \parallel p \mid \mathsf{rec}x.p \mid x, \qquad g ::= \mathsf{0} \mid a.p \mid \tau.p \mid g + g \tag{3}$$

as well as its standard LTS[7] whose properties we discuss in detail in Appendix F. This is exactly the syntax used in [92, 24], without the operators of restriction and relabelling. Here the syntactic category g defines *guards*, *i.e.* the terms that may be used as arguments for the $+$ operator. Note that, apart from $\mathsf{0}$, only input-prefixed and τ-prefixed terms are allowed as guards, and that the output prefix operator is replaced by *atoms* $\overline{a}$. In fact, this syntax is completely justified by Selinger axioms, which, as we argued above, specify that outputs cannot cause any other action, nor be in conflict with it.

▶ **Definition 6.** *Given an LTS* $\langle A, L, \longrightarrow\rangle$ *and state* $p_0 \in A$*, a* transition sequence *of* p_0 *is a finite or infinite sequence of the form* $p_0\alpha_1p_1\alpha_2p_2\cdots$ *with* $p_i \in A$ *and* $\alpha_i \in L$*, and such that, for every* $n \geq 1$ *such that* p_n *is in the sequence we have* $p_{n-1} \xrightarrow{\alpha_n} p_n$. ■

If a transition sequence is made only of τ-transitions, it is called a *computation*, the idea being that usually τ-steps should be related to reductions via the Harmony lemma.

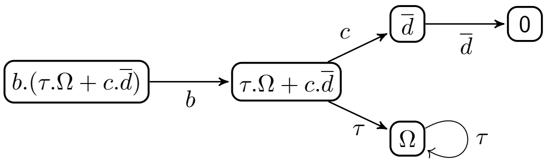
We give now an example that illustrates the use of the testing machinery in our asynchronous setting. This is also a counter-example to the completeness of the alternative preorder proposed in [39], as discussed in detail in Appendix H.

▶ **Example 7.** Let $\Omega = \mathsf{rec}x.\tau.x$ and $\mathit{Pierre} = b.(\tau.\Omega + c.\overline{d})$. The LTS of *Pierre* is as follows:

[5] In the calculus `TACCS` of [39] there is a construct of asynchronous output prefix, but its behaviour is to spawn the corresponding atom in parallel with the continuation, so it does not act as a prefix

[6] The syntax of `ACCS`, which is closely inspired by that of the asynchronous π-calculus with input- and τ-guarded choice [4, 5], is given in Equation (3) and discussed later.

[7] Where the recursion rule is replaced by the one usually adopted for testing semantics, which introduces a τ-transition before each unfolding.



Pierre models a citizen confronted with an unpopular pension reform. To begin with, *Pierre* can only do the input b, which models his getting aware of the brute-force imposition of the reform by the government. After performing the input, *Pierre* reaches the state $\tau.\Omega + c.\overline{d}$, where he behaves in a nondeterministic manner. He can internally choose not to trust the government for any positive change, in which case he will diverge, refusing any further interaction. But this need not happen: in case the government offers the action $\overline{c}$, which models a positive change in political decision, *Pierre* can decide to accept this change, and then he expresses his agreement with the output $\overline{d}$, which stands for "done". ◀

▶ **Example 8.** We prove now the inequality $\mathit{Pierre} \sqsubseteq_{\text{MUST}} \mathsf{0}$ by leveraging the possibility of divergence of *Pierre* after the input b. Fix an r such that $\mathit{Pierre}\ \textsc{must}\ r$. We distinguish two cases, according to whether $r \xrightarrow{\overline{b}}$ or $r \not\xrightarrow{\overline{b}}$.

i) Let $r \xrightarrow{\overline{b}} r'$ for some r'. Consider the maximal computation $\mathit{Pierre} \parallel r \longrightarrow \tau.\Omega + c.\overline{d} \parallel r' \longrightarrow \Omega \parallel r' \longrightarrow \ldots$ in which *Pierre* diverges and r does not move after the first output. Since $\mathit{Pierre}\ \textsc{must}\ r$, either $\textsc{good}(r)$ or $\textsc{good}(r')$. In case $\textsc{good}(r')$, by Lemma 86 we get also $\textsc{good}(r)$. Hence $\mathsf{0}\ \textsc{must}\ r$.

ii) Let $r \not\xrightarrow{\overline{b}}$. Suppose $r = r_0 \xrightarrow{\tau} r_1 \xrightarrow{\tau} r_2 \xrightarrow{\tau} \ldots$ is a maximal computation of r. Then $\mathit{Pierre} \parallel r$ has a maximal computation $\mathit{Pierre} \parallel r_0 \longrightarrow \mathit{Pierre} \parallel r_1 \longrightarrow \mathit{Pierre} \parallel r_2 \longrightarrow \ldots$. As $\mathit{Pierre}\ \textsc{must}\ r$, there must exist an $i \in \mathbb{N}$ such that $\textsc{good}(r_i)$. Hence $\mathsf{0}\ \textsc{must}\ r$. ◀

The argument in Example 8 can directly use Definition (3) because it is very simple to reason on the process $\mathsf{0}$. The issues brought about by the contextuality of Definition (3), though, hinder showing general properties of $\sqsubseteq_{\text{MUST}}$. Even proving the following seemingly obvious fact is already cumbersome:

$$\tau.(\overline{a} \parallel \overline{b}) + \tau.(\overline{a} \parallel \overline{c}) \sqsubseteq_{\text{MUST}} \overline{a} \parallel (\tau.\overline{b} + \tau.\overline{c}) \tag{4}$$

This motivates the study of alternative characterisations for $\sqsubseteq_{\text{MUST}}$, and in the rest of the paper we present two preorders that fit the purpose, and let us establish Equation (4).

We conclude this section by recalling auxiliary and rather standard notions: given an LTS $\langle A, L, \longrightarrow\rangle$, the weak transition relation $p \overset{s}{\Longrightarrow} p'$, where $s \in \mathsf{Act}^\star$, is defined via the rules

[wt-refl] $p \overset{\varepsilon}{\Longrightarrow} p$

[wt-tau] $p \overset{s}{\Longrightarrow} q$ if $p \xrightarrow{\tau} p'$ and $p' \overset{s}{\Longrightarrow} q$

[wt-mu] $p \overset{\mu.s}{\Longrightarrow} q$ if $p \xrightarrow{\mu} p'$ and $p' \overset{s}{\Longrightarrow} q$

We write $p \overset{s}{\Longrightarrow}$ to mean $\exists p'.\, p \overset{s}{\Longrightarrow} p'$, where $s \in \mathsf{Act}^\star$.

We write $p \downarrow$ and say that p *converges* if every computation of p is finite, and we lift the convergence predicate to finite traces by letting the relation $\Downarrow \subseteq A \times \mathsf{Act}^\star$ be the least one that satisfies the following rules

[cnv-epsilon] $p \Downarrow \varepsilon$ if $p \downarrow$,

[cnv-mu] $p \Downarrow \mu.s$ if $p \downarrow$ and $p \overset{\mu}{\Longrightarrow} p'$ implies $p' \Downarrow s$.

To understand the next section, one should keep in mind that all the predicates defined above have an implicit parameter: the LTS of programs. By changing this parameter, we may change the meaning of the predicates. For instance, letting Ω be the ACCS process $\mathsf{rec}x.\tau.x$, in the standard LTS $\langle \mathtt{ACCS}, \longrightarrow, \mathsf{Act}_\tau \rangle$ we have $\Omega \xrightarrow{\tau} \Omega$ and $\neg(\Omega \downarrow)$, while in the LTS $\langle \mathtt{ACCS}, \emptyset, \mathsf{Act}_\tau \rangle$ we have $\Omega \stackrel{\tau}{\not\longrightarrow}$ and thus $\Omega \downarrow$. In other words, the *same* predicates can be applied to different LTSs, and since the alternative characterisations of $\sqsubseteq_{\text{MUST}}$ are defined using such predicates, they can relate different LTSs.

3 Behavioural characterisations

We first recall the definition of the standard alternative preorder $\preccurlyeq_{\mathsf{AS}}$, and show how to use it to characterise $\sqsubseteq_{\text{MUST}}$ in our asynchronous setting. Then we recall the other standard alternative preorder, namely $\preccurlyeq_{\mathsf{MS}}$, and prove that it also captures $\sqsubseteq_{\text{MUST}}$, by applying our first characterisation.

3.1 The acceptance-set approach

The *ready set* of a program p is defined as $R(p) = I(p) \cup O(p)$, and it contains all the *visible* actions that p can immediately perform. If a program p is stable, *i.e.* it cannot perform any τ-transition, we say that it is a *potential deadlock*. In general, the ready set of a potential deadlock p shows how to make p move to a different state, possibly one that can perform further computation: if $R(p) = \emptyset$ then there is no way to make p move on, while if $R(p)$ contains some action, then p is a state waiting for the environment to interact with it. Indeed, potential deadlocks are called *waiting states* in [64]. In particular, in an asynchronous setting the outputs of a potential deadlock p show how it can unlock the inputs of a client, which in turn may lead the client to a novel state that can make p move, possibly to a state that can perform further computation. A standard manner to capture all the ways out of the potential deadlocks that a program p encounters after executing a trace s is its *acceptance set*: $\mathcal{A}(p, s, \longrightarrow) = \{R(p') \mid p \stackrel{s}{\Longrightarrow} p' \stackrel{\tau}{\not\longrightarrow}\}$.

In our presentation we indicate explicitly the third parameter of $\mathcal{A}$, *i.e.* the transition relation of the LTS at hand, because when necessary we will manipulate this parameter. For any two LTSs $\mathcal{L}_A, \mathcal{L}_B$ and servers $p \in A, q \in B$, we write $\mathcal{A}(p, s, \longrightarrow_A) \ll \mathcal{A}(q, s, \longrightarrow_B)$ if for every $R \in \mathcal{A}(q, s)$ there exists $\widehat{R} \in \mathcal{A}(p, s)$ such that $\widehat{R} \subseteq R$. We can now recall the definition of the behavioural preorder à la Hennessy, $\preccurlyeq_{\mathsf{AS}}$, which is based on acceptance sets [55].

▶ **Definition 9.** *We write*

- $p \preccurlyeq_{\mathsf{cnv}} q$ *whenever* $\forall s \in \mathsf{Act}^\star.\, p \Downarrow_A s$ *implies* $q \Downarrow_B s$*,*
- $p \preccurlyeq_{\mathsf{acc}} q$ *whenever* $\forall s \in \mathsf{Act}^\star.\, p \Downarrow_A s$ *implies* $\mathcal{A}(p, s, \longrightarrow_A) \ll \mathcal{A}(q, s, \longrightarrow_B)$*,*
- $p \preccurlyeq_{\mathsf{AS}} q$ *whenever* $p \preccurlyeq_{\mathsf{cnv}} q$ *and* $p \preccurlyeq_{\mathsf{acc}} q$*.* ◼

In the synchronous setting, the behavioural preorder $\preccurlyeq_{\mathsf{AS}}$ is closely related to the denotational semantics based on Acceptance Trees proposed by Hennessy in [54, 55]. There the predicates need not be annotated with the LTS that they are used on, because those works treat a unique LTS. Castellani and Hennessy [39] show in their Example 4 that the condition on acceptance sets, *i.e.* $\preccurlyeq_{\mathsf{acc}}$, is too demanding in an asynchronous setting.

Letting $p = a.\mathsf{0}$ and $q = \mathsf{0}$, they show that $p \sqsubseteq_{\text{MUST}} q$ but $p \not\preccurlyeq_{\mathsf{AS}} q$, because $\mathcal{A}(p, \epsilon) = \{\{a\}\}$ and $\mathcal{A}(q, \epsilon) = \{\emptyset\}$, and corresponding to the ready set $\emptyset \in \mathcal{A}(q, \epsilon)$ there is no ready set $\widehat{R} \in \mathcal{A}(p, s)$ such that $\widehat{R} \subseteq \emptyset$. Intuitively this is the case because acceptance sets treat inputs and outputs similarly, while in an asynchronous setting only outputs can be tested.

Nevertheless $\preccurlyeq_{\mathsf{AS}}$ characterises $\sqsubseteq_{\text{MUST}}$, if servers are enhanced as with forwarding. We now introduce this concept.

Forwarders. We say that an LTS $\mathcal{L}$ is of output-buffered agents *with forwarding*, for short is OW, if it satisfies all the axioms in Figure 2 except FEEDBACK, and also the two following axioms:

$$
\begin{array}{cc}
\begin{tikzcd} p \arrow[r, bend left, "a"] & p' \arrow[l, bend left, "\overline{a}"] \end{tikzcd}
&
\begin{tikzcd} p \arrow[r, "\overline{a}"] & p' \arrow[d, "a"] \\ & q \end{tikzcd} \Rightarrow\ p \xrightarrow{\tau} q \text{ or } p = q \\
\textsc{Input-Boomerang} & \textsc{Fwd-Feedback}
\end{array}
\tag{5}
$$

The INPUT-BOOMERANG axiom states a kind of input-enabledness property, which is however more specific as it stipulates that the target state of the input should loop back to the source state via a complementary output. This is the essence of the behaviour of a forwarder, whose role is simply to pass on a message and then get back to its original state. The FWD-FEEDBACK axiom is a weak form of Selinger's FEEDBACK axiom, which is better understood in conjunction with the INPUT-BOOMERANG axiom: if the sequence of transitions $p \xrightarrow{\overline{a}} p' \xrightarrow{a} q$ in the FWD-FEEDBACK axiom is taken to be the sequence of transitions $p' \xrightarrow{\overline{a}} p \xrightarrow{a} p'$ in the INPUT-BOOMERANG axiom, then we see that it must be $q = p$ in the FWD-FEEDBACK axiom. Moreover, no τ action is issued when moving from p to q, since no synchronisation occurs in this case: the message is just passed on.

We mechanise all this via the typeclass `LtsObaFW`. The overall structure of our typeclasses to reason on LTSs is thus `Lts` $\geq$ `LtsEq` $\geq$ `LtsOba` and `LtsOba` is a super-class of both `LtsObaFB` and `LtsObaFW`. We defer the details to Appendix I.

To prove that $\preccurlyeq_{\mathsf{AS}}$ is sound and complete with respect to $\sqsubseteq_{\text{MUST}}$:

1. we define an operation to lift any LTS $\mathcal{L} \in$ OF into a suitable LTS $\mathcal{L}_{\mathsf{fw}} \in$ OW, and
2. we check the predicates $\Downarrow$ and $\mathcal{A}(-,-,-)$ over the LTS $\mathcal{L}_{\mathsf{fw}}$.

Let MO denote the set of all finite multisets of output actions, for instance we have $\varnothing, \{\!|\overline{a}|\!\}, \{\!|\overline{a}, \overline{a}|\!\}, \{\!|\overline{a}, \overline{b}, \overline{a}, \overline{b}|\!\} \in MO$. We let $M, N, \ldots$ range over MO. The symbol M stands for *mailbox*. We denote with $\uplus$ the multiset union. We assume a function $\mathsf{mbox} : A \to MO$ defined for any LTS $\mathcal{L}_A$ of output-buffered agents such that

(i) $\overline{a} \in O(p)$ if and only if $\overline{a} \in \mathsf{mbox}(p)$, and
(ii) for every p', if $p \xrightarrow{\overline{a}} p'$ then $\mathsf{mbox}(p) = \{\!|\overline{a}|\!\} \uplus \mathsf{mbox}(p')$.

Note that by definition $\mathsf{mbox}(p)$ is a finite multiset.

▶ **Definition 10.** *Let $FW(\mathcal{L}) = \langle A \times MO, L, \longrightarrow_{\mathsf{fw}}\rangle$ for every LTS $\mathcal{L} = \langle A, L, \longrightarrow\rangle$, where the states in $FW(\mathcal{L})$ are pairs denoted $p \triangleright M$, such that $p \in A$ and $M \in MO$, and the transition relation $\longrightarrow_{\mathsf{fw}}$ is defined via the rules in Figure 6.* ■

▶ **Example 11.** If a calculus is fixed, then the function FW may have a simpler definition. For instance Castellani and Hennessy [39] define it in their calculus TACCS by letting $\xrightarrow{\alpha}_{\mathsf{fw}}$ be the least relation over TACCS such that (1) for every $\alpha \in \mathsf{Act}_\tau$. $\xrightarrow{\alpha} \subseteq \xrightarrow{\alpha}_{\mathsf{fw}}$, and (2) for every $a \in \mathcal{N}. p \xrightarrow{a}_{\mathsf{fw}} p \parallel \overline{a}$. ◀

The transition relation $\longrightarrow_{\mathsf{fw}}$ is reminiscent of the one introduced in Definition 8 by Honda and Tokoro in [64]. The construction given in our Definition (10), though, does not yield the LTS of Honda and Tokoro, as $\longrightarrow_{\mathsf{fw}}$ adds the forwarding capabilities to the states only at the top-level, instead of descending structurally into terms. As a consequence, in the LTS of [64] $a.\,0 + 0 \xrightarrow{b} \overline{b}$, while $a.\,0 + 0 \not\xrightarrow{b}_{\mathsf{fw}} \overline{b}$.

$$\frac{p \xrightarrow{\alpha} p'}{p \triangleright M \xrightarrow{\alpha}_{\mathsf{fw}} p' \triangleright M} \qquad \frac{p \xrightarrow{a} p'}{p \triangleright (\{\!|\overline{a}|\!\} \uplus M) \xrightarrow{\tau}_{\mathsf{fw}} p' \triangleright M}$$

$$\frac{}{p \triangleright (\{\!|\overline{a}|\!\} \uplus M) \xrightarrow{\overline{a}}_{\mathsf{fw}} p \triangleright M} \qquad \frac{}{p \triangleright M \xrightarrow{a}_{\mathsf{fw}} p \triangleright (\{\!|\overline{a}|\!\} \uplus M)}$$

Figure 6 Lifting of a transition relation to transitions of forwarders.

▶ **Example 12.** As the set $\mathcal{N}$ is countable, every process p in the LTS $\langle \texttt{ACCS} \times MO, \mathsf{Act}_\tau, \longrightarrow_{\mathsf{fw}} \rangle$ is infinitely-branching, for instance for every p and every input μ we have $p \xrightarrow{\mu} p \parallel \overline{\mu}$, hence $p \xrightarrow{a_0} p \parallel \overline{a_0}$, $p \xrightarrow{a_1} p \parallel \overline{a_1}$, $p \xrightarrow{a_2} p \parallel \overline{a_2}$, ... ◀

The intuition behind Definition (10) is that, when a client interacts with a server asynchronously, the client can send any message it likes, regardless of the inputs that the server can actually perform. In fact, asynchronous clients behave as if the server was saturated with *forwarders*, namely processes of the form $a.\overline{a}$, for any $a \in \mathcal{N}$.
We are ready to state two main properties of the function FW: it lifts any LTS of output-buffered agents with feedback to an LTS of forwarders, and the lifting preserves the MUST predicate. We can therefore reason on $\sqsubseteq_{\text{MUST}}$ using LTSs of forwarders.

▶ **Lemma 13.** *For every LTS $\mathcal{L} \in OF$, $FW(\mathcal{L}) \in OW$.*

▶ **Lemma 14.** *For every $\mathcal{L}_A, \mathcal{L}_B, \mathcal{L}_C \in OF, p \in A, q \in B, r \in C$,*
1. *p MUST_i r if and only if $FW(p)$ MUST_i r,*
2. *$p \sqsubseteq_{\text{MUST}} q$ if and only if $FW(p) \sqsubseteq_{\text{MUST}} FW(q)$.*

We now simplify the definition of acceptance sets to reason on forwarders: for any two LTS $\mathcal{L}_A, \mathcal{L}_B \in$ OW and servers $p \in A$, and $q \in B$ we let $\mathcal{A}_{\mathsf{fw}}(p, s, \longrightarrow) = \{O(p') \mid p \stackrel{s}{\Longrightarrow} p' \stackrel{\tau}{\not\longrightarrow}\}$. This definition suffices to characterise $\sqsubseteq_{\text{MUST}}$ because in each LTS that is OW every state performs every input, thus comparing inputs has no impact on the preorder $\preccurlyeq_{\mathsf{acc}}$ of Definition (9). More formally, for every $\mathcal{L}_A, \mathcal{L}_B \in$ OW and every $p \in A$ and $q \in B$, we let

$$p \preccurlyeq^{\mathsf{fw}}_{\mathsf{acc}} q \text{ iff } \forall s \in \mathsf{Act}^\star.\, p \Downarrow s \text{ implies } \mathcal{A}_{\mathsf{fw}}(p, s, \longrightarrow_A) \ll \mathcal{A}_{\mathsf{fw}}(q, s, \longrightarrow_B)$$

Then we have the following logical equivalence.

▶ **Lemma 15.** *Let $\mathcal{L}_A, \mathcal{L}_B \in OW$. For every $p \in A, q \in B, p \preccurlyeq_{\mathsf{acc}} q$ if and only if $p \preccurlyeq^{\mathsf{fw}}_{\mathsf{acc}} q$.*

Proof. The *only if* implication is trivial, so we discuss the *if* one. Suppose that $p \preccurlyeq^{\mathsf{fw}}_{\mathsf{acc}} q$ and that for some s we have that $R \in \mathcal{A}(q, s, \longrightarrow_B)$. Let X be the possibly empty subset of R that contains only output actions. Note that since $\mathcal{L}_B$ is OW we know by definition that $R = X \cup \mathcal{N}$. By definition $X \in \mathcal{A}_{\mathsf{fw}}(q, s, \longrightarrow_B)$, and thus by hypothesis there exists some set of output actions $Y \in \mathcal{A}_{\mathsf{fw}}(p, s, \longrightarrow_A)$ such that $Y \subseteq X$. It follows that the set $Y \cup \mathcal{N} \in \mathcal{A}(p, s, \longrightarrow_A)$, and trivially $Y \cup \mathcal{N} \subseteq X \cup \mathcal{N} = R$. ◀

In view of the second point of Lemma 14, to prove completeness it suffices to show that $\preccurlyeq_{\mathsf{AS}}$ includes $\sqsubseteq_{\text{MUST}}$ in the LTS of forwarders. This is indeed true:

▶ **Lemma 16.** *For every $\mathcal{L}_A, \mathcal{L}_B \in OW$ and servers $p \in A, q \in B$, if $p \sqsubseteq_{\text{MUST}} q$ then $p \preccurlyeq_{\mathsf{AS}} q$.*

By a slight abuse of notation, given an LTS $\mathcal{L} = \langle A, L, \longrightarrow\rangle$ and a state $p \in A$, we denote with $\mathrm{FW}(p)$ the LTS rooted at $p \triangleright \varnothing$ in $\mathrm{FW}(\mathcal{L})$.

▶ **Theorem 17.** *For every $\mathcal{L}_A, \mathcal{L}_B \in OF$ and $p \in A, q \in B$, $p \sqsubseteq_{\text{MUST}} q$ if and only if $FW(p) \preccurlyeq_{\mathsf{AS}} FW(q)$.*

The proof of completeness is given in Appendix C, where the main aim is to show Lemma 16. The proof of soundness, instead, requires much more auxiliary machinery than the one used to state Lemma 16, so we defer it entirely to Appendix D. Here we highlight the major novelty with respect to the literature, via a little digression. All the soundness arguments for behavioural characterisations of $\sqsubseteq_{\text{MUST}}$ in non-deterministic settings, for instance [44, 57, 59, 23, 14] but to cite a few, are rooted in classical logic, because they (1) unzip maximal computations of $p \parallel r \longrightarrow \cdots$ to produce traces $p \stackrel{s}{\Longrightarrow}$ and $r \stackrel{\overline{s}}{\Longrightarrow}$ that may be infinite; (2) use the excluded middle on an undecidable property, namely the infinity of the traces at hand; and (3) in case of infinite traces apply Kőnig's lemma (see for instance lemmas 4.4.12 and 4.4.13 of [55]). Our proof replaces Kőnig's lemma with induction and works on infinite branching STS. This is possible thanks to the bar-induction principle, which we outline in Section 4.

From Lemma 5 and Theorem 17 we immediately get a characterisation of $\sqsubseteq_{\text{MUST}}$ for `ACCS`:

▶ **Corollary 18.** *For every $p, q \in \mathtt{ACCS}_{\equiv}$, $p \sqsubseteq_{\text{MUST}} q$ if and only if $FW(p) \preccurlyeq_{\mathsf{AS}} FW(q)$.*

In Appendix E we present what, to the best of our knowledge, are the first behavioural characterisations of the MUST-preorder that fully exploit asynchrony, *i.e.* disregard irrelevant (that is, non-causal) orders of visible actions in traces. Due to space constraints, here we omit these additional results.

3.2 The must-set approach

As first application of Theorem 17, we prove that the second standard way to characterise the preorder $\sqsubseteq_{\text{MUST}}$, *i.e.* the one based on MUST-sets, is indeed sound and complete.

For every $X \subseteq_{fin} \mathsf{Act}$, that is for every finite set of visible actions, with a slightly abuse of notation we write p MUST X whenever $p \stackrel{\varepsilon}{\Longrightarrow} p'$ implies that $p' \stackrel{\mu}{\longrightarrow}$ for some $\mu \in X$, and we say that X is a MUST-set of p. Let $(p\,\mathsf{after}\,s, \longrightarrow) = \{p' \mid p \stackrel{s}{\Longrightarrow} p'\}$. For every $\mathcal{L}_A, \mathcal{L}_B$ and $p \in A, q \in B$, let $p \preccurlyeq_{\mathrm{M}} q$ whenever $\forall s \in \mathsf{Act}^\star$ we have that $p \Downarrow s$ implies that ($\forall X \subseteq_{fin} \mathsf{Act}$ if $(p\,\mathsf{after}\,s, \longrightarrow_A)$ MUST X then $(q\,\mathsf{after}\,s, \longrightarrow_B)$ MUST X).

▶ **Definition 19.** *For every $\mathcal{L}_A, \mathcal{L}_B \in OF$, and server $p \in A$ and $q \in B$ we let $p \preccurlyeq_{\mathsf{MS}} q$ whenever $p \preccurlyeq_{\mathsf{cnv}} q \wedge p \preccurlyeq_M q$.* ■

▶ **Lemma 20.** *Let $\mathcal{L}_A, \mathcal{L}_B \in OF$. For every $p \in A, q \in B$ such that $FW(p) \preccurlyeq_{\mathsf{cnv}} FW(q)$, we have that $FW(p) \preccurlyeq_M FW(q)$ if and only if $FW(p) \preccurlyeq^{\mathsf{fw}}_{\mathsf{acc}} FW(q)$.*

As a direct consequence, we obtain our second result.

▶ **Theorem 21.** *Let $\mathcal{L}_A, \mathcal{L}_B \in OF$. For every $p \in A$ and $q \in B$, we have that $p \sqsubseteq_{\text{MUST}} q$ if and only if $FW(p) \preccurlyeq_{\mathsf{MS}} FW(q)$.*

Failure refinement. MUST-sets have been used mainly by De Nicola and collaborators, for instance in [45, 24], and are closely related to the failure refinement proposed in [34] by Hoare, Brookes and Roscoe for TCSP (the process algebra based on Hoare's language CSP

[63, 32]). Following [34], a *failure* of a process p is a pair (s, X) such that $p \stackrel{s}{\Longrightarrow} p'$ and $p' \stackrel{\mu}{\not\to}$ for all $\mu \in X$. Then, failure refinement is defined by letting $p \leq_{\mathsf{fail}} q$ whenever the failures of q are also failures of p. This refinement was designed to give a denotational semantics to processes, and mechanisations in Isabelle/HOL have been developed to ensure that the refinement is well defined [93, 12]. Both Hennessy [55, pag. 260] and [38] highlight that the failure model can be justified operationally via the MUST testing equivalence: it is folklore dating back to [44, Section 4] that failure equivalence and $\eqsim$ coincide. Thanks to Theorem 21 we conclude that in fact $\sqsubseteq_{\text{MUST}}$ coincides with $\leq_{\mathsf{fail}}$ in conjunction with $\preccurlyeq_{\mathsf{cnv}}$.[8]

▶ **Corollary 22.** *Let $\mathcal{L}_A, \mathcal{L}_B \in OF$. For every $p \in A$ and $q \in B$, we have that $p \sqsubseteq_{\text{MUST}} q$ if and only if $FW(p) \preccurlyeq_{\mathsf{cnv}} FW(q)$ and $FW(p) \leq_{\mathsf{fail}} FW(q)$.*

4 Bar-induction: from extensional to intensional definitions

Two predicates are crucial to reason on the MUST-preorder, namely passing a test, *i.e.* MUST, and convergence, *i.e.* $\downarrow$. Both predicates are defined in an *extensional* manner, *i.e.* by requiring that for every infinite sequence there exists a state that is in some sense good. These are respectively the predicate GOOD in the definition of MUST and the predicate of stability, *i.e.* $\not\to$, in the definition of convergence.

Both extensional predicates can actually be defined inductively, following an *intensional* approach. Let int_Q be the inductive predicate (least fixpoint) defined by the following rules:

$$\text{[AXIOM]}\ \frac{Q(s)}{\mathsf{int}_Q(s)} \qquad \text{[IND-RULE]}\ \frac{s \to \qquad \forall s'.\, s \to s' \text{ implies } \mathsf{int}_Q(s')}{\mathsf{int}_Q(s)}$$

and we define our inductive predicates via int by letting $p \downarrow_i \stackrel{\mathsf{def}}{=} \mathsf{int}_{Q_1}(p)$ and $p\,\text{MUST}_i\,r \stackrel{\mathsf{def}}{=} \mathsf{int}_{Q_2}(p, r)$, where $Q_1(p) \stackrel{\mathsf{def}}{=} p \not\to$ and $Q_2(p, r) \stackrel{\mathsf{def}}{=} \text{GOOD}(r)$.

While proving that the intensional predicates (MUST_i and $\downarrow_i$) imply the extensional ones (MUST and $\downarrow$) are easy arguments by induction, proving the converse implications is a known problem. Its constructive solution rests on either the fan-theorem or the bar-induction principle. The first applies to finite branching trees, while the second to countably infinite branching trees. We favour bar-induction because in calculi like infinitary CCS computations can form countably branching trees.

▶ **Proposition 23.** *Given a countably branching STS $\langle S, \to \rangle$, and a decidable predicate Q on S, for all $s \in S$, $\mathsf{ext}_Q(s)$ implies $\mathsf{int}_Q(s)$.*

▶ **Corollary 24.** *For every $p \in A$, (1) $p \downarrow$ if and only if $p \downarrow_i$, (2) for every r we have that $p\,\text{MUST}\,r$ if and only if $p\,\text{MUST}_i\,r$.*

Thanks to this corollary, in the proofs of the characterisations of $\sqsubseteq_{\text{MUST}}$, and in our code, we use the predicates MUST_i and $\downarrow_i$. In other terms, we reason by induction.

The details about bar-induction, our mechanisation, and the proofs of the above results are deferred to Appendix A.

[8] The preorder becomes then the "failure divergence" refinement formalised as $\sqsubseteq_{\text{FD}}$ in `https://www.isa-afp.org/sessions/hol-csp/#Process_Order.html`.

5 Conclusion

In this paper we have shown that the standard characterisations of the MUST-preorder by De Nicola and Hennessy [44, 55] are sound and complete also in an asynchronous setting, provided servers are enhanced with the forwarding ability. Lemma 13 shows that this lifting is always possible. Our results are supported by the first mechanisation of the MUST-preorder, and increase proof (i.e. code) factorisation and reusability since the alternative preorders do not need to be changed when shifting between synchronous and asynchronous semantics: it is enough to parametrise the proofs on the set of non-blocking actions. Corollary 22 states that MUST-preorder and failure refinement essentially coincide. This might spur further interest in the mechanisations of the latter [93, 12], possibly leading to a joint development.

Proof method for must-preorder. Theorem 17 and Theorem 21 endow researchers in programming languages for message-passing software with a proof method for $\sqsubseteq_{\text{MUST}}$, namely: to define an LTS that enjoys the axioms of output-buffered agents with feedback for the language at issue. A concrete example of this approach is Corollary 18.

Live programs have barred trees. We argued that a proof of $p \,\text{MUST}\, r$ is a proof of liveness (of the client). This paper is thus de facto an exemple that proving liveness amounts to prove that a computational tree has a bar (identified by the predicate GOOD), and hence bar-induction is a natural way to reason constructively on liveness-preserving manipulations. While this fact seems to be by and large unexploited by the PL community, we believe that it may be of interest to practitioners reasoning on liveness properties in theorem provers in particular, and to the PL community at large.

Mechanisation. As observed by Boreale and Gadducci [22], the MUST-preorder lacks a tractable proof method. We thus argue that our contributions, being fully mechanised in Coq, are crucial to pursue non-trivial results about testing preorders for real-world programming languages. Our mechanisation lowers the barrier to entry for researchers versed into theorem provers and wishing to use testing preorders; adds to the toolkit of Coq users an alternative to the well-known (and already mechanised) bisimulation equivalence [80]; and provides a starting point for researchers willing to study testing preorders and analogous refinements within type theory. Our code is open-source and available on-line. Practitioners working on testing preorders may benefit from it, as there are analogies between reasoning techniques for MAY, MUST, COMPLIANCE, SHOULD, and FAIR testing. For instance Baldan et al. show with pen and paper that a technique similar to forwarding works to characterise the MAY-preorder [8].

Future work. Thanks to Theorem 17 and Theorem 21 we can now set out to (1) develop a coinductive characterisation for $\sqsubseteq_{\text{MUST}}$ adapting the one in [2, 17]; (2) devise an axiomatisation of $\sqsubseteq_{\text{MUST}}$ for asynchronous calculi, as done in [59, 23, 55, 56] for synchronous ones; (3) study for which asynchronous calculi $\sqsubseteq_{\text{MUST}}$ is a pre-congruence; (4) machine-check semantic models of subtyping for session types [17]; (5) study the decidability of $\sqsubseteq_{\text{MUST}}$. We conjecture that in Selinger asynchronous setting the MUST-preorder is undecidable.

Related work. Appendix G contains a detailed discussion of related works. Here we highlight that the notion of forwarder was outlined in the original paper on testing-preorders for asynchrony [39], and then used in the saturated LTS of [8] to reason on the MAY-preorder, and in [95] to reason on a version of the MUST-preorder parametrised on the set of tests. Forwarders, also called “links”, have applications outside of testing theory, as shown by [73] and the recent [49]. Characterising $\sqsubseteq_{\text{MUST}}$ directly on LTSs instead of calculi was suggested already in [55, 13]. Selinger axioms, discussed also by [9], are crucial in our completeness proof. Brouwer bar-induction principle is paramount to prove soundness constructively.

References

1 Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfsdóttir, and Karoliina Lehtinen. Adventures in monitorability: from branching to linear time and back again. *Proc. ACM Program. Lang.*, 3(POPL):52:1–52:29, 2019. doi:10.1145/3290365.

2 Luca Aceto and Matthew Hennessy. Termination, Deadlock, and Divergence. *J. ACM*, 39(1):147–187, 1992. doi:10.1145/147508.147527.

3 Reynald Affeldt and Naoki Kobayashi. A Coq Library for Verification of Concurrent Programs. In Carsten Schürmann, editor, *Proceedings of the Fourth International Workshop on Logical Frameworks and Meta-Languages, LFM@IJCAR 2004. Cork, Ireland, July 5, 2004*, volume 199 of *Electronic Notes in Theoretical Computer Science*, pages 17–32. Elsevier, 2004. URL: https://doi.org/10.1016/j.entcs.2007.11.010, doi:10.1016/J.ENTCS.2007.11.010.

