



HAL
open science

Constructive characterisations of the must-preorder for asynchrony

Giovanni Bernardi, Ilaria Castellani, Paul Laforgue, Léo Stefanescu

► **To cite this version:**

Giovanni Bernardi, Ilaria Castellani, Paul Laforgue, Léo Stefanescu. Constructive characterisations of the must-preorder for asynchrony. 2024. hal-04642776

HAL Id: hal-04642776

<https://hal.science/hal-04642776>

Preprint submitted on 10 Jul 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License


1 Constructive characterisations of the must-preorder 2 for asynchrony

3 Giovanni Bernardi ✉ 

4 Université Paris Cité, CNRS, IRIF, F-75013, Paris, France

5 Ilaria Castellani ✉ 

6 INRIA, Université Côte d'Azur, France

7 Paul Laforgue ✉ 

8 Université Paris Cité, CNRS, IRIF, F-75013, Paris, France

9 Nomadic Labs, Paris, France

10 Léo Stefanescu ✉ 

11 MPI-SWS

12 — Abstract —

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

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

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

26 **2012 ACM Subject Classification** Program verification, Constructive mathematics, Operational
27 semantics

28 **Keywords and phrases** Software Verification, Observational equivalence, Asynchrony

29 **Digital Object Identifier** 10.4230/LIPIcs...

30 **1** Introduction

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

36 **Comparing servers.** This paper studies a version of Morris preorder for nondeterministic
37 asynchronous client-server systems. In this setting it is natural to reformulate the preorder
38 by replacing reduction to normal forms (*i.e.* termination) with a suitable *liveness* property.
39 Let $p \parallel r$ denote a *client-server system*, that is a parallel composition in which the identities
40 of the server p and the client r are distinguished, and whose computations have the form
41 $p \parallel r = p_0 \parallel r_0 \longrightarrow p_1 \parallel r_1 \longrightarrow p_2 \parallel r_2 \longrightarrow \dots$, where each step represents either an internal
42 computation of one of the two components, or an interaction between them. Interactions
43 correspond to handshakes, where two components ready to perform matching input/output
44 actions advance together. We express liveness by saying that p *must pass* r , denoted $p \text{ MUST } r$,

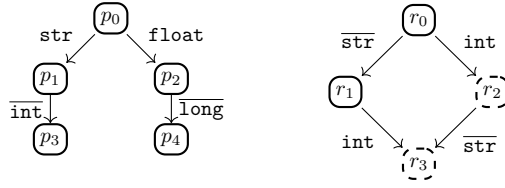


© Author: Please provide a copyright holder;

licensed under Creative Commons License CC-BY 4.0

Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



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

45 if in every maximal computation of $p \parallel r$ there exists a state $p_i \parallel r_i$ such that $\text{GOOD}(r_i)$,
 46 where GOOD is a decidable predicate indicating that the client has reached a successful state.
 47 Servers are then compared according to their capacity to satisfy clients, *i.e.* via contexts of
 48 the form $[-] \parallel r$ and the predicate MUST . Morris preorder then becomes the MUST -preorder
 49 by De Nicola and Hennessy [44] : $p \sqsubseteq_{\text{MUST}} q$ when $\forall r. p \text{ MUST } r$ implies $q \text{ MUST } r$.

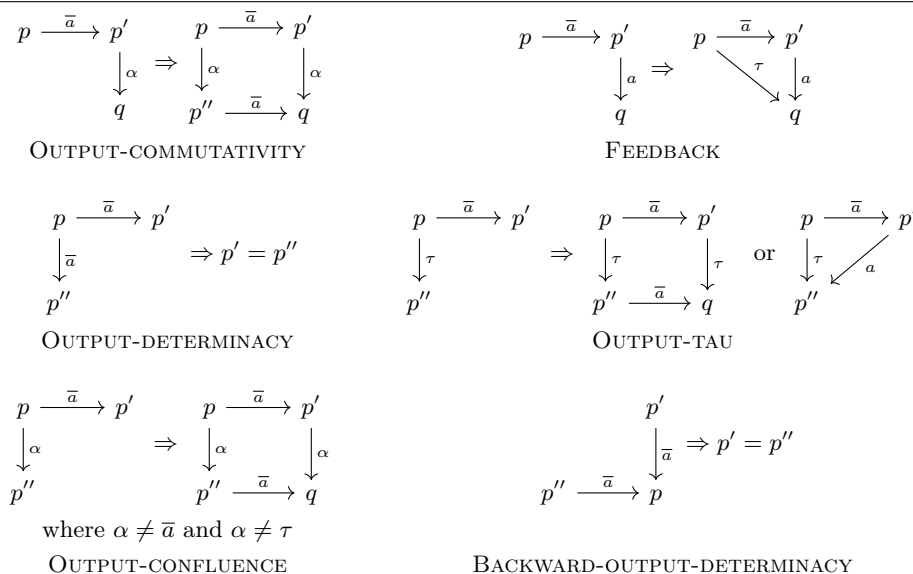
50 **Advantages.** The MUST -preorder is by definition liveness preserving, because $p \text{ MUST } r$
 51 literally means that “in every execution something good must happen (on the client side)”.
 52 Results on $\sqsubseteq_{\text{MUST}}$ thus shed light on liveness-preserving program transformations.

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

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

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

78 **Alternative preorders for synchrony.** Both approaches were originally proposed for
 79 the calculus CCS [75], where communication is synchronous. The first alternative preorder,
 80 which we denote by \preceq_{MS} , was put forth by De Nicola [44], and it compares server behaviours
 81 according to their MUST -sets, *i.e.* the sets of actions that they may perform. The second
 82 alternative preorder, which we denote by \preceq_{AS} , was put forth by Hennessy [55], and it



■ **Figure 2** First-order axioms for output-buffered agents with feedback as given by Selinger [92], extended with the BACKWARD-OUTPUT-DETERMINACY axiom.

83 compares the acceptance sets of servers, *i.e.* how servers can be moved out of their potentially
 84 deadlocked states. Both these preorders characterise $\sqsubseteq_{\text{MUST}}$ in the following sense:

85 $\forall p, q \in \text{CCS}. p \sqsubseteq_{\text{MUST}} q \text{ iff } \text{LTS}(p) \preceq_{\text{MS}} \text{LTS}(q)$ (1)

86 $\forall p, q \in \text{CCS}. p \sqsubseteq_{\text{MUST}} q \text{ iff } \text{LTS}(p) \preceq_{\text{AS}} \text{LTS}(q)$ (2)

88 **Asynchrony.** In distributed systems, however, communication is inherently asynchronous.
 89 For instance, the standard TCP transmission on the Internet is asynchronous. Actor languages
 90 like ELIXIR and ERLANG implement asynchrony via mailboxes, and both PYTHON and
 91 JAVASCRIPT offer developers the constructs ASYNC/WAIT, to return promises (of results) or
 92 wait for them. In this paper we model asynchrony via *output-buffered agents with feedback*,
 93 as introduced by Selinger [92]. These are LTSs obeying the axioms in Figure 2, where a
 94 denotes an input action, \bar{a} denotes an output action, τ denotes the internal action, and α
 95 ranges over all these actions. For instance, the OUTPUT-COMMUTATIVITY axiom states that
 96 an output \bar{a} can always be postponed: if \bar{a} is followed by any action α , it can commute with
 97 it. In other terms, outputs are non-blocking, as illustrated by the LTS for r_0 in Figure 1.