4 Roberto M. Amadio, Ilaria Castellani, and Davide Sangiorgi. On Bisimulations for the Asynchronous pi-calculus. In U. Montanari and V. Sassone, editors, *Proceedings CONCUR 96,* Pisa, volume 1119 of *Lecture Notes in Computer Science*, pages 147–162. Springer Verlag, 1996.

5 Roberto M. Amadio, Ilaria Castellani, and Davide Sangiorgi. On Bisimulations for the Asynchronous pi-calculus. *Theoretical Computer Science*, 195:291–324, 1998.

6 Anish Athalye. CoqIOA: A formalization of IO Automata in the Coq Proof Assistant. Master's thesis, Massachusetts Institute of Technology, June 2017.

7 Clément Aubert and Daniele Varacca. Processes against tests: On defining contextual equivalences. *J. Log. Algebraic Methods Program.*, 129:100799, 2022. URL: https://doi.org/10.1016/j.jlamp.2022.100799, doi:10.1016/J.JLAMP.2022.100799.

8 Paolo Baldan, Filippo Bonchi, Fabio Gadducci, and Giacoma Valentina Monreale. Asynchronous Traces and Open Petri Nets. In Chiara Bodei, Gian-Luigi Ferrari, and Corrado Priami, editors, *Programming Languages with Applications to Biology and Security - Essays Dedicated to Pierpaolo Degano on the Occasion of His 65th Birthday*, volume 9465 of *Lecture Notes in Computer Science*, pages 86–102. Springer, 2015. doi:10.1007/978-3-319-25527-9_8.

9 Paolo Baldan, Filippo Bonchi, Fabio Gadducci, and Giacoma Valentina Monreale. Concurrency cannot be observed, asynchronously. *Math. Struct. Comput. Sci.*, 25(4):978–1004, 2015. doi:10.1017/S0960129513000108.

10 Franco Barbanera and Ugo de'Liguoro. Two notions of sub-behaviour for session-based client/server systems. In Temur Kutsia, Wolfgang Schreiner, and Maribel Fernández, editors, *Proceedings of the 12th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, July 26-28, 2010, Hagenberg, Austria*, pages 155–164. ACM, 2010. doi:10.1145/1836089.1836109.

11 Henk Barendregt and Giulio Manzonetto. *A Lambda Calculus Satellite*. College Publications, 2022. Chapter 11. URL: https://www.collegepublications.co.uk/logic/mlf/?00035.

12 James Baxter, Pedro Ribeiro, and Ana Cavalcanti. Sound reasoning in tock-CSP. *Acta Informatica*, 59(1):125–162, 2022. doi:10.1007/s00236-020-00394-3.

13 Giovanni Bernardi. *Behavioural equivalences for Web services*. PhD thesis, Trinity College Dublin, 2013. URL: http://www.tara.tcd.ie/handle/2262/77595.

14 Giovanni Bernardi and Matthew Hennessy. Mutually Testing Processes. *Log. Methods Comput. Sci.*, 11(2), 2015. doi:10.2168/LMCS-11(2:1)2015.

15 Giovanni Bernardi and Matthew Hennessy. Using higher-order contracts to model session types. *Log. Methods Comput. Sci.*, 12(2), 2016. doi:10.2168/LMCS-12(2:10)2016.

16 Giovanni Tito Bernardi and Adrian Francalanza. Full-abstraction for client testing preorders. *Sci. Comput. Program.*, 168:94–117, 2018. doi:10.1016/j.scico.2018.08.004.

17 Giovanni Tito Bernardi and Matthew Hennessy. Modelling session types using contracts. *Math. Struct. Comput. Sci.*, 26(3):510–560, 2016. doi:10.1017/S0960129514000243.

18 Gérard Berry and Gérard Boudol. The Chemical Abstract Machine. *Theor. Comput. Sci.*, 96(1):217–248, 1992. doi:10.1016/0304-3975(92)90185-I.

19 Aleš Bizjak, Lars Birkedal, and Marino Miculan. A model of countable nondeterminism in guarded type theory. In Gilles Dowek, editor, *Rewriting and Typed Lambda Calculi*, pages 108–123, Cham, 2014. Springer International Publishing.

20 Filippo Bonchi, Georgiana Caltais, Damien Pous, and Alexandra Silva. Brzozowski's and Up-To Algorithms for Must Testing. In Chung-chieh Shan, editor, *Programming Languages and Systems - 11th Asian Symposium, APLAS 2013, Melbourne, VIC, Australia, December 9-11, 2013. Proceedings*, volume 8301 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2013. `doi:10.1007/978-3-319-03542-0\_1`.

21 Filippo Bonchi, Ana Sokolova, and Valeria Vignudelli. The Theory of Traces for Systems with Nondeterminism, Probability, and Termination. *Log. Methods Comput. Sci.*, 18(2), 2022. URL: `https://doi.org/10.46298/lmcs-18(2:21)2022`, `doi:10.46298/LMCS-18(2:21)2022`.

22 Michele Boreale and Fabio Gadducci. Processes as formal power series: A coinductive approach to denotational semantics. *Theor. Comput. Sci.*, 2006. `doi:10.1016/j.tcs.2006.05.030`.

23 Michele Boreale and Rocco De Nicola. Testing Equivalence for Mobile Processes. *Inf. Comput.*, 120(2):279–303, 1995. `doi:10.1006/inco.1995.1114`.

24 Michele Boreale, Rocco De Nicola, and Rosario Pugliese. Trace and Testing Equivalence on Asynchronous Processes. *Inf. Comput.*, 172(2):139–164, 2002. `doi:10.1006/inco.2001.3080`.

25 Gérard Boudol. Asynchrony and the Pi-calculus. Research Report RR-1702, INRIA, 1992. URL: `https://hal.inria.fr/inria-00076939`.

26 Gérard Boudol. The pi-Calculus in Direct Style. *High. Order Symb. Comput.*, 11(2):177–208, 1998. `doi:10.1023/A:1010064516533`.

27 Gérard Boudol and Carolina Lavatelli. Full Abstraction for Lambda Calculus with Resources and Convergence Testing. In Hélène Kirchner, editor, *Trees in Algebra and Programming - CAAP'96, 21st International Colloquium, Linköping, Sweden, April, 22-24, 1996, Proceedings*, volume 1059 of *Lecture Notes in Computer Science*, pages 302–316. Springer, 1996. `doi:10.1007/3-540-61064-2\_45`.

28 M. Bravetti, J. Lange, and G. Zavattaro. Fair Refinement for Asynchronous Session Types. In *FOSSACS*, 2021.

29 Nuria Brede and Hugo Herbelin. On the logical structure of choice and bar induction principles. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–13. IEEE, 2021. `doi:10.1109/LICS52264.2021.9470523`.

30 Flavien Breuvart, Giulio Manzonetto, Andrew Polonsky, and Domenico Ruoppolo. New Results on Morris's Observational Theory: The Benefits of Separating the Inseparable. In Delia Kesner and Brigitte Pientka, editors, *1st International Conference on Formal Structures for Computation and Deduction, FSCD 2016, June 22-26, 2016, Porto, Portugal*, volume 52 of *LIPIcs*, pages 15:1–15:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. `doi:10.4230/LIPIcs.FSCD.2016.15`.

31 Flavien Breuvart, Giulio Manzonetto, and Domenico Ruoppolo. Relational Graph Models at Work. *Log. Methods Comput. Sci.*, 14(3), 2018. `doi:10.23638/LMCS-14(3:2)2018`.

32 Stephen D. Brookes. On the Relationship of CCS and CSP. In Josep Díaz, editor, *Automata, Languages and Programming, 10th Colloquium, Barcelona, Spain, July 18-22, 1983, Proceedings*, volume 154 of *Lecture Notes in Computer Science*, pages 83–96. Springer, 1983. `doi:10.1007/BFb0036899`.

33 Stephen D. Brookes. Deconstructing CCS and CSP Asynchronous Communication, Fairness, and Full Abstraction. 2002.

34 Stephen D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A Theory of Communicating Sequential Processes. *J. ACM*, 31(3):560–599, 1984. `doi:10.1145/828.833`.

35 Caroline Caruana. Compositional Reasoning about Actor Based Systems, 2019.

36 Giuseppe Castagna, Mariangiola Dezani-Ciancaglini, Elena Giachino, and Luca Padovani. Foundations of session types. In António Porto and Francisco Javier López-Fraguas, editors, *Proceedings of the 11th International ACM SIGPLAN Conference on Principles and Practice*

of *Declarative Programming, September 7-9, 2009, Coimbra, Portugal*, pages 219–230. ACM, 2009. doi:10.1145/1599410.1599437.

37 Giuseppe Castagna, Nils Gesbert, and Luca Padovani. A theory of contracts for Web services. *ACM Trans. Program. Lang. Syst.*, 31(5):19:1–19:61, 2009. doi:10.1145/1538917.1538920.

38 Simon Castellan, Pierre Clairambault, and Glynn Winskel. The Mays and Musts of Concurrent Strategies. In Alessandra Palmigiano and Mehrnoosh Sadrzadeh, editors, *Samson Abramsky on Logic and Structure in Computer Science and Beyond*, pages 327–361, Cham, 2023. Springer International Publishing. doi:10.1007/978-3-031-24117-8_9.

39 Ilaria Castellani and Matthew Hennessy. Testing Theories for Asynchronous Languages. In Vikraman Arvind and Ramaswamy Ramanujam, editors, *Foundations of Software Technology and Theoretical Computer Science, 18th Conference, Chennai, India, December 17-19, 1998, Proceedings*, volume 1530 of *Lecture Notes in Computer Science*, pages 90–101. Springer, 1998. doi:10.1007/978-3-540-49382-2_9.

40 Andrea Cerone and Matthew Hennessy. Process Behaviour: Formulae vs. Tests. Technical report, Trinity College Dublin, School of Computer Science and Statistics, 2010.

41 R. Cleaveland and A. E. Zwarico. A Theory of Testing for Real-Time. In *LICS*, 1991.

42 Rance Cleaveland and Matthew Hennessy. Testing Equivalence as a Bisimulation Equivalence. In Joseph Sifakis, editor, *Automatic Verification Methods for Finite State Systems, International Workshop, Grenoble, France, June 12-14, 1989, Proceedings*, volume 407 of *Lecture Notes in Computer Science*, pages 11–23. Springer, 1989. doi:10.1007/3-540-52148-8_2.

43 T. Coquand. About Brouwer's Fan Theorem. https://www.cairn-int.info/journal-revue-internationale-de-philosophie-2004-4-page-483.htm, 2003.

44 Rocco De Nicola and Matthew Hennessy. Testing Equivalences for Processes. *Theor. Comput. Sci.*, 34:83–133, 1984. doi:10.1016/0304-3975(84)90113-0.

45 Rocco De Nicola and Hernan C. Melgratti. Multiparty testing preorders. *Log. Methods Comput. Sci.*, 19(1), 2023. doi:10.46298/lmcs-19(1:1)2023.

46 Rocco De Nicola and Rosario Pugliese. Linda-based applicative and imperative process algebras. *Theor. Comput. Sci.*, 238(1-2):389–437, 2000. doi:10.1016/S0304-3975(99)00339-4.

47 D. Dreyer, G. Neis, and L. Birkedal. The impact of higher-order state and control effects on local relational reasoning. *J. Funct. Program.*, 22(4-5):477–528, 2012. doi:10.1017/S095679681200024X.

48 Michael Dummett. *Elements of Intuitionism*. Oxford logic guides. Clarendon Press, 2000. URL: https://books.google.fr/books?id=JVFzknbGBVAC.

49 Adrien Durier, Daniel Hirschkoff, and Davide Sangiorgi. Eager Functions as Processes. *Theor. Comput. Sci.*, 913:8–42, 2022. URL: https://doi.org/10.1016/j.tcs.2022.01.043, doi:10.1016/J.TCS.2022.01.043.

50 Adrian Francalanza. A theory of monitors. *Inf. Comput.*, 281:104704, 2021. doi:10.1016/j.ic.2021.104704.

51 D. Fridlender. An Interpretation of the Fan Theorem in Type Theory. In *TYPES*, 1998. URL: https://doi.org/10.1007/3-540-48167-2_7.

52 Dan Frumin, Robbert Krebbers, and Lars Birkedal. ReLoC: A Mechanised Relational Logic for Fine-Grained Concurrency. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 442–451. ACM, 2018. doi:10.1145/3209108.3209174.

53 Simon J. Gay and Malcolm Hole. Subtyping for session types in the pi calculus. *Acta Informatica*, 42(2-3):191–225, 2005. doi:10.1007/s00236-005-0177-z.

54 Matthew Hennessy. Acceptance Trees. *J. ACM*, 32(4):896–928, 1985. doi:10.1145/4221.4249.

55 Matthew Hennessy. *Algebraic theory of processes*. MIT Press series in the foundations of computing. MIT Press, 1988.

56 Matthew Hennessy. A fully abstract denotational semantics for the pi-calculus. *Theor. Comput. Sci.*, 278(1-2):53–89, 2002. doi:10.1016/S0304-3975(00)00331-5.

57 Matthew Hennessy. The security pi-calculus and non-interference. *J. Log. Algebraic Methods Program.*, 2005. doi:10.1016/j.jlap.2004.01.003.

58 Matthew Hennessy. *A distributed Pi-calculus*. Cambridge University Press, 2007.

59 Matthew Hennessy and Anna Ingólfsdóttir. Communicating Processes with Value-passing and Assignments. *Formal Aspects Comput.*, 5(5):432–466, 1993. doi:10.1007/BF01212486.

60 Matthew Hennessy and Gordon D. Plotkin. A Term Model for CCS. In Piotr Dembinski, editor, *Mathematical Foundations of Computer Science 1980 (MFCS'80), Proceedings of the 9th Symposium, Rydzyna, Poland, September 1-5, 1980*, volume 88 of *Lecture Notes in Computer Science*, pages 261–274. Springer, 1980. doi:10.1007/BFb0022510.

61 Matthew Hennessy and James Riely. Information flow vs. resource access in the asynchronous pi-calculus. *ACM Trans. Program. Lang. Syst.*, 24(5):566–591, 2002. doi:10.1145/570886.570890.

62 Daniel Hirschkoff, Guilhem Jaber, and Enguerrand Prebet. Deciding Contextual Equivalence of ν-Calculus with Effectful Contexts. In Orna Kupferman and Pawel Sobocinski, editors, *Foundations of Software Science and Computation Structures - 26th International Conference, FoSSaCS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2023, Paris, France, April 22-27, 2023, Proceedings*, volume 13992 of *Lecture Notes in Computer Science*, pages 24–45. Springer, 2023. doi:10.1007/978-3-031-30829-1_2.

63 C. A. R. Hoare. Communicating Sequential Processes (Reprint). *Commun. ACM*, 1983.

64 Kohei Honda and Mario Tokoro. An Object Calculus for Asynchronous Communication. In Pierre America, editor, *ECOOP'91 European Conference on Object-Oriented Programming, Geneva, Switzerland, July 15-19, 1991, Proceedings*, volume 512 of *Lecture Notes in Computer Science*, pages 133–147. Springer, 1991. doi:10.1007/BFb0057019.

65 Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty Asynchronous Session Types. In George C. Necula and Philip Wadler, editors, *POPL*, pages 273–284, New York, 2008. ACM Press.

66 Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty Asynchronous Session Types. *Journal of ACM*, 63(1):9:1–9:67, 2016.

67 Benedetto Intrigila, Giulio Manzonetto, and Andrew Polonsky. Degrees of extensionality in the theory of Böhm trees and Sallé's conjecture. *Log. Methods Comput. Sci.*, 15(1), 2019. doi:10.23638/LMCS-15(1:6)2019.

68 S. C. Kleene and R. E. Vesley. *The Foundations of Intuitionistic Mathematics: Especially in Relation to Recursive Functions*. Studies in logic and the foundations of mathematics. North-Holland Publishing Company, 1965. URL: https://books.google.fr/books?id=2EHVxQEACAAJ.

69 Vasileios Koutavas and Nikos Tzevelekos. Fully Abstract Normal Form Bisimulation for Call-by-Value PCF. In *LICS*, 2023.

70 Robbert Krebbers, Amin Timany, and Lars Birkedal. Interactive proofs in higher-order concurrent separation logic. In Giuseppe Castagna and Andrew D. Gordon, editors, *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, pages 205–217. ACM, 2017. doi:10.1145/3009837.3009855.

71 Leslie Lamport. *Specifying Systems, The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley, 2002. URL: http://research.microsoft.com/users/lamport/tla/book.html.

72 Cosimo Laneve and Luca Padovani. The *Must* Preorder Revisited. In Luís Caires and Vasco Thudichum Vasconcelos, editors, *CONCUR 2007 - Concurrency Theory, 18th International Conference, CONCUR 2007, Lisbon, Portugal, September 3-8, 2007, Proceedings*, volume 4703 of *Lecture Notes in Computer Science*, pages 212–225. Springer, 2007. doi:10.1007/978-3-540-74407-8_15.

73 Massimo Merro and Davide Sangiorgi. On asynchrony in name-passing calculi. *Math. Struct. Comput. Sci.*, 14(5):715–767, 2004. doi:10.1017/S0960129504004323.

74 Robin Milner. Functions as Processes. In Mike Paterson, editor, *Automata, Languages and Programming, 17th International Colloquium, ICALP90, Warwick University, England, UK, July 16-20, 1990, Proceedings*, volume 443 of *Lecture Notes in Computer Science*, pages 167–180. Springer, 1990. doi:10.1007/BFb0032030.

75 Robin Milner. *Communicating and Mobile Systems - the Pi-Calculus*. Cambridge University Press, 1999.

76 James H. Morris. *Lambda-calculus models of programming languages*. PhD thesis, Massachusetts Institute of Technology, 1969. URL: https://dspace.mit.edu/handle/1721.1/64850.

77 Keiko Nakata, Tarmo Uustalu, and Marc Bezem. A Proof Pearl with the Fan Theorem and Bar Induction - Walking through Infinite Trees with Mixed Induction and Coinduction. In Hongseok Yang, editor, *Programming Languages and Systems - 9th Asian Symposium, APLAS 2011, Kenting, Taiwan, December 5-7, 2011. Proceedings*, volume 7078 of *Lecture Notes in Computer Science*, pages 353–368. Springer, 2011. doi:10.1007/978-3-642-25318-8_26.

78 Rocco De Nicola and Matthew Hennessy. CCS without tau's. In Hartmut Ehrig, Robert A. Kowalski, Giorgio Levi, and Ugo Montanari, editors, *TAPSOFT'87: Proceedings of the International Joint Conference on Theory and Practice of Software Development, Pisa, Italy, March 23-27, 1987, Volume 1: Advanced Seminar on Foundations of Innovative Software Development I and Colloquium on Trees in Algebra and Programming (CAAP'87)*, volume 249 of *Lecture Notes in Computer Science*, pages 138–152. Springer, 1987. doi:10.1007/3-540-17660-8_53.

79 Catuscia Palamidessi. Comparing the Expressive Power of the Synchronous and Asynchronous *pi*-calculi. *Mathematical Structures in Computer Science*, 13(5):685–719, 2003. doi:10.1017/S0960129503004043.

80 Damien Pous. Coinduction All the Way Up. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 307–316. ACM, 2016. doi:10.1145/2933575.2934564.

81 Enguerrand Prebet. Functions and References in the Pi-Calculus: Full Abstraction and Proof Techniques. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 130:1–130:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ICALP.2022.130.

82 R. Pugliese. *Semantic Theories for Asynchronous Languages*. PhD thesis, Università di Roma "La Sapienza", 1996.

83 Vincent Rahli, Mark Bickford, Liron Cohen, and Robert L. Constable. Bar Induction is Compatible with Constructive Type Theory. *J. ACM*, 66(2):13:1–13:35, 2019. doi:10.1145/3305261.

84 António Ravara, Pedro Resende, and Vasco Thudichum Vasconcelos. An Algebra of Behavioural Types. *Inf. Comput.*, 212:64–91, 2012.

85 Arend Rensink and Walter Vogler. Fair testing. *Inf. Comput.*, 205(2):125–198, 2007. doi:10.1016/j.ic.2006.06.002.

86 David Sabel and Manfred Schmidt-Schauß. A call-by-need lambda calculus with locally bottom-avoiding choice: context lemma and correctness of transformations. *Math. Struct. Comput. Sci.*, 18(3):501–553, 2008. doi:10.1017/S0960129508006774.

87 Davide Sangiorgi. *Introduction to Bisimulation and Coinduction*. Cambridge University Press, 2011.

88 Davide Sangiorgi. Asynchronous pi-calculus at Work: The Call-by-Need Strategy. In Mário S. Alvim, Kostas Chatzikokolakis, Carlos Olarte, and Frank Valencia, editors, *The Art of Modelling Computational Systems: A Journey from Logic and Concurrency to Security and Privacy - Essays Dedicated to Catuscia Palamidessi on the Occasion of Her 60th Birthday*,

volume 11760 of *Lecture Notes in Computer Science*, pages 33–49. Springer, 2019. doi:10.1007/978-3-030-31175-9_3.
89 Davide Sangiorgi and David Walker. *The Pi-Calculus - a Theory of Mobile Processes*. Cambridge University Press, 2001.
90 Manfred Schmidt-Schauß and David Sabel. Correctly Implementing Synchronous Message Passing in the Pi-Calculus By Concurrent Haskell's MVars. In Ornela Dardha and Jurriaan Rot, editors, *Proceedings Combined 27th International Workshop on Expressiveness in Concurrency and 17th Workshop on Structural Operational Semantics, EXPRESS/SOS 2020, and 17th Workshop on Structural Operational SemanticsOnline, 31 August 2020*, volume 322 of *EPTCS*, pages 88–105, 2020. doi:10.4204/EPTCS.322.8.
91 Manfred Schmidt-Schauß, David Sabel, and Nils Dallmeyer. Sequential and Parallel Improvements in a Concurrent Functional Programming Language. In David Sabel and Peter Thiemann, editors, *Proceedings of the 20th International Symposium on Principles and Practice of Declarative Programming, PPDP 2018, Frankfurt am Main, Germany, September 03-05, 2018*, pages 20:1–20:13. ACM, 2018. doi:10.1145/3236950.3236952.
92 Peter Selinger. First-Order Axioms for Asynchrony. In Antoni W. Mazurkiewicz and Józef Winkowski, editors, *CONCUR '97: Concurrency Theory, 8th International Conference, Warsaw, Poland, July 1-4, 1997, Proceedings*, volume 1243 of *Lecture Notes in Computer Science*, pages 376–390. Springer, 1997. doi:10.1007/3-540-63141-0_26.
93 S. Taha, L. Ye, and B. Wolff. HOL-CSP Version 2.0. *Archive of Formal Proofs*, April 2019.
94 Erica Tanti and Adrian Francalanza. Towards Sound Refactoring in Erlang. 2015. URL: https://api.semanticscholar.org/CorpusID:63046364.
95 Prasannaa Thati. *A Theory of Testing for Asynchronous Concurrent Systems*. PhD thesis, University of Illinois Urbana-Champaign, USA, 2003. URL: https://hdl.handle.net/2142/81630.
96 Aaron Joseph Turon, Jacob Thamsborg, Amal Ahmed, Lars Birkedal, and Derek Dreyer. Logical Relations for Fine-Grained Concurrency. In Roberto Giacobazzi and Radhia Cousot, editors, *The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '13, Rome, Italy - January 23 - 25, 2013*, pages 343–356. ACM, 2013. doi:10.1145/2429069.2429111.
97 Rob van Glabbeek. Just Testing. In Orna Kupferman and Pawel Sobocinski, editors, *Foundations of Software Science and Computation Structures - 26th International Conference, FoSSaCS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2023, Paris, France, April 22-27, 2023, Proceedings*, volume 13992 of *Lecture Notes in Computer Science*, pages 498–519. Springer, 2023. doi:10.1007/978-3-031-30829-1_24.

A Bar-Induction

In this appendix we present our treatment of the bar-induction principle. Section A.1 is an informal introduction to the intuitions behind bar-induction. A reader already acquainted with this principle may read directly Section A.2.

A.1 A visual introduction

We explain the difference between *extensional* definitions of predicates and *intensional* ones, by discussing how the two different approaches make us reason on computational trees.

Suppose that we have a client-server system $p \parallel r$ and that we want to prove either$p \,\textsc{must}\, r$ or $p \,\textsc{must}_i\, r$. For both proofs, what matters is the state transition system (STS) of $p \parallel r$, i.e. the computation steps performed by the client-server system at issue. In fact it is customary to treat this STS as a computational tree, as done for instance in the proofs of [55, Lemma 4.4.12] and [40, Theorem 2.3.3]. In the rest of this subsection we discuss the tree depicted

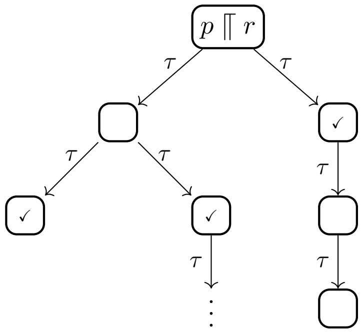


Figure 7 The state transition system of client-server system.

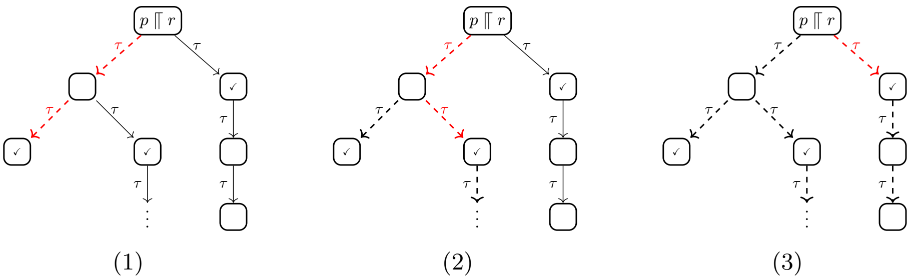


Figure 8 Extensional approach: finding successful prefixes in every maximal path of the computational tree.

813 in Figure 7. It contains three maximal computations, the middle one being infinite. In the 814 figures of this subsection, the states in which the client is successful (i.e. in the predicate 815 GOOD) contain the symbol ✓.

816 A.1.0.1 The extensional approach

817 To prove p MUST r, the extensional definition of MUST requires checking that every maximal 818 path in the tree in Figure 7 starts with a finite prefix that leads to a successful state. The 819 proof that p MUST r amounts to looking for a suitable prefix maximal path by maximal path, 820 via a loop whose iterations are suggested in Figure 8. There at every iteration a different 821 maximal path (highlighted by dashed arrows) is checked, and each time a successful prefix 822 is found (indicated by a red arrow), the loop moves on to the next maximal path. Once 823 a maximal path is explored, it remains dashed, to denote that there a succesful prefix has 824 been found. The first iteration looks for a successful prefix in the left-most maximal path, 825 while the last iteration looks for a successful prefix in the right-most path. In the current 826 example the loop terminates because the tree in Figure 7 has conveniently a finite number 827 of maximal paths, but in general the mathematical reasoning has to deal with an infinite 828 amount of maximal path. An archetypal example is the tree in Figure 10: it has countably 829 many maximal paths, each one starting with a successful prefix.

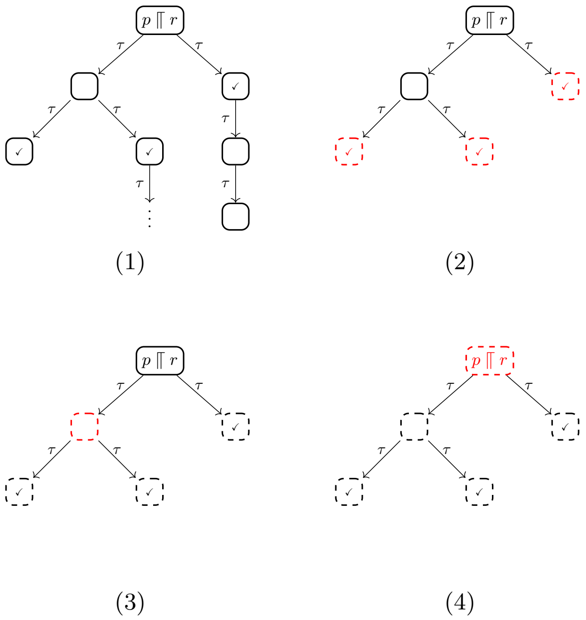


Figure 9 Intensional approach: visiting the tree bottom-up, starting from the bar.

A.1.0.2 The intensional approach

Consider now the predicate MUST_i - which is defined intensional ly - and a proof that $p\, \text{MUST}_i\, r$. The base case of MUST_i ensures that all the nodes that contain a successful client (i.e. that satisfies the predicate Q_2, defined on line 553 of the submission) are in MUST_i. Pictorially, this is the step from (1) to (2) in Figure 9, where the nodes in MUST_i are drawn using dashed borders, and the freshly added ones are drawn in red. Once the base case is established, the inductive rule of MUST_i ensures that any node that inevitably goes to nodes that are in MUST_i, is also in the predicate MUST_i. This leads to the step from (2) to (3) and then from (3) to (4). Note that the argument is concise, for in the tree the depth at which successful states can be found is finite. In general though is may not be the case. The tree in Figure 10 is again the archetypal example: every maximal path there contains a finite prefix that leads to a successful state, but there is no upper bound on the length on those prefixes.

A.1.0.3 Do extensional and intensional predicates coincide ?

Extensional and intensional definitions make us reason on computational trees in strikingly different fashions: *extensionally* we reason maximal path by maximal path, while *intensional ly* we reason bottom-up, starting from the nodes in a predicate that bars the tree.[9] It is natural to ask whether reasoning in these different manners ultimately leads to the same outcomes. In our setting this amounts to proving that the predicates MUST and MUST_i are logically equivalent, and similarly for the convergence predicates $\downarrow$ and $\downarrow_i$. The proof that $p\, \text{MUST}_i\, r$ implies $p\, \text{MUST}\, r$ is - obviously - by induction on the derivation of $p\, \text{MUST}_i\, r$. Proving that the extensional predicates imply the intensional ones is, on the other hand,

[9] Whence the name *bar*-induction.

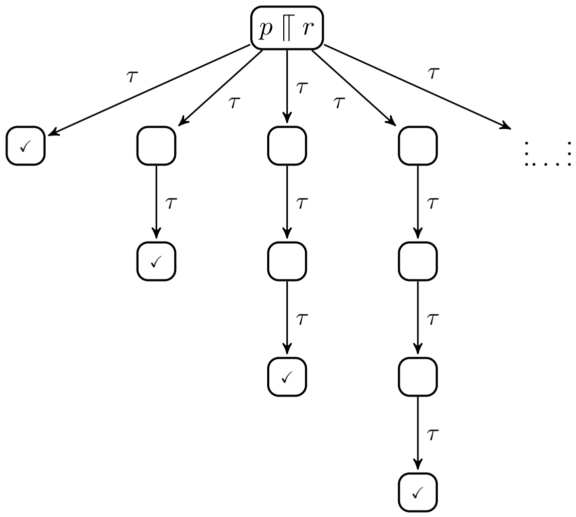


Figure 10 An infinite branching computational tree where the *bar* ✓ is at unbounded depth.

delicate, because we may have to deal with unbounded structures. The tree in Figure 10 is once more the archetypal example: it has countably many maximal paths, and there is no upper bound on the depth at which successful states (i.e. nodes in the *bar*) are found.

In classical logic one can prove that $p \,\textsc{must}\, r$ implies $p \,\textsc{must}_i\, r$ by contradiction. As we wish to avoid this reasoning principle, the only tool we have is the axiom of Bar-induction, which states exactly that under suitable hypotheses, extensionally defined predicates imply their intensional ly defined counter-parts.

A.2 Inductive definitions of predicates

We present the inductive characterisations of $\downarrow$ and $\textsc{must}$ in any state transition system (STS) $\langle S, \rightarrow \rangle$ that is countably branching. In practice, this condition is satisfied by most concrete LTS of programming languages, which usually contain countably many terms; this is the case for `ACCS` and for the asynchronous π-calculus.

Following the terminology of [29] we introduce extensional and intensional predicates associated to any decidable predicate $Q : S \rightarrow \mathbb{B}$ over an STS $\langle S, \rightarrow \rangle$.

▶ **Definition 25.** *The* extensional predicate $\mathsf{ext}_Q(s)$ *is defined, for* $s \in S$*, as*

$$\forall \eta \text{ maximal execution of } S.\, \eta_0 = s \text{ implies } \exists n \in \mathbb{N},\ Q(\eta_n)$$

The intensional predicate int_Q *is the inductive predicate (least fixpoint) defined by the following rules:*

$$[\textsc{axiom}]\ \frac{Q(s)}{\mathsf{int}_Q(s)} \qquad [\textsc{ind-rule}]\ \frac{s \rightarrow \qquad \forall s'.\, s \rightarrow s' \text{ implies } \mathsf{int}_Q(s')}{\mathsf{int}_Q(s)}$$

■

For instance, by letting

$$Q_1(p) \iff p \not\rightarrow \qquad\qquad Q_2(p, r) \iff \textsc{good}(r)$$

we have by definition that

$$p \downarrow \iff \mathsf{ext}_{Q_1}(p) \qquad\qquad p \,\text{MUST}\, r \iff \mathsf{ext}_{Q_2}(p, r) \qquad\qquad \text{(ext-preds)}$$

that is the standard definitions of $\downarrow$ and MUST are extensional. Our aim now is to prove that they coincide with their intensional counterparts. Since we will use the intensional predicates in the rest of the paper a little syntactic sugar is in order, let

$$p \downarrow_i \iff \mathsf{int}_{Q_1}(p) \qquad\qquad p \,\text{MUST}_i\, r \iff \mathsf{int}_{Q_2}(p, r) \qquad\qquad \text{(int-preds)}$$

The proofs of soundness, *i.e.* that the inductively defined predicates imply the extensional ones, are by rule induction:

▶ **Lemma 26.** *For $p \in S$,*
(a) *$p \downarrow_i$ implies $p \downarrow$,*
(b) *for every r. $p \,\text{MUST}_i\, r$ implies $p \,\text{MUST}\, r$.*

The proofs of completeness are more delicate. To the best of our knowledge, the ones about CCS [40, 13] proceed by induction on the greatest number of steps necessary to arrive at termination or at a successful state. Since the STS of $\langle \text{CCS}, \xrightarrow{\tau} \rangle$ is finite branching, Kőnig's lemma guarantees that such a bound exists. This technique does not work on infinite-branching STSs, for example the one of CCS with infinite sums [14]. If we reason in classical logic, we can prove completeness without Kőnig's lemma and also over infinite-branching STSs via a proof *ad absurdum*: suppose $p \downarrow$. If $\neg(p \downarrow_i)$ no finite derivation tree exists to prove $p \downarrow_i$, and then we construct an infinite sequence of τ moves starting with p, thus $\neg(p \downarrow)$. Since we strive to be constructive we replace reasoning *ad absurdum* with a constructive axiom: (decidable) *bar-induction*. In the rest of this section we discuss this axiom, and adapt it to our client-server setting. This requires a little terminology.

A.2.0.1 Bar-induction

The axiom we want to use is traditionally stated using natural numbers. We use the standard notations $\mathbb{N}^\star$ for finite sequences of natural numbers, $\mathbb{N}^\omega$ for infinite sequences, and $\mathbb{N}^\infty = \mathbb{N}^\star \cup \mathbb{N}^\omega$ for finite or infinite sequences. Remark that, in constructive logics, given $u \in \mathbb{N}^\infty$, we cannot do a case analysis on whether u is finite or infinite. The set $\mathbb{N}^\infty$ equipped with the prefix order can be seen as a *tree*, denoted $T_\mathbb{N}$, in the sense of set theory: a tree is an ordered set $(A, \leq)$ such that, for each $a \in A$, the set $\{b \mid b < a\}$ is well-ordered by $<$. A *path* in a tree A is a maximal element in A. In the tree $\mathbb{N}^\infty$, each node has ω children, and the paths are exactly the infinite sequences $\mathbb{N}^\omega$.

A predicate $P \subseteq \mathbb{N}^\star$ over finite words is a *bar* if every infinite sequence of natural numbers has a finite prefix in P. Note that a bar defines a subtree of $T_\mathbb{N}$ *extensionally*, because it defines each path of the tree, as a path $u \in \mathbb{N}^\omega$ is in the tree if and only if there exists a finite prefix which is in the bar P.

A predicate $Q \subseteq \mathbb{N}^\star$ is *hereditary* if

$$\forall w \in \mathbb{N}^\star, \text{ if } \forall n \in \mathbb{N}, w \cdot n \in Q \text{ then } w \in Q.$$

Bar-induction states that the extensional predicate associated to a bar implies its *intensional* counterpart: a predicate $P_{int} \subseteq \mathbb{N}^\star$ which contains Q and which is hereditary.

▶ **Axiom 27** (Decidable bar induction over $\mathbb{N}$). *Given two predicates P_{int}, Q over $\mathbb{N}^\star$, such that:*

1. *for all* $\pi \in \mathbb{N}^\omega$*, there exists* $n \in \mathbb{N}$ *such that* $(\pi_1, \ldots, \pi_n) \in Q$*;*
2. *for all* $w \in \mathbb{N}^*$*, it is decidable whether* $Q(w)$ *or* $\neg Q(w)$*;*
3. *for all* $w \in \mathbb{N}^*$*,* $Q(w) \Rightarrow P_{int}(w)$*;*
4. P_{int} *is hereditary;*

then P_{int} *holds over the empty word:* $P_{int}(\varepsilon)$*.*

Bar-induction is a generalisation of the fan theorem, i.e. the constructive version of Kőnig's lemma [48, pag. 56], and states that any extensionally well-founded tree T can be turned into an inductively-defined tree t that realises T [29, 68].

Our mechanisation of bar-induction principle is formulated as a Proposition that is proved using classical reasoning, since it is not provable directly in the type theory of Coq. This principle though has a computational content, bar recursion, which, currently, cannot be used in mainstream proof assistants such as Coq.

A.2.0.2 Admissibility.

To show that the principle is admissible, we prove that it follows from the Classical Epsilon (CE) axiom of the Coq standard library. In short, CE gives a choice function ϵ such that if p is a proof of $\exists x : A, Px$, then $\epsilon(p)$ is an element of A such that $P(\epsilon(p))$ holds. It implies Excluded Middle, and thus classical reasoning, because $A \vee \neg A$ is equivalent to $\exists b : \mathit{bool}, (b = \mathit{true} \wedge A) \vee (b = \mathit{false} \wedge \neg A)$. Since CE is guaranteed by the Coq developers to be admissible, our statement of bar-induction is also admissible.

A.2.0.3 Encoding states

The version of bar-induction we just outlined is not directly suitable for our purposes, as we need to reason about sequences of reductions rather than sequences of natural numbers. The solution is to encode STS states by natural numbers. This leads to the following issue: the nodes of the tree $T_{\mathbb{N}}$ have a fixed arity, namely $\mathbb{N}$, while processes have variably many reducts, including zero if they are stable. To deal with this glitch, it suffices to assume that there exists the following family of surjections:

$$F(p) : \mathbb{N} \to \{q \mid p \to q\} \tag{6}$$

where a surjection is defined as follows.

▶ **Definition 28.** *A map* $f : A \to B$ *is a surjection if it has a section* $g : B \to A$*, that is,* $f \circ g = \mathrm{Id}_B$*.*

This definition implies the usual one which states the existence of an antecedent $x \in A$ for any $y \in B$, and it is equivalent to it if we assume the Axiom of Choice.

Using this map F as a decoding function, any sequence of natural numbers corresponds to a path in the STS. Its subjectivity means that all paths of the LTS can be represented as such a sequence. This correspondence allows us to transport bar induction from sequences of natural numbers to executions of processes.

Note that such a family of surjections F exists for `ACCS` processes, and generally to most programming languages, because the set Act_τ is countable, and so are processes. This leads to the following version of bar-induction where words and sequences are replaced by finite and infinite executions.

▶ **Proposition 29** (Decidable bar induction over an STS)**.** *Let* $\langle S, \to \rangle$ *be an STS such that a surjection as in (6) exists. Given two predicates* Q, P_{int} *over finite executions, if*

1. *for all infinite execution η, there exists $n \in \mathbb{N}$ such that $(\eta_1, \ldots, \eta_n) \in Q$;*
2. *for all finite execution ζ, $Q(\zeta)$ or $\neg Q(\zeta)$ is decidable;*
3. *for all finite execution ζ, $Q(\zeta) \Rightarrow P_{int}(\zeta)$;*
4. *P_{int} is hereditary, as defined above except that $\zeta \cdot q$ is a partial operation defined when ζ is empty or its last state is p and $p \to q$;*

then P_{int} holds over the empty execution: that is $P_{int}(\varepsilon)$ holds.

The last gap towards a useful principle is the requirement that every state in our STS has an outgoing transition. This condition is necessary to ensure the existence of the surjection in Equation (6). To ensure this requirement given any countably-branching STS, we enrich it by adding a *sink* state, which (a) is only reachable from stable states of the original STS, and (b) loops. This is a typical technique, see for instance [71, pag. 17].

▶ **Definition 30.** *Define $Sink(S, \to) \coloneqq \langle S \cup \{\top\}, \to^\top \rangle$, where $\to^\top$ is defined inductively as follows:*

$$p \to q \implies p \to^\top q \qquad p \not\to \implies p \to^\top \top \qquad \top \to^\top \top$$

A maximal execution of $Sink(S, \to)$ is always infinite, and it corresponds (in classical logic) to either an infinite execution of S or a maximal execution of S followed by infinitely many $\top$. We finally prove the converse of Lemma 26.

▶ **Proposition 31.** *Given a countably branching STS $\langle S, \to \rangle$, and a decidable predicate Q on S, we have that, for all $s \in S$, $\mathsf{ext}_Q(s)$ implies $\mathsf{int}_Q(s)$.*

Now we easily obtain completeness of the intensional predicates.

▶ **Corollary 32.** *For every $p \in C$,*
1. *$p \downarrow$ implies $p \downarrow_i$,*
2. *for every r. p $\textsc{must}$ r implies p $\textsc{must}_i$ r.*

Proof. Direct consequence of Proposition 31, and Equation (ext-preds) and Equation (int-preds) above. ◀

As we have outlined why Corollary 32 is true, from now on we use $\downarrow_i$ and $\textsc{must}_i$ instead of $\downarrow$ and $\textsc{must}$. We now present the properties of these predicates that we use in the rest of the paper.

Convergence along traces is obviously preserved by the strong transitions $\longrightarrow$.

▶ **Lemma 33.** *In every LTS, for every $p, p' \in C$ and $s \in \mathsf{Act}^\star$ the following facts are true,*
1. *if $p \Downarrow s$ and $p \xrightarrow{\tau} p'$ then $p' \Downarrow s$,*
2. *for every $\mu \in \mathsf{Act}$. $p \Downarrow \mu.s$ and $p \xrightarrow{\mu} p'$ imply $p \Downarrow s$.*

▶ **Lemma 34.** *For every $s \in \mathsf{Act}^\star$ and $p \in \mathit{ACCS}$, if $p \Downarrow s$ then $|\{q \mid p \overset{s}{\Longrightarrow} q\}| \in \mathbb{N}$.*

The hypothesis of convergence in Lemma 34 is necessary. This is witnessed by the process $p = \mathsf{rec}x.(x \parallel \overline{a})$, which realises an ever lasting addition of a message to the mailbox:

$$p \xrightarrow{\tau} p \parallel \overline{a} \xrightarrow{\tau} p \parallel \overline{a} \parallel \overline{a} \xrightarrow{\tau} p \parallel \overline{a} \parallel \overline{a} \parallel \overline{a} \xrightarrow{\tau} \ldots$$

In more general languages also image-finiteness may fail. An example is given on page 267 of [60].

The predicate $\textsc{must}_i$ is preserved by atoms freely changing their locations in systems. This is coherent with the intuition that the mailbox is a global and shared one. For instance the systems $a.\mathsf{0} \parallel \overline{d} \mathrel{[\![} d.\mathsf{1}$ and $a.\mathsf{0} \mathrel{[\![} d.\mathsf{1} \parallel \overline{d}$, which in the mechanisation are respectively

```
(pr_par (pr_input a pr_nil) (pr_out d), pr_input d pr_succes)
```

and

```
(pr_input a pr_nil, pr_par (pr_input a pr_succes) (pr_out d))
```

have the same mailbox, namely $\overline{d}$.

The predicate MUST_i enjoys three useful properties: it ensures convergence of servers interacting with clients that are not in a good state; it is preserved by internal computation of servers; and it is preserved also by interactions with unhappy clients. The arguments to show these facts are by rule induction on the hypothesis $p \, \text{MUST}_i \, r$. The last fact is a consequence of a crucial property of MUST_i, namely Lemma 44.

▶ **Lemma 35.** *Let $\mathcal{L}_A \in OW$ and $\mathcal{L}_B \in OF$. For every $p \in A$, $r \in B$ we have that $p \, \text{MUST}_i \, r$ implies that $p \downarrow_i$ or $\text{GOOD}(r)$.*

▶ **Lemma 36.** *Let $\mathcal{L}_A \in OW$ and $\mathcal{L}_B \in OF$. For every $p, p' \in A$, $r \in B$ we have that $p \, \text{MUST}_i \, r$ and $p \xrightarrow{\tau} p'$ imply $q \, \text{MUST}_i \, r$.*

▶ **Lemma 37.** *For every $\mathcal{L}_B \in OBA$, $r \in B$ and name $a \in \mathcal{N}$ such that $p \xrightarrow{\overline{a}} p'$ then $\text{GOOD}(p)$ iff $\text{GOOD}(p')$.*

Proof. This is a property of `Good`, more specifically `good_preserved_by_lts_output` and `good_preserved_by_lts_output_converse`. ◀

Lemma 44 Let $\mathcal{L}_A \in$ OW and $\mathcal{L}_B \in$ OF. For every $p_1, p_2 \in A$, every $r_1, r_2 \in B$ and name $a \in \mathcal{N}$ such that $p_1 \xrightarrow{\overline{a}} p_2$ and $r_1 \xrightarrow{\overline{a}} r_2$, if $p_1 \, \text{MUST}_i \, r_2$ then $p_2 \, \text{MUST}_i \, r_1$.

Proof. We proceed by induction on $p_1 \, \text{MUST}_i \, r_2$. In the base case $p_1 \, \text{MUST}_i \, r_2$ is derived using the rule [AXIOM] and thus $\text{GOOD}(r_2)$. Lemma 37 implies that $\text{GOOD}(r_1)$, and so we prove $p_2 \, \text{MUST}_i \, r_1$ using rule [AXIOM]. We are done with the base case.

In the inductive case, the hypothesis $p_2 \, \text{MUST}_i \, r_1$ has been derived via an rule [IND-RULE], and we therefore know the following facts:

1. $p_1 \parallel r_2 \xrightarrow{\tau} \hat{p} \parallel \hat{r}$, and
2. For every p', r' such that $p_1 \parallel r_2 \xrightarrow{\tau} p' \parallel r'$ we have that $p' \, \text{MUST}_i \, r'$.

We prove $p_2 \, \text{MUST}_i \, r_1$ by applying rule [IND-RULE]. In turn this requires us to show that

(i) $p_2 \parallel r_1 \xrightarrow{\tau}$, and that

(ii) for each p' and r' such that $p_2 \parallel r_1 \xrightarrow{\tau} p' \parallel r'$, we have $p' \, \text{MUST}_i \, r'$.

We prove (i). The argument starts with a case analysis on how the transition (1) has been derived. There are the following three cases:

[S-Srv] a τ-transition performed by the server such that $p_1 \xrightarrow{\tau} \hat{p}$ and that $\hat{r} = r_2$, or

[S-Clt] a τ-transition performed by the client such that $r_2 \xrightarrow{\tau} \hat{r}$ and that $\hat{p} = p_1$, or

[S-com] an interaction between the server p_1 and the client r_2 such that $p_1 \xrightarrow{\mu} \hat{p}$ and that $r_2 \xrightarrow{\overline{\mu}} \hat{r}$.

In case [S-SRV] we use the OUTPUT-TAU axiom together with the transitions $p_1 \xrightarrow{\overline{a}} p_2$ and $p_1 \xrightarrow{\tau} \hat{p}$ to obtain that either:

- there exists a p_3 such that $p_2 \xrightarrow{\tau} p_3$ and $\hat{p} \xrightarrow{\overline{a}} p_3$, or
- $p_2 \xrightarrow{a} p_3$.

In the first case $p_2 \xrightarrow{\tau} p_3$ let us construct the transition $p_2 \parallel r_1 \xrightarrow{\tau} p_3 \parallel r_1$ as required. In the second case recall that by hypothesis $r_1 \xrightarrow{\overline{a}} r_2$, and thus the transition $p_2 \xrightarrow{a} \hat{p}$ and rule [S-COM] let us construct the desired reduction $p_2 \parallel r_1 \xrightarrow{\tau} \hat{p} \parallel r_2$.

In case [S-CLT] we use the OUTPUT-COMMUTATIVITY axiom together with the transitions $r_1 \xrightarrow{\overline{a}} r_2 \xrightarrow{\tau} \hat{r}$ to obtain a r_3 such that $r_1 \xrightarrow{\tau} r_3 \xrightarrow{\overline{a}} \hat{r}$ and it follows that there exists the silent move $p_2 \parallel r_1 \xrightarrow{\tau} p_2 \parallel r_3$.

In case [S-COM] we have that $p_1 \xrightarrow{\mu} \hat{p}$ and $r_2 \xrightarrow{\overline{\mu}} \hat{r}$. We distinguish whether $\mu = \overline{a}$ or not. If $\mu = \overline{a}$ then observe that $r_1 \xrightarrow{\overline{a}} r_2 \xrightarrow{a} \hat{r}$. Since by hypothesis $r_1, r_2 \in B$ and $\mathcal{L}_B \in \mathrm{OF}$ we apply FEEDBACK axiom to these transitions and obtain $r_1 \xrightarrow{\tau} \hat{r}$. An application of [S-COM] let us construct the desired transition $p_2 \parallel r_1 \xrightarrow{\tau} p_2 \parallel \hat{r}$.

If $\mu \neq \overline{a}$ we apply the OUTPUT-CONFLUENCE axiom to the transitions $p_1 \xrightarrow{\overline{a}} p_2$ and $p_1 \xrightarrow{\mu} \hat{p}$ to obtain a p_3 such that $p_2 \xrightarrow{\mu} p_3$ and $\hat{p} \xrightarrow{\overline{a}} p_3$. We then apply the OUTPUT-COMMUTATIVITY axiom to obtain $r_1 \xrightarrow{\overline{\mu}} r_3 \xrightarrow{\overline{a}} \hat{r}$ for some r_3. Finally, we have the desired $p_2 \parallel r_1 \xrightarrow{\tau} \hat{p} \parallel r_3$ thanks to the existence of an interaction between p_2 and r_1 that follows from $p_2 \xrightarrow{\mu} p_3$ and $r_1 \xrightarrow{\overline{\mu}} r_2$. This concludes the proof of (i).

We now tackle (ii). First of all, note that the inductive hypothesis states the following fact,

For every p', r', p_0 and r_0, such that $p_1 \parallel r_2 \xrightarrow{\tau} p' \parallel r'$, $p' \xrightarrow{\overline{a}} p_0$ and $r_0 \xrightarrow{\overline{a}} r'$ then
$$p_0 \text{ MUST}_i\, r_0.$$

Fix a transition
$$p_2 \parallel r_1 \xrightarrow{\tau} p' \parallel r',$$
we must show $p' \text{ MUST}_i\, r'$. We proceed by case analysis on the rule used to derive the transition at issue, and the cases are as follows,

- **(a)** a τ-transition performed by the server such that $p_2 \xrightarrow{\tau} p'$ and that $r' = r_1$, or
- **(b)** a τ-transition performed by the client such that $r_1 \xrightarrow{\tau} r'$ and that $p' = p_2$, or
- **(c)** an interaction between the server p_2 and the client r_1 such that $p_2 \xrightarrow{\mu} p'$ and that $r_1 \xrightarrow{\overline{\mu}} r'$.

In case (a) we have $p_2 \xrightarrow{\tau} p'$ and $r' = r_1$ and hence we must show $p' \text{ MUST}_i\, r_1$. We apply the OUTPUT-COMMUTATIVITY axiom to the transitions $p_1 \xrightarrow{\overline{a}} p_2 \xrightarrow{\tau} p'$ to obtain a p_3 such that $p_1 \xrightarrow{\tau} p_3 \xrightarrow{\overline{a}} p'$. We apply the inductive hypothesis with $p' = p_3, r' = r_2, p_0 = p'$ and $r_0 = r_1$ and obtain $p_2 \text{ MUST}_i\, r_1$ as required.

In case (b) we have $r_1 \xrightarrow{\tau} r'$ and $p' = p_2$, we therefore must show $p_2 \text{ MUST}_i\, r'$. We apply the OUTPUT-TAU axiom to the transitions $r_1 \xrightarrow{\tau} r'$ and $r_1 \xrightarrow{\overline{a}} r_2$ to obtain that

- either there exists a $\hat{r}$ such that $r_2 \xrightarrow{\tau} \hat{r}$ and $r' \xrightarrow{\overline{a}} \hat{r}$,
- or $r_2 \xrightarrow{a} r'$.

In the first case we apply the inductive hypothesis with $p' = p_1, r' = \hat{r}, p_0 = p_2$ and $r_0 = r'$ and obtain $p_2 \text{ MUST}_i\, r'$ as required. In the second case, the transitions $p_1 \xrightarrow{\overline{a}} p_2$ and $r_2 \xrightarrow{a} r'$ and rule [S-COM] let us prove $p_1 \parallel r_2 \xrightarrow{\tau} p_2 \parallel r'$. We apply part (2) to obtain $p_2 \text{ MUST}_i\, r'$ as required.

We now consider the case (c) in which $p_2 \xrightarrow{\mu} p'$ and $r_1 \xrightarrow{\overline{\mu}} r'$. We must show $p' \text{ MUST}_i\, r'$ and to do so we distinguish whether $\mu = a$ or not.

If $\mu = a$ then we apply the OUTPUT-DETERMINACY axiom to the transitions $r_1 \xrightarrow{\overline{a}} r_2$ and $r_1 \xrightarrow{\overline{\mu}} r'$ to obtain that $r_2 = r'$. Since by hypothesis $p_1, p_2 \in A$ and $\mathcal{L}_A \in \mathrm{OW}$ we apply the FWD-FEEDBACK axiom to the transitions $p_1 \xrightarrow{\overline{a}} p_2 \xrightarrow{a} p'$ to prove that either $p_1 \xrightarrow{\tau} p'$

or $p_1 = p'$ must hold. If $p_1 \xrightarrow{\tau} p'$ then we have that $p_1 \parallel r_2 \xrightarrow{\tau} p' \parallel r_2$. The property in (2) ensures that $p' \,\text{MUST}_i\, r_2$ and from $r_2 = r'$ we have that the required $p' \,\text{MUST}_i\, r'$ holds too. If $p_1 = p'$ then $p' \,\text{MUST}_i\, r_2$ is a direct consequence of the hypothesis $p_1 \,\text{MUST}_i\, r_2$.

If $\mu \neq a$ then we are allowed to apply the OUTPUT-CONFLUENCE axiom to the transitions $r_1 \xrightarrow{\overline{a}} r_2$ and $r_1 \xrightarrow{\overline{\mu}} r'$ to obtain a $\hat{r}$ such that $r_2 \xrightarrow{\overline{\mu}} \hat{r}$ and $r' \xrightarrow{\overline{a}} \hat{r}$. An application of the OUTPUT-COMMUTATIVITY axiom to the transitions $p_1 \xrightarrow{\overline{a}} p_2 \xrightarrow{\mu} p'$ provides us with a $\hat{p}$ such that $p_1 \xrightarrow{\mu} \hat{p} \xrightarrow{\overline{a}} p'$. We now apply the inductive hypothesis with $p' = \hat{p}, r' = \hat{r}, p_0 = p'$ and $r_0 = r'$ and obtain $p_2 \,\text{MUST}_i\, r'$ as required. This concludes the proof of (ii), and therefore of the lemma. ◀

▶ **Lemma 38.** *Let $\mathcal{L}_A \in OW$ and $\mathcal{L}_B \in OF$. For every $p, p' \in A$, $r, r' \in B$ and every action $\mu \in \mathsf{Act}$ such that $p \xrightarrow{\mu} p'$ and $r \xrightarrow{\overline{\mu}} r'$ we have that $p \,\text{MUST}_i\, r$ and $\neg\text{GOOD}(r)$ implies $p' \,\text{MUST}_i\, r'$.*

Proof. By case analysis on the hypothesis that $p \,\text{MUST}_i\, r$. ◀

B Forwarders

The intuition behind forwarders, quoting [64], is that "any message can come into the configuration, regardless of the forms of inner receptors. [...] As the experimenter is not synchronously interacting with the configuration [...], he may send any message as he likes."

In this appendix we give the technical results to ensure that the function FW(−) builds an LTS that satisfies the axioms of the class `LtsEq`.

▶ **Definition 39.** *We define the function $\mathsf{strip} : C \longrightarrow C$ by induction on $\mathsf{mbox}(p)$ as follows: if $\mathsf{mbox}(p) = \varnothing$ then $\mathsf{strip}(p) = p$, while if $\exists \overline{a} \in \mathsf{mbox}(p)$ and $p \xrightarrow{\overline{a}} p'$ then $\mathsf{strip}(p) = \mathsf{strip}(p')$. Note that $\mathsf{strip}(p)$ is well-defined thanks to the OUTPUT-DETERMINACY and the OUTPUT-COMMUTATIVITY axioms.* ■

We now wish to show that $\text{FW}(\mathcal{L}) \in \text{OW}$ for any LTS $\mathcal{L}$ of output-buffered agents with feedback. Owing to the structure of our typeclasses, we have first to construct an equivalence $\doteq$ over $\text{FW}(\mathcal{L})$ that is compatible with the transition relation, *i.e.* satisfies the axiom in Figure 5. We do this in the obvious manner, *i.e.* by combining the equivalence $\simeq$ over the states of $\mathcal{L}$ with an equivalence over mailboxes.

▶ **Definition 40.** *For any LTS $\mathcal{L}$, two states $p \triangleright M$ and $q \triangleright N$ of $FW(\mathcal{L})$ are equivalent, denoted $p \triangleright M \doteq q \triangleright N$, if $\mathsf{strip}(p) \simeq \mathsf{strip}(q)$ and $M \uplus \mathsf{mbox}(p) = N \uplus \mathsf{mbox}(q)$.* ■

▶ **Lemma 41.** *For every $\mathcal{L}_A$ and every $p \triangleright M, q \triangleright N \in A \times MO$, and every $\alpha \in L$, if $p \triangleright M \;(\doteq \cdot \xrightarrow{\alpha}_{\mathsf{fw}})\; q \triangleright N$ then $p \triangleright M \;(\xrightarrow{\alpha}_{\mathsf{fw}} \cdot \doteq)\; q' \triangleright N'$.*

C Completeness

This section is devoted to the proof that the alternative preorder given in Definition (9) includes the MUST-preorder. First we present a general outline of the main technical results to obtain the proof we are after. Afterwards, in Subsection (C) we discuss in detail on all the technicalities.

Proofs of completeness of characterisations of contextual preorders usually require using, as the name suggests, syntactic contexts. Our calculus-independent setting, though, does not allow us to define them. Instead we phrase our arguments using two functions $tc : \mathsf{Act}^\star \to C$,

$\forall s \in \mathsf{Act}^\star, \forall a \in \mathcal{N},$
(1) $\neg\textsc{good}(f(s))$
(2) $\forall \mu \in \mathsf{Act}, f(\mu.s) \xrightarrow{\overline{\mu}} f(s)$
(3) $f(\overline{a}.s) \xrightarrow{\tau}$
(4) $\forall r \in C, f(\overline{a}.s) \xrightarrow{\tau} r$ implies $\textsc{good}(r)$
(5) $\forall r \in C, \mu \in \mathsf{Act},\ f(\overline{a}.s) \xrightarrow{\mu} r$ implies $\mu = a$ and $r = f(s)$

$\forall E \subseteq \mathcal{N},$
(t1) $ta(\varepsilon, E) \stackrel{\tau}{\not\longrightarrow}$
(t2) $\forall a \in \mathcal{N}, ta(\varepsilon, E) \stackrel{\overline{a}}{\not\longrightarrow}$
(t3) $\forall a \in \mathcal{N}, ta(\varepsilon, E) \xrightarrow{a}$ if and only if $a \in E$
(t4) $\forall \mu \in \mathsf{Act}, r \in C, ta(\varepsilon, E) \xrightarrow{\mu} r$ implies $\textsc{good}(r)$

(c1) $\forall \mu \in \mathsf{Act}, tc(\varepsilon) \stackrel{\tau}{\not\longrightarrow}$
(c2) $\exists r, tc(\varepsilon) \xrightarrow{\tau} r$
(c3) $\forall r, tc(\varepsilon) \xrightarrow{\tau} r$ implies $\textsc{good}(r)$

Table 1 Properties of the functions that generate clients.

and $ta : \mathsf{Act}^\star \times \mathcal{P}(\mathcal{N}) \to C$ where $\langle C, L, \longrightarrow\rangle$ is some LTS of OF. In Table 1 we gather all the *properties* of tc and ta that are sufficient to give our arguments. The properties (1) - (5) must hold for both tc and $ta(\varepsilon, -)$ for every set of names O, the properties (c1) - (c2) must hold for tc, and (t1) - (t4) must hold for ta.

We use the function tc to test the convergence of servers, and the function ta to test the acceptance sets of servers.

A natural question is whether such tc and ta can actually exist. The answer depends on the LTS at hand. In Appendix F.2, and in particular Figure 17, we define these functions for the standard LTS of `ACCS`, and it should be obvious how to adapt those definitions to the asynchronous π-calculus [57].

In short, our proofs show that $\preccurlyeq_{\mathsf{AS}}$ is complete with respect to $\sqsubseteq_{\textsc{must}}$ in any LTS of output-buffered agents with feedback wherein the functions tc and ta enjoying the properties in Table 1 can be defined.

First, converging along a finite trace s is logically equivalent to passing the client $tc(s)$. In other words, there exists a bijection between the proofs (i.e. finite derivation trees of $p\,\textsc{must}_i\, tc(s)$) and the ones of $p \Downarrow s$. We first give the proposition, and then discuss the auxiliary lemmas to prove it.

▶ **Proposition 42.** *For every $\mathcal{L}_A \in OW$, $p \in A$, and $s \in \mathsf{Act}^\star$ we have that $p\,\textsc{must}_i\, tc(s)$ if and only if $p \Downarrow s$.*

The *if* implication is Lemma 59 and the *only if* implication is Lemma 56. The hypothesis that $\mathcal{L}_A \in \mathrm{OW}$, *i.e.* the use of forwarders, is necessary to show that convergence implies passing a client, as shown by the next example.

▶ **Example 43.** Consider a server p in an LTS $\mathcal{L} \in \mathrm{OF}$ whose behaviour amounts to the following transitions: $p \xrightarrow{b} \Omega \xrightarrow{\tau} \Omega \xrightarrow{\tau} \ldots$ Note that this entails that $\mathcal{L}$ does not *not* enjoy the axioms of forwarders.

Now let $s = a.b$. Since $p \downarrow$ and $p \stackrel{a}{\not\Longrightarrow}$ we know that $p \Downarrow a.b$. On the other hand Table 1(2) implies that the client $tc(s)$ performs the transitions $tc(s) \xrightarrow{\overline{a}} tc(b) \xrightarrow{\overline{b}} tc(\varepsilon)$. Thanks to the OUTPUT-COMMUTATIVITY axiom we obtain $tc(s) \xrightarrow{\overline{b}}\xrightarrow{\overline{a}} tc(\varepsilon)$. Table 1(1) implies that the states reached by the client are unsuccessful, and so by zipping the traces performed by p and by $tc(s)$ we build a maximal computation of $p \parallel tc(s)$ that is unsuccessful, and thus $p \not\text{MUST}_i\ tc(s)$. ◀

This example explains why in spite of Lemma 14 output-buffered agents with feedback do not suffice to use the standard characterisations of the MUST-preorder.

We move on to the more involved technical results, *i.e.* the next three lemmas, that we use to reason on acceptance sets of servers. We wish to stress Lemma 44: it states that, when reasoning on MUST_i, outputs can be freely moved from the client to the server side of systems, if servers have the forwarding ability. Its proof uses *all* the axioms for output-buffered agents with feedback, and the lemma itself is used in the proof of the main result on acceptance sets, namely Lemma 46.

▶ **Lemma 44** (Output swap). *Let $\mathcal{L}_A \in OW$ and $\mathcal{L}_B \in OF$. For every $p_1, p_2 \in A$, every $r_1, r_2 \in B$ and name $a \in \mathcal{N}$ such that $p_1 \xrightarrow{\overline{a}} p_2$ and $r_1 \xrightarrow{\overline{a}} r_2$, if $p_1\, \text{MUST}_i\, r_2$ then $p_2\, \text{MUST}_i\, r_1$.*

▶ **Lemma 45.** *Let $\mathcal{L}_A \in OW$. For every $p \in A$, $s \in \mathsf{Act}^\star$, and every $L, E \subseteq \mathcal{N}$, if $\overline{L} \in \mathcal{A}_{\mathsf{fw}}(p, s)$ then $p \not\text{MUST}_i\ ta(s, E \setminus L)$.*

▶ **Lemma 46.** *Let $\mathcal{L}_A \in OW$. For every $p \in A, s \in \mathsf{Act}^\star$, and every finite set $O \subseteq \overline{\mathcal{N}}$, if $p \Downarrow s$ then either*

(i) *$p\, \text{MUST}_i\, ta(s, \overline{\bigcup \mathcal{A}_{\mathsf{fw}}(p, s) \setminus O})$, or*

(ii) *there exists $\widehat{O} \in \mathcal{A}_{\mathsf{fw}}(p, s)$ such that $\widehat{O} \subseteq O$.*

We can now show that the alternative preorder $\preccurlyeq_{\mathsf{AS}}$ includes $\sqsubseteq_{\text{MUST}}$ when used over LTSs of forwarders.

▶ **Lemma 47.** *For every $\mathcal{L}_A, \mathcal{L}_B \in OW$ and servers $p \in A, q \in B$, if $p \sqsubseteq_{\text{MUST}} q$ then $p \preccurlyeq_{\mathsf{AS}} q$.*

Proof. Let $p \sqsubseteq_{\text{MUST}} q$. To prove that $p \preccurlyeq_{\mathsf{cnv}} q$, suppose $p \Downarrow s$ for some trace s. Proposition 42 implies $p\, \text{MUST}_i\, tc(s)$, and so by hypothesis $q\, \text{MUST}_i\, tc(s)$. Proposition 42 ensures that $q \Downarrow s$.

We now show that $p \preccurlyeq_{\mathsf{acc}} q$. Thanks to Lemma 15, it is enough to prove that $p \preccurlyeq^{\mathsf{fw}}_{\mathsf{acc}} q$. So, we show that for every trace $s \in \mathsf{Act}^\star$, if $p \Downarrow s$ then $\mathcal{A}_{\mathsf{fw}}(p, s) \ll \mathcal{A}_{\mathsf{fw}}(q, s)$. Fix an $O \in \mathcal{A}_{\mathsf{fw}}(q, s)$. We have to exhibit a set $\widehat{O} \in \mathcal{A}_{\mathsf{fw}}(p, s)$ such that $\widehat{O} \subseteq O$.

By definition $O \in \mathcal{A}_{\mathsf{fw}}(q, s)$ means that for some q' we have $q \stackrel{s}{\Longrightarrow} q' \stackrel{\tau}{\not\longrightarrow}$ and $O(q') = O$. Let $E = \bigcup \mathcal{A}_{\mathsf{fw}}(p, s)$ and $X = E \setminus O$. The hypothesis that $p \Downarrow s$, and the construction of the set X let us apply Lemma 46, which implies that either

(a) $p\, \text{MUST}_i\, ta(s, \overline{X})$, or

(b) there exists a $\widehat{O} \in \mathcal{A}_{\mathsf{fw}}(p, s)$ such that $\widehat{O} \subseteq O(q')$.

Since (b) is exactly what we are after, to conclude the argument it suffices to prove that (a) is false. This follows from Lemma 45, which proves $q \not\text{MUST}_i\ ta(s, \overline{X})$, and the hypothesis $p \sqsubseteq_{\text{MUST}} q$, which ensures $p \not\text{MUST}_i\ ta(s, \overline{X})$. ◀

The fact that the MUST-preorder can be captured via the function $\text{FW}(-)$ and $\preccurlyeq_{\mathsf{AS}}$ is a direct consequence of Lemma 14 and Lemma 47.

▶ **Proposition 48** (Completeness). *For every $\mathcal{L}_A, \mathcal{L}_B \in OF$ and servers $p \in A, q \in B$, if $p \sqsubseteq_{\text{MUST}} q$ then $FW(p) \preccurlyeq_{\mathsf{AS}} FW(q)$.*

We now gather all the technical auxiliary lemmas and then discuss the proofs of the main ones.

By assumption, outputs preserve the predicate GOOD. For stable clients, they also preserve the negation of this predicate.

▶ **Lemma 49.** *For all $r, r' \in A$ and trace $s \in \overline{\mathcal{N}}^\star$, $r \overset{\tau}{\not\longrightarrow}$, $\neg GOOD(r)$ and $r \overset{s}{\Longrightarrow} r'$ implies $\neg GOOD(r')$.*

C.1 Testing convergence

We start with preliminary facts, in particular two lemmas that follow from the properties in Table 1.

A process p *converges along* a trace s if for every p' reached by p performing any prefix of s, the process p' converges.

▶ **Lemma 50.** *For every $\langle A, L, \longrightarrow \rangle$, $p \in A$, and $s \in \mathsf{Act}^\star$, $p \Downarrow s$ if and only if $p \overset{s'}{\Longrightarrow} p'$ implies $p' \downarrow$ for every s' prefix of s.*

Traces of output actions impact neither the stability of servers, nor their input actions.

▶ **Lemma 51.** *For every $\mathcal{L}_A$, every $p, p' \in A$ and every trace $s \in \overline{\mathcal{N}}^\star$,*
1. *$p \overset{\tau}{\not\longrightarrow}$ and $p \overset{s}{\Longrightarrow} p'$ implies $p' \overset{\tau}{\not\longrightarrow}$.*
2. *$p \overset{\tau}{\not\longrightarrow}$ and $p \overset{s}{\Longrightarrow} p'$ implies $I(p) = I(p')$.*

▶ **Lemma 52.** *For every $s \in \mathsf{Act}^\star$, $tc(s) \overset{\tau}{\longrightarrow}$.*

The BACKWARD-OUTPUT-DETERMINACY axiom is used in the proof of the next lemma.