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

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

- 108 ■ The first behavioural characterisations of the MUST-preorder (Theorem 17, Theorem 21),
 109 that are calculus independent, in that both our definitions and our proofs work directly

110 on LTSs. Contrary to all the previous works on the topic, we show that the *standard*
 111 alternative preorders characterise the MUST-preorder also in Selinger asynchronous setting.
 112 To this end, it suffices to enrich the server semantics with *forwarding*, i.e. ensure that
 113 servers are ready to receive any input message, as long as they store it back in a global
 114 shared buffer. This idea, although we use it here in a slightly different form, was pioneered
 115 by Honda et al. [64]. In this paper we propose a construction that works on any LTS
 116 (Lemma 13) and we show the following counterparts of Equations (1) and (2), where OF
 117 denotes the LTSs of output-buffered agents with feedback, and FW is the function that
 118 enhances them with forwarding:

$$119 \quad \forall p, q \in \text{OF}. p \sqsubseteq_{\text{MUST}} q \text{ iff } \text{FW}(p) \preceq_{\text{MS}} \text{FW}(q) \quad (\text{a})$$

$$120 \quad \forall p, q \in \text{OF}. p \sqsubseteq_{\text{MUST}} q \text{ iff } \text{FW}(p) \preceq_{\text{AS}} \text{FW}(q) \quad (\text{b})$$

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

- 125 ■ The first constructive account of the MUST-preorder. We show that if the MUST and
 126 termination predicates are defined *intensionally* (in the sense of Brede and Herbelin [29]),
 127 then $\sqsubseteq_{\text{MUST}}$ can be characterised constructively. The original definitions of MUST and
 128 termination given by De Nicola [44], though, are *extensional*. Showing that intensional and
 129 extensional definitions are logically equivalent is a known problem, discussed for instance
 130 by Coquand [43] and Brede and Herbelin [29]. We follow their approach and adapt the
 131 bar-induction principle to our setting to prove the desired equivalences. Our treatment
 132 shows that König’s lemma, which is a mainstay in the literature on the MUST-preorder, is
 133 actually unnecessary: the bar-induction principle suffices for our purposes¹. Since Rahl
 134 et al. [83] have shown bar-induction to be compatible with constructive type theory, we
 135 argue that our development is entirely constructive. Due to space constraints, we explain
 136 the principle of bar-induction and how to adapt it to our usage in Appendix A, while in
 137 this extended abstract we merely employ the principle.
- 138 ■ The first mechanisation of the theory of MUST-preorder in a fully nondeterministic setting,
 139 which consists of around 8000 lines of Coq. In Appendix J we gather the Coq versions of
 140 all the definitions and the results used in the main body of the paper.

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

144 2 Preliminaries

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

148 **Labelled transition systems.** A *labelled transition system* (LTS) is a triple $\mathcal{L} =$
 149 $\langle A, L, \longrightarrow \rangle$ where A is the set of states, L is the set of labels and $\longrightarrow \subseteq A \times L \times A$ is the
 150 transition relation. When modelling programs as LTSs, we use transition labels to represent
 151 program actions. The set of labels in Selinger LTSs has the same structure as the set of
 152 actions in Milner’s calculus CCS: one assumes a set of names \mathcal{N} , denoting input actions

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

```

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

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

Class Label (L: Type) :=          Class Lts (A L : Type) {Label L} :=
MkLabel {                          MkLts {
  label_eqdec: EqDecision L;       lts_step: A → Act L → A → Prop;
  label_countable: Countable L;    lts_outputs: A → finite_set L;
                                   lts_performs: A → (Act L) → Prop; }.

```

■ **Figure 3** Highlights of our Sts and Lts typeclasses.

$$\begin{array}{c}
\text{[S-SRV]} \quad \frac{p \xrightarrow{\tau} p'}{p \parallel r \longrightarrow p' \parallel r} \quad \text{[S-CLT]} \quad \frac{r \xrightarrow{\tau} r'}{p \parallel r \longrightarrow p \parallel r'} \quad \text{[S-COM]} \quad \frac{p \xrightarrow{\mu} p' \quad r \xrightarrow{\bar{\mu}} r'}{p \parallel r \longrightarrow p' \parallel r'}
\end{array}$$

■ **Figure 4** The STS of server-client systems.

153 and ranged over by a, b, c , a complementary set of conames $\bar{\mathcal{N}}$, denoting output actions and
 154 ranged over by $\bar{a}, \bar{b}, \bar{c}$, and an *invisible* action τ , representing internal computation. The set
 155 of all actions, ranged over by α, β, γ , is given by $\text{Act}_\tau \stackrel{\text{def}}{=} \mathcal{N} \uplus \bar{\mathcal{N}} \uplus \{\tau\}$. We use μ, μ' to
 156 range over the set of visible actions $\mathcal{N} \uplus \bar{\mathcal{N}}$, and we extend the complementation function $\bar{\cdot}$ to
 157 this set by letting $\bar{\bar{a}} \stackrel{\text{def}}{=} a$. In the following, we will always assume $L = \text{Act}_\tau$. Once the LTS is
 158 fixed, we write $p \xrightarrow{\alpha} p'$ to mean that $(p, \alpha, p') \in \longrightarrow$ and $p \xrightarrow{\alpha}$ to mean $\exists p'. p \xrightarrow{\alpha} p'$.

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

161 In our mechanisation LTSs are borne out by the typeclass `Lts` in Figure 3. The states of
 162 the LTS have type A , labels have type L , and `lts_step` is the characteristic function of the
 163 transition relation, which we assume to be decidable. We let $O(p) = \{\bar{a} \in \bar{\mathcal{N}} \mid p \xrightarrow{\bar{a}}\}$ and
 164 $I(p) = \{a \in \mathcal{N} \mid p \xrightarrow{a}\}$ be respectively the set of outputs and the set of inputs of state p . We
 165 assume that the set $O(p)$ is finite for any p . In our mechanisation, the set $O(p)$ is rendered by
 166 the function `lts_outputs`, and we shall also use a function `lts_performs` that lets us decide
 167 whether a state can perform a transition labelled by a given action.

168 **Client-server systems.** A *client-server* system (or *system*, for short) is a pair $p \parallel r$ in
 169 which p is deemed to be the server of client r . In general, every system $p \parallel r$ is the root of a
 170 *state transition system* (STS), $\langle S, \longrightarrow \rangle$, where S is the set of states and \longrightarrow is the reduction
 171 relation. For the sake of simplicity² we derive the reduction relation from the LTS semantics
 172 of servers and clients as specified by the rules in Figure 4. In our mechanisation (Figure 3),

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.

XX:6 Constructive characterisations of the must-preorder for asynchrony

173 `sts_step` is the characteristic function of the reduction relation \longrightarrow , and `sts_stable` is the
174 function that states whether a state can reduce or not. Both functions are assumed decidable.

175 ► **Definition 1** (Computation). *Given an STS $\langle S, \longrightarrow \rangle$ and a state $s_0 \in S$, a computation of s_0
176 is a finite or infinite reduction sequence³, i.e. a partial function η from \mathbb{N} to S whose domain
177 is downward-closed, such that $s_0 = \eta(0)$ and for each $n \in \text{dom}(\eta) \setminus \{0\}$, $\eta(n-1) \longrightarrow \eta(n)$. ■*

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

189 ► **Definition 2** (Client satisfaction). *We write $p \text{ MUST } r$ if every maximal computation of $p \parallel r$
190 is successful. ■*

191 ► **Definition 3** (MUST-preorder). *We let $p \sqsubseteq_{\text{MUST}} q$ whenever for every client r we have that
192 $p \text{ MUST } r$ implies $q \text{ MUST } r$. ■*

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

199 **A closer look at Selinger axioms.** Let us now discuss the axioms in Figure 2. The
200 `OUTPUT-COMMUTATIVITY` axiom expresses the non-blocking behaviour of outputs: an output
201 cannot be a cause of any subsequent transition, since it can also be executed after it, leading
202 to the same resulting state. Hence, outputs are concurrent with any subsequent transition.
203 The `FEEDBACK` axiom says that an output followed by a complementary input can also
204 synchronise with it to produce a τ -transition. These first two axioms specify properties
205 of outputs that are followed by another transition. Instead, the following three axioms,
206 `OUTPUT-CONFLUENCE`, `OUTPUT-DETERMINACY` and `OUTPUT-TAU`, specify properties of
207 outputs that are co-initial with another transition⁴. The `OUTPUT-DETERMINACY` and
208 `OUTPUT-TAU` axioms apply to the case where the co-initial transition is an identical output or
209 a τ -transition respectively, while the `OUTPUT-CONFLUENCE` axiom applies to the other cases.
210 When taken in conjunction, these three axioms state that outputs cannot be in conflict with
211 any co-initial transition, except when this is a τ -transition: in this case, the `OUTPUT-TAU`
212 axiom allows for a confluent nondeterminism between the τ -transition on one side and the
213 output followed by the complementary input on the other side.

³ Which is defined as a coinductive type in our Coq development.

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

$$\begin{array}{ccc}
p & & p \xrightarrow{\alpha} p' \\
\vdots & & \vdots \\
\approx & \Rightarrow & \approx \\
\vdots & & \vdots \\
q \xrightarrow{\alpha} q' & & q \xrightarrow{\alpha} q'
\end{array}$$

■ **Figure 5** Axiom stating that equivalence \approx is compatible with a transition relation.

214 We now explain the novel BACKWARD-OUTPUT-DETERMINACY axiom. It is the dual of
 215 OUTPUT-DETERMINACY, as it states that also backward transitions with identical outputs
 216 lead to the same state. The intuition is that if two programs arrive at the same state by
 217 removing the same message from the mailbox, then they must coincide. This axiom need not
 218 be assumed in [92] because it can be derived from Selinger axioms when modelling a calculus
 219 like ACCS equipped with a parallel composition operator \parallel (see Lemma 87 in Appendix F).
 220 We use the BACKWARD-OUTPUT-DETERMINACY axiom only to prove a technical property of
 221 clients (Lemma 54) that is used to prove our completeness result.

222 **Calculi.** A number of asynchronous calculi [64, 25, 39, 61, 79, 88] have an LTS that
 223 enjoys the axioms in Figure 2, at least up to some structural equivalence \equiv . The reason is
 224 that these calculi syntactically enforce outputs to have no continuation, *i.e.* outputs can only
 225 be composed in parallel with other processes.⁵ For example, Selinger [92] shows that the
 226 axioms of Figure 2 hold for the LTS of the calculus ACCS (the asynchronous variant of CCS⁶)
 227 modulo bisimulation, and in Lemma 90 we prove this for the LTS of ACCS modulo \equiv :

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

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

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

$$235 \quad p, q, r ::= \bar{a} \mid g \mid p \parallel p \mid \text{rec}x.p \mid x, \quad g ::= 0 \mid a.p \mid \tau.p \mid g + g \quad (3)$$

236 as well as its standard LTS⁷ whose properties we discuss in detail in Appendix F. This
 237 is exactly the syntax used in [92, 24], without the operators of restriction and relabelling.
 238 Here the syntactic category g defines *guards*, *i.e.* the terms that may be used as arguments
 239 for the $+$ operator. Note that, apart from 0 , only input-prefixed and τ -prefixed terms are
 240 allowed as guards, and that the output prefix operator is replaced by *atoms* \bar{a} . In fact, this
 241 syntax is completely justified by Selinger axioms, which, as we argued above, specify that
 242 outputs cannot cause any other action, nor be in conflict with it.

243 ► **Definition 6.** *Given an LTS $\langle A, L, \longrightarrow \rangle$ and state $p_0 \in A$, a transition sequence of p_0 is a
 244 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,
 245 for every $n \geq 1$ such that p_n is in the sequence we have $p_{n-1} \xrightarrow{\alpha_n} p_n$. ■*

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.

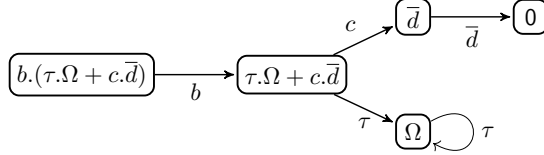
⁷ Where the recursion rule is replaced by the one usually adopted for testing semantics, which introduces a τ -transition before each unfolding.

XX:8 Constructive characterisations of the must-preorder for asynchrony

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

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

251 ► **Example 7.** Let $\Omega = \text{rec } x.\tau.x$ and $Pierre = b.(\tau.\Omega + c.\bar{d})$. The LTS of $Pierre$ is as follows:



252

253 $Pierre$ models a citizen confronted with an unpopular pension reform. To begin with, $Pierre$
 254 can only do the input b , which models his getting aware of the brute-force imposition of the
 255 reform by the government. After performing the input, $Pierre$ reaches the state $\tau.\Omega + c.\bar{d}$,
 256 where he behaves in a nondeterministic manner. He can internally choose not to trust the
 257 government for any positive change, in which case he will diverge, refusing any further
 258 interaction. But this need not happen: in case the government offers the action \bar{c} , which
 259 models a positive change in political decision, $Pierre$ can decide to accept this change, and
 260 then he expresses his agreement with the output \bar{d} , which stands for “done”. ◀

261 ► **Example 8.** We prove now the inequality $Pierre \sqsubseteq_{\text{MUST}} 0$ by leveraging the possibility of
 262 divergence of $Pierre$ after the input b . Fix an r such that $Pierre \text{ MUST } r$. We distinguish two
 263 cases, according to whether $r \xrightarrow{\bar{b}}$ or $r \not\xrightarrow{\bar{b}}$.

264 *i)* Let $r \xrightarrow{\bar{b}} r'$ for some r' . Consider the maximal computation $Pierre \parallel r \longrightarrow \tau.\Omega +$
 265 $c.\bar{d} \parallel r' \longrightarrow \Omega \parallel r' \longrightarrow \dots$ in which $Pierre$ diverges and r does not move after the first
 266 output. Since $Pierre \text{ MUST } r$, either $\text{GOOD}(r)$ or $\text{GOOD}(r')$. In case $\text{GOOD}(r')$, by Lemma 87
 267 we get also $\text{GOOD}(r)$. Hence $0 \text{ MUST } r$.

268 *ii)* Let $r \not\xrightarrow{\bar{b}}$. Suppose $r = r_0 \xrightarrow{\tau} r_1 \xrightarrow{\tau} r_2 \xrightarrow{\tau} \dots$ is a maximal computation of r . Then
 269 $Pierre \parallel r$ has a maximal computation $Pierre \parallel r_0 \longrightarrow Pierre \parallel r_1 \longrightarrow Pierre \parallel r_2 \longrightarrow \dots$
 270 As $Pierre \text{ MUST } r$, there must exist an $i \in \mathbb{N}$ such that $\text{GOOD}(r_i)$. Hence $0 \text{ MUST } r$. ◀

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

$$275 \quad \tau.(\bar{a} \parallel \bar{b}) + \tau.(\bar{a} \parallel \bar{c}) \sqsubseteq_{\text{MUST}} \bar{a} \parallel (\tau.\bar{b} + \tau.\bar{c}) \quad (4)$$