▶ **Lemma 53.** *For every $s \in \mathsf{Act}^\star$, if $tc(s) \overset{\mu}{\longrightarrow} r$ then either*
(a) *$GOOD(r)$, or*
(b) *$s = s_1.\overline{\mu}.s_2$ for some $s_1 \in \mathcal{N}^\star$ and $s_2 \in \mathsf{Act}^\star$ such that $r \simeq tc(s_1.s_2)$.*

▶ **Lemma 54.** *For every $s \in \mathsf{Act}^\star$, if $tc(s) \overset{\tau}{\longrightarrow} r$ then either:*
(a) *$GOOD(r)$, or*
(b) *there exist b, s_1, s_2 and s_3 with $s_1.b.s_2 \in \mathcal{N}^\star$ such that $s = s_1.b.s_2.\overline{b}.s_3$ and $r \simeq tc(s_1.s_2.s_3)$.*

▶ **Lemma 55.** *Let $\mathcal{L}_A \in OW$. For every server $p, p' \in A$, trace $s \in \mathsf{Act}^\star$ and action $\mu \in \mathsf{Act}$ such that $p \overset{\mu}{\Longrightarrow} p'$ we have that $p \, MUST_i \, tc(\mu.s)$ implies $p'p' \, MUST_i \, tc(s)$.*

Proof. By rule induction on the reduction $p \overset{\mu}{\Longrightarrow} p'$ together with Lemma 36 and Lemma 38. ◀

▶ **Lemma 56.** *Let $\mathcal{L}_A \in OW$. For every server $p \in A$, trace $s \in \mathsf{Act}^\star$ we have that $p \, MUST_i \, tc(s)$ implies $p \Downarrow s$.*

Proof. We proceed by induction on the trace s. In the base case s is ε. Table 1(1) states that $\neg$GOOD$(tc(\varepsilon))$ and we apply Lemma 35 to obtain $p \downarrow_i$, and thus $p \Downarrow \varepsilon$. In the inductive case s is $\mu.s'$ for some $\mu \in \mathsf{Act}$ and $s' \in \mathsf{Act}^\star$. We must show the following properties,
1. $p \downarrow_i$, and
2. for every p' such that $p \overset{\mu}{\Longrightarrow} p'$, $p' \Downarrow s'$.

We prove the first property as we did in the base case, and we apply Lemma 55 to prove the second property. ◀

▶ **Lemma 57.** *Let $\mathcal{L}_A \in OW$. For every $p \in A$, $s_1 \in \mathcal{N}^\star$ and $s_3 \in \mathsf{Act}^\star$ we have that*

1. *for every $\mu \in \mathsf{Act}$, if $p \Downarrow s_1.\mu.s_3$ and $p \xrightarrow{\mu} q$ then $q \Downarrow s_1.s_3$,*
2. *for every $a.s_2 \in \mathcal{N}^\star$ if $p \Downarrow s_1.a.s_2.\overline{a}.s_3$ then $p \Downarrow s_1.s_2.s_3$.*

▶ **Lemma 58.** *For every LTS $\mathcal{L}_A$ and every $p \in A$, $p \downarrow_i$ implies $p \, \textsc{must}_i \, tc(\varepsilon)$.*

Proof. Rule induction on the derivation of $p \downarrow_i$. ◀

▶ **Lemma 59.** *For every $\mathcal{L}_A \in OW$, every $p \in A$, and $s \in \mathsf{Act}^\star$, if $p \Downarrow s$ then $p \, \textsc{must}_i \, tc(s)$.*

Proof. The hypothesis $p \Downarrow s$ ensures $p \downarrow_i$. We show that $p \, \textsc{must}_i \, tc(s)$ reasoning by complete induction on the length of the trace s. The base case is Lemma 58 and here we discuss the inductive case, i.e. when $\mathsf{len}(s) = n + 1$ for some $n \in \mathbb{N}$.

We proceed by rule induction on $p \downarrow_i$. In the base case $p \stackrel{\tau}{\not\longrightarrow}$, and the reduction at hand is due to either a τ transition in $tc(s)$, or a communication between p and $tc(s)$.

In the first case $tc(s) \xrightarrow{\tau} r$, and so Lemma 54 ensures that one of the following conditions holds,

1. $\textsc{good}(r)$, or
2. there exist $a \in \mathcal{N}$, s_1, s_2 and s_3 with $s = s_1.a.s_2.\overline{a}.s_3$ and $r \simeq tc(s_1.s_2.s_3)$.

If $\textsc{good}(r)$ then we conclude via rule [axiom]; otherwise Lemma 57(2) and the hypothesis that $p \Downarrow s$ imply $p \Downarrow s_1.s_2.s_3$, thus prove $p \, \textsc{must}_i \, r$ via the inductive hypothesis of the complete induction on s.

We now consider the case when the transition is due to a communication, *i.e.* $p \xrightarrow{\mu} p'$ and $tc(s) \xrightarrow{\overline{\mu}} r$. Lemma 53 tells us that either $\textsc{good}(r)$ or there exist s_1 and s_2 such that $s = s_1.\mu.s_2$ and $r \simeq tc(s_1.s_2)$. In the first case we conclude via rule [axiom]. In the second case we apply Lemma 57(1) to prove $p' \Downarrow s_1.s_2$, and thus $p' \, \textsc{must}_i \, r$ follows from the inductive hypothesis of the complete induction. In the inductive case of the rule induction on $p \downarrow_i$, we know that $p \xrightarrow{\tau} p'$ for some process p'. We reason again by case analysis on how the reduction we fixed has been derived, *i.e.* either via a τ transition in $tc(s)$, or via a communication between p and $tc(s)$, or via a τ transition in p. In the first two cases we reason as we did for the base case of the rule induction. In the third case $p \Downarrow s$ and $p \xrightarrow{\tau} p'$ imply $p' \Downarrow s$, we thus obtain $p' \, \textsc{must}_i \, tc(s)$ thanks to the inductive hypothesis of the rule induction which we can apply because the tree to derive $p' \downarrow_i$ is smaller than the tree to derive that $p \downarrow_i$. ◀

C.2 Testing acceptance sets

In this section we present the properties of the function $ta(-,-)$ that are sufficient to obtain completeness. To begin with, $ta(-,-)$ function enjoys a form of monotonicity with respect to its second argument.

▶ **Lemma 60.** *Let $\mathcal{L}_A \in OF$. For every $p \in A$, trace $s \in \mathsf{Act}^\star$, and sets of outputs O_1, O_2, if $p \, \textsc{must}_i \, ta(s, O_1)$ and $O_1 \subseteq O_2$ then $p \, \textsc{must}_i \, ta(s, O_2)$.*

Proof. Induction on the derivation of $p \, \textsc{must}_i \, ta(s, O_1)$. ◀

Let OBA denote the set of LTS of output-buffered agents. Note that any $\mathcal{L} \in \text{OBA}$ need not enjoy the Feedback axiom.

▶ **Lemma 61.** *Let $\mathcal{L}_A \in OBA$, and $\mathcal{L}_B \in OBA$. For every $p \in A$, trace $s \in \mathsf{Act}^\star$, set of outputs O and name $a \in \mathcal{N}$, such that*

(i) $p \downarrow_i$ *and,*

(ii) *For every* $p' \in A$, $p \stackrel{\overline{a}}{\Longrightarrow} p'$ *implies* $p' \textsc{must}_i\ ta(s, \overline{O})$,

we have that $p \textsc{must}_i\ ta(\overline{a}.s, \overline{O})$.

Proof. We proceed by induction on the hypothesis $p \downarrow_i$.

C.2.0.1 (Base case: p is stable)

We prove $p \textsc{must}_i\ ta(\overline{a}.s, \overline{O})$ by applying rule [IND-RULE]. Since Table 1(3) implies that $p \parallel ta(\overline{a}.s, \overline{O}) \xrightarrow{\tau}$, all we need to prove is the following fact,

$$\forall p' \in A, r \in B.\ \text{if } p \parallel ta(\overline{a}.s, \overline{O}) \xrightarrow{\tau} p' \parallel r \text{ then } p' \textsc{must}_i\ r. \qquad (\star)$$

Fix a transition $p \parallel ta(\varepsilon, \overline{O}) \xrightarrow{\tau} p' \parallel r$. As p is stable, this transition can either be due to:

1. a τ-transition performed by the client such that $ta(\overline{a}.s, \overline{O}) \xrightarrow{\tau} r$, or
2. an interaction between the server p and the client $ta(\overline{a}.s, \overline{O})$.

In the first case Table 1(4) implies $\textsc{good}(r)$, and hence we obtain $p' \textsc{must}_i\ r$ via rule [AXIOM]. In the second case there exists an action μ such that

$$p \xrightarrow{\mu} p' \quad \text{and} \quad ta(\overline{a}.s, \overline{O}) \xrightarrow{\overline{\mu}} r$$

Table 1(5) implies μ is $\overline{a}$ and $r = ta(s, \overline{O})$. We then have $p \xrightarrow{\overline{a}} p'$ and thus the reduction $p \stackrel{\overline{a}}{\Longrightarrow} p'$, which allows us to apply the hypothesis (ii) and obtain $p' \textsc{must}_i\ r$ as required.

C.2.0.2 (Inductive case: $p \xrightarrow{\tau} p'$ implies p')

The argument is similar to one for the base case, except that we must also tackle the case when the transition $p \parallel ta(\overline{a}.s, \overline{O}) \xrightarrow{\tau} p' \parallel r$ is due to a τ action performed by p, *i.e.* $p \xrightarrow{\tau} p'$ and $r = ta(\overline{a}.s, \overline{O})$. The inductive hypothesis tells us the following fact:

For every p_1 and a, such that $p \xrightarrow{\tau} p_1$, for every p_2, if $p_1 \stackrel{\overline{a}}{\Longrightarrow} p_2$ then $p_2 \textsc{must}_i\ ta(s, O)$.

To apply the inductive hypothesis we have to show that for every p_2 such that $p' \stackrel{\overline{a}}{\Longrightarrow} p_2$ we have that $p_2 \textsc{must}_i\ ta(s, \overline{O})$. This is a consequence of the hypothesis (ii) together with the reduction $p \xrightarrow{\tau} p' \stackrel{\overline{a}}{\Longrightarrow} p_2$, and thus concludes the proof. ◀

▶ **Lemma 62.** *Let $\mathcal{L}_A \in OF$. For every $p \in A$ and set of outputs O, if p is stable then either*

(a) $p \textsc{must}_i\ ta(\varepsilon, \overline{O(p) \setminus O})$, *or*

(b) $O(p) \subseteq O$.

Proof. We distinguish whether $O(p) \setminus O$ is empty or not. In the first case, $O(p) \setminus O = \emptyset$ implies $O(p) \subseteq O$, and we are done.

In the second case, there exists $\overline{a} \in O(p)$ such that $\overline{a} \notin O$. Note also that Table 1(1) ensures that $\neg\textsc{good}(ta(\varepsilon, \overline{O(p) \setminus O}))$, and thus we construct a derivation of $p \textsc{must}_i\ ta(\varepsilon, \overline{O(p) \setminus O})$ by applying the rule [IND-RULE]. This requires us to show the following facts,

1. $p \parallel ta(s, \overline{O(p) \setminus O}) \longrightarrow$, and
2. for each p', r such that $p \parallel ta(s, \overline{O(p) \setminus O}) \xrightarrow{\tau} p' \parallel r$, $p' \textsc{must}_i\ r$ holds.

To prove (1), we show that an interaction between the server p and the test $ta(s, \overline{O(p) \setminus O})$ exists. As $\overline{a} \in O(p)$, we have that $p \xrightarrow{\overline{a}}$. Then $\overline{a} \in O(p) \setminus O$ together with (3) ensure that $ta(s, \overline{O(p) \setminus O}) \xrightarrow{a}$. An application of the rule [S-COM] gives us the required transition $p \parallel ta(s, \overline{O(p) \setminus O}) \longrightarrow$.

To show (2), fix a silent transition $p \parallel ta(s, \overline{O(p) \setminus O}) \xrightarrow{\tau} p' \parallel r$. We proceed by case analysis on the rule used to derive the transition under scrutiny. Recall that the server p is stable by hypothesis, and that $ta(s, \overline{O(p) \setminus O})$ is stable thanks to Table 1(1). This means that the silent transition must have been derived via rule [S-COM]. Furthermore, Table 1(2) implies that the test $ta(s, \overline{O(p) \setminus O})$ does not perform any output. As a consequence, if there is an interaction it must be because the test performs an input and becomes r. Table 1(4) implies that $\text{GOOD}(r)$, and hence we obtain the required $p' \text{ MUST}_i\, r$ applying rule [AXIOM]. ◀

Lemma 45 Let $\mathcal{L}_A \in \text{OW}$. For every $p \in A$, $s \in \mathsf{Act}^\star$, and every $L, E \subseteq \mathcal{N}$, if $\overline{L} \in \mathcal{A}_{\mathsf{fw}}(p, s)$ then $p \not\text{MUST}_i\, ta(s, E \setminus L)$.

Proof. By hypothesis there exists a set $\overline{L} \in \mathcal{A}_{\mathsf{fw}}(p, s)$, *i.e.* for some p' we have $p \stackrel{s}{\Longrightarrow} p' \stackrel{\tau}{\not\longrightarrow}$ and $O(p') = \overline{L}$. We have to show that $p \not\text{MUST}_i\, ta(s, E \setminus L)$, *i.e.* $p \text{ MUST}_i\, ta(s, E \setminus L)$ implies $\perp$. For convenience, let $X = E \setminus L$.

We proceed by induction on the derivation of the weak transitions $p \stackrel{s}{\Longrightarrow} p'$. In the base case the derivation consists in an application of rule [WT-REFL], which implies that $p = p'$ and $s = \varepsilon$. We show that there exists no derivation of judgement $p \text{ MUST}_i\, ta(s, X)$. By definition, $\neg\text{GOOD}(ta(s, X))$ and thus no tree that ends with [AXIOM] can have $p \text{ MUST}_i\, ta(s, X)$ as conclusion. The hypotheses ensure that p is stable, and $ta(\varepsilon, X)$ is stable by definition. The set of inputs of $ta(\varepsilon, X)$ is X, which prevents an interaction between p and $ta(s, X)$, i.e. an application of rule [S-COM]. This proves that $p \parallel ta(s, X)$ is stable, thus a side condition of [IND-RULE] is false, and the rule cannot be employed to prove $p \text{ MUST}_i\, ta(s, X)$.

In the inductive cases $p \stackrel{s}{\Longrightarrow} p'$ is derived using either:

(i) rule [WT-TAU] such that $p \xrightarrow{\tau} \widehat{p} \stackrel{s}{\Longrightarrow} p'$, or

(ii) rule [WT-MU] such that $p \xrightarrow{\mu} \widehat{p} \stackrel{t}{\Longrightarrow} p'$, with $s = \mu.t$.

In the first case, applying the inductive hypothesis requires us to show $\widehat{p} \text{ MUST}_i\, ta(s, X)$, which is true since $p \text{ MUST}_i\, ta(s, X)$ is preserved by the τ-transitions performed by the server.

In the second case, applying the inductive hypothesis requires us to show $\widehat{p} \text{ MUST}_i\, ta(t, X)$. Table 1(2) implies that $ta(\mu.t, X) \xrightarrow{\overline{\mu}} ta(t, X)$. Then we derive via [S-COM] the transition $p \parallel ta(\mu.t, E) \xrightarrow{\tau} \widehat{p} \parallel ta(t, E)$. Since $p \text{ MUST}_i\, ta(s, X)$ is preserved by the interactions occurring between the server and the client, which implies $\widehat{p} \text{ MUST}_i\, ta(t, X)$ as required. ◀

Lemma 46 Let $\mathcal{L}_A \in \text{OW}$. For every $p \in A, s \in \mathsf{Act}^\star$, and every finite set $O \subseteq \overline{\mathcal{N}}$, if $p \Downarrow s$ then either

(i) $p \text{ MUST}_i\, ta(s, \overline{\bigcup \mathcal{A}_{\mathsf{fw}}(p, s) \setminus O})$, or

(ii) there exists $\widehat{O} \in \mathcal{A}_{\mathsf{fw}}(p, s)$ such that $\widehat{O} \subseteq O$.

Proof. We proceed by induction on the trace s.

C.2.0.3 (Base case, $s = \varepsilon$)

The hypothesis $p \Downarrow \varepsilon$ implies $p \downarrow_i$ and we continue by induction on the derivation of $p \downarrow_i$.[10] In the base case $p \downarrow_i$ was proven using rule [AXIOM], and hence $p \stackrel{\tau}{\not\longrightarrow}$. We apply Lemma 62 to obtain either:

(i) $p \text{ MUST}_i\, ta(\varepsilon, \overline{O(p) \setminus O})$, or

(ii) $O(p) \subseteq O$.

[10] Recall that the definition of $\downarrow_i$ is in Equation (int-preds)

In case (i) we are done. In case (ii), as p is stable we have $\{p' \mid p \stackrel{\varepsilon}{\Longrightarrow} p' \stackrel{\tau}{\not\longrightarrow}\} = \{p\}$ and thus $\mathcal{A}_{\mathsf{fw}}(p, \varepsilon) = \{O(p)\}$ and we conclude by letting $\widehat{O} = O(p)$.

In the inductive case $p \downarrow_i$ was proven using rule [IND-RULE]. We know that $p \xrightarrow{\tau}$, and the inductive hypothesis states that for any p' such that $p \xrightarrow{\tau} p'$, either:

(a) $p' \text{ MUST}_i\ ta(\varepsilon, \overline{O(p') \setminus O})$, or

(b) there exists $\widehat{O} \in \mathcal{A}_{\mathsf{fw}}(p', s)$ such that $\widehat{O} \subseteq O$.

It follows that either

($\forall$) for each $p' \in \{p' \mid p \xrightarrow{\tau} p'\}$, $p' \text{ MUST}_i\ ta(\varepsilon, \bigcup \overline{\mathcal{A}_{\mathsf{fw}}(p', s) \setminus O})$, or

($\exists$) there exists a $p' \in \{p' \mid p \xrightarrow{\tau} p'\}$ and a $\widehat{O} \in \mathcal{A}_{\mathsf{fw}}(p', \varepsilon)$ such that $\widehat{O} \subseteq O$,

We discuss the two cases. If ($\exists$) the argument is straightforward: we pick the existing p' such that $p \xrightarrow{\tau} p'$. The definition of $\mathcal{A}_{\mathsf{fw}}(-, -)$ ensures that and show that $\mathcal{A}_{\mathsf{fw}}(p', \varepsilon) \subseteq \mathcal{A}_{\mathsf{fw}}(p, \varepsilon)$, and thus we conclude by choosing $\widehat{O}$.

Case ($\forall$) requires more work. We are going to show that $p \text{ MUST}_i\ ta(\varepsilon, \bigcup \overline{\mathcal{A}_{\mathsf{fw}}(p, s) \setminus O})$ holds. To do so we apply the rule [IND-RULE] and we need to show the following facts,

(a) $p \parallel ta(\varepsilon, \bigcup \overline{\mathcal{A}_{\mathsf{fw}}(p, s) \setminus O}) \xrightarrow{\tau}$, and

(b) for each $p' \parallel r'$ such that $p \parallel ta(\varepsilon, \bigcup \overline{\mathcal{A}_{\mathsf{fw}}(p, s) \setminus O}) \xrightarrow{\tau} p' \parallel r'$, we have $p' \text{ MUST}_i\ r'$.

The first requirement follows from the fact that p is not stable. To show the second requirement we proceed by case analysis on the transition $p \parallel ta(\varepsilon, \bigcup \overline{\mathcal{A}_{\mathsf{fw}}(p, s) \setminus O}) \xrightarrow{\tau} p' \parallel r'$. As $ta(\varepsilon, \overline{O(p) \setminus O})$ is stable by (1), it can either be due to:

1. a τ-transition performed by the server p such that $p \xrightarrow{\tau} p'$, or
2. an interaction between the server p and the client $ta(\varepsilon, \bigcup \overline{\mathcal{A}_{\mathsf{fw}}(p, s) \setminus O})$.

In the first case we apply the first part of the inductive hypothesis to prove that $p' \text{ MUST}_i\ ta(\varepsilon, \bigcup \overline{\mathcal{A}_{\mathsf{fw}}(p', s) \setminus O})$, and we conclude via Lemma 60 to get the required

$$p' \text{ MUST}_i\ ta(\varepsilon, \bigcup \overline{\mathcal{A}_{\mathsf{fw}}(p, s) \setminus O}).$$

In the second case, there exists a $\mu \in \mathsf{Act}$ such that

$$p \xrightarrow{\mu} p' \text{ and } ta(\varepsilon, \bigcup \overline{\mathcal{A}_{\mathsf{fw}}(p, s) \setminus O}) \xrightarrow{\overline{\mu}} r$$

Thanks to Table 1(4) we apply rule [AXIOM] to prove that $p' \text{ MUST}_i\ r$ and we are done with the base case of the main induction on the trace s.

C.2.0.4 (Inductive case, $s = \mu.s'$)

By induction on the set $\{p' \mid p \stackrel{\mu}{\Longrightarrow} p'\}$ and an application of the inductive hypothesis we know that either:

(i) there exists $p' \in \{p' \mid p \stackrel{\mu}{\Longrightarrow} p'\}$ and $\widehat{O} \in \mathcal{A}_{\mathsf{fw}}(p', s')$ such that $\widehat{O} \subseteq O$, or

(ii) for each $p' \in \{p' \mid p \stackrel{\mu}{\Longrightarrow} p'\}$ we have that $p' \text{ MUST}_i\ ta(s', \bigcup \overline{\mathcal{A}_{\mathsf{fw}}(p', s')})$.

In the first case, the inclusion $\mathcal{A}_{\mathsf{fw}}(p', s') \subseteq \mathcal{A}_{\mathsf{fw}}(p, \mu.s')$ and $\widehat{O} \in \mathcal{A}_{\mathsf{fw}}(p', s')$ imply $\widehat{O} \in \mathcal{A}_{\mathsf{fw}}(p, s)$ and we are done.

In the second case, we show $p \text{ MUST}_i\ ta(s, \bigcup \overline{\mathcal{A}_{\mathsf{fw}}(p, s)})$ by case analysis on the action μ, which can be either an input or an output.

- If μ is an input, $\mu = a$ for some $a \in \mathcal{N}$. An application of the axiom of forwarders gives us a p' such that $p \xrightarrow{a} p' \xrightarrow{\overline{a}} p$. An application of Table 1(2) gives us the following transition,

$$ta(a.s', \bigcup \overline{\mathcal{A}_{\mathsf{fw}}(p, a.s') \setminus O}) \xrightarrow{\overline{a}} ta(s', \bigcup \overline{\mathcal{A}_{\mathsf{fw}}(p, a.s') \setminus O})$$

$$\text{[MSET-NOW]}\quad \frac{\textsc{good}(r)}{X \;\textsc{must}_{\mathsf{aux}}\; r}$$

$$\text{[MSET-STEP]}\quad \frac{\begin{array}{ll} \neg\textsc{good}(r) & \forall X'.\, X \xrightarrow{\tau} X' \text{ implies } X' \;\textsc{must}_{\mathsf{aux}}\; r \\ \forall\, p \in X.\, p \parallel r \xrightarrow{\tau} & \forall r'.\, r \xrightarrow{\tau} r' \text{ implies } X \;\textsc{must}_{\mathsf{aux}}\; r' \\ \multicolumn{2}{l}{\forall X', \mu \in \mathsf{Act}^{\star}.\, X \xRightarrow{\overline{\mu}} X' \text{ and } r \xrightarrow{\mu} r' \text{ imply } X' \;\textsc{must}_{\mathsf{aux}}\; r'} \end{array}}{X \;\textsc{must}_{\mathsf{aux}}\; r}$$

Figure 11 Rules to define inductively the predicate $\textsc{must}_{\mathsf{aux}}$.

By an application of Lemma 44 it is enough to show

$$p' \;\textsc{must}_i\; ta(s', \bigcup \overline{\mathcal{A}_{\mathsf{fw}}(p, a.s') \setminus O})$$

to obtain the required $p\,\textsc{must}_i\; ta(a.s', \bigcup \overline{\mathcal{A}_{\mathsf{fw}}(p, a.s') \setminus O})$.

- If μ is an output, $\mu = \overline{a}$ for some $a \in \mathcal{N}$ and we must show that

$$p\,\textsc{must}_i\; ta(\overline{a}.s', \bigcup \overline{\mathcal{A}_{\mathsf{fw}}(p, \overline{a}.s') \setminus O}).$$

We apply Lemma 61 together with (ii) to obtain $p\,\textsc{must}_i\; ta(\overline{a}.s', \bigcup \overline{\mathcal{A}_{\mathsf{fw}}(p', s') \setminus O})$. Again, Lemma 60 together with the inclusion $\mathcal{A}_{\mathsf{fw}}(p', s') \subseteq \mathcal{A}_{\mathsf{fw}}(p, \overline{a}.s')$ ensures the required $p\,\textsc{must}_i\; ta(s, \bigcup \overline{\mathcal{A}_{\mathsf{fw}}(p, s) \setminus O})$.

◀

D Soundness

In this section we prove the converse of Proposition 48, i.e. that $\preccurlyeq_{\mathsf{AS}}$ is included in $\underset{\sim}{\sqsubset}_{\textsc{must}}$. We remark immediately that a naïve reasoning does not work. Fix two servers p and q such that $p \preccurlyeq_{\mathsf{AS}} q$. We need to prove that for every client r, if $p\,\textsc{must}_i\, r$ then $p\,\textsc{must}_i\, r$. The reasonable first proof attempt consisting in proceeding by induction on $p\,\textsc{must}_i\, r$ fails, as demonstrated by the following example.

▶ **Example 63.** Consider the two servers $p = \tau.(\overline{a} \parallel \overline{b}) + \tau.(\overline{a} \parallel \overline{c})$ and $q = \overline{a} \parallel (\tau.\overline{b} + \tau.\overline{c})$ of Equation (4). Fix a client r such that $p\,\textsc{must}_i\, r$. Rule induction yields the following inductive hypothesis:

$$\forall p', q'.\; p \parallel r \xrightarrow{\tau} p' \parallel r' \wedge\; p' \preccurlyeq_{\mathsf{AS}} q' \;\Rightarrow\; q'\,\textsc{must}_i\, r'.$$

In the proof of $q\,\textsc{must}_i\, r$ we have to consider the case where there is a communication between q and r such that, for instance, $q \xrightarrow{\overline{a}} \tau.\overline{b} + \tau.\overline{c}$ and $r \xrightarrow{a} r'$. In that case, we need to show that $\tau.\overline{b} + \tau.\overline{c}\,\textsc{must}_i\, r'$. Ideally, we would like to use the inductive hypothesis. This requires us to exhibit a p' such that $p \parallel r \xrightarrow{\tau} p' \parallel r'$ and $p' \preccurlyeq_{\mathsf{acc}} \tau.\overline{b} + \tau.\overline{c}$. However, note that there is no way to derive $p \parallel r \xrightarrow{\tau} p' \parallel r'$, because $p \not\xrightarrow{\overline{a}}$. The inductive hypothesis thus cannot be applied, and the naïve proof does not go through. ◀

This example suggests that defining an auxiliary predicate $\textsc{must}_{\mathsf{aux}}$ in some sense equivalent to $\textsc{must}_i$, but that uses explicitly *weak* outputs of servers, should be enough to prove that $\preccurlyeq_{\mathsf{AS}}$ is sound with respect to $\underset{\sim}{\sqsubset}_{\textsc{must}}$. Unfortunately, though, there is an additional nuisance to tackle: server nondeterminism.

▶ **Example 64.** Assume that we defined the predicate $\textsc{must}_i$ using weak transitions on the server side for the case of communications. Recall the argument put forward in the previous example. The inductive hypothesis now becomes the following:

$$\text{For every } p', q', \mu \text{ such that } p \stackrel{\mu}{\Longrightarrow} p' \text{ and } r \xrightarrow{\mu} r',\ p' \preccurlyeq_{\mathsf{AS}} q' \text{ implies } q' \ \textsc{must}_i\ r'.$$

To use the inductive hypothesis we have to choose a p' such that $p \stackrel{\overline{a}}{\Longrightarrow} p'$ and $p' \preccurlyeq_{\mathsf{AS}} \tau.\overline{b} + \tau.\overline{c}$. This is still not enough for the entire proof to go through, because (modulo further τ-moves) the particular p' we pick has to be related also to either $\overline{b}$ or $\overline{c}$. It is not possible to find such a p', because the two possible candidates are either $\overline{b}$ or $\overline{c}$; neither of which can satisfy $p' \preccurlyeq_{\mathsf{AS}} \tau.\overline{b} + \tau.\overline{c}$, as the right-hand side has not committed to a branch yet.

If instead of a single state p in the novel definition of $\textsc{must}_i$ we used a set of states and a suitable transition relation, the choice of either $\overline{b}$ or $\overline{c}$ will be suitably delayed. It suffices for instance to have the following states and transitions: $\{p\} \stackrel{\overline{a}}{\Longrightarrow} \{\overline{b}, \overline{c}\}$. ◀

Now that we have motivated the main intuitions behind the definition of our novel auxiliary predicate $\textsc{must}_{\mathsf{aux}}$, we proceed with the formal definitions.

The LTS of sets. Let $\mathcal{P}^+(Z)$ be the set of *non-empty* parts of Z. For any LTS $\langle A, L, \longrightarrow \rangle$, we define for every $X \in \mathcal{P}^+(A)$ and every α the sets

$$\begin{array}{lcl} D(\alpha, X) & = & \{p' \mid \exists p \in X.\, p \xrightarrow{\alpha} p'\}, \\ WD(\alpha, X) & = & \{p' \mid \exists p \in X.\, p \stackrel{\alpha}{\Longrightarrow} p'\}. \end{array}$$

Essentially we lift the standard notion of state derivative to sets of states. We construct the LTS $\langle \mathcal{P}^+(A), \mathsf{Act}_\tau, \longrightarrow \rangle$ by letting $X \xrightarrow{\alpha} D(\alpha, X)$ whenever $D(\alpha, X) \neq \emptyset$. Similarly, we have $X \stackrel{\alpha}{\Longrightarrow} WD(\alpha, X)$ whenever $WD(\alpha, X) \neq \emptyset$. This construction is standard [42, 20, 21] and goes back to the determinisation of nondeterministic automata.

Let $\textsc{must}_{\mathsf{aux}}$ be defined via the rules in Figure 11. This predicate let us reason on $\textsc{must}_i$ via sets of servers, in the following sense.

▶ **Lemma 65.** *For every LTS $\mathcal{L}_A, \mathcal{L}_B$ and every $X \in \mathcal{P}^+(A)$, we have that $X \ \textsc{must}_{\mathsf{aux}}\ r$ if and only if for every $p \in X.\, p \ \textsc{must}_i\ r$.*

To lift the predicates $\preccurlyeq_{\mathsf{cnv}}$ and $\preccurlyeq_{\mathsf{acc}}$ to sets of servers, we let $\mathcal{A}_{\mathsf{fw}}(X, s) = \{O \mid \exists p \in X. O \in \mathcal{A}_{\mathsf{fw}}(p, s)\}$, and for every finite $X \in \mathcal{P}^+(A)$, we write $X \downarrow$ to mean $\forall p \in X.\, p \downarrow$, we write $X \Downarrow s$ to mean $\forall p \in X.\, p \Downarrow s$, and let

- $X \preccurlyeq^{\mathsf{set}}_{\mathsf{cnv}} q$ to mean $\forall s \in \mathsf{Act}^\star$, if $X \Downarrow s$ then $q \Downarrow s$,
- $X \preccurlyeq^{\mathsf{set}}_{\mathsf{acc}} q$ to mean $\forall s \in \mathsf{Act}^\star, X \Downarrow s$ implies $\mathcal{A}_{\mathsf{fw}}(X, s) \ll \mathcal{A}_{\mathsf{fw}}(q, s)$,
- $X \preccurlyeq^{\mathsf{set}}_{\mathsf{AS}} q$ to mean $X \preccurlyeq^{\mathsf{set}}_{\mathsf{cnv}} q \wedge X \preccurlyeq^{\mathsf{set}}_{\mathsf{acc}} q$.

These definitions imply immediately the following equivalences, $\{p\} \preccurlyeq^{\mathsf{set}}_{\mathsf{acc}} q \iff p \preccurlyeq_{\mathsf{cnv}} q$, $\{p\} \preccurlyeq^{\mathsf{set}}_{\mathsf{cnv}} q \iff p \preccurlyeq_{\mathsf{acc}} q$ and thereby the following lemma.

▶ **Lemma 66.** *For every LTS $\mathcal{L}_A, \mathcal{L}_B, p \in A$, $q \in B$, $p \preccurlyeq_{\mathsf{AS}} q$ if and only if $\{p\} \preccurlyeq^{\mathsf{set}}_{\mathsf{AS}} q$.*

The preorder $\preccurlyeq^{\mathsf{set}}_{\mathsf{AS}}$ is preserved by τ-transitions on its right-hand side, and by visible transitions on both sides. We reason separately on the two auxiliary preorders $\preccurlyeq^{\mathsf{set}}_{\mathsf{cnv}}$ and $\preccurlyeq^{\mathsf{set}}_{\mathsf{acc}}$. We need one further notion.

▶ **Lemma 67.** *Let $\mathcal{L}_A, \mathcal{L}_B \in OW$. For every set $X \in \mathcal{P}^+(A)$, and $q \in B$, such that $X \preccurlyeq^{\mathsf{set}}_{\mathsf{cnv}} q$,*

1. *$q \xrightarrow{\tau} q'$ implies $X \preccurlyeq^{\mathsf{set}}_{\mathsf{cnv}} q'$,*
2. *$X \downarrow_i$, $X \stackrel{\mu}{\Longrightarrow} X'$ and $q \xrightarrow{\mu} q'$ imply $X' \preccurlyeq^{\mathsf{set}}_{\mathsf{cnv}} q'$.*

▶ **Lemma 68.** *Let $\mathcal{L}_A, \mathcal{L}_B \in OW$. For every $X, X' \in \mathcal{P}^+(A)$ and $q \in B$, such that $X \preccurlyeq^{\mathsf{set}}_{\mathsf{acc}} q$, then*

1. $q \xrightarrow{\tau} q'$ *implies* $X \preccurlyeq^{\mathsf{set}}_{\mathsf{acc}} q'$,
2. *if $X \downarrow_i$ then for every $\mu \in \mathsf{Act}$, every q' and X' such that $q \xrightarrow{\mu} q'$ and $X \xRightarrow{\mu} X'$ we have $X' \preccurlyeq^{\mathsf{set}}_{\mathsf{acc}} q'$.*

The main technical work for the proof of soundness is carried out by the next lemma.

▶ **Lemma 69.** *Let $\mathcal{L}_A, \mathcal{L}_B \in OW$ and $\mathcal{L}_C \in OF$. For every set of servers $X \in \mathcal{P}^+(A)$, server $q \in B$ and client $r \in C$, if $X \text{ MUST}_{\mathsf{aux}}\, r$ and $X \preccurlyeq^{\mathsf{set}}_{\mathsf{AS}} q$ then $q \text{ MUST}_i\, r$.*

▶ **Proposition 70** (Soundness). *For every $\mathcal{L}_A, \mathcal{L}_B \in OF$ and servers $p \in A, q \in B$, if $FW(p) \preccurlyeq_{\mathsf{AS}} FW(q)$ then $p \sqsubseteq_{\text{MUST}} q$.*

Proof. Lemma 14 ensures that the result follows if we prove that $\mathrm{FW}(p) \sqsubseteq_{\text{MUST}} \mathrm{FW}(q)$. Fix a client r such that $\mathrm{FW}(p) \text{ MUST}_i\, r$. Lemma 69 implies the required $\mathrm{FW}(q) \text{ MUST}_i\, r$, if we show that

(i) $\{\mathrm{FW}(p)\} \text{ MUST}_{\mathsf{aux}}\, r$, and that
(ii) $\{\mathrm{FW}(p)\} \preccurlyeq^{\mathsf{set}}_{\mathsf{AS}} \mathrm{FW}(q)$.