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

278 We conclude this section by recalling auxiliary and rather standard notions: given an LTS
 279 $\langle A, L, \longrightarrow \rangle$, the weak transition relation $p \xRightarrow{s} p'$, where $s \in \text{Act}^*$, is defined via the rules

$$280 \quad [\text{wt-refl}] \quad p \xRightarrow{\varepsilon} p$$

$$281 \quad [\text{wt-tau}] \quad p \xRightarrow{s} q \text{ if } p \xrightarrow{\tau} p' \text{ and } p' \xRightarrow{s} q$$

$$282 \quad [\text{wt-mu}] \quad p \xRightarrow{\mu.s} q \text{ if } p \xrightarrow{\mu} p' \text{ and } p' \xRightarrow{s} q$$

283 We write $p \xRightarrow{s}$ to mean $\exists p'. p \xRightarrow{s} p'$, where $s \in \text{Act}^*$.

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 $\downarrow \subseteq A \times \text{Act}^*$ 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 $p \xrightarrow{\mu} p'$ 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 $\text{rec}x.\tau.x$, in the standard LTS $\langle \text{ACCS}, \longrightarrow, \text{Act}_\tau \rangle$ we have $\Omega \xrightarrow{\tau} \Omega$ and $\neg(\Omega \downarrow)$, while in the LTS $\langle \text{ACCS}, \emptyset, \text{Act}_\tau \rangle$ we have $\Omega \xrightarrow{\tau}$ 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 \preceq_{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 \preceq_{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*: $\mathcal{A}(p, s, \longrightarrow) = \{R(p') \mid p \xrightarrow{s} p' \xrightarrow{\tau}\}$.

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 $\hat{R} \in \mathcal{A}(p, s)$ such that $\hat{R} \subseteq R$. We can now recall the definition of the behavioural preorder à la Hennessy, \preceq_{AS} , which is based on acceptance sets [55].

► **Definition 9.** *We write*

- $p \preceq_{\text{cnv}} q$ whenever $\forall s \in \text{Act}^*. p \downarrow_A s$ implies $q \downarrow_B s$,
- $p \preceq_{\text{acc}} q$ whenever $\forall s \in \text{Act}^*. p \downarrow_A s$ implies $\mathcal{A}(p, s, \longrightarrow_A) \ll \mathcal{A}(q, s, \longrightarrow_B)$,
- $p \preceq_{\text{AS}} q$ whenever $p \preceq_{\text{cnv}} q$ and $p \preceq_{\text{acc}} q$. ■

In the synchronous setting, the behavioural preorder \preceq_{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

XX:10 Constructive characterisations of the must-preorder for asynchrony

326 treat a unique LTS. Castellani and Hennessy [39] show in their Example 4 that the condition
 327 on acceptance sets, *i.e.* \preceq_{acc} , is too demanding in an asynchronous setting.

328 Letting $p = a.0$ and $q = 0$, they show that $p \sqsubset_{\text{MUST}} q$ but $p \not\preceq_{\text{AS}} q$, because $\mathcal{A}(p, \epsilon) = \{\{a\}\}$
 329 and $\mathcal{A}(q, \epsilon) = \{\emptyset\}$, and corresponding to the ready set $\emptyset \in \mathcal{A}(q, \epsilon)$ there is no ready set
 330 $\widehat{R} \in \mathcal{A}(p, \epsilon)$ such that $\widehat{R} \subseteq \emptyset$. Intuitively this is the case because acceptance sets treat inputs
 331 and outputs similarly, while in an asynchronous setting only outputs can be tested.

332 Nevertheless \preceq_{AS} characterises \sqsubset_{MUST} , if servers are enhanced as with forwarding. We now
 333 introduce this concept.

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

$$\begin{array}{ccc}
 \begin{array}{c} \xrightarrow{a} \\ p \quad p' \\ \xleftarrow{\bar{a}} \end{array} & \begin{array}{c} p \xrightarrow{\bar{a}} p' \\ \downarrow a \\ q \end{array} & \Rightarrow p \xrightarrow{\tau} q \text{ or } p = q \\
 \text{INPUT-BOOMERANG} & \text{FWD-FEEDBACK} & (5)
 \end{array}$$

338 The INPUT-BOOMERANG axiom states a kind of input-enabledness property, which is
 339 however more specific as it stipulates that the target state of the input should loop back
 340 to the source state via a complementary output. This is the essence of the behaviour of
 341 a forwarder, whose role is simply to pass on a message and then get back to its original
 342 state. The FWD-FEEDBACK axiom is a weak form of Selinger's FEEDBACK axiom, which
 343 is better understood in conjunction with the INPUT-BOOMERANG axiom: if the sequence
 344 of transitions $p \xrightarrow{\bar{a}} p' \xrightarrow{a} q$ in the FWD-FEEDBACK axiom is taken to be the sequence of
 345 transitions $p' \xrightarrow{\bar{a}} p \xrightarrow{a} p'$ in the INPUT-BOOMERANG axiom, then we see that it must be
 346 $q = p$ in the FWD-FEEDBACK axiom. Moreover, no τ action is issued when moving from p to
 347 q , since no synchronisation occurs in this case: the message is just passed on.

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

351 To prove that \preceq_{AS} is sound and complete with respect to \sqsubset_{MUST} :

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

354 Let MO denote the set of all finite multisets of output actions, for instance we have
 355 $\emptyset, \{\bar{a}\}, \{\bar{a}, \bar{a}\}, \{\bar{a}, \bar{b}\}, \{\bar{a}, \bar{b}, \bar{a}, \bar{b}\} \in MO$. We let M, N, \dots range over MO . The symbol M stands
 356 for *mailbox*. We denote with \uplus the multiset union. We assume a function `mbox` : $A \rightarrow MO$
 357 defined for any LTS \mathcal{L}_A of output-buffered agents such that

- 358 (i) $\bar{a} \in O(p)$ if and only if $\bar{a} \in \text{mbox}(p)$, and
- 359 (ii) for every p' , if $p \xrightarrow{\bar{a}} p'$ then $\text{mbox}(p) = \{\bar{a}\} \uplus \text{mbox}(p')$.

360 Note that by definition `mbox`(p) is a finite multiset.

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

364 **Example 11.** If a calculus is fixed, then the function `FW` may have a simpler definition.
 365 For instance Castellani and Hennessy [39] define it in their calculus `TACCS` by letting $\xrightarrow{\alpha}_{\text{fw}}$

$$\begin{array}{c}
\text{[L-PROC]} \quad \frac{p \xrightarrow{\alpha} p'}{p \triangleright M \xrightarrow{\alpha}_{\text{fw}} p' \triangleright M} \qquad \text{[L-COMM]} \quad \frac{p \xrightarrow{a} p'}{p \triangleright (\{\bar{a}\} \uplus M) \xrightarrow{\tau}_{\text{fw}} p' \triangleright M} \\
\text{[L-MOUT]} \quad \frac{}{p \triangleright (\{\bar{a}\} \uplus M) \xrightarrow{\bar{a}}_{\text{fw}} p \triangleright M} \qquad \text{[L-MINP]} \quad \frac{}{p \triangleright M \xrightarrow{a}_{\text{fw}} p \triangleright (\{\bar{a}\} \uplus M)}
\end{array}$$

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

366 be the least relation over TACCS such that (1) for every $\alpha \in \text{Act}_\tau$, $\xrightarrow{\alpha} \subseteq \xrightarrow{\alpha}_{\text{fw}}$, and (2)
367 for every $a \in \mathcal{N}$, $p \xrightarrow{a}_{\text{fw}} p \parallel \bar{a}$. ◀

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

373 ▶ **Example 12.** As the set \mathcal{N} is countable, every process p in the LTS $\langle \text{ACCS} \times \text{MO}, \text{Act}_\tau, \xrightarrow{\cdot}_{\text{fw}} \rangle$
374 is infinitely-branching, for instance for every p and every input μ we have $p \xrightarrow{\mu} p \parallel \bar{\mu}$, hence
375 $p \xrightarrow{a_0} p \parallel \bar{a}_0$, $p \xrightarrow{a_1} p \parallel \bar{a}_1$, $p \xrightarrow{a_2} p \parallel \bar{a}_2$, ... ◀

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

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

384 ▶ **Lemma 14.** For every $\mathcal{L}_A, \mathcal{L}_B, \mathcal{L}_C \in \text{OF}$, $p \in A, q \in B, r \in C$,

- 385 1. $p \text{ MUST}_i r$ if and only if $\text{FW}(p) \text{ MUST}_i r$,
- 386 2. $p \sqsubseteq_{\text{MUST}} q$ if and only if $\text{FW}(p) \sqsubseteq_{\text{MUST}} \text{FW}(q)$.

We now simplify the definition of acceptance sets to reason on forwarders: for any two
LTS $\mathcal{L}_A, \mathcal{L}_B \in \text{OW}$ and servers $p \in A$, and $q \in B$ we let $\mathcal{A}_{\text{fw}}(p, s, \longrightarrow) = \{O(p') \mid p \xrightarrow{s} p' \xrightarrow{\tau} \}$. 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 \preceq_{acc} of
Definition (9). More formally, for every $\mathcal{L}_A, \mathcal{L}_B \in \text{OW}$ and every $p \in A$ and $q \in B$, we let

$$p \preceq_{\text{acc}}^{\text{fw}} q \text{ iff } \forall s \in \text{Act}^*. p \downarrow s \text{ implies } \mathcal{A}_{\text{fw}}(p, s, \longrightarrow_A) \ll \mathcal{A}_{\text{fw}}(q, s, \longrightarrow_B)$$

387 Then we have the following logical equivalence.

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

389 **Proof.** The *only if* implication is trivial, so we discuss the *if* one. Suppose that $p \preceq_{\text{acc}}^{\text{fw}} q$
390 and that for some s we have that $R \in \mathcal{A}(q, s, \longrightarrow_B)$. Let X be the possibly empty subset
391 of R that contains only output actions. Note that since \mathcal{L}_B is OW we know by definition

XX:12 Constructive characterisations of the must-preorder for asynchrony

392 that $R = X \cup \mathcal{N}$. By definition $X \in \mathcal{A}_{\text{fw}}(q, s, \rightarrow_B)$, and thus by hypothesis there exists
 393 some set of output actions $Y \in \mathcal{A}_{\text{fw}}(p, s, \rightarrow_A)$ such that $Y \subseteq X$. It follows that the set
 394 $Y \cup \mathcal{N} \in \mathcal{A}(p, s, \rightarrow_A)$, and trivially $Y \cup \mathcal{N} \subseteq X \cup \mathcal{N} = R$. \blacktriangleleft

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

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

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

400 **► Theorem 17.** *For every $\mathcal{L}_A, \mathcal{L}_B \in \text{OF}$ and $p \in A, q \in B$, $p \sqsubseteq_{\text{MUST}} q$ if and only if
 401 $\text{FW}(p) \preceq_{\text{AS}} \text{FW}(q)$.*

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

414 From Lemma 5 and Theorem 17 we immediately get a characterisation of $\sqsubseteq_{\text{MUST}}$ for ACCS:
 415

416 **► Corollary 18.** *For every $p, q \in \text{ACCS}_{\exists}$, $p \sqsubseteq_{\text{MUST}} q$ if and only if $\text{FW}(p) \preceq_{\text{AS}} \text{FW}(q)$.*

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

421 3.2 The must-set approach

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

424 For every $X \subseteq_{\text{fin}} \text{Act}$, that is for every finite set of visible actions, with a slightly abuse of
 425 notation we write $p \text{ MUST } X$ whenever $p \xrightarrow{\varepsilon} p'$ implies that $p' \xrightarrow{\mu}$ for some $\mu \in X$, and we
 426 say that X is a MUST-set of p . Let $(p \text{ after } s, \rightarrow) = \{p' \mid p \xrightarrow{s} p'\}$. For every $\mathcal{L}_A, \mathcal{L}_B$ and
 427 $p \in A, q \in B$, let $p \preceq_{\text{M}} q$ whenever $\forall s \in \text{Act}^*$ we have that $p \downarrow s$ implies that $(\forall X \subseteq_{\text{fin}} \text{Act}$ if
 428 $(p \text{ after } s, \rightarrow_A) \text{ MUST } X$ then $(q \text{ after } s, \rightarrow_B) \text{ MUST } X$).

429 **► Definition 19.** *For every $\mathcal{L}_A, \mathcal{L}_B \in \text{OF}$, and server $p \in A$ and $q \in B$ we let $p \preceq_{\text{MS}} q$
 430 whenever $p \preceq_{\text{cnv}} q \wedge p \preceq_{\text{M}} q$. \blacksquare*

431 **► Lemma 20.** *Let $\mathcal{L}_A, \mathcal{L}_B \in \text{OF}$. For every $p \in A, q \in B$ such that $\text{FW}(p) \preceq_{\text{cnv}} \text{FW}(q)$, we
 432 have that $\text{FW}(p) \preceq_{\text{M}} \text{FW}(q)$ if and only if $\text{FW}(p) \preceq_{\text{acc}}^{\text{fw}} \text{FW}(q)$.*

433 As a direct consequence, we obtain our second result.

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

436 **Failure refinement.** MUST-sets have been used mainly by De Nicola and collaborators,
 437 for instance in [45, 24], and are closely related to the failure refinement proposed in [34] by
 438 Hoare, Brookes and Roscoe for TCSP (the process algebra based on Hoare’s language CSP
 439 [63, 32]). Following [34], a *failure* of a process p is a pair (s, X) such that $p \xrightarrow{s} p'$ and $p' \not\xrightarrow{\mu}$
 440 for all $\mu \in X$. Then, failure refinement is defined by letting $p \leq_{fail} q$ whenever the failures
 441 of q are also failures of p . This refinement was designed to give a denotational semantics
 442 to processes, and mechanisations in Isabelle/HOL have been developed to ensure that the
 443 refinement is well defined [93, 12]. Both Hennessy [55, pag. 260] and [38] highlight that the
 444 failure model can be justified operationally via the MUST testing equivalence: it is folklore
 445 dating back to [44, Section 4] that failure equivalence and \approx coincide. Thanks to Theorem 21
 446 we conclude that in fact \sqsubseteq_{MUST} coincides with \leq_{fail} in conjunction with \preceq_{cnv} .⁸

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

449 **4 Bar-induction: from extensional to intensional definitions**

450 Two predicates are crucial to reason on the MUST-preorder, namely passing a test, *i.e.* MUST,
 451 and convergence, *i.e.* \downarrow . Both predicates are defined in an *extensional* manner, *i.e.* by
 452 requiring that for every infinite sequence there exists a state that is in some sense good.
 453 These are respectively the predicate GOOD in the definition of MUST and the predicate of
 454 stability, *i.e.* $\dashv\rightarrow$, 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:

$$[\text{AXIOM}] \frac{Q(s)}{\text{int}_Q(s)} \quad [\text{IND-RULE}] \frac{s \rightarrow \quad \forall s'. s \rightarrow s' \text{ implies } \text{int}_Q(s')}{\text{int}_Q(s)}$$

455 and we define our inductive predicates via int by letting $p \downarrow_i \stackrel{\text{def}}{=} \text{int}_{Q_1}(p)$ and $p \text{MUST}_i r \stackrel{\text{def}}{=} \text{int}_{Q_2}(p, r)$,
 456 where $Q_1(p) \stackrel{\text{def}}{=} p \dashv\rightarrow$ and $Q_2(p, r) \stackrel{\text{def}}{=} \text{GOOD}(r)$.