The first fact follows from the assumption that $\mathrm{FW}(p) \text{ MUST}_i\, r$ and Lemma 65 applied to the singleton $\{\mathrm{FW}(p)\}$. The second fact follows from the hypothesis that $\mathrm{FW}(p) \preccurlyeq_{\mathsf{AS}} \mathrm{FW}(q)$ and Lemma 66. ◀

D.1 Technical results to prove soundness

We now discuss the proofs of the main technical results behind Proposition 70. The predicate $\text{MUST}_{\mathsf{aux}}$ is monotonically decreasing with respect to its first argument, and it enjoys properties analogous to the ones of MUST_i that have been shown in Lemma 35 and Lemma 36.

▶ **Lemma 71.** *For every LTS $\mathcal{L}_A, \mathcal{L}_B$ and every set $X_1 \subseteq X_2 \subseteq A$, client $r \in B$, if $X_2 \text{ MUST}_{\mathsf{aux}}\, r$ then $X_1 \text{ MUST}_{\mathsf{aux}}\, r$.*

▶ **Lemma 72.** *Let $\mathcal{L}_A \in OW$ and $\mathcal{L}_B \in OF$. For every set $X \in \mathcal{P}^+(A)$, client $r \in B$, if $\neg\text{GOOD}(r)$ and $X \text{ MUST}_{\mathsf{aux}}\, r$ then $X \downarrow_i$.*

▶ **Lemma 73.** *For every $\mathcal{L}_A, \mathcal{L}_B$, every set $X_1, X_2 \in \mathcal{P}^+(A)$, and client $r \in B$, if $X_1 \text{ MUST}_{\mathsf{aux}}\, r$ and $X_1 \xRightarrow{\varepsilon} X_2$ then $X_2 \text{ MUST}_{\mathsf{aux}}\, r$.*

▶ **Lemma 74.** *For every LTS $\mathcal{L}_A, \mathcal{L}_B$ and every $X \in \mathcal{P}^+(A)$ and $r \in B$, if $X \text{ MUST}_{\mathsf{aux}}\, r$ then for every X' such that*

(a) *If $X \xRightarrow{\varepsilon} X'$ then $X' \text{ MUST}_{\mathsf{aux}}\, r$,*
(b) *For any $\mu \in \mathsf{Act}$ and client r', if $X \xRightarrow{\mu} X'$, $r \xrightarrow{\overline{\mu}} r'$ and $\neg\text{GOOD}(r)$, then $X' \text{ MUST}_{\mathsf{aux}}\, r'$.*

▶ **Lemma 75.** *Given two LTS $\mathcal{L}_A$ and $\mathcal{L}_B$ then for every $X \in \mathcal{P}^+(A)$ and $r \in B$, if for each $p \in X$ we have that $p \text{ MUST}_i\, r$, then $X \text{ MUST}_{\mathsf{aux}}\, r$.*

▶ **Lemma 76.** *Let $\mathcal{L}_A, \mathcal{L}_B \in OW$ and $\mathcal{L}_C \in OF$. For every $X \in \mathcal{P}^+(A)$ and $q \in B$ such that $X \preccurlyeq^{\mathsf{set}}_{\mathsf{AS}} q$, for every $r \in C$ if $\neg\text{GOOD}(r)$ and $X \text{ MUST}_{\mathsf{aux}}\, r$ then $q \parallel r \xrightarrow{\tau}$.*

Proof. If either $q \xrightarrow{\tau}$ or $r \xrightarrow{\tau}$ then we prove that $q \parallel r$ performs a τ-transition vis [S-Srv]or [S-Clt], so suppose that both q and r are stable. Since q is stable we know that

$$\mathcal{A}_{\mathsf{fw}}(q, \varepsilon) = \{O(q)\}$$

The hypotheses $\neg\textsc{good}(r)$ and X $\textsc{must}_{\mathsf{aux}}$ r together with Lemma 72 imply $X \downarrow_i$ and thus $X \Downarrow \varepsilon$. The hypothesis $X \preccurlyeq^{\mathsf{set}}_{\mathsf{acc}} q$ with $s = \varepsilon$, gives us a p' such that $p \stackrel{\varepsilon}{\Longrightarrow} p' \stackrel{\tau}{\not\longrightarrow}$ and $O(p') \subseteq O(q)$. By definition there exists the weak silent trace $X \Longrightarrow X'$ for some set X' such that $\{p'\} \subseteq X'$. The hypothesis X $\textsc{must}_{\mathsf{aux}}$ r together with Lemma 73 and Lemma 71 ensure that $\{p'\}$ $\textsc{must}_{\mathsf{aux}}$ r.

As $\neg\textsc{good}(r)$, $\{p'\}$ $\textsc{must}_{\mathsf{aux}}$ r must have been derived using rule [ind-rule] which implies that $p' \parallel r \stackrel{\tau}{\longrightarrow}$. As both r is stable by assumption, and p' is stable by definition, this τ-transition must have been derived using [s-com], and so $p' \stackrel{\mu}{\longrightarrow}$ and $r \stackrel{\overline{\mu}}{\longrightarrow}$ for some $\mu \in \mathsf{Act}$. Now we distinguish whether μ is an input or an output. In the first case μ is an input. Since $\mathcal{L}_B \in \mathrm{OW}$ we use the Input-Boomerang axiom to prove $q \stackrel{\mu}{\longrightarrow}$, and thus $q \parallel r \stackrel{\tau}{\longrightarrow}$ via rule [s-com]. In the second case μ is an output, and so the inclusion $O(p') \subseteq O(q)$ implies that $q \stackrel{\mu}{\longrightarrow}$, and so we conclude again applying rule [s-com]. ◀

▶ **Lemma 77.** *Let $\mathcal{L}_A, \mathcal{L}_B \in OW$. For every $X \in \mathcal{P}^+(A)$ and $q, q' \in B$, such that $X \preccurlyeq^{\mathsf{set}}_{\mathsf{AS}} q$, then for every $\mu \in \mathsf{Act}$, if $X \Downarrow \mu$ and $q \stackrel{\mu}{\longrightarrow} q'$ then $X \stackrel{\mu}{\Longrightarrow}$.*

Proof. Then, from $X \preccurlyeq^{\mathsf{set}}_{\mathsf{cnv}} q$ and $X \Downarrow \mu$ we have that $q \Downarrow \mu$ and thus $q' \downarrow_i$. As q' converges, there must exist q'' such that

$$q \stackrel{\mu}{\Longrightarrow} q' \stackrel{\varepsilon}{\Longrightarrow} q'' \stackrel{\tau}{\not\longrightarrow}$$

and so $\mathcal{A}_{\mathsf{fw}}(q, \mu, \longrightarrow_B) \neq \emptyset$. An application of the hypothesis $X \preccurlyeq^{\mathsf{set}}_{\mathsf{acc}} q$ implies that there exists a set $\widehat{O} \in \mathcal{A}_{\mathsf{fw}}(X, \mu, \longrightarrow_A)$, and thus there exist two servers $p' \in X$ and p'' such that $p' \stackrel{\mu}{\Longrightarrow} p'' \stackrel{\tau}{\not\longrightarrow}$. Since $p' \in X$ it follows that $X \stackrel{\mu}{\Longrightarrow}$. ◀

Lemma 67 Let $\mathcal{L}_A, \mathcal{L}_B \in \mathrm{OW}$. For every set $X \in \mathcal{P}^+(A)$, and $q \in B$, such that $X \preccurlyeq^{\mathsf{set}}_{\mathsf{cnv}} q$ then

1. $q \stackrel{\tau}{\longrightarrow} q'$ implies $X \preccurlyeq^{\mathsf{set}}_{\mathsf{cnv}} q'$,
2. if $X \downarrow_i$ and $q \stackrel{\mu}{\longrightarrow} q'$ then for every set $X \stackrel{\mu}{\Longrightarrow} X'$ we have that $X' \preccurlyeq^{\mathsf{set}}_{\mathsf{cnv}} q'$.

Proof. We first prove part (1). Let us fix a trace s such that $X \Downarrow s$. We must show $q' \Downarrow s$. An application of the hypothesis $X \preccurlyeq^{\mathsf{set}}_{\mathsf{cnv}} q$ ensures $q \Downarrow s$. From the transition $q \stackrel{\tau}{\longrightarrow} q'$ and the fact that convergence is preserved by the τ-transitions we have that $q' \Downarrow s$ as required.

We now prove part (2). Fix a trace s such that $X' \Downarrow s$. Since $q \stackrel{\mu}{\longrightarrow} q'$, the required $q' \Downarrow s$ follows from $q \Downarrow \mu.s$. Thanks to the hypothesis $X \preccurlyeq^{\mathsf{set}}_{\mathsf{cnv}} q$ it suffices to show that $X \Downarrow \mu.s'$, *i.e.* that

$$\forall p \in X. p \Downarrow \mu.s'$$

Fix a server $p \in X$. We must show that

1. $p \downarrow_i$ and that
2. for any p' such that $p \stackrel{\mu}{\Longrightarrow} p'$ we have $p' \Downarrow s$.

The first requirement follows from the hypothesis $X \downarrow_i$. The second requirement follows from the transition $p \stackrel{\mu}{\Longrightarrow} p'$, from the assumption $X' \Downarrow s$, and the hypothesis that $X \stackrel{\mu}{\Longrightarrow} X'$, which ensures that $p' \in X'$ and thus by definition of $X' \Downarrow s$ that $p' \Downarrow s$. ◀

Lemma 68 Let $\mathcal{L}_A, \mathcal{L}_B \in \mathrm{OW}$. For every $X, X' \in \mathcal{P}^+(A)$ and $q \in B$, such that $X \preccurlyeq^{\mathsf{set}}_{\mathsf{acc}} q$, then

1. $q \stackrel{\tau}{\longrightarrow} q'$ implies $X \preccurlyeq^{\mathsf{set}}_{\mathsf{acc}} q'$,
2. for every $\mu \in \mathsf{Act}$, if $X \downarrow_i$, then for every $q \stackrel{\mu}{\longrightarrow} q'$ and set $X \stackrel{\mu}{\Longrightarrow} X'$ we have $X' \preccurlyeq^{\mathsf{set}}_{\mathsf{acc}} q'$.

Proof. To prove part (1) fix a trace $s \in \mathsf{Act}^\star$ such that $X \Downarrow s$. We have to explain why $\mathcal{A}_{\mathsf{fw}}(X, s) \ll \mathcal{A}_{\mathsf{fw}}(q', s)$. By unfolding the definitions, this amounts to showing that

$$\forall O \in \mathcal{A}_{\mathsf{fw}}(q', s).\, \exists p_{\mathsf{attaboy}} \in X.\, \exists \widehat{O} \in \mathcal{A}_{\mathsf{fw}}(p_{\mathsf{attaboy}}, s).\, \widehat{O} \subseteq O \qquad (\star)$$

Fix a set $O \in \mathcal{A}_{\mathsf{fw}}(q', s)$. By definition there exists some q'' such that $q' \stackrel{s}{\Longrightarrow} q'' \stackrel{\tau}{\not\longrightarrow}$, and that $O = O(q'')$. The definition of $\mathcal{A}_{\mathsf{fw}}(-,-)$ and the silent move $q \xrightarrow{\tau} q'$ ensures that $O \in \mathcal{A}_{\mathsf{fw}}(q, s)$. The hypothesis $X \preccurlyeq^{\mathsf{set}}_{\mathsf{acc}} q$ and that $X \Downarrow s$ now imply that $\mathcal{A}_{\mathsf{fw}}(X, s) \ll \mathcal{A}_{\mathsf{fw}}(q, s)$, which together with $O \in \mathcal{A}_{\mathsf{fw}}(q, s)$ implies exactly Equation $(\star)$.

We now prove part (2). To show $X' \preccurlyeq^{\mathsf{set}}_{\mathsf{acc}} q'$ fix a trace $s \in \mathsf{Act}^\star$ such that $X' \Downarrow s$.

We have to explain why $\mathcal{A}_{\mathsf{fw}}(X, s) \ll \mathcal{A}_{\mathsf{fw}}(q', s)$. By unfolding the definitions we obtain our aim,

$$\forall O \in \mathcal{A}_{\mathsf{fw}}(q', s).\, \exists p_{\mathsf{attaboy}} \in X'.\, \exists \widehat{O} \in \mathcal{A}_{\mathsf{fw}}(p_{\mathsf{attaboy}}, s).\, \widehat{O} \subseteq O \qquad (\star\star)$$

To begin with, we prove that $X \Downarrow \mu.s$. Since $X \stackrel{\mu}{\Longrightarrow} X'$ we know that $X \stackrel{\mu}{\Longrightarrow} X'$. This, together with $X \downarrow_i$ and $X' \Downarrow s$ implies the convergence property we are after.

Now fix a set $O \in \mathcal{A}_{\mathsf{fw}}(q', s)$. Thanks to the transition $q \xrightarrow{\mu} q'$, we know that $O \in \mathcal{A}_{\mathsf{fw}}(q, \mu.s)$. The hypothesis $X \preccurlyeq^{\mathsf{set}}_{\mathsf{acc}} q$ together with $X \Downarrow \mu.s$ implies that there exists a server $p_{\mathsf{attaboy}} \in X$ such that there exists an $\widehat{O} \in \mathcal{A}_{\mathsf{fw}}(p_{\mathsf{attaboy}}, \mu.s)$. This means that $p_{\mathsf{attaboy}} \stackrel{\mu}{\Longrightarrow} p'_{\mathsf{attaboy}}$ and that $\widehat{O} \in \mathcal{A}_{\mathsf{fw}}(p'_{\mathsf{attaboy}}, s)$. Since $X \stackrel{\mu}{\Longrightarrow} X'$ we know that $p'_{\mathsf{attaboy}} \in X'$ and this concludes the argument.

◀

▶ **Lemma 78.** *For every $\mathcal{L}_A \in OW$, $\mathcal{L}_B \in OF$, every set of processes $X \in \mathcal{P}^+(A)$, every $r \in B$, and every $\mu \in \mathsf{Act}$, if X $\textsc{must}_{\mathsf{aux}}$ r, $\neg\textsc{good}(r)$ and $r \xrightarrow{\mu}$ then $X \Downarrow \overline{\mu}$.*

Lemma 69 Let $\mathcal{L}_A, \mathcal{L}_B \in \mathrm{OW}$ and $\mathcal{L}_C \in \mathrm{OF}$. For every set of processes $X \in \mathcal{P}^+(A)$, server $q \in B$ and client $r \in C$, if X $\textsc{must}_{\mathsf{aux}}$ r and $X \preccurlyeq^{\mathsf{set}}_{\mathsf{AS}} q$ then $q \, \textsc{must}_i \, r$.

Proof. We proceed by induction on the derivation of X $\textsc{must}_{\mathsf{aux}}$ r. In the base case, $\textsc{good}(r)$ so we trivially derive $q \, \textsc{must}_i \, r$. In the inductive case the proof of the hypothesis X $\textsc{must}_{\mathsf{aux}}$ r terminates with an application of [Mset-step]. Since $\neg\textsc{good}(r)$, we show the result applying [ind-rule]. This requires us to prove that

(1) $q \parallel r \xrightarrow{\tau}$, and that

(2) for all q', r' such that $q \parallel r \xrightarrow{\tau} q' \parallel r'$ we have $q' \, \textsc{must}_i \, r'$.

The first fact is a consequence of Lemma 76, which we can apply because $\neg\textsc{good}(r)$ and thanks to the hypothesis $X \preccurlyeq^{\mathsf{set}}_{\mathsf{acc}} q$ and X $\textsc{must}_{\mathsf{aux}}$ r. To prove the second fact, fix a transition $q \parallel r \xrightarrow{\tau} q' \parallel r'$. We have to explain why the following properties are true,

(a) for every q'. $q \xrightarrow{\tau} q'$ implies $q' \, \textsc{must}_i \, r$,

(b) for every r. $r \xrightarrow{\tau} r'$ implies $q \, \textsc{must}_i \, r'$,

(c) for every q', r' and $\mu \in \mathsf{Act}$, $q \xrightarrow{\mu} q'$ and $r \xrightarrow{\overline{\mu}} r'$ imply $q' \, \textsc{must}_i \, r'$.

First, note that X $\textsc{must}_{\mathsf{aux}}$ r, $\neg\textsc{good}(r)$, and Lemma 72 imply $X \downarrow$. Second, the inductive hypotheses state that for every r', non-empty set X', and q the following facts hold,

(i) $X \xrightarrow{\tau} X'$ and $X' \preccurlyeq^{\mathsf{set}}_{\mathsf{AS}} q$ implies $q \, \textsc{must}_i \, r$,

(ii) $r \xrightarrow{\tau} r'$ and $X \preccurlyeq^{\mathsf{set}}_{\mathsf{AS}} q$ implies $q \, \textsc{must}_i \, r'$,

(iii) for every and $\mu \in \mathsf{Act}$, $X \stackrel{\overline{\mu}}{\Longrightarrow} X'$ and $r \xrightarrow{\mu} r'$, and $X' \preccurlyeq^{\mathsf{set}}_{\mathsf{AS}} q$ implies $q \, \textsc{must}_i \, r'$.

$$\begin{tikzcd} p \arrow[r, "a"] & p_1 \arrow[d, "b"] \\ & p_3 \end{tikzcd} \text{ implies } \begin{tikzcd} p \arrow[r, "a"] \arrow[d, "b"] & p_1 \arrow[d, "b"] \\ p_2 \arrow[r, "a"] & p_3 \end{tikzcd} \qquad p \text{ implies } p \xrightarrow{a} p'$$

INPUT-COMMUTATIVITY INPUT-RECEPTIVITY

Figure 12 The INPUT-RECEPTIVITY axiom of [92], and our version of input-commutativity, which allows swapping only consecutive inputs.

To prove (a) we use $X \downarrow$ and the hypothesis $X \preccurlyeq^{\mathsf{set}}_{\mathsf{cnv}} q$ to obtain $q \downarrow_i$. A rule induction on $q \downarrow_i$ now suffices: in the base case (a) is trivially true and in the inductive case (a) follows from Lemma 67(1) and Lemma 68(1), and the inductive hypothesis.

The requirement (b) follows directly from the hypothesis $X \preccurlyeq^{\mathsf{set}}_{\mathsf{AS}} q$ and part (ii) of the inductive hypothesis.

To see why (c) holds, fix an action μ such that $q \xrightarrow{\mu} q'$ and $r \xrightarrow{\overline{\mu}} r'$. Since $\neg\textsc{good}(r)$ Lemma 78 implies that $X \Downarrow \mu$, and so Lemma 77 proves that $X \stackrel{\mu}{\Longrightarrow}$. In turn this implies that there exists a X' such that $X \stackrel{\mu}{\Longrightarrow} X'$, and thus Lemma 67(2) and Lemma 68(2) prove that $X' \preccurlyeq^{\mathsf{set}}_{\mathsf{AS}} q'$ holds, and (iii) ensures the result, *i.e.* that q' $\textsc{must}_i$ r'. ◀

E Traces in normal form and further alternative characterisations

As hinted at in the main body of the paper, we characterise the MUST-preorder using only the causal order of actions on traces. In this appendix we outline the necessary constructions and our reasoning. All the results are mechanised.

Let $\mathsf{nf} : \mathsf{Act}^\star \longrightarrow (MI \times MO)^\star$ be the function

$$\mathsf{nf}(s) = (I_0, M_0), (I_1, M_2), \ldots, (I_n, M_n)$$

which is defined inductively in Figure 13. The intuition is that given a trace s, the function nf forgets the orders of actions in sequences of consecutive inputs, and in sequences of consecutive outputs, thereby transforming them in multisets. On the other hand nf preserves the order among these sequences, for instance

$$\mathsf{nf}(ca\overline{b}\overline{d}\overline{d}ae\overline{e}\overline{f}e) = (\{\!|c, a|\!\}, \{\!|\overline{b}, \overline{d}, \overline{d}|\!\}), (\{\!|a|\!\}, \{\!|\overline{e}, \overline{f}|\!\}), (\{\!|e|\!\}, \varnothing)$$

Let σ range over the set $(MI \times MO)^\star$. We say that σ is a trace in *normal form*, and we write $p \stackrel{\sigma}{\Longrightarrow} q$, whenever there exists $s \in \mathsf{Act}^\star$ such that $p \stackrel{s}{\Longrightarrow} q$ and $\mathsf{nf}(s) = \sigma$.

We lift in the obvious way the predicates $\preccurlyeq_{\mathsf{cnv}}, \preccurlyeq_{\mathsf{acc}}$, and $\preccurlyeq^{\mathsf{fw}}_{\mathsf{acc}}$ to traces in nformal forms. For every $\mathcal{L}_A, \mathcal{L}_B$ and $p \in A, q \in B$ let

- $p \preccurlyeq^{\mathsf{cnv}}_{\mathsf{asyn}} q$ to mean $\forall \sigma \in (MI \times MO)^\star.\, p \Downarrow \sigma$ implies $q \Downarrow \sigma$,
- $p \preccurlyeq^{\mathsf{acc}}_{\mathsf{asyn}} q$ to mean $\forall \sigma \in (MI \times MO)^\star.\, p \Downarrow \sigma$ implies $\mathcal{A}_{\mathsf{fw}}(p, \sigma) \ll \mathcal{A}_{\mathsf{fw}}(q, \sigma)$,
- $p \preccurlyeq^{\mathsf{asyn}}_{\mathrm{MS}} q$ to mean $\forall \sigma \in (MI \times MO)^\star.\, p \Downarrow \sigma$ implies that if $\forall L.(p \,\mathsf{after}\, \sigma, \longrightarrow_A)$ MUST L then $(q \,\mathsf{after}\, \sigma, \longrightarrow_B)$ MUST L

▶ **Definition 79.** *Let*

- $p \preccurlyeq^{\mathsf{NF}}_{\mathsf{AS}} q$ *whenever* $q \preccurlyeq^{cnv}_{\mathsf{asyn}} p \wedge p \preccurlyeq^{acc}_{\mathsf{asyn}} q$,
- $p \preccurlyeq^{\mathsf{NF}}_{\mathsf{MS}} q$ *whenever* $q \preccurlyeq^{cnv}_{\mathsf{asyn}} p \wedge p \preccurlyeq^{asyn}_{MS} q$. ■

$$\begin{array}{rcl} \mathsf{nf}(\varepsilon) & = & \varepsilon \\ \mathsf{nf}(s) & = & \mathsf{nf}'(s, \varnothing, \varnothing) \\ \\ \mathsf{nf}'(\varepsilon, I, M) & = & (I, M) \\ \mathsf{nf}'(\overline{a}.b.s, I, M) & = & (I, \{\!|a|\!\} \uplus M), \mathsf{nf}'(s, \{\!|b|\!\}, \varnothing) \\ \mathsf{nf}'(a.s, I, M) & = & \mathsf{nf}'(s, \{\!|a|\!\} \uplus I, M) \\ \mathsf{nf}'(\overline{a}.s, I, M) & = & \mathsf{nf}'(s, I, \{\!|b|\!\} \uplus M) \end{array}$$

Figure 13 Definition of the trace normalization function nf

If an LTS is of forwarders, *i.e.* $\mathcal{L} \in \mathrm{OW}$, the transition relation $\longrightarrow$ is *input-receptive* (Axiom (IB4), Table 2 of [92]), and in Lemma 80 we prove that it enjoys a restricted version of Input-commutativity, and that so does its weak version. Sequences of input actions $s \in \mathcal{N}^\star$ enjoy a form of diamond property in $\Longrightarrow$. The crucial fact pertains consecutive input actions.

▶ **Lemma 80.** *For every $\mathcal{L}_A \in OW$, every $p, q \in A$ and every $a, b \in \mathcal{N}$, if $p \stackrel{a.b}{\Longrightarrow} q$ then $p \stackrel{b.a}{\Longrightarrow} \cdot \simeq q$.*

Lemma 80, together with an induction on traces, allows us to prove that nf preserves convergence and acceptance sets.

▶ **Lemma 81.** *For every $\mathcal{L}_A \in OW$, every $p \in A$ and every $s \in \mathsf{Act}^\star$ and we have that*
1. *$p \stackrel{s}{\Longrightarrow} q$ iff $p \stackrel{\mathsf{nf}(s)}{\Longrightarrow} \cdot \simeq q$, and if the first trace does not pass through a successful state then the normal form does not either,*
2. *$p \Downarrow \mathsf{nf}(s)$ iff $p \Downarrow s$,*
3. *$\mathcal{A}_{\mathsf{fw}}(p, s) = \mathcal{A}_{\mathsf{fw}}(p, \mathsf{nf}(s))$.*

We thereby obtain two other characterisations of the contextual preorder $\sqsubseteq_{\text{MUST}}$: Theorem 17 and Lemma 81 ensure that the preorders $\sqsubseteq_{\text{MUST}}$, $\preccurlyeq^{\mathsf{NF}}_{\mathsf{AS}}$, and $\preccurlyeq^{\mathsf{NF}}_{\mathsf{MS}}$ coincide.

▶ **Corollary 82.** *For every $\mathcal{L}_A, \mathcal{L}_B \in OF$, every $p \in A$ and $q \in B$, the following facts are equivalent:*
1. *$p \sqsubseteq_{\text{MUST}} q$,*
2. *$FW(p) \preccurlyeq^{\mathsf{NF}}_{\mathsf{AS}} FW(q)$,*
3. *$FW(p) \preccurlyeq^{\mathsf{NF}}_{\mathsf{MS}} FW(q)$.*

F Asynchronous CCS

Here we recall the syntax and the LTS of asynchronous CCS, or `ACCS` for short, a version of CCS where outputs have no continuation and sum is restricted to input- and τ-guards. This calculus, which is inspired by the variant of the asynchronous π-calculus considered by [4, 5] for their study of asynchronous bisimulation, was first investigated by [92], and subsequently resumed by other authors such as [24]. Different asynchronous variants of CCS were studied in the same frame of time by [82], whose calculus included output prefixing and operators from ACP, and by [39], whose calculus TACCS included asynchronous output prefixing and featured two forms of choice, internal and external, in line with previous work on testing semantics [78].

$$[\textsc{Input}]\ \frac{}{a.p \xrightarrow{a} p} \qquad [\textsc{Tau}]\ \frac{}{\tau.p \xrightarrow{\tau} p}$$

$$[\textsc{Mb-Out}]\ \frac{}{\boxed{\overline{a}} \xrightarrow{\overline{a}} \mathsf{0}} \qquad [\textsc{Unf}]\ \frac{}{\mathsf{rec}x.p \xrightarrow{\tau} p[{}^{\mathsf{rec}x.p}/_{x}]}$$

$$[\textsc{Sum-L}]\ \frac{p \xrightarrow{\alpha} p'}{p + q \xrightarrow{\alpha} p'} \qquad [\textsc{Sum-R}]\ \frac{q \xrightarrow{\alpha} q'}{p + q \xrightarrow{\alpha} q'}$$

$$[\textsc{Par-L}]\ \frac{p \xrightarrow{\alpha} p'}{p \parallel q \xrightarrow{\alpha} p' \parallel q} \qquad [\textsc{Par-R}]\ \frac{q \xrightarrow{\alpha} q'}{p \parallel q \xrightarrow{\alpha} p \parallel q'}$$

$$[\textsc{Com}]\ \frac{p \xrightarrow{\mu} p' \quad q \xrightarrow{\overline{\mu}} q'}{p \parallel p' \xrightarrow{\tau} q \parallel q'}$$

Figure 14 The LTS of processesThe meta-variables are $a \in \mathcal{N}, \mu \in \mathsf{Act}, \alpha \in \mathsf{Act}_\tau$.

The syntax of terms is given in Equation (3). As usual, $\mathsf{rec}x.p$ binds the variable x in p, and we use standard notions of free variables, open and closed terms. Processes, ranged over by $p, q, r, \ldots$ are *closed* terms. The operational semantics of processes is given by the LTS $\langle \mathtt{ACCS}, \mathsf{Act}_\tau, \longrightarrow \rangle$ specified by the rules in Figure 14.

The prefix $a.p$ represents a *blocked* process, which waits to perform the input a, *i.e.* to interact with the atom $\overline{a}$, and then becomes p; and atoms $\boxed{\overline{a}}, \boxed{\overline{b}}, \ldots$ represent output messages. We will discuss in detail the role played by atoms in the calculus, but we first overview the rest of the syntax. We include $\mathsf{1}$ to syntactically denote successful states. The prefix $\tau.p$ represents a process that does one step of internal computation and then becomes p. The sum $g_1 + g_2$ is a process that can behave as g_1 or g_2, but not both. Thus, for example $\tau.p + \tau.q$ models an `if ...then ...else`, while $a.p + b.q$ models a `match ...with`. Note that the sum operator is only defined on *guards*, namely it can only take as summands $\mathsf{0}, \mathsf{1}$ or input-prefixed and τ-prefixed processes. While the restriction to guarded sums is a standard one, widely adopted in process calculi, the restriction to input and τ guards is specific to asynchronous calculi. We will come back to this point after discussing atoms and mailboxes. Parallel composition $p \parallel q$ runs p and q concurrently, allowing them also to interact with each other, thanks to rule [Com]. For example

$$b.a.\mathsf{0} \parallel b.c.\mathsf{0} \parallel \boxed{(\overline{a} \parallel \overline{b} \parallel \overline{c})} \tag{7}$$

represents a system in which two concurrent processes, namely $b.a.\mathsf{0}$ and $b.c.\mathsf{0}$, are both ready to consume the message $\overline{b}$ from a third process, namely $\boxed{\overline{a} \parallel \overline{b} \parallel \overline{c}}$. This last process is a parallel product of atoms, and it is not guarded, hence it is best viewed as an unordered mailbox shared by *all* the processes running in parallel with it. For instance in (7) the terms $b.a.\mathsf{0}$ and $b.c.\mathsf{0}$ share the mailbox $\boxed{\overline{a} \parallel \overline{b} \parallel \overline{c}}$. Then, depending on which process consumes $\overline{b}$, the overall process will evolve to either $b.c.\mathsf{0} \parallel \boxed{\overline{c}}$ or $b.a.\mathsf{0} \parallel \boxed{\overline{a}}$, which are both stuck.[11]

[11] The global shared mailbox that we treat is reminiscent but less general than the chemical "soup" of [18].

Concerning the sum construct, we follow previous work on asynchronous calculi ([4, 5, 89, 24]) and only allow input-prefixed or τ-prefixed terms as summands. The reason for forbidding atoms in sums is that the nondeterministic sum is essentially a synchronising operator: the choice is solved by executing an action in one of the summands and *simultaneously* discarding all the other summands. Then, if an atom were allowed to be a summand, this atom could be discarded by performing an action in another branch of the choice. This would mean that a process would have the ability to withdraw a message from the mailbox without consuming it, thus contradicting the intuition that the mailbox is a shared entity which is out of the control of any given process, and with which processes can only interact by feeding a message into it or by consuming a message from it. In other words, this restriction on the sum operator ensures that atoms indeed represent messages in a global mailbox. For further details see the discussion on page 191 of [89].

A structural induction on the syntax ensures that processes perform only a finite number of outputs:

▶ **Lemma 83.** *For every* $p \in \texttt{ACCS}. \mid O(p) \mid \in \mathbb{N}$.

Together with Lemma 86, this means that at any point of every execution the global mailbox contains a finite number of messages. Since the LTS is image-finite under any visible action, a consequence of Lemma 83 is that the number of reducts of a program is finite.

▶ **Lemma 84.** *For every* $p \in \texttt{ACCS}. \mid \{p' \in \texttt{ACCS} \mid p \xrightarrow{\tau} p'\} \mid \in \mathbb{N}$.

Proof. Structural induction on p. The only non-trivial case is if $p = p_1 \parallel p_2$. In this case the result is a consequence of the inductive hypothesis, of Lemma 83 and of the following fact: $p \xrightarrow{\tau} q$ iff

1. $p_1 \xrightarrow{\tau} p_1'$ and $q = p_1' \parallel p_2$,
2. $p_2 \xrightarrow{\tau} p_2'$ and $q = p_1 \parallel p_2'$,
3. $p_1 \xrightarrow{a} p_1'$ and $p_2 \xrightarrow{\overline{a}} p_2'$ and $q = p_1' \parallel p_2'$,
4. $p_1 \xrightarrow{\overline{a}} p_1'$ and $p_2 \xrightarrow{a} p_2'$ and $q = p_1' \parallel p_2'$.

In the third case the number of possible output actions $\overline{a}$ is finite thanks to Lemma 83, and so is the number of reducts p_1' and p_2', so the set of term $p_1' \parallel p_2'$ is decidable. The same argument works for the fourth case. ◀

Thanks to Lemma 83 and Lemma 84, Lemma 84 holds also for the LTS modulo structural congruence, i.e. $\langle \texttt{ACCS}_\equiv, \longrightarrow_\equiv, \mathsf{Act}_\tau \rangle$.

F.1 Structural equivalence and its properties

To manipulate the syntax of processes we use a standard structural congruence denoted $\equiv$, stating that ACCS is a commutative monoid with identity 0 with respect to both sum and parallel composition.

A first fact is the following one.

▶ **Lemma 85.** *For every* $\mu \in \overline{\mathcal{N}}$ *and* $\alpha \in \mathsf{Act}_\tau$, *if* $p \overset{\mu.\alpha}{\Longrightarrow} q$ *then* $p \overset{\alpha.\mu}{\Longrightarrow} \cdot \equiv q$.

In that context the components of the soup are not just atoms, but whole parallel components: in fact, the chemical soup allows parallel components to come close in order to react with each other, exactly as the structural congruence of [74], which indeed was inspired by the Chemical Abstract Machine.

```
Class LtsEq (A L : Type) `{Lts A L} := {
  eq_rel : A → A → Prop;

  eq_rel_refl p : eq_rel p p;
  eq_symm p q : eq_rel p q → eq_rel q p;
  eq_trans p q r :
  eq_rel p q → eq_rel q r → eq_rel p r;

  eq_spec p q (α : Act L) :
  (∃ p', (eq_rel p p') ∧ p' ⟶{α} q)
  →
  (∃ q', p ⟶{α} q' ∧ (eq_rel q' q))
}.
```

Figure 15 A typeclass for LTSs where a structural congruence exists over states.

As sum and parallel composition are commutative monoids, we use the notation

$$\begin{array}{lll} \Sigma\{g_0, g_1, \ldots g_n\} & \text{to denote} & g_0 + g_1 + \ldots + g_n \\ \Pi\{p_0, p_1, \ldots p_n\} & \text{to denote} & p_0 \parallel p_1 \parallel \ldots \parallel p_n \end{array}$$

This notation is useful to treat the global shared mailbox. In particular, if $\{\!|\mu_0, \mu_1, \ldots \mu_n|\!\}$ is a multiset of output actions, then the syntax $\Pi\{\!|\mu_0, \mu_1, \ldots \mu_n|\!\}$ represents the shared mailbox that contains the messages μ_i; for instance $\Pi\{\!|\overline{a}, \overline{a}, \overline{c}|\!\} = \boxed{\overline{a} \parallel \overline{a} \parallel \overline{c}}$. We use the colour $\boxed{-}$ to highlight the content of the mailbox. Intuitively a shared mailbox contains the messages that are ready to be read, i.e. the outputs that are immediately available (i.e. not guarded by any prefix operation). For example in

$$\boxed{\overline{c}} \parallel a.(\overline{b} \parallel c.d.\,1) \parallel \boxed{\overline{d}} \parallel \tau.\overline{e}$$

the mailbox is $\boxed{\overline{c} \parallel \overline{d}}$. The global mailbox that we denote with $\boxed{-}$ is exactly the buffer B in the *configurations* of [95], and reminiscent of the ω used by [28]. The difference is that ω represents an unbounded *ordered* queue, while our mailbox is an unbounded *unordered* buffer.

As for the relation between output actions in the LTS and the global mailbox, an output $\overline{a}$ can take place if and only if the message $\overline{a}$ appears in the mailbox:

▶ **Lemma 86.** *For every $p \in ACCS$,*

1. *for every $a \in \mathcal{N}.\, p \xrightarrow{\overline{a}} p'$ implies $p \equiv p' \parallel \boxed{\overline{a}}$,*
2. *there exists p' such that $p \equiv p' \parallel \boxed{\Pi M}$, and p' performs no output action.*

This lemma and Lemma 85 essentially hold, because, as already pointed out in Section 2, the syntax enforces outputs to have no continuation.

The following lemma states a fundamental fact ([58, Lemma 2.13], [75, Proposition 5.2], [89, Lemma 1.4.15]). Its proof is so tedious that even the references we have given only sketch it. In this paper we follow the masters example, and give merely a sketch. However, we have a complete machine-checked proof.

▶ **Lemma 87.** *For every $p, q \in ACCS$ and $\alpha \in \mathsf{Act}_\tau.\, p \equiv \cdot \xrightarrow{\alpha} q$ implies $p \xrightarrow{\alpha} \cdot \equiv q$.*

$$
\begin{array}{rll}
[\textsc{S-szero}] & p + 0 \equiv p & \\
[\textsc{S-scom}] & p + q \equiv q + p & \\
[\textsc{S-sass}] & (p + q) + r \equiv p + (q + r) & \\
\\
[\textsc{S-pzero}] & p \parallel 0 \equiv p & \\
[\textsc{S-pcom}] & p \parallel q \equiv q \parallel p & \\
[\textsc{S-pass}] & (p \parallel q) \parallel r \equiv p \parallel (q \parallel r) & \\
\\
[\textsc{S-refl}] & p \equiv p & \\
[\textsc{S-symm}] & p \equiv q & \textit{if } q \equiv p \\
[\textsc{S-trans}] & p \equiv q & \textit{if } p \equiv p' \textit{ and } p' \equiv q \\
\\
[\textsc{S-prefix}] & \alpha.p \equiv \alpha.q & \textit{if} p \equiv q \\
[\textsc{S-sum}] & p + q \equiv p' + q & \textit{if} p \equiv p' \\
[\textsc{S-ppar}] & p \parallel q \equiv p' \parallel q & \textit{if} p \equiv p'
\end{array}
$$

Figure 16 Rules to define structural congruence on `ACCS`.

Proof sketch. We need to show that if there exists a process p' such that $p \equiv p'$ and $p' \xrightarrow{\alpha} q$ then there exists a process q' such that $p \xrightarrow{\alpha} p'$ and $p' \equiv q$. The proof is by induction on the derivation $p \equiv p'$.

We illustrate one case with the rule [S-TRANS]. The hypotheses tell us that there exists $\hat{p}$ such that $p \equiv \hat{p}$ and $\hat{p} \equiv p'$, that $p' \xrightarrow{\alpha} q$, and the inductive hypotheses that

(a) for all q' $s.t$ $\hat{p} \xrightarrow{\alpha} q'$ implies that there exists a $\hat{q}$ such that $p \xrightarrow{\alpha} \hat{q}$ and $\hat{q} \equiv q'$

(b) for all q' $s.t$ $p' \xrightarrow{\alpha} q'$ implies that there exists a $\hat{q}$ such that $\hat{p} \xrightarrow{\alpha} \hat{q}$ and $\hat{q} \equiv q'$

By combining part (b) and $p' \xrightarrow{\alpha} q$ we obtain a $\hat{q}_1$ such that $\hat{p} \xrightarrow{\alpha} \hat{q}_1$ and $\hat{q}_1 \equiv q$. Using part (a) together with $\hat{p}_1 \xrightarrow{\alpha} \hat{q}_1$ we have that there exists a $\hat{q}_2$ such that $p \xrightarrow{\alpha} \hat{q}_2$ and $\hat{q}_2 \equiv \hat{q}_1$. We then have that $p \xrightarrow{\alpha} \hat{q}_2$ and it remains to show that $\hat{q}_2 \equiv q$. We use the transitivity property of the structural congruence relation to show that $\hat{q}_2 \equiv \hat{q}_1$ and $\hat{q}_1 \equiv q$ imply $\hat{q}_2 \equiv q$ as required and we are done with this case. ◀

Time is a finite resource. The one spent to machine check Lemma 87 would have been best invested into bibliographical research. Months after having implemented the lemma we realised that [3] already had an analogous result for a mechanisation of the π-calculus. Lemma 87 is crucial to prove the Harmony Lemma, which states that τ-transitions coincide with the standard reduction relation of `ACCS`. This is out of the scope of our discussion, and we point the interested reader to Lemma 1.4.15 of [89], and to the list of problems presented on the web-page of The Concurrent Calculi Formalisation Benchmark.[12]

We give a corollary that is useful to prove Lemma 89.

▶ **Corollary 88.** *For every $p, q \in$ ACCS$, \alpha \in \mathsf{Act}_\tau . p \equiv q$ implies that $p \xrightarrow{\alpha} \cdot \equiv r$ if and only if $q \xrightarrow{\alpha} \cdot \equiv r$.*

Proof. Since $q \equiv p \xrightarrow{\alpha} p' \equiv r$ Lemma 87 implies $q \xrightarrow{\alpha} \cdot \equiv p'$, thus $q \xrightarrow{\alpha} \cdot \equiv r$ by transitivity of $\equiv$. The other implication follows from the same argument and the symmetry of $\equiv$. ◀

---

[12] `https://concurrentbenchmark.github.io/`

A consequence of Lemma 86 is that the LTS $\langle \mathtt{ACCS}_{\equiv}, \longrightarrow_{\equiv}, \mathsf{Act}_\tau \rangle$ enjoys the axioms in Figure 2, and thus it is OF. [92, Theorem 4.3] proves it reasoning modulo bisimilarity, while we reason modulo structural equivalence.

▶ **Lemma 89.** *For every $p \in \mathit{ACCS}$, and $a \in \mathcal{N}$ the following properties are true,*

- *for every $\alpha \in \mathsf{Act}_\tau$. $p \xrightarrow{\overline{a}}\xrightarrow{\alpha} p_3$ implies $p \xrightarrow{\alpha}\xrightarrow{\overline{a}} \cdot \equiv p_3$;*
- *for every $\alpha \in \mathsf{Act}_\tau . \alpha \notin \{\tau, \overline{a}\}$. $p \xrightarrow{\overline{a}} p'$ and $p \xrightarrow{\alpha} p''$ imply that $p'' \xrightarrow{\overline{a}} q$ and $p' \xrightarrow{\alpha} q$ for some q;*
- *$p \xrightarrow{\overline{a}} p'$ and $p \xrightarrow{\overline{a}} p''$ imply $p' \equiv p''$;*
- *$p \xrightarrow{\overline{a}} p' \xrightarrow{a} q$ implies $p \xrightarrow{\tau} \cdot \equiv q$;*
- *$p \xrightarrow{\overline{a}} p'$ and $p \xrightarrow{\tau} p''$ imply that $p' \xrightarrow{\tau} q$ and $p'' \xrightarrow{\overline{a}} q$; or that $p' \xrightarrow{\tau} p''$.*
- *for every p' if there exists a $\hat{p}$ such that $p \xrightarrow{\overline{a}} \hat{p}$ and $p' \xrightarrow{\overline{a}} \hat{p}$ then $p \equiv p'$*

Proof. To show FEEDBACK we begin via Lemma 86 which proves $p \equiv p' \parallel \boxed{\overline{a}}$. We derive $p' \parallel \boxed{\overline{a}} \xrightarrow{\tau} q'$ and apply Corollary 88 to obtain $p \xrightarrow{\tau} \cdot \equiv q$.

We prove OUTPUT-TAU. The hypothesis and Lemma 86 imply that $p \equiv p' \parallel \boxed{a}$. Since $p \xrightarrow{\tau} p''$ it must be the case that $p' \xrightarrow{\tau} \hat{p}$ for some $\hat{p}$, and $p'' = \hat{p} \parallel \boxed{a}$. Let $q = \hat{p}$. We have that $p'' \xrightarrow{a} \hat{p} \parallel 0 \equiv q$. ◀

Processes that enjoy OUTPUT-TAU are called *non-preemptive* in [41, Definition 10].

Each time a process p reduces to a *stable* process p', it does so by consuming at least part of the mailbox, for instance a multiset of outputs N, thereby arriving in a state q whose inputs cannot interact with what remains of the mailbox, i.e. $M \setminus N$, where M is the original mailbox.

▶ **Lemma 90.** *For every $M \in MO$, $p, p' \in \mathit{ACCS}$, if $p \parallel \boxed{\Pi M} \stackrel{\varepsilon}{\Longrightarrow} p' \stackrel{\tau}{\not\longrightarrow}$ then there exist an $N \subseteq M$ and some $q \in \mathit{ACCS}$ such that $p \stackrel{\overline{N}}{\Longrightarrow} q \stackrel{\tau}{\not\longrightarrow}$, $O(q) \subseteq O(p')$, and $\overline{I(q)} \# (M \setminus N)$.*

Proof. By induction on the derivation of $p \parallel \boxed{\Pi M} \stackrel{\varepsilon}{\Longrightarrow} p'$.In the base case this is due to [WT-REFL], which ensures that

$$p \parallel \boxed{\Pi M} = p',$$

from which we obtain $p \parallel \boxed{\Pi M} \stackrel{\tau}{\not\longrightarrow}$. This ensures that $\overline{I(p)} \# M$.We pick as q and N respectively $p \parallel \boxed{\Pi M}$ and $\varnothing$ as $p \parallel \boxed{\Pi M} \stackrel{\varnothing}{\Longrightarrow} p \parallel \boxed{\Pi M}$ by reflexivity, and $O(p \parallel \boxed{\Pi M}) = O(p')$.

In the inductive case the derivation ends with an application of [WT-TAU] and

$$\frac{p \parallel \boxed{\Pi M} \xrightarrow{\tau} p' \qquad \dfrac{\vdots}{p' \stackrel{\varepsilon}{\Longrightarrow} p'}}{p \parallel \boxed{\Pi M} \stackrel{\varepsilon}{\Longrightarrow} p'}$$

We continue by case analysis on the rule used to infer the transition $p \parallel \boxed{\Pi M} \xrightarrow{\tau} p'$. As by definition $\boxed{\Pi M} \stackrel{\tau}{\not\longrightarrow}$, the rule is either [PAR-L], i.e. a τ-transition performed by p, or [COM], i.e. an interaction between p and $\boxed{\Pi M}$.

F.1.0.1 Rule [Par-L]:

In this case $p \xrightarrow{\tau} p''$ for some p'', thus $p'' \parallel \boxed{\Pi M} \stackrel{\varepsilon}{\Longrightarrow} p'$ and the result follows from the inductive hypothesis.

F.1.0.2 Rule [Com]:

The hypothesis of the rule ensure that $p \xrightarrow{a} p''$ and $\boxed{\Pi M} \xrightarrow{\overline{a}} p'$, and as the process $\boxed{\Pi M}$ does not perform any input, it must be the case that $a \in \mathcal{N}$, that $\overline{a} \in M$, and that $q \equiv \boxed{\Pi(M \setminus \{\!|\overline{a}|\!\})}$.[13] Note that $p' \equiv p'' \parallel \boxed{\Pi(M \setminus \{\!|\overline{a}|\!\})}$.

The inductive hypothesis ensures that for some $N' \subseteq M \setminus \{\!|\overline{a}|\!\}$ and some $q_3 \in \texttt{ACCS}$ we have

(a) $p' \stackrel{\overline{N'}}{\Longrightarrow} q_3 \stackrel{\tau}{\not\longrightarrow}$,

(b) $O(q_3) \subseteq O(p')$, and

(c) $\overline{I(q_3)} \# ((M \setminus \{\!|\overline{a}|\!\}) \setminus N')$

We conclude by letting $q = q_3$, and $N = \{\!|a|\!\} \uplus N'$. The trace $p \xrightarrow{a} p' \stackrel{N'}{\Longrightarrow} q_3$ proves that $p \stackrel{\{\!|a|\!\} \uplus N'}{\Longrightarrow} q_3$, moreover we already know that q_3 is stable. The set inclusion $O(q_3) \subseteq O(p')$ follows from b, and lastly $\overline{I(q)} \# (M \setminus (\{\!|\overline{a}|\!\} \uplus N'))$ is a consequence of $\overline{I(q_3)} \# ((M \setminus \{\!|\overline{a}|\!\}) \setminus N')$ and of $(M \setminus \{\!|\overline{a}|\!\}) \setminus N' = (I \setminus (\{\!|\overline{a}|\!\} \uplus N'))$. ◀

We define the predicate GOOD,

$$
\begin{array}{ll}
\textsc{good}(1) & \\
\textsc{good}(p \parallel q) & \textit{if } \textsc{good}(p) \textit{ or } \textsc{good}(q) \\
\textsc{good}(p + q) & \textit{if } \textsc{good}(p) \textit{ or } \textsc{good}(q)
\end{array}
$$

This predicate is preserved by structural congruence.

▶ **Lemma 91.** *For every $p, q \in ACCS$. $p \equiv q$ and $\textsc{good}(p)$ imply $\textsc{good}(q)$.*

▶ **Lemma 92.** *For every $p, q \in ACCS$. $p \equiv q$ and $p \downarrow_i$ imply $q \downarrow_i$.*

▶ **Lemma 93.** *For every $p, q \in ACCS$ and $s \in \mathsf{Act}^\star$, we have that $p \equiv q$ and $FW(p) \Downarrow s$ imply $FW(q) \Downarrow s$.*

▶ **Lemma 94.** *For every $p, r, r' \in ACCS$. $r \equiv r'$ and $p \,\textit{MUST}_i\, r$ then $p \,\textit{MUST}_i\, r'$.*

▶ **Lemma 95.** *For every $p, q, r \in ACCS$. $p \equiv q$ and $p \,\textit{MUST}_i\, r$ then $q \,\textit{MUST}_i\, r$.*

A typical technique to reason on the LTS of concurrent processes, and so also of client-server systems, is trace zipping: if $p \stackrel{s}{\Longrightarrow} p'$ and $q \stackrel{\overline{s}}{\Longrightarrow} q'$, an induction on s ensures that $p \parallel q \Longrightarrow p' \parallel q'$. Zipping together different LTS is slightly more delicate: we can zip weak transitions $\stackrel{s}{\Longrightarrow}_{\mathsf{fw}}$ together with the co-transitions $\stackrel{\overline{s}}{\Longrightarrow}$, but possibly moving inside equivalence classes of $\equiv$ instead of performing actual transitions in $\longrightarrow$.

▶ **Lemma 96** (Zipping)**.** *For every $p, q \in ACCS$*

1. *for every $\mu \in \mathsf{Act}$. if $p \xrightarrow{\mu}_{\mathsf{fw}} p'$ and $q \xrightarrow{\overline{\mu}} q'$ then $p \parallel q \xrightarrow{\tau} p' \parallel q'$ or $p \parallel q \equiv p' \parallel q'$;*
2. *for every $s \in \mathsf{Act}^\star$. if $p \stackrel{s}{\Longrightarrow}_{\mathsf{fw}} p'$ and $q \stackrel{\overline{s}}{\Longrightarrow} q'$ then $p \parallel q \stackrel{\varepsilon}{\Longrightarrow} \cdot \equiv p' \parallel q'$.*

Obviously, for every $p, q \in A$ and output $a \in \mathcal{N}$ we have

$$p \xrightarrow{\tau}_{\mathsf{fw}} q \text{ if and only if } p \xrightarrow{\tau} q \tag{8}$$

$$p \stackrel{\varepsilon}{\Longrightarrow}_{\mathsf{fw}} q \text{ if and only if } p \stackrel{\varepsilon}{\Longrightarrow} q \tag{9}$$

$$p \xrightarrow{\overline{a}}_{\mathsf{fw}} q \text{ if and only if } p \xrightarrow{\overline{a}} q \tag{10}$$

[13] In terms of LTS with mailboxes, $p' = (M \setminus \{\!|\overline{a}|\!\})$.

$$
\begin{array}{rcl}
g(\varepsilon, r) & = & r \\
g(a.s, r) & = & \boxed{\overline{a}} \parallel g(s, r) \\
g(\overline{a}.s, r) & = & a.g(s, r) + \tau.1
\end{array}
$$

$$c(s) = g(s, \tau.1) \tag{14}$$

$$ta(s, L) = g(s, h(L)) \text{ where } h(L) = \Pi\{\mu.1 \mid \mu \in L\} \tag{15}$$

Figure 17 Functions to generate clients.

together with the expected properties of finiteness, the first one amounting to the finiteness of the global mailbox in any state:

$$| \{\overline{a} \in \overline{\mathcal{N}} \mid p \xrightarrow{\overline{a}}_{\mathsf{fw}}\} | \in \mathbb{N} \tag{11}$$

$$\text{For every } \mu \in \mathsf{Act}. | \{q \mid p \xrightarrow{\mu}_{\mathsf{fw}} q\} | \in \mathbb{N} \tag{12}$$

$$| \{q \in A \mid p \xrightarrow{\tau}_{\mathsf{fw}} q\} | \in \mathbb{N} \tag{13}$$

F.2 Client generators and their properties

This subsection is devoted to the study of the semantic properties of the clients produced by the function g. In general these are the properties sufficient to obtain our completeness result.

▶ **Lemma 97.** *For every $p \in ACCS$ and $s \in \mathsf{Act}^\star$, if $p \xrightarrow{\tau}$ then $g(p, s) \xrightarrow{\tau}$.*

Proof. By induction on the sequence s. In the base case $s = \varepsilon$. The test generated by g is p, which reduces by hypothesis, and so does $g(\varepsilon, p)$. In the inductive case $s = \alpha.s_2$, and we proceed by case-analysis on α. If α is an output then $g(\alpha.s_2, p) = \overline{\alpha}.(g(s_2, p)) + \tau.1$ which reduces to 1 using the transition rule [Sum-R]. If μ is an input then $g(\alpha.s_2, p) = \boxed{\overline{\alpha}} \parallel g(s_2, p)$ which reduces using the transition rule [Par-R] and the inductive hypothesis, which ensures that $g(s_2, p)$ reduces. ◀

▶ **Lemma 98.** *For every p and s, if $\neg\text{GOOD}(p)$ then for every s, $\neg\text{GOOD}(g(s, p))$.*

Proof. The argument is essentially the same of Lemma 97 ◀

▶ **Lemma 99.** *For every $s \in \mathsf{Act}^\star$, if $g(s, q) \xrightarrow{\mu} o$ then either*

(a) *$q \xrightarrow{\mu} q'$, $s \in \mathcal{N}^\star$, and $o = \boxed{\Pi\overline{s}} \parallel q'$, or*

(b) *$s = s_1.\boxed{\mu}.s_2$ for some $s_1 \in \mathcal{N}^\star$ and $s_2 \in \mathsf{Act}^\star$, and $o \equiv \boxed{\Pi\overline{s_1}} \parallel g(s_2, q)$, and*

  (i) *$\mu \in \mathcal{N}$ implies $g(s, q) \equiv \boxed{\Pi\overline{s_1}} \parallel (\tau.1 + \mu.g(s_2, q))$,*

  (ii) *$\mu \in \boxed{\overline{\mathcal{N}}}$ implies $g(s, q) \equiv \boxed{\Pi\overline{s_1} \parallel \mu} \parallel (\tau.1 + \mu.g(s_2, q))$.*

Proof. The proof is by induction on s.

In the case, $s = \varepsilon$, and hence by definition $g(s, q) = q$. The hypotheses $g(s, q) \xrightarrow{\mu} o$ implies $q \xrightarrow{\mu} o$, and $o \equiv 0 \parallel o \equiv \Pi\varepsilon \parallel o$.

In the inductive case, $s = \nu.s'$. We have two cases, depending on whether ν is an output action or an input action.

Suppose ν is an output. In this case $g(s,q) = \tau.\,1 + \overline{\nu}.g(s',q)$. The hypothesis $g(s,q) \xrightarrow{\mu} o$ ensures that $\boxed{\overline{\nu}} = \mu$, thus μ is an input action. By letting $s_1 = \varepsilon$ and $s_2 = s'$ we obtain the required

$$g(s,q) = \tau.\,1 + \mu.g(s_2,q) \equiv \Pi\overline{s_1} \parallel (\tau.\,1 + \mu.g(s_2,q))$$

and $o \equiv \Pi\overline{s_1} \parallel g(s_2,q)$.

Now suppose that ν is an input action. By definition

$$g(s,q) = \boxed{\overline{\nu}} \parallel g(s',q) \tag{16}$$

and the inductive hypothesis ensures that either

(1) $q \xrightarrow{\mu} q'$, $s' \in \mathcal{N}^\star$, and $o' = \Pi\overline{s'} \parallel q'$, or

(2) $s' = s'_1.\boxed{\mu}.s'_2$, for some $s'_1 \in \mathcal{N}^\star$ and $s_2 \in \mathsf{Act}^\star$, and

$$o' \equiv \Pi\boxed{\overline{s'_1}} \parallel g(s'_2,q) \tag{17}$$

and

$$\mu \in \mathcal{N} \text{ implies } g(s',q) \equiv \Pi\boxed{\overline{s'_1}} \parallel (\tau.\,1 + \mu.g(s'_2,q)) \tag{18}$$

$$\mu \in \overline{\mathcal{N}} \text{ implies } g(s',q) \equiv \Pi\boxed{\overline{s'_1}} \parallel \mu \parallel (\tau.\,1 + \mu.g(s'_2,q)) \tag{19}$$

The action μ is either an input or an output, and we organise the proof accorrdingly.

Suppose μ is an input. Since $\boxed{\overline{\nu}}$ is an output, the transition $g(s,q) \xrightarrow{\mu} o$ must be due to a transition $g(s',q) \xrightarrow{\mu} o'$, thus Equation (16) implies

$$o = \boxed{\overline{\nu}} \parallel o' \tag{20}$$

In case 1, then $s' \in \mathcal{N}^\star$ and $\nu \in \mathcal{N}$ ensure $s \in \mathcal{N}^\star$ and the equality $o \equiv \Pi\overline{s} \parallel q'$ follows from $o' = \Pi\overline{s'} \parallel q'$ and Equation (20).

In case 2, let $s_1 = \nu.s'_1$, $s_2 = s'_2$. Since ν is an input we have $s_1 \in \mathcal{N}^\star$. The equalities $s = \nu.s'$ and $s' = s'_1.\boxed{\mu}.s'_2$ imply that $s = s_1\boxed{\mu}s_2$. The required $o \equiv \Pi\boxed{\overline{s_1}} \parallel g(s'_2,q)$ follows from $o' \equiv \Pi\boxed{\overline{s'_1}} \parallel g(s'_2,q)$ and Equation (20).

Now we proceed as follows,

$$\begin{array}{rcll}
g(s,q) & = & \boxed{\overline{\nu}} \parallel g(s',q) & \text{By } \textit{Equation } (16) \\
 & \equiv & \boxed{\overline{\nu}} \parallel (\Pi\boxed{\overline{s'_1}} \parallel (\tau.\,1 + \mu.g(s'_2,q))) & \text{By } \textit{Equation } (18) \\
 & \equiv & (\boxed{\overline{\nu}} \parallel \Pi\boxed{\overline{s'_1}}) \parallel (\tau.\,1 + \mu.g(s'_2,q)) & \text{Associativity} \\
 & \equiv & \Pi\boxed{\overline{s_1}} \parallel (\tau.\,1 + \mu.g(s'_2,q)) & \text{Because } s_1 = \nu.s'_1 \\
 & \equiv & \Pi\boxed{\overline{s_1}} \parallel (\tau.\,1 + \mu.q(s_2,q)) & \text{Because } s_2 = s'_2
\end{array}$$

Now suppose that μ is an output. Then either $\overline{\nu} = \mu$ or $\overline{\nu} \neq \mu$.

In the first case we let $s_1 = \varepsilon$ and $s_2 = s'$. Equation (16) and $\overline{\nu} = \mu$ imply $g(s,q) = \mu \parallel g(s_2,q)$ from which we obtain the required $g(s,q) \equiv \Pi\boxed{\overline{s_1}} \parallel \mu \parallel g(s_2,q)$, and $o \equiv \Pi\boxed{\overline{s_1}} \parallel g(s_2,q)$.

If $\overline{\nu} \neq \mu$ then Equation (16) ensures that the transition $g(s,q) \xrightarrow{\mu} o$ must be due to $g(s',q) \xrightarrow{\mu} o'$ and Equation (20) holds. We use the inductive hypothesis.

If 1 is true, we proceed as already discussed. In case 2 holds, let $s_1 = \nu.s'_1$, we have that $s_1 \in \mathcal{N}^\star$. Let $s_2 = s'_2$, now we have that

$$\begin{array}{rcll} g(s,q) & = & \boxed{\overline{\nu}} \parallel g(s',q) & \text{By } \textit{Equation } (16) \\ & \equiv & \boxed{\overline{\nu}} \parallel (\Pi \boxed{\overline{s'_1}} \parallel \mu \parallel g(s'_2,q)) & \text{By } \textit{Equation } (19) \\ & \equiv & (\boxed{\overline{\nu}} \parallel \Pi \boxed{\overline{s'_1}}) \parallel \mu \parallel g(s'_2,q) & \text{Associativity} \\ & = & \Pi \boxed{\overline{s_1}} \parallel \mu \parallel g(s'_2,q) & \text{Because } s_1 = \nu.s'_1 \\ & = & \Pi \boxed{\overline{s_1}} \parallel \mu \parallel g(s_2,q) & \text{Because } s_2 = s'_2 \end{array}$$

◀

▶ **Lemma 100.** *For every $s \in \mathsf{Act}^\star$, if $g(s,q) \xrightarrow{\mu} p$ then either:*

(a) *there exists q' such that $q \xrightarrow{\mu} q'$, $s \in \mathcal{N}^\star$ with $p \equiv \Pi\overline{s} \parallel q'$, or*

(b) *$s = s_1.\boxed{\overline{\mu}}.s_2$ for some $s_1 \in \mathcal{N}^\star$ and $s_2 \in \mathsf{Act}^\star$ with $p \equiv g(s_1.s_2,q)$.*

Proof. The proof is by induction over the sequence s.

In the base case $s = \varepsilon$ and we have $g(\varepsilon,q) = q$. We show part (a) and choose $q' = p$. We have $g(\varepsilon,q) = q \xrightarrow{\mu} p$, $p \equiv \Pi\overline{\varepsilon} \parallel q'$ and $\varepsilon \in \mathcal{N}^\star$ as required.

In the inductive case $s = \nu.s'$. We proceed by case-analysis on ν. If ν is an input, then $g(\nu.s',q) = \overline{\nu} \parallel g(s',q)$. The hypothesis $g(\nu.s',q) \xrightarrow{\mu} p$ implies that either:

(i) $\overline{\nu} \xrightarrow{\mu} \mathsf{0}$ with $p = \mathsf{0} \parallel g(s',q)$ and $\overline{\nu} = \mu$, or

(ii) $g(s',q) \xrightarrow{\mu} \hat{p}$ with $p = \overline{\nu} \parallel \hat{p}$.

In the first case we show part (b). We choose $s_1 = \varepsilon$, $s_2 = s'$. We have $s = \mu.s' = \overline{\nu}.s'$, $p = \mathsf{0} \parallel g(s',q) \equiv g(\varepsilon.s',q)$ and $\varepsilon \in \mathcal{N}^\star$ as required.

In the second case the inductive the hypothesis tells us that either:

(H-a) there exists q' such that $q \xrightarrow{\mu} q'$, $s' \in \mathcal{N}^\star$ with $\hat{p} \equiv \Pi\overline{s'} \parallel q'$, or

(H-b) $s = s_1.\boxed{\overline{\mu}}.s_2$ for some $s_1 \in \mathcal{N}^\star$ and $s_2 \in \mathsf{Act}^\star$ with $\hat{p} \equiv g(s_1.s_2,q)$.

part (a) or part (b) is true.

If part (a) is true then $s' \in \mathcal{N}^\star$ and there exists q'' such that $q \xrightarrow{\mu} q''$ with $\hat{p} \equiv \Pi\overline{s'} \parallel q''$. We prove part (a). We choose $q' = q''$ and $s = \nu.s'$. We have $p \equiv \overline{\nu} \parallel \hat{p} \equiv \overline{\nu} \parallel \Pi\overline{s'} \parallel q'' \equiv \Pi\overline{\nu.s'} \parallel q''$ and $\nu.s' \in \mathcal{N}^\star$ as required.

If part (b) is true then $s' = s'_1.\boxed{\overline{\mu}}.s'_2$ for some $s'_1 \in \mathcal{N}^\star$ and $s'_2 \in \mathsf{Act}^\star$ with $\hat{p} \equiv g(s'_1.s'_2,q)$. We prove part (a). We choose $s_1 = \nu.s'_1$ and $s_2 = s'_2$. We have $p \equiv \overline{\nu} \parallel \hat{p} \equiv \overline{\nu} \parallel g(s'_1.s'_2,q) \equiv g(\nu.s'_1.s'_2,q)$ as required.

If ν is an output, then $g(\nu.s',q) = \overline{\nu}.(g(s',q)) + \tau.\mathsf{1}$. We prove part (b) and choose $s_1 = \varepsilon$, $s_2 = s'$. The hypothesis $g(\nu.s',q) \xrightarrow{\mu} p$ implies that $\mu = \overline{\nu}$ and $p \equiv g(s',q) \equiv g(\varepsilon.s',q)$ as required. ◀

▶ **Lemma 101.** *For every $s \in \mathsf{Act}^\star$, if $g(s,q) \xrightarrow{\tau} o$ then either*

(a) *GOOD(o), or*

(b) *$s \in \mathcal{N}^\star$, $q \xrightarrow{\tau} q'$, and $o = \Pi\overline{s} \parallel q'$, or*

(c) *$s \in \mathcal{N}^\star$, $q \xrightarrow{\nu} q'$, and $s = s_1.\nu.s_2$, and $o \equiv \Pi\overline{s_1.s_2} \parallel q'$, or*

(d) *$o \equiv g(s_1.s_2.s_3,q)$ where $s = s_1.\mu.s_2.\overline{\mu}.s_3$ with $s_1.\mu.s_2 \in \mathcal{N}^\star$.*

Proof. By induction on the structure of s.

In the base case $s = \varepsilon$. We prove b. Trivially $s \in \mathcal{N}^\star$, and by definition $g(\varepsilon,q) = q$, the hypothesis implies therefore that $q \xrightarrow{\tau} o$. The q' we are after is o itself, for $o \equiv \mathsf{0} \parallel o = \Pi\overline{s} \parallel o$.

In the inductive case $s = \nu.s'$. We proceed by case analysis on whether $\nu \in \overline{\mathcal{N}}$ or $\nu \in \mathcal{N}$.

If ν is an output, by definition $g(s,q) = \tau.1 + \overline{\nu}.gens'q$. Since $\overline{\nu}.g(s',q) \stackrel{\tau}{\not\longrightarrow}$, the silent move $g(s,q) \xrightarrow{\tau} o$ is due to rule [SUM-L], thus $o = 1 + \overline{\nu}.g(s',q)$, and thus $\textsc{good}(o)$. We have proven a.

Suppose now that ν is an input, by definition

$$g(s,q) = \overline{\nu} \parallel g(s',q) \tag{21}$$

The silent move $g(s,q) \xrightarrow{\tau} o$ must have been derived via the rule [COM], or the rule [PAR-R].

If [COM] was employed we know that

$$\dfrac{\overline{\nu} \xrightarrow{\overline{\nu}} 0 \qquad \dfrac{\vdots}{g(s',q) \xrightarrow{\nu} o'}}{\overline{\nu} \parallel g(s',q) \xrightarrow{\tau} 0 \parallel o'}$$

and thus $o \equiv o'$. Since ν is an input and $g(s',q) \xrightarrow{\nu} o'$, Lemma 99 ensures that either

(1) $q \xrightarrow{\nu} q'$, $s' \in \mathcal{N}^\star$, and $o' = \Pi\overline{s'} \parallel q'$, or

(2) $s' = s_1'.\overline{\nu}.s_2'$ for some $s_1' \in \mathcal{N}^\star$ and $s_2' \in \mathsf{Act}^\star$, and

$$o' \equiv \Pi\overline{s_1'} \parallel g(s_2',q) \tag{22}$$

$$g(s,q) \equiv \Pi\overline{s_1'} \parallel (\tau.1 + \nu.g(s_2',q)) \tag{23}$$

In case 1 we prove part (c). Since ν is an input, $s' \in \mathcal{N}^\star$ ensures that $s \in \mathcal{N}^\star$. By letting $s_1 = \varepsilon$ and $s_2 = s'$ we obtain $s = s_1.\nu.s_2$. We have to explain why $o \equiv \Pi\overline{s_1.s_2} \parallel q'$. This follows from the definitions of s_1 and s_2, from $o \equiv 0 \parallel o'$ and from $o' \equiv \Pi\overline{s'} \parallel q'$.

In case 2 we prove part (d). Let $s_1 = \varepsilon$, $s_2 = s_1'$ and $s_3 = s_2'$.

$$\begin{array}{lcll} s' & = & \nu s_1'.\overline{\nu}.s_2' & \text{By inductive hypothesis} \\ \nu.s' & = & \nu.s_1'.\overline{\nu}.s_2' & \\ s & = & \nu.s_1'.\overline{\nu}.s_2' & \text{Because } s = \nu.s' \\ s & = & s_1.\nu.s_2.\overline{\nu}.s_3 & \text{By definition} \end{array}$$

and $s_1.\nu.s_2 \in \mathcal{N}^\star$ as required. Moreover $o' = \Pi\overline{s_1'} \parallel g(s_2',q) = \Pi\overline{s_2} \parallel g(s_3,q) = g(s_2.s_3,q) = g(s_1.s_2.s_3,q)$ as required. This concludes the argument due to an applicaton of [COM].

If [PAR-R] was employed we know that

$$\dfrac{\dfrac{\vdots}{g(s',q) \xrightarrow{\tau} o'}}{\overline{\nu} \parallel g(s',q) \xrightarrow{\tau} \overline{\nu} \parallel o'}$$

thus $g(s',q) \xrightarrow{\tau} o'$ and

$$o = \overline{\nu} \parallel o' \tag{24}$$

Since s' is smaller than s, thanks to $g(s',q) \xrightarrow{\tau} o'$ we apply the inductive hypothesis to obtain either

(i) $\textsc{good}(o')$, or

(ii) $s' \in \mathcal{N}^\star$, $q \xrightarrow{\tau} q'$, and $o' = \Pi\overline{s'} \parallel q'$, or
(iii) $s' \in \mathcal{N}^\star$, $q \xrightarrow{\mu} q'$, and $s' = s'_1.\mu.s'_2$, or
(iv) $o' \equiv g(s'_1.s'_2.s'_3, q)$ where $s' = s'_1.\mu.s'_2.\overline{\mu}.s'_3$ with $s'_1.\mu.s'_2 \in \mathcal{N}^\star$,

If i then Equation (24) implies a. If ii then $s = \nu.s'$ and the assumption that ν is input imply that $s \in \mathcal{N}^\star$. Equation (24) and $o' = \Pi\overline{s'} \parallel q'$ imply that $o = \Pi\overline{s} \parallel q'$. We have proven b. If ii we prove b, because $s' \in \mathcal{N}^\star$ ensures $s \in \mathcal{N}^\star$ and $s' = s'_1.\mu.s'_2$ let use prove $s = s_1.\nu.s_2$ by letting $s_1 = \nu.s'_1$ and $s_2 = s'_2$.