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

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

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

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

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

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

471 5 Conclusion

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

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

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

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

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

510 **Related work.** Appendix G contains a detailed discussion of related works. Here we
 511 highlight that the notion of forwarder was outlined in the original paper on testing-preorders

512 for asynchrony [39], and then used in the saturated LTS of [8] to reason on the MAY-preorder,
 513 and in [95] to reason on a version of the MUST-preorder parametrised on the set of tests.
 514 Forwarders, also called “links”, have applications outside of testing theory, as shown by [73]
 515 and the recent [49]. Characterising $\sqsubseteq_{\text{MUST}}$ directly on LTSs instead of calculi was suggested
 516 already in [55, 13]. Selinger axioms, discussed also by [9], are crucial in our completeness
 517 proof. Brouwer bar-induction principle is paramount to prove soundness constructively.

518 ——— References ———

- 519 **1** Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfssdóttir, and Karoliina Lehtinen.
 520 Adventures in monitorability: from branching to linear time and back again. *Proc. ACM*
 521 *Program. Lang.*, 3(POPL):52:1–52:29, 2019. doi:10.1145/3290365.
- 522 **2** Luca Aceto and Matthew Hennessy. Termination, Deadlock, and Divergence. *J. ACM*,
 523 39(1):147–187, 1992. doi:10.1145/147508.147527.
- 524 **3** Reynald Affeldt and Naoki Kobayashi. A Coq Library for Verification of Concurrent Programs.
 525 In Carsten Schürmann, editor, *Proceedings of the Fourth International Workshop on Logical*
 526 *Frameworks and Meta-Languages, LFM@IJCAR 2004. Cork, Ireland, July 5, 2004*, volume
 527 199 of *Electronic Notes in Theoretical Computer Science*, pages 17–32. Elsevier, 2004. URL:
 528 <https://doi.org/10.1016/j.entcs.2007.11.010>, doi:10.1016/J.ENTCS.2007.11.010.
- 529 **4** Roberto M. Amadio, Iliaria Castellani, and Davide Sangiorgi. On Bisimulations for the
 530 Asynchronous pi-calculus. In U. Montanari and V. Sassone, editors, *Proceedings CONCUR*
 531 *96*, Pisa, volume 1119 of *Lecture Notes in Computer Science*, pages 147–162. Springer Verlag,
 532 1996.
- 533 **5** Roberto M. Amadio, Iliaria Castellani, and Davide Sangiorgi. On Bisimulations for the
 534 Asynchronous pi-calculus. *Theoretical Computer Science*, 195:291–324, 1998.
- 535 **6** Anish Athalye. CoqIOA: A formalization of IO Automata in the Coq Proof Assistant. Master’s
 536 thesis, Massachusetts Institute of Technology, June 2017.
- 537 **7** Clément Aubert and Daniele Varacca. Processes against tests: On defining contextual
 538 equivalences. *J. Log. Algebraic Methods Program.*, 129:100799, 2022. URL: [https://doi.org/](https://doi.org/10.1016/j.jlamp.2022.100799)
 539 [10.1016/j.jlamp.2022.100799](https://doi.org/10.1016/j.jlamp.2022.100799), doi:10.1016/J.JLAMP.2022.100799.
- 540 **8** Paolo Baldan, Filippo Bonchi, Fabio Gadducci, and Giacoma Valentina Monreale. Asyn-
 541 chronous Traces and Open Petri Nets. In Chiara Bodei, Gian-Luigi Ferrari, and Corrado Priami,
 542 editors, *Programming Languages with Applications to Biology and Security - Essays Dedicated*
 543 *to Pierpaolo Degano on the Occasion of His 65th Birthday*, volume 9465 of *Lecture Notes in*
 544 *Computer Science*, pages 86–102. Springer, 2015. doi:10.1007/978-3-319-25527-9_8.
- 545 **9** Paolo Baldan, Filippo Bonchi, Fabio Gadducci, and Giacoma Valentina Monreale. Concurrency
 546 cannot be observed, asynchronously. *Math. Struct. Comput. Sci.*, 25(4):978–1004, 2015.
 547 doi:10.1017/S0960129513000108.
- 548 **10** Franco Barbanera and Ugo de’Liguoro. Two notions of sub-behaviour for session-based
 549 client/server systems. In Temur Kutsia, Wolfgang Schreiner, and Maribel Fernández, editors,
 550 *Proceedings of the 12th International ACM SIGPLAN Conference on Principles and Practice*
 551 *of Declarative Programming, July 26-28, 2010, Hagenberg, Austria*, pages 155–164. ACM, 2010.
 552 doi:10.1145/1836089.1836109.
- 553 **11** Henk Barendregt and Giulio Manzonetto. *A Lambda Calculus Satellite*. College Publications,
 554 2022. Chapter 11. URL: <https://www.collegepublications.co.uk/logic/mlf/?00035>.
- 555 **12** James Baxter, Pedro Ribeiro, and Ana Cavalcanti. Sound reasoning in tock-CSP. *Acta*
 556 *Informatica*, 59(1):125–162, 2022. doi:10.1007/s00236-020-00394-3.
- 557 **13** Giovanni Bernardi. *Behavioural equivalences for Web services*. PhD thesis, Trinity College
 558 Dublin, 2013. URL: <http://www.tara.tcd.ie/handle/2262/77595>.
- 559 **14** Giovanni Bernardi and Matthew Hennessy. Mutually Testing Processes. *Log. Methods Comput.*
 560 *Sci.*, 11(2), 2015. doi:10.2168/LMCS-11(2:1)2015.