If iv we prove d. We have $o' \equiv g(s'_1.s'_2.s'_3, q)$ and $s' = s'_1.\lambda.s'_2.\overline{\lambda}.s'_3$ with $s'_1.\lambda.s'_2 \in \mathcal{N}^\star$.
Let $s_1 = \nu.s'_1$, $s_2 = s'_2$ and $s_3 = s'_3$. Since $s'_1.\mu.s'_2 \in \mathcal{N}^\star$, we have $s_1.\mu.s_2 \in \mathcal{N}^\star$. We also have

$$\begin{array}{lcll} s' & = & s'_1.\mu.s'_2.\overline{\mu}.s'_3 & \text{By inductive hypothesis} \\ \nu.s' & = & \nu.s'_1.\mu.s'_2.\overline{\mu}.s'_3 & \\ s & = & \nu.s'_1.\mu.s'_2.\overline{\mu}.s'_3 & \text{Because } s = \nu.s' \\ s & = & s_1.\mu.s_2.\overline{\mu}.s_3 & \text{By definition} \end{array}$$

It remains to prove that $o \equiv g(s_1.s_2.s_3, q)$. This is a consequence of Equation (24), of $o' \equiv g(s'_1.s'_2.s'_3, q)$, and of the definitions of s_1, s_2, and s_3. ◀

▶ **Lemma 102.** *For every $s \in \mathsf{Act}^\star$, and process q such that or $q \xrightarrow{\tau} q'$ implies* $\textsc{good}(q')$, *and for every $\mu \in \mathcal{N}.q \xrightarrow{\mu} q'$ implies* $\textsc{good}(q')$, *if $g(s, q) = o_0 \xrightarrow{\tau} o_1 \xrightarrow{\tau} o_2 \xrightarrow{\tau} \ldots o_n \overset{\tau}{\not\longrightarrow}$ and $n > 0$ then* $\textsc{good}(o_i)$ *for some $i \in [1, n]$.*

Proof. Lemma 101 implies that one of the following is true,
(a) $\textsc{good}(o_1)$, or
(b) $s \in \mathcal{N}^\star$, $q \xrightarrow{\tau} q'$, and $o_1 = \Pi\overline{s} \parallel q'$, or
(c) $s \in \mathcal{N}^\star$, $q \xrightarrow{\mu} q'$, and $s = s_1.\mu.s_2$, and $o_1 \equiv \Pi\overline{s_1.s_2} \parallel q'$, or
(d) $o_1 \equiv g(s_1.s_2.s_3, q)$ where $s = s_1.\mu.s_2.\overline{\mu}.s_3$ with $s_1.\mu.s_2 \in \mathcal{N}$.
If a we are done. If b or c then $\textsc{good}(q')$, and thus $\textsc{good}(o_1)$.

In the base case $\mathsf{len}(s) = 0$, thus d is false. It follows that $\textsc{good}(o_1)$.

In the inductive case $s = \nu.s'$. We have to discuss only the case in which d is true. The inductive hypothesis ensures that

For every $s' \in \bigcup_{i=0}^{\mathsf{len}(s)-1}$, if $g(s', q) \xrightarrow{\tau} o'_1 \xrightarrow{\tau} o'_2 \xrightarrow{\tau} \ldots o'_m \overset{\tau}{\not\longrightarrow}$ and $m > 0$ then $\textsc{good}(o'_j)$ for some j.

Note that $o_1 \xrightarrow{\tau}$ so the reduction sequence $o_1 \Longrightarrow o_n$ cannot be empty, thus $m > 0$. This and $\mathsf{len}(s_1 s_2 s_3) < \mathsf{len}(s)$ let us apply the inductive hypothesis to state that

$$g(s_1.s_2.s_3, q) \xrightarrow{\tau} o'_1 \xrightarrow{\tau} o'_2 \xrightarrow{\tau} \ldots o'_m \overset{\tau}{\not\longrightarrow} \text{ implies } o'_j \text{ for some } j.$$

We conclude the argument via Lemma 87 and because $\equiv$ preserves success. ◀

▶ **Lemma 103.** *For every $s \in \mathsf{Act}^\star$ and process q, if $g(s, q) \overset{\tau}{\not\longrightarrow}$ then*
1. *$s \in \mathcal{N}^\star$,*
2. *$q \overset{\tau}{\not\longrightarrow}$,*
3. *$I(q) \cap \overline{s} = \emptyset$,*
4. *$R(g(s, q)) = \overline{s} \cup R(q)$.*

Proof. By induction on s. In the base case $\varepsilon \in \mathcal{N}^\star$, and $g(\varepsilon, q) = q$, thus $q \overset{\tau}{\not\longrightarrow}$. The last two points follow from this equality and from ε containing no actions.

In the inductive case $s = \mu.s'$. The hypothesis $g(\mu.s', q) \overset{\tau}{\not\longrightarrow}$ and the definition of g imply that $g(\mu.s', q) = \overline{\mu} \parallel g(s', q)$, thus $\mu \in \mathcal{N}$. The inductive hypothesis ensures that

1. $s' \in \mathcal{N}^\star$,
2. $q \stackrel{\tau}{\not\longrightarrow}$,
3. for every $I(q) \cap \overline{s'} = \emptyset$,
4. for every $R(g(s', q)) = \overline{s'} \cup R(q)$

Since $\overline{\mu} \parallel g(s', q) \stackrel{\tau}{\not\longrightarrow}$ rule [Com] cannot be applied, thus $q \stackrel{\mu}{\not\longrightarrow}$, and so $I(q) \cap \overline{s} = \emptyset$. From $R(g(s', q)) = \overline{s'} \cup R(q)$ we obtain $R(g(s, q)) = \overline{s} \cup R(q)$. ◀

▶ **Lemma 104.** *For every $\mu \in \mathsf{Act}$, s and p, $g(\mu.s, p) \xrightarrow{\overline{\mu}} g(s, p)$.*

Proof. We proceed by case-analysis on μ. If μ is an input then $g(\mu.s, p) = \overline{\mu} \parallel g(s, p)$. We have $\boxed{\overline{\mu}} \parallel g(s, p) \xrightarrow{\overline{\mu}} 0 \parallel g(s, p) \equiv g(s, p)$ as required. If μ is an output then $g(\mu.s, p) = \overline{\mu}.g(s, p) + \tau.1$. We have $\overline{\mu}.g(s, p) + \tau.1 \xrightarrow{\overline{\mu}} g(s, p)$ as required. ◀

▶ **Lemma 105.** *For every $s \in \mathsf{Act}^\star, q \in \mathit{ACCS}.\, c(s) \xrightarrow{\tau}_{\mathsf{fw}} q$ either*

(a) *$\textsc{good}(q)$, or*

(b) *there exist b, s_1, s_2 and s_3 with $s_1.b.s_2 \in \mathcal{N}^\star$ such that $s = s_1.b.s_2.\overline{b}.s_3$ and $q \equiv c(s_1.s_2.s_3)$.*

Proof. The proof is by induction on s.

In the base case $s = \varepsilon$, $c(\varepsilon) = \tau.1$ and then $q = 1$. We prove a with $\textsc{good}(1)$.

In the inductive case $s = \mu.s'$. We proceed by case-analysis over μ.

If μ is an input then $c(\mu.s') = \overline{\mu} \parallel c(s')$. We continue by case-analysis over the reduction $\overline{\mu} \parallel c(s') \xrightarrow{\tau} q$. It is either due to:

(i) a communication between $\overline{\mu}$ and $c(s')$ such that $\overline{\mu} \xrightarrow{\overline{\mu}} 0$ and $c(s') \xrightarrow{\mu} q'$ with $q = 0 \parallel q'$, or

(ii) a reduction of $c(s')$ such that $c(s') \xrightarrow{\tau} q'$ with $q = \overline{\mu} \parallel q'$.

If i is true then Lemma 100 tells us that there exist s'_1 and s'_2 such that $s' = s'_1.\overline{\mu}.s'_2$ and $q' \equiv c(s'_1.s'_2)$ with $s'_1 \in \mathcal{N}^*$. We prove (b). We choose $b = \mu$, $s_1 = \varepsilon$, $s_2 = s'_1$, $s_3 = s'_2$. We show the first requirement by $s = \mu.s' = \mu.s'_1.\overline{\mu}.s'_2 = \varepsilon.\mu.s'_1.\overline{\mu}.s'_2 = s_1.b.s_2.\overline{b}.s_3$. The second requirement is $q = 0 \parallel q' \equiv c(s'_1.s'_2) = c(\varepsilon.s'_1.s'_2) = c(s_1.s_2.s_3)$.

We now consider the case (ii). The inductive hypothesis tells us that either:

1. $\textsc{good}(q')$, or
2. there exist ι, s'_1, s'_2 and s'_3 with $s'_1.\iota.s'_2 \in \mathcal{N}^*$ such that $s' = s'_1.\iota.s'_2.\overline{\iota}.s'_3$ and $q' \equiv c(s'_1.s'_2.s'_3)$.

If (1) is true then we prove a with $q = \boxed{\overline{\mu}} \parallel q'$ and $\boxed{\overline{\mu}} \parallel q'$ and $\textsc{good}(q')$. If (2) is true then we prove (b). We choose $b = \iota$, $s_1 = \mu.s'_1$, $s_2 = s'_2$, $s_3 = s'_3$. We show the first requirement with $s = \mu.s' = \mu.s'_1.\iota.s'_2.\overline{\iota}.s'_3 = s_1.b.s_2.\overline{b}.s_3$. The second requirement is $q = \boxed{\overline{\mu}} \parallel q' \equiv \boxed{\overline{\mu}} \parallel c(s'_1.s'_2.s'_3) = c(\mu.s'_1.s'_2.s'_3) = c(s_1.s_2.s_3)$.

If μ is an output then $c(\mu.s') = \overline{\mu}.(cs') + \tau.1$. The hypothesis $c(\mu.s') \xrightarrow{\tau} q$ implies $q = 1$. We prove (a) with $\textsc{good}(1)$. ◀

G Further related works

Contextual preorders in functional languages. Morris preorder is actively studied in the pure λ-calculus [30, 31, 67, 11], λ-calculus with references [81, 62], in PCF [69] as well as in languages supporting shared memory concurrency [96], and mutable references [47]. The more sophisticated the languages, the more intricate and larger the proofs. The need for mechanisation became thus apparent, in particular to prove that complex logical relations defined in the framework Iris (implemented in Coq) are sound, *i.e.* included in the preorder

[70, 52]. [7] provide a framework to study contextual equivalences in the setting of process calculi. It is worth noting, though, that as argued by [26] (in Section 3 of that paper), Morris equivalence coincides with MAY-equivalence, at least if the operational semantics at hand enjoys the Church-Rosser property. In fact [27] define Morris preorder literally as a testing one, via tests for convergence. The studies of the MUST-preorder in process calculi can thus be seen as providing proof methods to adapt Morris equivalence to nondeterministic settings, and using contexts that are really external observers. To sum up, one may say that Morris equivalence coincides with MAY-equivalence when nondeterminism is confluent and all states are viewed as accepting states, while it coincides with MUST equivalence in the presence of true nondeterminism and when only successful states are viewed as accepting states.

In the setting of nondeterministic and possibly concurrent applicative programming languages [91, 90, 19], also a contextual preorder based on may and must-termination has been studied [86, 19]. Our preorder $\preccurlyeq_{\mathsf{cnv}}$ is essentially a generalisation of the must-termination preorder of [86] to traces of visible actions.

Models of asynchrony. While synchronous (binary) communication requires the simultaneous occurrence of a send and a receive action, asynchronous communication allows a delay between a send action and the corresponding receive action. Different models of asynchrony exist, depending on which medium is assumed for storing messages in transit. In this paper, following the early work on the asynchronous π-calculus [64, 25, 4], we assume the medium to be an unbounded unordered mailbox, shared by all processes. Thus, no process needs to wait to send a message, namely the send action is non-blocking. This model of communication is best captured via the output-buffered agents with feedback of [92]. The early style LTS of the asynchronous π-calculus is a concrete example of this kind of LTSs. A similar global unordered mailbox is used also in Chapter 5 of [95], by [33], which relies explicitly on a mutable global state, and by [79], which manipulates it via two functions *get* and *set*.

More deterministic models of asynchrony are obtained assigning a data structure to every channel. For example [65, 66] use an even more deterministic model in which each ordered pair of processes is assigned a dedicated channel, equipped with an *ordered* queue. Hence, messages along such channels are received in the same order in which they were sent. This model is used for asynchronous session calculi, and mimics the communication mode of the TCP/IP protocol. The obvious research question here is how to adapt our results to the different communication mechanisms and different classes of LTSs. For instance, both [94] and [35] define LTSs for ERLANG. We will study whether at least one of these LTSs is an instance of output-buffered agents with feedback. If this is not the case, we will first try to adapt our results to ERLANG LTSs.

Mutable state. Prebet [81] has recently shown an encoding of the asynchronous π-calculus into a λ-calculus with references, which captures Morris equivalence via a bisimulation. This renders vividly the intuition that output-buffered agents manipulate a shared common state. We therefore see our work also as an analysis of the MUST-preorder for a language in which programs manipulate a global mutable store. Since the store is what contains output messages, and our formal development shows that only outputs are observable, our results suggest that characterisations of testing preorders for impure programming languages should predicate over the content of the mutable store, *i.e.* the values written by programs. Another account of π-calculus synchronisation via a functional programming language is provided in [90], that explains how to use Haskell M-VARs to implement π-calculus message passing.

Theories for synchronous semantics. Both [72] and [37] employed LTSs as a model of contracts for web-services (*i.e.* `WSCL`), and the MUST-preorder as refinement for contracts.

The idea is that a search engine asked to look for a service described by a contract p_1 can actually return a service that has a contract p_2, provided that $p_1 \sqsubsetsim_{\text{MUST}} p_2$.

The MUST-preorder for *clients* proposed by [14] has partly informed the theory of monitors by [1], in particular the study of preorders for monitors by [50]. Our results concern LTSs that are more general than those of monitors, and thus our code could provide the basis to mechanise the results of [1].

The first subtyping relation for binary session types was presented in [53] using a syntax-oriented definition. The semantic model of that subtyping is a refinement very similar to the MUST-preorder. The idea is to treat types as CCS terms, assign them an LTS [36, 10, 84, 17], and use the resulting testing preorders as semantic models of the subtyping. In the setting of coinductively defined higher-order session types, the correspondence is implicitly addressed in [36]. In the setting of recursive higher-order session types, it is given by Theorem 4.10 of [15].

We would like to mechanise in our framework these results, in particular the ones about asynchronous semantics, and contrast and compare the various testing preorders used in the literature. More in general, given the practical relevance of asynchronous communication, it seems crucial not only to adapt the large body of theory outlined above to the asynchronous setting but also to resort to machine supported reasoning to do it. This paper is meant to be a step forward in this direction.

Must-preorder and asynchrony. The first investigation on the MUST-preorder in an asynchronous setting was put forth by [39]. While their very clear examples shed light on the preorder, their alternative preorder (Definition 6 in that paper) is more complicated than necessary: it uses the standard LTS of ACCS, the LTS of forwarders, a somewhat ad-hoc predicate $\overset{I}{\rightsquigarrow}$, and a condition on multisets of inputs, that we do not use. Moreover that preorder is not complete because of a glitch in the treatment of divergence. The details of the counter-example we found to that completeness result are in Appendix H.

In [57] Hennessy outlines how to adapt the approach of [39] to a typed asynchronous π-calculus. While the LTS of forwarders is replaced by a Context LTS, the predicates to define the alternative preorder are essentially the same used in the preceding work with Castellani. Acceptance sets are given in Definition 3.19 there, and the predicate $\rightsquigarrow$ is denoted $\searrow$, while the generalised acceptance sets of [39] are given in Definition 3.20. Owing to the glitch in the completeness of [39], it is not clear that Theorem 3.28 of [57] is correct either.

Also the authors of [24] the MUST-preorder in ACCS. There is a major difference between their approach and ours. When studying theories for asynchronous programs, one can either

(1) keep the definitions used for synchronous programs, and enhance the LTS with forwarders; or

(2) adapt the definitions, and keep the standard LTS.

In the first case, the complexity is moved into the LTS, which becomes infinite-branching and infinite-state. In the second case, the complexity is moved into the definitions used to reason on the LTS (i.e. in the meta-language), and in particular in the definition of the alternative preorder, which deviates from the standard one. The authors of [24] follow the second approach. This essentially explains why they employ the standard LTS of CCS and to tackle asynchrony they reason on traces via

(i) a preorder $\preceq$ (Table 2 of that paper) that defines on *input* actions the phenomena due to asynchrony, for instance their *annihilation* rule (i.e. TO3) is analogous to the FEEDBACK axiom, and their *postponement* (i.e. TO2) is analogous to the OUTPUT-COMMUTATIVITY axiom; and

(ii) a rather technical operation on traces, namely $s \ominus s' = (\{\!|s|\!\}_i \setminus \{\!|s'|\!\}_i) \setminus \overline{(\{\!|s|\!\}_o \setminus \{\!|s'|\!\}_o)}$.

We favour instead the first approach, for, as we already argued, it helps us achieve a modular mechanisation.

The authors of [46] give yet another account of the MUST-preorder. Even though non-blocking outputs can be written in their calculus, they use a left-merge operator that allows writing *blocking* outputs. The contexts that they use to prove the completeness of their alternative preorder use such blocking outputs, consequently their arguments need not tackle the asymmetric treatment of input and output actions. This explains why they can use smoothly a standard LTS, while [39] and [24] have to resort to more complicated structures.

Theorem 5.3 of the PhD thesis by [95] states an alternative characterisation of the MUST-preorder, but it is given with no proof. The alternative preorder given in Definition 5.8 of that thesis turns out to be a mix of the ones by [39] and [24]. In particular, the definition of the alternative preorder relies on the LTS of forwarders, there denoted $\longrightarrow_A$ (Point 1. in Definition 5.1 defines exactly the input transitions that forward messages into the global buffer). The condition that compares convergence of processes is the same as in [39], while server actions are compared using MUST-sets, and not acceptance sets. In fact, Definition 5.7 there is titled "acceptance sets" but it actually defines MUST-sets.

May-preorder. MAY testing and the MAY-preorder, have been widely studied in asynchronous settings. The first characterisation for ACCS appeared in [39] and relies on comparing traces and asynchronous traces of servers. Shortly after [24] presented a characterisation based on operation on traces. A third characterisation appeared in [8], where the saturated LTS $\longrightarrow_s$ is essentially out $\longrightarrow_{\mathsf{fw}}$. That characterisation supports our claim that results about synchronous semantics are true also for asynchronous ones, modulo forwarding. Compositionality of trace inclusion, i.e. the alternative characterisation of the MAY-preorder, has been partly investigated in Coq by [6] in the setting of IO-automata. The MAY-preorder has also been studied in the setting of actor languages by [35, 94].

Fairness. Van Glabbeek [97] argues that by amending the semantics of parallel composition (i.e. the scheduler) different notions of fairness can be embedded in the MUST-preorder. We would like to investigate which notion of fairness makes the MUST-preorder coincide with the FAIR-preorder of [85].

Bar-induction. A mainstay in the literature on the MUST-preorder is Kőnig's lemma, see for example Theorem 2.3.3 in [40], and Theorem 1 in [16]. [48], though, explains in detail why Kőnig's lemma is not constructive. Instead, we use in this paper the constructive bar-induction principle, whose fundamental use is to prove that if every path in a tree T is finite, then T is well-founded, as discussed by [77, 29] and [68]. Unfortunately, while it is a constructive principle, mainstream proof assistants do not support it, which is why we had to postulate it as a proof principle that we proved using the Excluded Middle axiom. One consequence of using an axiom is that they do not have computational content. Developing a type theory with a principle of bar-induction is the subject of recent and ongoing works [51, 83].

H Counter-example to existing completeness result

In this section we recall the definition of the alternative preorder $\ll_{\mathsf{ch}}$ by [39], and show that it is not complete with respect to $\sqsubseteq_{\text{MUST}}$, *i.e.* $\sqsubseteq_{\text{MUST}} \not\subseteq \ll_{\mathsf{ch}}$. We start with some auxiliary definitions.

The predicate $\overset{I}{\rightsquigarrow}$ is defined by the following two rules:

- $p \overset{I}{\rightsquigarrow} p$ if $p \overset{\tau}{\not\longrightarrow}$ and $I(p) \cap I = \emptyset$,
- $p \overset{I \uplus \{\!|a|\!\}}{\rightsquigarrow} p''$ if $p \overset{a}{\Longrightarrow} p'$ and $p' \overset{I}{\rightsquigarrow} p'$

The *generalised acceptance set* of a process p after a trace s with respect to a multiset of input actions I is defined by

$$\mathcal{GA}(p, s, I) = \{O(p'') \mid p \overset{s}{\Longrightarrow}_{\mathsf{fw}} p' \overset{I}{\rightsquigarrow} p''\}$$

The set of *input multisets* of a process p after a trace s is defined by

$$IM(p, s) = \{\{\!|a_1, \ldots, a_n|\!\} \mid a_i \in \mathcal{N}, p \overset{s}{\Longrightarrow}_{\mathsf{fw}} \overset{a_1}{\Longrightarrow} \ldots \overset{a_n}{\Longrightarrow}\}$$

The convergence predicate over traces performed by forwarders is denoted $\Downarrow_a$, and defined as $\Downarrow$, but over the LTS given in Example 11.

The preorder $\ll_{\mathsf{ch}}$ is now defined as follows:

▶ **Definition 106** (Alternative preorder $\ll_{\mathsf{ch}}$ [39])**.** *Let $p \ll_{\mathsf{ch}} q$ if for every $s \in \mathsf{Act}^{\star}$. $p \Downarrow_a s$ implies*

1. *$q \Downarrow_a s$,*
2. *for every $R \in \mathcal{A}(q, s)$ and every $I \in IM(p, s)$ such that $I \cap R = \emptyset$ there exists some $O \in \mathcal{GA}(p, s, I)$ such that $O \setminus \overline{I} \subseteq R$.* ■

We illustrate the three auxiliary definitions using the process $\mathit{Pierre} = b.(\tau.\Omega + c.\overline{d})$ introduced in Example 8. We may infer that

$$\mathit{Pierre} \overset{\{\!|b,c|\!\}}{\rightsquigarrow} \boxed{\overline{d}} \tag{25}$$

thanks to the following derivation tree

$$\dfrac{\dfrac{\dfrac{}{\boxed{\overline{d}} \overset{\emptyset}{\rightsquigarrow} \boxed{\overline{d}}}\ \boxed{\overline{d}} \overset{\tau}{\not\longrightarrow} \text{ and } I(\boxed{\overline{d}}) \cap \emptyset = \emptyset \qquad \tau.\Omega + c.\overline{d} \overset{c}{\Longrightarrow} \boxed{\overline{d}}}{\tau.\Omega + c.\overline{d} \overset{\{\!|c|\!\}}{\rightsquigarrow} \boxed{\overline{d}}} \qquad \mathit{Pierre} \overset{b}{\Longrightarrow} \tau.\Omega + c.\overline{d}}{\mathit{Pierre} \overset{\{\!|b,c|\!\}}{\rightsquigarrow} \boxed{\overline{d}}}$$

Let us now consider the generalised acceptance set of *Pierre* after the trace ε with respect to the multiset $\{\!|b, c|\!\}$. We prove that

$$\mathcal{GA}(\mathit{Pierre}, \varepsilon, \{\!|b, c|\!\}) = \{\{\overline{d}\}\} \tag{26}$$

By definition $\mathcal{GA}(\mathit{Pierre}, \varepsilon, \{\!|b, c|\!\}) = \{O(p'') \mid \mathit{Pierre} \overset{\varepsilon}{\Longrightarrow}_{\mathsf{fw}} p' \overset{\{\!|b,c|\!\}}{\rightsquigarrow} p''\}$. Since $\mathit{Pierre} \overset{\tau}{\not\longrightarrow}$, we have

$$\mathcal{GA}(\mathit{Pierre}, \varepsilon, \{\!|b, c|\!\}) = \{O(p'') \mid \mathit{Pierre} \overset{\{\!|b,c|\!\}}{\rightsquigarrow} p''\} \tag{27}$$

Then, thanks to Equation (25) we get $O(\overline{d}) = \{\overline{d}\} \in \mathcal{GA}(\mathit{Pierre}, \varepsilon, \{\!|b, c|\!\})$. We show now that $\{\overline{d}\}$ is the only element of this acceptance set. By (27) above, it is enough to show that $\mathit{Pierre} \overset{\{\!|b,c|\!\}}{\rightsquigarrow} p''$ implies $p'' = \boxed{\overline{d}}$. Observe that

1. $I(\mathit{Pierre}) \cap \{\!|b, c|\!\} \neq \emptyset$,
2. $\mathit{Pierre} \overset{a}{\Longrightarrow} p'$ implies $a = b$, and
3. There are two different states p' such that $\mathit{Pierre} \overset{b}{\Longrightarrow} p'$, but the only one that can do the input c is $p' = \tau.\Omega + c.\overline{d}$.

This implies that the only way to infer $\mathit{Pierre} \overset{\{\!|b,c|\!\}}{\rightsquigarrow} p''$ is via the derivation tree that proves Equation (25) above. Thus $p'' = \overline{d}$.

▶ **Counterexample 107.** *The alternative preorder $\ll_{\mathsf{ch}}$ is not complete for $\sqsubseteq_{\text{MUST}}$, namely $p \sqsubseteq_{\text{MUST}} q$ does not imply $p \ll_{\mathsf{ch}} q$.*

Proof. The cornestone of the proof is the process $Pierre = b.(\tau.\Omega + c.\overline{d})$ discussed above. In Example 8 we have shown that $Pierre \sqsubseteq_{\text{MUST}} 0$. Here we show that $Pierre \not\ll_{\mathsf{ch}} 0$, because the pair $(Pierre, 0)$ does not satisfy Condition 2 of Definition (106).

Since $Pierre \xrightarrow{\tau}\!\!\!\!\not\,\,$, we know that $Pierre \downarrow$, and thus by definition $Pierre \Downarrow_a \varepsilon$. We also have by definition $\mathcal{A}(0, \varepsilon) = \{\emptyset\}$, and $IM(Pierre, \varepsilon) = \{\emptyset, \{\!|b|\!\}, \{\!|b, c|\!\}\}$.

Let us check Condition 2 of Definition (106) for $p = Pierre$ and $q = 0$. Since there is a unique $R \in \mathcal{A}(0, \varepsilon)$, which is $\emptyset$, and $I \cap \emptyset = \emptyset$ for any I, we only have to check that for every $I \in IM(Pierre, \varepsilon)$ there exists some $O \in \mathcal{GA}(Pierre, \varepsilon, I)$ such that $O \setminus \overline{I} \subseteq \emptyset$.

Let $I = \{\!|b, c|\!\}$. By Equation (26) it must be $O = \{\overline{d}\}$. Since $\{\overline{d}\} \setminus \overline{I} = \{\overline{d}\} \setminus \overline{\{\!|b, c|\!\}} = \{\overline{d}\} \not\subseteq \emptyset$, the condition is not satisfied. Thus $Pierre \not\ll_{\mathsf{ch}} 0$. ◀

I Highlights of the Coq mechanisation

I.1 Preliminaries

We begin this section recalling the definition of MUST, which is given in Definition (2). It is noteworthy that the mechanised definition, i.e. `must_extensional`, depends on the typeclass `Sts` (Figure I.1.1), and *not* the type class `Lts`. This lays bare what stated in Section 1: to define MUST a reduction semantics (i.e. a state transition system), and a predicate GOOD over clients suffice.

I.1.1 State Transition Systems

The typeclass for state transition systems (Sts) is defined as follows, where `A` is the set of states of the Sts. It included a notion of stability which is axiomatized and decidable.

```
Class Sts (A: Type) := {
    sts_step: A → A → Prop;
    sts_state_eqdec: EqDecision A;
    sts_step_decidable: RelDecision sts_step;

    sts_stable: A → Prop;
    sts_stable_decidable p : Decision (sts_stable p);
    sts_stable_spec1 p : ¬ sts_stable p -> { q | sts_step p q };
    sts_stable_spec2 p : { q | sts_step p q } → ¬ sts_stable p;
  }.
```

I.1.2 Maximal computations

A computation is maximal if it is infinite or if its last state is stable. Given a state `s`, the type `max_exec_from s` contains all the maximal traces that start from `s`. Note the use of a coinductive type to allow for infinite executions.

```
Context `{Sts A}.

CoInductive max_exec_from: A -> Type :=
```

```
  | MExStop s (Hstable: sts_stable s) : max_exec_from s
  | MExStep s s' (Hstep: sts_step s s') (η: max_exec_from s') :
    max_exec_from s.
```

I.2 The must-preorder

I.2.1 Client satisfaction

The predicate GOOD is defined as any predicate over the states of an LTS that satisfies certain properties: it is preserved by structural congruence, by outputs in both directions (if $p \xrightarrow{\overline{a}} p'$ then $\textsc{good}(p) \Leftrightarrow \textsc{good}(p')$).

It is defined as a typeclass indexed over the type of states and labels, because we expect a practitioner to reason on a single canonical notion of "good" at a time.

```
Class Good (A L : Type) `{Lts A L, ! LtsEq A L} := {
    good : A -> Prop;
    good_preserved_by_eq p q : good p -> p ≡ q -> good q;
    good_preserved_by_lts_output p q a :
      p ⟶[ActOut a] q -> good p -> good q;
    good_preserved_by_lts_output_converse p q a :
      p ⟶[ActOut a] q -> good q -> good p
}.
```

I.2.2 Must testing

Definition (2): We write $p \,\textsc{must}\, r$ if every maximal computation of $p \parallel r$ is successful.

Given an integer `n` and a maximal execution η, the function `mex_take_from n` applied to η returns `None` if η is shorter than `n` and `Some p`, where `p` is a finite execution corresponding to the first `n` steps of η.

Then, we define the extensional version of $p \,\textsc{must}\, e$ by stating that, for all maximal executions η starting from (p, e), there exists an integer n such that the n-th element of η is good. The nth element is obtained by taking the last element of the finite prefix of length n computed using the function above.

```
Context `{good : B -> Prop}.

Fixpoint mex_take_from (n: nat) {x} (η: max_exec_from x) :
  option (finexec_from x) :=
 match n with
  | 0 => Some $ FExSingl x
  | S n => match η with
           | MExStop x Hstable => None
           | MExStep x x' Hstep η' =>
               let p' := mex_take_from n η' in
               (λ p', FExStep x x' (bool_decide_pack _ Hstep) p') <$> p'
          end
 end.
```

```
Definition must_extensional (p : A) (e : B) : Prop :=
  forall η : max_exec_from (p, e), exists n fex,
    mex_take_from n η = Some fex /\ good (fex_from_last fex).2.
```

I.2.3 The preorder

Definition 3 is mechanised in a straightforward way:

```
Definition pre_extensional (p : A) (q : R) : Prop :=
   forall (r : B), must_extensional p r -> must_extensional q r.

Notation "p ⊑_e q" := (pre_extensional p q).
```

I.3 Behavioural characterizations

I.3.1 Labeled Transition Systems

An LTS is a typeclass indexed by the type of states and the type of labels. The type of labels must be equipped with decidable equality and be countable, as enforced by the `Label` typeclass. An action `a : Act L` is either an internal action τ or an external action: an input or an output of a label in `L`.

```
Class Label (L: Type) := {
  label_eqdec: EqDecision L;
  label_countable: Countable L;
}.

Inductive Act (A: Type) := ActExt (μ: ExtAct A) | τ.

Class Lts (A L : Type) `{Label L} := {
    lts_step: A → Act L → A → Prop;
    lts_state_eqdec: EqDecision A;

    lts_step_decidable a α b : Decision (lts_step a α b);

    lts_outputs : A -> gset L;
    lts_outputs_spec1 p1 x p2 :
      lts_step p1 (ActExt (ActOut x)) p2 -> x ∈ lts_outputs p1;
    lts_outputs_spec2 p1 x :
      x ∈ lts_outputs p1 -> {p2 | lts_step p1 (ActExt (ActOut x)) p2};

    lts_stable: A → Act L → Prop;
    lts_stable_decidable p α : Decision (lts_stable p α);
    lts_stable_spec1 p α : ¬ lts_stable p α → { q | lts_step p α q };
    lts_stable_spec2 p α : { q | lts_step p α q } → ¬ lts_stable p α;
  }.

Notation "p ⟶ q"      := (lts_step p τ q).
```

```
Notation "p ⟶{ α } q" := (lts_step p α q).
Notation "p ⟶[ α ] q" := (lts_step p (ActExt μ) q).
```

An LTS L is cast into an STS by taking only the τ-transitions, as formalised by the following instance, which says that `A` can be equipped with an STS structure when, together with some labels `L`, `A` is equipped with a LTS structure.

```
Program Instance sts_of_lts `{Label L} (M: Lts A L): Sts A :=
  {|
    sts_step p q := sts_step p τ q;
    sts_stable s := lts_stable s τ;
  |}.
```

I.3.2 Weak transitions

Let $\Longrightarrow \subseteq A \times \mathsf{Act}^\star \times A$ denote the least relation such that:

[wt-refl] $p \stackrel{\varepsilon}{\Longrightarrow} p'$,

[wt-tau] $p \stackrel{s}{\Longrightarrow} q$ if $p \stackrel{\tau}{\longrightarrow} p'$, and $p' \stackrel{s}{\Longrightarrow} q$

[wt-mu] $p \stackrel{\mu.s}{\Longrightarrow} q$ if $p \stackrel{\mu}{\longrightarrow} p'$ and $p' \stackrel{s}{\Longrightarrow} q$.

```
Definition trace L := list (ExtAct L).

Inductive wt : A -> trace L -> A -> Prop :=
| wt_nil p : wt p [] p
| wt_tau s p q t (l : p ⟶ q) (w : wt q s t) : wt p s t
| wt_act μ s p q t (l : p ⟶[μ] q) (w : wt q s t) : wt p (μ :: s) t.

Notation "p ⟹[s] q" := (wt p s q).
```

I.3.3 Product of LTS

The characteristic function of the transition relation of the LTS resulting from the parallel composition of two LTS. States of the parallel product of L_1 and L_2 are pairs $(a, b) \in L_1 \times L_2$. The first two cases correspond to unsynchronized steps from either LTS, and the third case corresponds to the LTS taking steps with dual actions. The predicate `act_match l1 l2` states that the two actions are visible and are dual of each other.

```
Inductive parallel_step `{M1: Lts A L, M2: Lts B L} :
  A * B → Act L → A * B → Prop :=
| ParLeft l a1 a2 b: a1 -[l]→ a2 → parallel_step (a1, b) l (a2, b)
| ParRight l a b1 b2: b1 -[l]→ b2 → parallel_step (a, b1) l (a, b2)
| ParSync l1 l2 a1 a2 b1 b2:
  act_match l1 l2 → a1 -[l1]→ a2 → b1 -[l2]→ b2 →
  parallel_step (a1, b1) τ (a2, b2)
.
```

I.4 Typeclasses for LTS

The Selinger axioms for LTSs are represented as three typeclasses in our Coq development.

$$\begin{tikzcd}[column sep=small] & \texttt{Lts} \arrow[d] & \\ & \texttt{LtsEq} \arrow[d] & \\ & \texttt{LtsOba} \arrow[dl] \arrow[dr] & \\ \texttt{LtsObaFb} & & \texttt{LtsObaFw} \end{tikzcd}$$

Figure 18 Typeclasses to formalise LTSs.

```
Class LtsOba (A L : Type) `{Lts A L, !LtsEq A L} :=
    MkOBA {
      lts_oba_output_commutativity {p q r a α} :
        p ⟶[ActOut a] q → q ⟶{α} r →
        ∃ t, p ⟶{α} t ∧ t ⟶≡[ActOut a] r ;
      lts_oba_output_confluence {p q1 q2 a μ} :
        μ ≠ ActOut a → p ⟶[ActOut a] q1 → p ⟶[μ] q2 →
        ∃ r, q1 ⟶[μ] r ∧ q2 ⟶≡[ActOut a] r ;
      lts_oba_output_tau {p q1 q2 a} :
        p ⟶[ActOut a] q1 → p ⟶ q2 →
        (∃ t, q1 ⟶ t ∧ q2 ⟶≡[ActOut a] t) ∨ q1 ⟶≡[ActIn a] q2 ;
      lts_oba_output_deter {p1 p2 p3 a} :
        p1 ⟶[ActOut a] p2 → p1 ⟶[ActOut a] p3 → p2 ≡ p3 ;
      lts_oba_output_deter_inv {p1 p2 q1 q2} a :
        p1 ⟶[ActOut a] q1 → p2 ⟶[ActOut a] q2 → q1 ≡ q2 → p1 ≡ p2;
      (* Multiset of outputs *)
      lts_oba_mo p : gmultiset L;
      lts_oba_mo_spec1 p a : a ∈ lts_oba_mo p <-> a ∈ lts_outputs p;
      lts_oba_mo_spec2 p a q :
        p ⟶[ActOut a] q -> lts_oba_mo p = {[+ a +]} ⊎ lts_oba_mo q;
    }.

Class LtsObaFB (A L: Type) `{LtsOba A L} :=
  MkLtsObaFB {
      lts_oba_fb_feedback {p1 p2 p3 a} :
        p1 ⟶[ActOut a] p2 → p2 ⟶[ActIn a] p3 → p1 ⟶≡ p3
    }.

Class LtsObaFW (A L : Type) `{LtsOba A L} :=
  MkLtsObaFW {
      lts_oba_fw_forward p1 a :
        ∃ p2, p1 ⟶[ActIn a] p2 ∧ p2 ⟶≡[ActOut a] p1;
      lts_oba_fw_feedback {p1 p2 p3 a} :
        p1 ⟶[ActOut a] p2 → p2 ⟶[ActIn a] p3 → p1 ⟶≡ p3 ∨ p1 ≡ p3;
    }.
```

I.4.1 Termination

We write $p \downarrow$ and say that p *converges* if every sequence of τ-transitions performed by p is finite. This is expressed extensionally by the property that all maximal computations starting from p contain a *stable* process, meaning that it is finite.

```
Definition terminate (p : A) : Prop :=
    forall η : max_exec_from p, exists n fex,
      mex_take_from n η = Some fex /\ lts_stable (fex_from_last fex) τ.
```

I.4.2 Convergence along a trace

To define the behavioural characterisation of the preorder, we first define $\Downarrow \subseteq A \times \mathsf{Act}^\star$ as the least relation such that,

[cnv-epsilon] $p \Downarrow \varepsilon$ if $p \downarrow$,

[cnv-mu] $p \Downarrow \mu.s$ if $p \downarrow$ and for each p', $p \stackrel{\mu}{\Longrightarrow} p'$ implies $p' \Downarrow s$.

This corresponds to the following inductive predicate in Coq:

```
Inductive cnv : A -> trace L -> Prop :=
  | cnv_ext_nil p : terminate p -> cnv p []
  | cnv_ext_act p μ s :
    terminate p -> (forall q, p ⟹{μ} q -> cnv q s) -> cnv p (μ :: s).

Notation "p ⇓ s" := (cnv p s).
```

I.5 Forwarders

We define a mailbox MO as a multiset of names.

```
Definition mb (L : Type) `{Label L} := gmultiset L.
```

Definition (10) and Figure 6. Lifting of a transition relation to transitions of forwarders.

```
Inductive lts_fw_step {A L : Type} `{Lts A L} :
  A * mb L -> Act L -> A * mb L -> Prop :=
| lts_fw_p p q m α:
  lts_step p α q -> lts_fw_step (p ▷ m) α (q ▷ m)
| lts_fw_out_mb m p a :
  lts_fw_step (p ▷ {[+ a +]} ⊎ m) (ActExt $ ActOut a) (p ▷ m)
| lts_fw_inp_mb m p a :
  lts_fw_step (p ▷ m) (ActExt $ ActIn a) (p ▷ {[+ a +]} ⊎ m)
| lts_fw_com m p a q :
  lts_step p (ActExt $ ActIn a) q ->
  lts_fw_step (p ▷ {[+ a +]} ⊎ m) τ (q ▷ m).
```

Definition (39) and Definition (40). For any LTS $\mathcal{L}$, two states of $\mathrm{FW}(\mathcal{L})$ are equivalent, denoted $p \triangleright M \doteq q \triangleright N$, if $\mathsf{strip}(p) \simeq \mathsf{strip}(q)$ and $M \uplus \mathsf{mbox}(p) = N \uplus \mathsf{mbox}(q)$.

```
Inductive strip `{Lts A L} : A -> gmultiset L -> A -> Prop :=
| strip_nil p : p ⇝{∅} p
| strip_step p1 p2 p3 a m :
  p1 ⟶[ActOut a] p2 -> p2 ⇝{m} p3 -> p1 ⇝{{[+ a +]} ⊎ m} p3

where "p ⇝{ m } q" := (strip p m q).

Definition fw_eq `{LtsOba A L} (p : A * mb L) (q : A * mb L) :=
  forall (p' q' : A),
    p.1 ⇝{lts_oba_mo p.1} p' ->
    q.1 ⇝{lts_oba_mo q.1} q' ->
    p' ≃ q' /\ lts_oba_mo p.1 ⊎ p.2 = lts_oba_mo q.1 ⊎ q.2.

Infix "≐" := fw_eq (at level 70).
```

Lemma 41. For every $\mathcal{L}_A$ and every $p \triangleright M, q \triangleright N \in A \times MO$, and every $\alpha \in L$, if $p \triangleright M \ (\doteq \cdot \xrightarrow{\alpha}_{\mathsf{fw}})\ q \triangleright N$ then $p \triangleright M\ (\xrightarrow{\alpha}_{\mathsf{fw}} \cdot \doteq)\ q' \triangleright N'$.

```
Lemma lts_fw_eq_spec `{LtsObaFB A L} p q t mp mq mt α :
  p ▷ mp ≐ t ▷ mt -> (t ▷ mt) ⟶{α} (q ▷ mq) -> p ▷ mp ⟶≐{α} q ▷ mq.
```

Lemma 13. For every LTS $\mathcal{L} \in \mathrm{OF}$, $\mathrm{FW}(\mathcal{L}) \in \mathrm{OW}$.

```
Program Instance LtsMBObaFW `{LtsObaFB A L} : LtsObaFW (A * mb L) L.
```

Lemma 14. For every $\mathcal{L}_A, \mathcal{L}_B \in \mathrm{OF}, p \in A, r \in B$, $p \,\textsc{must}_i\, r$ if and only if $\mathrm{FW}(p) \,\textsc{must}_i\, r$.

```
Lemma must_iff_must_fw
  {@LtsObaFB A L IL LA LOA V, @LtsObaFB B L IL LB LOB W,
   !FiniteLts A L, !Good B L good }
  (p : A) (e : B) : must p e ↔ must (p, ∅) e.
```

I.6 The Acceptance Set Characterisation

The behavioural characterisation with acceptance sets (Definition 9) is formalised as follows. Note that `lts_outputs`, used in the second part of the definition, is part of the definition of an `Lts`, and produces the finite set of outputs that a process can immediately produce.

```
Definition bhv_pre_cond1 `{Lts A L, Lts B L} (p : A) (q : B) :=
  forall s, p ⇓ s -> q ⇓ s.

Notation "p ≼₁ q" := (bhv_pre_cond1 p q) (at level 70).

Definition bhv_pre_cond2 `{Lts A L, Lts B L} (p : A) (q : B) :=
  forall s q', p ⇓ s -> q ⟹[s] q' -> q' ↛ ->
    ∃ p', p ⟹[s] p' /\ p' ↛ /\ lts_outputs p' ⊆ lts_outputs q'.

Notation "p ≼₂ q" := (bhv_pre_cond2 p q) (at level 70).
```

```
Definition bhv_pre `{@Lts A L HL, @Lts B L HL} (p : A) (q : B) :=
  p ≼₁ q /\ p ≼₂ q.

Notation "p ≼ q" := (bhv_pre p q) (at level 70).
```

Given an LTS that satisfies the right conditions, MUST-equivalence coincides with the behavioural characterisation above on the LTS of forwarders (Theorem 17).

```
Section correctness.
  Context `{LtsObaFB A L, LtsObaFB R L, LtsObaFB B L}.
  Context `{!FiniteLts A L, !FiniteLts B L, !FiniteLts R L, !Good B L}.
  (* The LTS can express the tests required for completeness *)
  Context `{!gen_spec_conv gen_conv, !gen_spec_acc gen_acc}.

  Theorem equivalence_bhv_acc_ctx (p : A) (q : R) :
    p ⊑ₑ q <-> (p, ∅) ≼ (q, ∅).
End correctness.
```

I.7 The Must Set characterisation

The behavioural characterisation with must sets (Definition 19) is formalised as follows.

```
Definition MUST `{Lts A L} (p : A) (G : gset (ExtAct L)) :=
    forall p', p ⟹ p' -> exists μ p0, μ ∈ G /\ p' ⟹{μ} p0.

Definition MUST__s `{FiniteLts A L} (ps : gset A) (G : gset (ExtAct L)) :=
  forall p, p ∈ ps -> MUST p G.

Definition AFTER `{FiniteLts A L} (p : A) (s : trace L) (hcnv : p ⇓ s) :=
  wt_set p s hcnv.

Definition bhv_pre_ms_cond2
  `{@FiniteLts A L HL LtsA, @FiniteLts B L HL LtsB} (p : A) (q : B) :=
  forall s h1 h2 G, MUST__s (AFTER p s h1) G -> MUST__s (AFTER q s h2) G.

Notation "p ≾₂ q" := (bhv_pre_ms_cond2 p q) (at level 70).

Definition bhv_pre_ms `{@FiniteLts A L HL LtsA, @FiniteLts B L HL LtsB}
  (p : A) (q : B) := p ≼₁ q /\ p ≾₂ q.

Notation "p ≾ q" := (bhv_pre_ms p q).
```

Lemma 20. Let $\mathcal{L}_A, \mathcal{L}_B \in \mathrm{OF}$. For every $p \in A$ and $q \in B$ such that $\mathrm{FW}(p) \preccurlyeq_{\mathsf{cnv}} \mathrm{FW}(q)$, we have that $\mathrm{FW}(p) \preccurlyeq_{\mathrm{M}} \mathrm{FW}(q)$ if and only if $\mathrm{FW}(p) \preccurlyeq^{\mathsf{fw}}_{\mathsf{acc}} \mathrm{FW}(q)$.

```
Context `{@LtsObaFB A L LL LtsA LtsEqA LtsObaA}.
Context `{@LtsObaFB B L LL LtsR LtsEqR LtsObaR}.
```

```
Lemma equivalence_bhv_acc_mst2 (p : A) (q : B) :
  (p, ∅) ≼₁ (q, ∅) -> (p, ∅) ≾₂ (q, ∅) <-> (p, ∅) ≼₂ (q, ∅).
```

Given an LTS that satisfies the right conditions, MUST-equivalence coincides with the behavioural characterisation above on the LTS of forwarders (Theorem 21).

```
Section correctness.
  Context `{LtsObaFB A L, LtsObaFB R L, LtsObaFB B L}.
  Context `{!FiniteLts A L, !FiniteLts B L, !FiniteLts R L, !Good B L}.
  (* The LTS can express the tests required for completeness. *)
  Context `{!gen_spec_conv gen_conv, !gen_spec_acc gen_acc}.

  Theorem equivalence_bhv_mst_ctx (p : A) (q : R) :
    p ⊑e q <-> (p, ∅) ≾ (q, ∅).
End correctness.
```

I.8 From extensional to intensional definitions

Proposition 31. Given a countably branching STS $\langle S, \rightarrow \rangle$, and a decidable predicate Q on S, for all $s \in S$, $\mathsf{ext}_Q(s)$ implies $\mathsf{int}_Q(s)$.

```
Context `{Hsts: Sts A, @CountableSts A Hsts}.
Context `{@Bar A Hsts}.

Theorem extensional_implies_intensional x:
  extensional_pred x -> intensional_pred x.
```

Corollary 24. For every $p \in A$,

1. $p \downarrow$ if and only if $p \downarrow_i$,
2. for every r we have that $p \, \textsc{must} \, r$ if and only if $p \, \textsc{must}_i \, r$.

```
Context `{Label L}.
Context `{!Lts A L, !FiniteLts A L}.

Lemma terminate_extensional_iff_terminate (p : A) :
  terminate_extensional p <-> terminate p.

Inductive must_sts `{Sts (A * B), good : B -> Prop} (p : A) (e : B) :
  Prop :=
| m_sts_now : good e -> must_sts p e
| m_sts_step
    (nh : ¬ good e)
    (nst : ¬ sts_stable (p, e))
    (l : forall p' e', sts_step (p, e) (p', e') -> must_sts p' e')
  : must_sts p e
.
```

```
Lemma must_extensional_iff_must_sts
  `{good : B -> Prop, good_decidable : forall (e : B), Decision (good e)}
  `{Lts A L, !Lts B L, !LtsEq B L, !Good B L good,
    !FiniteLts A L, !FiniteLts B L} (p : A) (e : B) :
  must_extensional p e <-> must_sts p e.
```

Equivalence between the inductive definitions of MUST defined using Sts and MUST defined using Lts.

```
Inductive must `{Lts A L, !Lts B L, !LtsEq B L, !Good B L good}
  (p : A) (e : B) : Prop :=
| m_now : good e -> must p e
| m_step
    (nh : ¬ good e)
    (ex : ∃ t, parallel_step (p, e) τ t)
    (pt : forall p', p ⟶ p' -> must p' e)
    (et : forall e', e ⟶ e' -> must p e')
    (com : forall p' e' μ, e ⟶[μ] e' -> p ⟶[co μ] p' -> must p' e')
  : must p e
.

Lemma must_sts_iff_must `{Lts A L, !Lts B L, !LtsEq B L, !Good B L good}
  (p : A) (e : B) : must_sts p e <-> must p e.
```

I.9 Completeness

Properties of the functions that generate clients (Table 1).

```
Class gen_spec {A L : Type} `{Lts A L, !LtsEq A L, !Good A L good}
  (gen : list (ExtAct L) -> A) := {
    gen_spec_ungood : forall s, ¬ good (gen s) ;
    gen_spec_mu_lts_co μ s : gen (μ :: s) ⟶≃[co μ] gen s;
    gen_spec_out_lts_tau_ex a s : ∃ e', gen (ActOut a :: s) ⟶ e';
    gen_spec_out_lts_tau_good a s e : gen (ActOut a :: s) ⟶ e -> good e;
    gen_spec_out_lts_mu_uniq {e a μ s} :
    gen (ActOut a :: s) ⟶[μ] e -> e = gen s /\ μ = ActIn a;
  }.

Class gen_spec_conv {A L : Type} `{Lts A L, ! LtsEq A L, !Good A L good}
  (gen_conv : list (ExtAct L) -> A) := {
    gen_conv_spec_gen_spec : gen_spec gen_conv ;
    gen_spec_conv_nil_stable_mu μ : gen_conv [] ↛[μ] ;
    gen_spec_conv_nil_lts_tau_ex : ∃ e', gen_conv [] ⟶ e';
    gen_spec_conv_nil_lts_tau_good e : gen_conv [] ⟶ e -> good e;
  }.

Class gen_spec_acc {A : Type} `{Lts A L, ! LtsEq A L, !Good A L good}
  (gen_acc : gset L -> list (ExtAct L) -> A) := {
```

```
    gen_acc_spec_gen_spec O : gen_spec (gen_acc O);
    gen_spec_acc_nil_stable_tau O : gen_acc O [] ↛;
    gen_spec_acc_nil_stable_out O a : gen_acc O [] ↛[ActOut a];
    gen_spec_acc_nil_mu_inv O a e : gen_acc O [] ⟶[ActIn a] e -> a ∈ O;
    gen_spec_acc_nil_mem_lts_inp O a :
      a ∈ O -> ∃ r, gen_acc O [] ⟶[ActIn a] r;
    gen_spec_acc_nil_lts_inp_good μ e' O :
      gen_acc O [] ⟶[μ] e' -> good e';
  }.
```

Proposition 42. For every $\mathcal{L}_A \in \text{OW}$, $p \in A$, and $s \in \mathsf{Act}^\star$ we have that $p \,\textsc{must}_i\, tc(s)$ if and only if $p \Downarrow s$.

```
Lemma must_iff_cnv
  `{@LtsObaFW A L IL LA LOA V, @LtsObaFB B L IL LB LOB W,
    !Good B L good, !gen_spec_conv gen_conv} (p : A) s :
   must p (gen_conv s) <-> p ⇓ s.
Proof. split; [eapply cnv_if_must | eapply must_if_cnv]; eauto. Qed.
```

Lemma 44. Let $\mathcal{L}_A \in \text{OW}$ and $\mathcal{L}_B \in \text{OF}$. For every $p_1, p_2 \in A$, every $r_1, r_2 \in B$ and name $a \in \mathcal{N}$ such that $p_1 \xrightarrow{\overline{a}} p_2$ and $r_1 \xrightarrow{\overline{a}} r_2$, if $p_1 \,\textsc{must}_i\, r_2$ then $p_2 \,\textsc{must}_i\, r_1$.

```
Lemma must_output_swap_l_fw
  `{@LtsObaFW A L IL LA LOA V, @LtsObaFB B L IL LB LOB W, !Good B L good}
  (p1 p2 : A) (e1 e2 : B) (a : L) :
  p1 ⟶[ActOut a] p2 -> e1 ⟶[ActOut a] e2 -> must p1 e2 -> must p2 e1.
```

Lemma 45. Let $\mathcal{L}_A \in \text{OW}$. For every $p \in A$, $s \in \mathsf{Act}^\star$, and every $L, E \subseteq \mathcal{N}$, if $\overline{L} \in \mathcal{A}_{\mathsf{fw}}(p, s)$ then $p \,\not\textsc{must}_i\, ta(s, E \setminus L)$.

```
Lemma not_must_gen_a_without_required_output
  `{@LtsObaFW A L IL LA LOA V, @LtsObaFB B L IL LB LOB W,
    !Good B L good, !gen_spec_acc gen_acc} (q q' : A) s O :
  q ⟹[s] q' -> q' ↛ -> ¬ must q (gen_acc (O \ lts_outputs q') s).
```

Lemma 46. Let $\mathcal{L}_A \in \text{OW}$. For every $p \in A, s \in \mathsf{Act}^\star$, and every finite set $O \subseteq \overline{\mathcal{N}}$, if $p \Downarrow s$ then either

(i) $p \,\textsc{must}_i\, ta(s, \overline{\bigcup \mathcal{A}_{\mathsf{fw}}(p, s) \setminus O})$, or

(ii) there exists $\widehat{O} \in \mathcal{A}_{\mathsf{fw}}(p, s)$ such that $\widehat{O} \subseteq O$.

```
Lemma must_gen_a_with_s
  `{@LtsObaFW A L IL LA LOA V, @LtsObaFB B L IL LB LOB W,
    !FiniteLts A L, !Good B L good, !gen_spec_acc gen_acc}
  s (p : A) (hcnv : p ⇓ s) O :
  (exists p', p ⟹[s] p' /\ lts_stable p' τ /\ lts_outputs p' ⊆ O)
    \/ must p (gen_acc (oas p s hcnv \ O) s).
```

Lemma 47. For every $\mathcal{L}_A, \mathcal{L}_B \in \text{OW}$ and servers $p \in A, q \in B$, if $p \sqsubseteq_{\textsc{must}} q$ then $p \preccurlyeq_{\mathsf{AS}} q$.

```
Lemma completeness_fw
  `{@LtsObaFW A L IL LA LOA V, @LtsObaFB B L IL LB LOB W,
    @LtsObaFW C L IL LC LOC VC, !FiniteLts A L, !FiniteLts C L,
    !FiniteLts B L, !Good B L good,
    !gen_spec_conv gen_conv, !gen_spec_acc gen_acc}
  (p : A) (q : C) : p ⊑ q -> p ≼ q.
```

Proposition 48. For every $\mathcal{L}_A, \mathcal{L}_B \in \text{OF}$ and servers $p \in A, q \in B$, if $p \sqsubseteq_{\text{MUST}} q$ then $\text{FW}(p) \preccurlyeq_{\mathsf{AS}} \text{FW}(q)$.

```
Lemma completeness
  `{@LtsObaFB A L IL LA LOA V, @LtsObaFB B L IL LB LOB W,
    @LtsObaFB C L IL LC LOC VC,
    !FiniteLts A L, !FiniteLts B L, !FiniteLts C L, !Good C L good,
    !gen_spec_conv gen_conv, !gen_spec_acc gen_acc}
  (p : A) (q : B) : p ⊑ q -> p ▷ ∅ ≼ q ▷ ∅.
```

I.10 Soundness

Figure 11. Rules to define inductively the predicate $\text{MUST}_{\mathsf{aux}}$.

```
Inductive mustx
  `{Lts A L, !FiniteLts A L, !Lts B L, !LtsEq B L, !Good B L good}
  (ps : gset A) (e : B) : Prop :=
| mx_now (hh : good e) : mustx ps e
| mx_step
    (nh : ¬ good e)
    (ex : forall (p : A), p ∈ ps -> ∃ t, parallel_step (p, e) τ t)
    (pt : forall ps',
        lts_tau_set_from_pset_spec1 ps ps' -> ps' ≠ ∅ ->
        mustx ps' e)
    (et : forall (e' : B), e ⟶ e' -> mustx ps e')
    (com : forall (e' : B) μ (ps' : gset A),
        lts_step e (ActExt μ) e' ->
        wt_set_from_pset_spec1 ps [co μ] ps' -> ps' ≠ ∅ ->
        mustx ps' e')
  : mustx ps e.
```

Lemma 65. For every LTS $\mathcal{L}_A, \mathcal{L}_B$ and every $X \in \mathcal{P}^+(A)$, we have that $X \ \text{MUST}_{\mathsf{aux}}\ r$ if and only if for every $p \in X. p \ \text{MUST}_i\ r$.

```
Lemma must_set_iff_must_for_all
  `{Lts A L, !FiniteLts A L, !Lts B L, !LtsEq B L, !Good B L good}
  (X : gset A) (e : B) : X ≠ ∅ ->
    (forall p, p ∈ X -> must p e) <-> mustx X e.
```

Lifting of the predicates $\preccurlyeq_{\mathsf{cnv}}$ and $\preccurlyeq_{\mathsf{acc}}$ to sets of servers.

```
Definition bhv_pre_cond1__x `{FiniteLts P L, FiniteLts Q L}
 (ps : gset P) (q : Q) := forall s, (forall p, p ∈ ps -> p ⇓ s) -> q ⇓ s.

Notation "ps ≼ₓ1 q" := (bhv_pre_cond1__x ps q) (at level 70).

Definition bhv_pre_cond2__x
  `{@FiniteLts P L HL LtsP, @FiniteLts Q L HL LtsQ}
  (ps : gset P) (q : Q) :=
  forall s q', q ⟹[s] q' -> q' ↛ ->
    (forall p, p ∈ ps -> p ⇓ s) ->
    exists p, p ∈ ps /\ exists p',
      p ⟹[s] p' /\ p' ↛ /\ lts_outputs p' ⊆ lts_outputs q'.

Notation "ps ≼ₓ2 q" := (bhv_pre_cond2__x ps q) (at level 70).

Notation "ps ≼ₓ q" := (bhv_pre_cond1__x ps q /\ bhv_pre_cond2__x ps q)
   (at level 70).
```

Lemma 66. For every LTS $\mathcal{L}_A, \mathcal{L}_B$ and servers $p \in A$, $q \in B$, $p \preccurlyeq_{\mathsf{AS}} q$ if and only if $\{p\} \preccurlyeq^{\mathsf{set}}_{\mathsf{AS}} q$.

```
Lemma alt_set_singleton_iff
  `{@FiniteLts P L HL LtsP, @FiniteLts Q L HL LtsQ}
  (p : P) (q : Q) : p ≼ q <-> {[ p ]} ≼ₓ q.
```

Lemma 67. Let $\mathcal{L}_A, \mathcal{L}_B \in \mathrm{OW}$. For every set $X \in \mathcal{P}^+(A)$, and $q \in B$, such that $X \preccurlyeq^{\mathsf{set}}_{\mathsf{cnv}} q$ then

1. $q \xrightarrow{\tau} q'$ implies $X \preccurlyeq^{\mathsf{set}}_{\mathsf{cnv}} q'$,
2. $X \downarrow_i$, $X \stackrel{\mu}{\Longrightarrow} X'$ and $q \xrightarrow{\mu} q'$ imply $X' \preccurlyeq^{\mathsf{set}}_{\mathsf{cnv}} q'$.

Lemma 68. Let $\mathcal{L}_A, \mathcal{L}_B \in \mathrm{OW}$. For every $X, X' \in \mathcal{P}^+(A)$ and $q \in B$, such that $X \preccurlyeq^{\mathsf{set}}_{\mathsf{acc}} q$, then

1. $q \xrightarrow{\tau} q'$ implies $X \preccurlyeq^{\mathsf{set}}_{\mathsf{acc}} q'$,
2. for every $\mu \in \mathsf{Act}$, if $X \downarrow_i$, then for every $q \xrightarrow{\mu} q'$ and set $X \stackrel{\mu}{\Longrightarrow} X'$ we have $X' \preccurlyeq^{\mathsf{set}}_{\mathsf{acc}} q'$.

```
Lemma bhvx_preserved_by_tau
  `{@FiniteLts P L HL LtsP, @FiniteLts Q L HL LtsQ}
  (ps : gset P) (q q' : Q) : q ⟶ q' -> ps ≼ₓ q -> ps ≼ₓ q'.

Lemma bhvx_preserved_by_mu
  `{@FiniteLts P L HL LtsP, @FiniteLts Q L HL LtsQ}
  (ps0 : gset P) (q : Q) μ ps1 q'
  (htp : forall p, p ∈ ps0 -> terminate p) :
  q ⟶[μ] q' -> wt_set_from_pset_spec ps0 [μ] ps1 ->
  ps0 ≼ₓ q -> ps1 ≼ₓ q'.
```

Lemma 76 Let $\mathcal{L}_A, \mathcal{L}_B \in \mathrm{OW}$ and $\mathcal{L}_C \in \mathrm{OF}$. For every $X \in \mathcal{P}^+(A)$ and $q \in B$ such that $X \preccurlyeq^{\mathsf{set}}_{\mathsf{AS}} q$, for every $r \in C$ if $\neg\textsc{good}(r)$ and X $\textsc{must}_{\mathsf{aux}}$ r then $q \parallel r \xrightarrow{\tau}$.

```
Lemma stability_nbhvleqtwo
  `{@LtsObaFW P L Lbl LtsP LtsEqP LtsObaP,
    @LtsObaFW Q L Lbl LtsQ LtsEqQ LtsObaQ,
    !FiniteLts P L, !FiniteLts Q L, !Lts B L, !LtsEq B L, !Good B L good}
  (X : gset P) (q : Q) e :
  ¬ good e -> mustx X e -> X ≼ₓ2 q -> exists t, (q, e) ⟶{τ} t.
```

Lemma 77 Let $\mathcal{L}_A, \mathcal{L}_B \in \mathrm{OW}$. For every $X \in \mathcal{P}^+(A)$ and $q, q' \in B$, such that $X \preccurlyeq^{\mathsf{set}}_{\mathsf{AS}} q$, then for every $\mu \in \mathsf{Act}$, if $X \Downarrow \mu$ and $q \xrightarrow{\mu} q'$ then $X \overset{\mu}{\Longrightarrow}$.

```
Lemma bhvx_mu_ex `{@FiniteLts P L HL LtsP, @FiniteLts Q L HL LtsQ}
  (ps : gset P) (q q' : Q) μ
  : ps ≼ₓ q -> (forall p, p ∈ ps -> p ⇓ [μ]) ->
    q ⟶[μ] q' -> exists p', wt_set_from_pset_spec1 ps [μ] {[ p' ]}.
```

Lemma 78 For every $\mathcal{L}_A \in \mathrm{OW}$, $\mathcal{L}_B \in \mathrm{OF}$, every set of processes $X \in \mathcal{P}^+(A)$, every $r \in B$, and every $\mu \in \mathsf{Act}$, if $X \ \textsc{must}_{\mathsf{aux}}\ r$, $\neg\textsc{good}(r)$ and $r \xrightarrow{\mu}$ then $X \Downarrow \overline{\mu}$.

```
Lemma ungood_acnv_mu `{LtsOba A L, !FiniteLts A L, !Lts B L, !LtsEq B L,
  !Good B L good} ps e e' : mustx ps e ->
  forall μ p, p ∈ ps -> e ⟶[co μ] e' -> ¬ good e -> p ⇓ [μ].
```

Lemma 69. Let $\mathcal{L}_A, \mathcal{L}_B \in \mathrm{OW}$ and $\mathcal{L}_C \in \mathrm{OF}$. For every set of processes $X \in \mathcal{P}^+(A)$, server $q \in B$ and client $r \in C$, if $X \ \textsc{must}_{\mathsf{aux}}\ r$ and $X \preccurlyeq^{\mathsf{set}}_{\mathsf{AS}} q$ then $q \ \textsc{must}_i\ r$.

```
Lemma soundnessx `{
    @LtsObaFW A L Lbl LtsA LtsEqA LtsObaA,
    @LtsObaFW C L Lbl LtsC LtsEqC LtsObaC,
    @LtsObaFB B L Lbl LtsB LtsEqB LtsObaB,
    !FiniteLts A L, !FiniteLts C L, !FiniteLts B L, !Good B L good}
    (ps : gset A) (e : B) : mustx ps e ->
      forall (q : C), ps ≼ₓ q -> must q e.
```

Proposition 70. For every $\mathcal{L}_A, \mathcal{L}_B \in \mathrm{OF}$ and servers $p \in A, q \in B$, if $\mathrm{FW}(p) \preccurlyeq_{\mathsf{AS}} \mathrm{FW}(q)$ then $p \sqsubseteq_{\textsc{must}} q$.

```
Lemma soundness
  `{@LtsObaFB A L IL LA LOA V, @LtsObaFB C L IL LC LOC T,
    @LtsObaFB B L IL LB LOB W,
    !FiniteLts A L, !FiniteLts C L, !FiniteLts B L, !Good B L good }
  (p : A) (q : C) : p ▷ ∅ ≼ q ▷ ∅ -> p ⊑ q.
```

▶ **Corollary 108.** *Let $\mathcal{L}_A, \mathcal{L}_B \in OF$. For every $p \in A$ and $q \in B$, we have that $p \sqsubseteq_{\textsc{must}} q$ if and only if $p \leq_{\mathit{fail}} q$.*

```
Section failure.
```

```
  Definition Failure `{FiniteLts A L} (p : A)
    (s : trace L) (G : gset (ExtAct L)) :=
    p ⇓ s -> exists p', p ⟹[s] p' /\
      forall μ, μ ∈ G -> ¬ exists p0, p' ⟹{μ} p0.

  Definition fail_pre_ms_cond2
    `{@FiniteLts A L HL LtsA, @FiniteLts B L HL LtsB}
    (p : A) (q : B) := forall s G, Failure q s G -> Failure p s G.

  Definition fail_pre_ms
    `{@FiniteLts A L HL LtsA, @FiniteLts B L HL LtsB} (p : A) (q : B) :=
    p ≼₁ q /\ fail_pre_ms_cond2 p q.

  Context `{LL : Label L}.
  Context `{LtsA : !Lts A L, !FiniteLts A L}.
  Context `{LtsR : !Lts R L, !FiniteLts R L}.

  Context `{@LtsObaFB A L LL LtsA LtsEqA LtsObaA}.
  Context `{@LtsObaFB R L LL LtsR LtsEqR LtsObaR}.

  Theorem equivalence_pre_failure_must_set (p : A) (q : R) :
   (p ▷ ∅) ≾ (q ▷ ∅) <-> (p ▷ ∅) ⋖ (q ▷ ∅).

End failure.
```

J Mapping of results from the paper to the Coq code

Paper	Coq File	Coq name
Figure 2	TransitionSystems.v	Class LtsOba
Figure 3	TransitionSystems.v	Class Sts, ExtAct, Act, Label, Lts
Definition (2)	Equivalence.v	must_extensional
Definition (3)	Equivalence.v	pre_extensional
Equation (3)	ACCSInstance.v	proc
Figure 5	TransitionSystems.v	LtsEq
Definition (2)	MustEx.v	must_extensional
Definition (6)	TransitionSystems.v	max_exec_from
$p \overset{s}{\Longrightarrow} p'$	TransitionSystems.v	wt
$p \downarrow$	Equivalence.v	terminate_extensional
$p \Downarrow s$	TransitionSystems.v	cnv
Lemma 50	TransitionSystems.v	cnv_iff_prefix_terminate
Lemma 51	TransitionSystems.v	stable_tau_preserved_by_wt_output, stable_tau_input_preserved_by_wt_output
Lemma 49	Must.v	ungood_preserved_by_wt_output
Equation (5)	TransitionSystems.v	Class LtsObaFW
Definition (9)	Must.v	bhv_pre
Figure 6	TransitionSystems.v	lts_fw_step
Definition (10)	TransitionSystems.v	MbLts
Definition (39)	TransitionSystems.v	strip
Definition (40)	TransitionSystems.v	fw_eq
Lemma 41	TransitionSystems.v	lts_fw_eq_spec
Lemma 13	TransitionSystems.v	Instance LtsMBObaFW
Lemma 14	Lift.v	must_iff_must_fw
Lemma 14	Lift.v	lift_fw_ctx_pre
Theorem 17	Equivalence.v	equivalence_bhv_acc_ctx
Definition (19)	Must.v	bhv_pre_ms
Lemma 20	Must.v	equivalence_bhv_acc_mst
Theorem 21	Must.v	equivalence_bhv_mst_ctx
Lemma 5	ACCSInstance.v	ACCS_ltsObaFB
Corollary 18	ACCSInstance.v	bhv_iff_ctx_ACCS
Proposition 31	Bar.v	extensional_implies_intensional
$p \downarrow_i$	TransitionSystems.v	terminate
$p \, \textsc{must}_i \, q$	Must.v	must_sts
Corollary 24	Equivalence.v	terminate_extensional_iff_terminate
Table 1	Completeness.v	Class gen_spec, gen_spec_conv, gen_spec_acc
Proposition 42	Completeness.v	must_iff_cnv
Lemma 44	Lift.v	must_output_swap_l_fw
Lemma 45	Completeness.v	not_must_gen_a_without_required_output
Lemma 46	Completeness.v	must_gen_a_with_s
Lemma 47	Completeness.v	completeness_fw
Proposition 48	Completeness.v	completeness
Lemma 65	Soundness.v	must_set_iff_must_for_all
Figure 11	Soundness.v	mustx
$X \preccurlyeq^{\mathsf{set}}_{\mathsf{cnv}} q$ and $X \preccurlyeq^{\mathsf{set}}_{\mathsf{acc}} q$	Soundness.v	bhv_pre_cond1$_x$ and bhv_pre_cond2$_x$
Lemma 66	Soundness.v	must_set_iff_must
Lemma 67, Lemma 68	Soundness.v	bhvx_preserved_by_tau, bhvx_preserved_by_mu
Lemma 76	Soundness.v	stability_nbhvleqtwo
Lemma 77	Soundness.v	bhvx_mu_ex
Lemma 69	Soundness.v	soundnessx
Proposition 70	Soundndess.v	soundness