- 561 15 Giovanni Bernardi and Matthew Hennessy. Using higher-order contracts to model session
562 types. *Log. Methods Comput. Sci.*, 12(2), 2016. doi:10.2168/LMCS-12(2:10)2016.
- 563 16 Giovanni Tito Bernardi and Adrian Francalanza. Full-abstraction for client testing preorders.
564 *Sci. Comput. Program.*, 168:94–117, 2018. doi:10.1016/j.scico.2018.08.004.
- 565 17 Giovanni Tito Bernardi and Matthew Hennessy. Modelling session types using contracts. *Math.*
566 *Struct. Comput. Sci.*, 26(3):510–560, 2016. doi:10.1017/S0960129514000243.
- 567 18 Gérard Berry and Gérard Boudol. The Chemical Abstract Machine. *Theor. Comput. Sci.*,
568 96(1):217–248, 1992. doi:10.1016/0304-3975(92)90185-I.
- 569 19 Aleš Bizjak, Lars Birkedal, and Marino Miculan. A model of countable nondeterminism in
570 guarded type theory. In Gilles Dowek, editor, *Rewriting and Typed Lambda Calculi*, pages
571 108–123, Cham, 2014. Springer International Publishing.
- 572 20 Filippo Bonchi, Georgiana Caltais, Damien Pous, and Alexandra Silva. Brzozowski’s and
573 Up-To Algorithms for Must Testing. In Chung-chieh Shan, editor, *Programming Languages and*
574 *Systems - 11th Asian Symposium, APLAS 2013, Melbourne, VIC, Australia, December 9-11,*
575 *2013. Proceedings*, volume 8301 of *Lecture Notes in Computer Science*, pages 1–16. Springer,
576 2013. doi:10.1007/978-3-319-03542-0\1.
- 577 21 Filippo Bonchi, Ana Sokolova, and Valeria Vignudelli. The Theory of Traces for Systems with
578 Nondeterminism, Probability, and Termination. *Log. Methods Comput. Sci.*, 18(2), 2022. URL:
579 [https://doi.org/10.46298/lmcs-18\(2:21\)2022](https://doi.org/10.46298/lmcs-18(2:21)2022), doi:10.46298/LMCS-18(2:21)2022.
- 580 22 Michele Boreale and Fabio Gadducci. Processes as formal power series: A coinductive approach
581 to denotational semantics. *Theor. Comput. Sci.*, 2006. doi:10.1016/j.tcs.2006.05.030.
- 582 23 Michele Boreale and Rocco De Nicola. Testing Equivalence for Mobile Processes. *Inf. Comput.*,
583 120(2):279–303, 1995. doi:10.1006/inco.1995.1114.
- 584 24 Michele Boreale, Rocco De Nicola, and Rosario Pugliese. Trace and Testing Equivalence on
585 Asynchronous Processes. *Inf. Comput.*, 172(2):139–164, 2002. doi:10.1006/inco.2001.3080.
- 586 25 Gérard Boudol. Asynchrony and the Pi-calculus. Research Report RR-1702, INRIA, 1992.
587 URL: <https://hal.inria.fr/inria-00076939>.
- 588 26 Gérard Boudol. The pi-Calculus in Direct Style. *High. Order Symb. Comput.*, 11(2):177–208,
589 1998. doi:10.1023/A:1010064516533.
- 590 27 Gérard Boudol and Carolina Lavatelli. Full Abstraction for Lambda Calculus with Resources
591 and Convergence Testing. In Hélène Kirchner, editor, *Trees in Algebra and Programming -*
592 *CAAP’96, 21st International Colloquium, Linköping, Sweden, April, 22-24, 1996, Proceedings*,
593 volume 1059 of *Lecture Notes in Computer Science*, pages 302–316. Springer, 1996. doi:
594 10.1007/3-540-61064-2\45.
- 595 28 M. Bravetti, J. Lange, and G. Zavattaro. Fair Refinement for Asynchronous Session Types. In
596 *FOSSACS*, 2021.
- 597 29 Nuria Brede and Hugo Herbelin. On the logical structure of choice and bar induction principles.
598 In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy,*
599 *June 29 - July 2, 2021*, pages 1–13. IEEE, 2021. doi:10.1109/LICS52264.2021.9470523.
- 600 30 Flavien Breuvert, Giulio Manzonetto, Andrew Polonsky, and Domenico Ruoppolo. New
601 Results on Morris’s Observational Theory: The Benefits of Separating the Inseparable. In
602 Delia Kesner and Brigitte Pientka, editors, *1st International Conference on Formal Structures*
603 *for Computation and Deduction, FSCD 2016, June 22-26, 2016, Porto, Portugal*, volume 52
604 of *LIPICs*, pages 15:1–15:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:
605 10.4230/LIPICs.FSCD.2016.15.
- 606 31 Flavien Breuvert, Giulio Manzonetto, and Domenico Ruoppolo. Relational Graph Models at
607 Work. *Log. Methods Comput. Sci.*, 14(3), 2018. doi:10.23638/LMCS-14(3:2)2018.
- 608 32 Stephen D. Brookes. On the Relationship of CCS and CSP. In Josep Díaz, editor, *Automata,*
609 *Languages and Programming, 10th Colloquium, Barcelona, Spain, July 18-22, 1983, Proceedings*,
610 volume 154 of *Lecture Notes in Computer Science*, pages 83–96. Springer, 1983. doi:10.1007/
611 BFB0036899.

- 612 33 Stephen D. Brookes. Deconstructing CCS and CSP Asynchronous Communication, Fairness,
613 and Full Abstraction. 2002.
- 614 34 Stephen D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A Theory of Communicating
615 Sequential Processes. *J. ACM*, 31(3):560–599, 1984. doi:10.1145/828.833.
- 616 35 Caroline Caruana. Compositional Reasoning about Actor Based Systems, 2019.
- 617 36 Giuseppe Castagna, Mariangiola Dezani-Ciancaglini, Elena Giachino, and Luca Padovani.
618 Foundations of session types. In António Porto and Francisco Javier López-Fraguas, editors,
619 *Proceedings of the 11th International ACM SIGPLAN Conference on Principles and Practice*
620 *of Declarative Programming, September 7-9, 2009, Coimbra, Portugal*, pages 219–230. ACM,
621 2009. doi:10.1145/1599410.1599437.
- 622 37 Giuseppe Castagna, Nils Gesbert, and Luca Padovani. A theory of contracts for Web services.
623 *ACM Trans. Program. Lang. Syst.*, 31(5):19:1–19:61, 2009. doi:10.1145/1538917.1538920.
- 624 38 Simon Castellan, Pierre Clairambault, and Glynn Winskel. The Mays and Musts of Concurrent
625 Strategies. In Alessandra Palmigiano and Mehrnoosh Sadrzadeh, editors, *Samson Abramsky on*
626 *Logic and Structure in Computer Science and Beyond*, pages 327–361, Cham, 2023. Springer
627 International Publishing. doi:10.1007/978-3-031-24117-8_9.
- 628 39 Ilaria Castellani and Matthew Hennessy. Testing Theories for Asynchronous Languages. In
629 Vikraman Arvind and Ramaswamy Ramanujam, editors, *Foundations of Software Technology*
630 *and Theoretical Computer Science, 18th Conference, Chennai, India, December 17-19, 1998,*
631 *Proceedings*, volume 1530 of *Lecture Notes in Computer Science*, pages 90–101. Springer, 1998.
632 doi:10.1007/978-3-540-49382-2_9.
- 633 40 Andrea Cerone and Matthew Hennessy. Process Behaviour: Formulae vs. Tests. Technical
634 report, Trinity College Dublin, School of Computer Science and Statistics, 2010.
- 635 41 R. Cleaveland and A. E. Zwarico. A Theory of Testing for Real-Time. In *LICS*, 1991.
- 636 42 Rance Cleaveland and Matthew Hennessy. Testing Equivalence as a Bisimulation Equivalence.
637 In Joseph Sifakis, editor, *Automatic Verification Methods for Finite State Systems, Interna-*
638 *tional Workshop, Grenoble, France, June 12-14, 1989, Proceedings*, volume 407 of *Lecture*
639 *Notes in Computer Science*, pages 11–23. Springer, 1989. doi:10.1007/3-540-52148-8_2.
- 640 43 T. Coquand. About Brouwer’s Fan Theorem. [https://www.cairn-int.info/
641 journal-revue-internationale-de-philosophie-2004-4-page-483.htm](https://www.cairn-int.info/journal-revue-internationale-de-philosophie-2004-4-page-483.htm), 2003.
- 642 44 Rocco De Nicola and Matthew Hennessy. Testing Equivalences for Processes. *Theor. Comput.*
643 *Sci.*, 34:83–133, 1984. doi:10.1016/0304-3975(84)90113-0.
- 644 45 Rocco De Nicola and Hernan C. Melgratti. Multiparty testing preorders. *Log. Methods Comput.*
645 *Sci.*, 19(1), 2023. doi:10.46298/lmcs-19(1:1)2023.
- 646 46 Rocco De Nicola and Rosario Pugliese. Linda-based applicative and imperative process algebras.
647 *Theor. Comput. Sci.*, 238(1-2):389–437, 2000. doi:10.1016/S0304-3975(99)00339-4.
- 648 47 D. Dreyer, G. Neis, and L. Birkedal. The impact of higher-order state and control effects
649 on local relational reasoning. *J. Funct. Program.*, 22(4-5):477–528, 2012. doi:10.1017/
650 S095679681200024X.
- 651 48 Michael Dummett. *Elements of Intuitionism*. Oxford logic guides. Clarendon Press, 2000.
652 URL: <https://books.google.fr/books?id=JVFzknGBVAC>.
- 653 49 Adrien Durier, Daniel Hirschhoff, and Davide Sangiorgi. Eager Functions as Processes.
654 *Theor. Comput. Sci.*, 913:8–42, 2022. URL: <https://doi.org/10.1016/j.tcs.2022.01.043>,
655 doi:10.1016/J.TCS.2022.01.043.
- 656 50 Adrian Francalanza. A theory of monitors. *Inf. Comput.*, 281:104704, 2021. doi:10.1016/j.
657 ic.2021.104704.
- 658 51 D. Fridlender. An Interpretation of the Fan Theorem in Type Theory. In *TYPES*, 1998. URL:
659 https://doi.org/10.1007/3-540-48167-2_7.
- 660 52 Dan Frumin, Robbert Krebbers, and Lars Birkedal. ReLoC: A Mechanised Relational Logic
661 for Fine-Grained Concurrency. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the*
662 *33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK,*
663 *July 09-12, 2018*, pages 442–451. ACM, 2018. doi:10.1145/3209108.3209174.

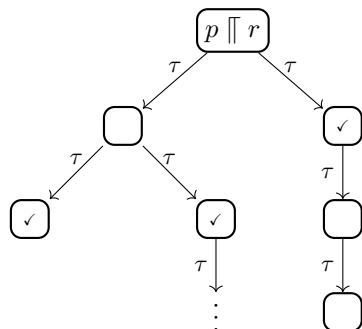
- 664 **53** Simon J. Gay and Malcolm Hole. Subtyping for session types in the pi calculus. *Acta*
665 *Informatica*, 42(2-3):191–225, 2005. doi:10.1007/s00236-005-0177-z.
- 666 **54** Matthew Hennessy. Acceptance Trees. *J. ACM*, 32(4):896–928, 1985. doi:10.1145/4221.4249.
- 667 **55** Matthew Hennessy. *Algebraic theory of processes*. MIT Press series in the foundations of
668 computing. MIT Press, 1988.
- 669 **56** Matthew Hennessy. A fully abstract denotational semantics for the pi-calculus. *Theor. Comput.*
670 *Sci.*, 278(1-2):53–89, 2002. doi:10.1016/S0304-3975(00)00331-5.
- 671 **57** Matthew Hennessy. The security pi-calculus and non-interference. *J. Log. Algebraic Methods*
672 *Program.*, 2005. doi:10.1016/j.jlap.2004.01.003.
- 673 **58** Matthew Hennessy. *A distributed Pi-calculus*. Cambridge University Press, 2007.
- 674 **59** Matthew Hennessy and Anna Ingólfssdóttir. Communicating Processes with Value-passing and
675 Assignments. *Formal Aspects Comput.*, 5(5):432–466, 1993. doi:10.1007/BF01212486.
- 676 **60** Matthew Hennessy and Gordon D. Plotkin. A Term Model for CCS. In Piotr Dembinski,
677 editor, *Mathematical Foundations of Computer Science 1980 (MFCS'80), Proceedings of*
678 *the 9th Symposium, Rydzyna, Poland, September 1-5, 1980*, volume 88 of *Lecture Notes in*
679 *Computer Science*, pages 261–274. Springer, 1980. doi:10.1007/BFb0022510.
- 680 **61** Matthew Hennessy and James Riely. Information flow vs. resource access in the asynchronous
681 pi-calculus. *ACM Trans. Program. Lang. Syst.*, 24(5):566–591, 2002. doi:10.1145/570886.
682 570890.
- 683 **62** Daniel Hirschhoff, Guilhem Jaber, and Enguerrand Prebet. Deciding Contextual Equivalence
684 of ν -Calculus with Effectful Contexts. In Orna Kupferman and Pawel Sobocinski, editors,
685 *Foundations of Software Science and Computation Structures - 26th International Conference,*
686 *FoSSaCS 2023, Held as Part of the European Joint Conferences on Theory and Practice of*
687 *Software, ETAPS 2023, Paris, France, April 22-27, 2023, Proceedings*, volume 13992 of *Lecture*
688 *Notes in Computer Science*, pages 24–45. Springer, 2023. doi:10.1007/978-3-031-30829-1\
689 _2.
- 690 **63** C. A. R. Hoare. Communicating Sequential Processes (Reprint). *Commun. ACM*, 1983.
- 691 **64** Kohei Honda and Mario Tokoro. An Object Calculus for Asynchronous Communication. In
692 Pierre America, editor, *ECOOP'91 European Conference on Object-Oriented Programming,*
693 *Geneva, Switzerland, July 15-19, 1991, Proceedings*, volume 512 of *Lecture Notes in Computer*
694 *Science*, pages 133–147. Springer, 1991. doi:10.1007/BFb0057019.
- 695 **65** Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty Asynchronous Session Types.
696 In George C. Necula and Philip Wadler, editors, *POPL*, pages 273–284, New York, 2008. ACM
697 Press.
- 698 **66** Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty Asynchronous Session Types.
699 *Journal of ACM*, 63(1):9:1–9:67, 2016.
- 700 **67** Benedetto Intrigila, Giulio Manzonetto, and Andrew Polonsky. Degrees of extensionality in
701 the theory of Böhm trees and Sallé’s conjecture. *Log. Methods Comput. Sci.*, 15(1), 2019.
702 doi:10.23638/LMCS-15(1:6)2019.
- 703 **68** S. C. Kleene and R. E. Vesley. *The Foundations of Intuitionistic Mathematics: Especially*
704 *in Relation to Recursive Functions*. Studies in logic and the foundations of mathematics.
705 North-Holland Publishing Company, 1965. URL: [https://books.google.fr/books?id=](https://books.google.fr/books?id=2EHVxQEACAAJ)
706 [2EHVxQEACAAJ](https://books.google.fr/books?id=2EHVxQEACAAJ).
- 707 **69** Vasileios Koutavas and Nikos Tzevelekos. Fully Abstract Normal Form Bisimulation for
708 Call-by-Value PCF. In *LICS*, 2023.
- 709 **70** Robbert Krebbers, Amin Timany, and Lars Birkedal. Interactive proofs in higher-order
710 concurrent separation logic. In Giuseppe Castagna and Andrew D. Gordon, editors, *Proceedings*
711 *of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL*
712 *2017, Paris, France, January 18-20, 2017*, pages 205–217. ACM, 2017. doi:10.1145/3009837.
713 3009855.

- 714 **71** Leslie Lamport. *Specifying Systems, The TLA+ Language and Tools for Hardware and Software*
715 *Engineers*. Addison-Wesley, 2002. URL: [http://research.microsoft.com/users/lamport/](http://research.microsoft.com/users/lamport/tla/book.html)
716 [tla/book.html](http://research.microsoft.com/users/lamport/tla/book.html).
- 717 **72** Cosimo Laneve and Luca Padovani. The *Must* Preorder Revisited. In Luís Caires and
718 Vasco Thudichum Vasconcelos, editors, *CONCUR 2007 - Concurrency Theory, 18th In-*
719 *ternational Conference, CONCUR 2007, Lisbon, Portugal, September 3-8, 2007, Proceed-*
720 *ings*, volume 4703 of *Lecture Notes in Computer Science*, pages 212–225. Springer, 2007.
721 doi:10.1007/978-3-540-74407-8_15.
- 722 **73** Massimo Merro and Davide Sangiorgi. On asynchrony in name-passing calculi. *Math. Struct.*
723 *Comput. Sci.*, 14(5):715–767, 2004. doi:10.1017/S0960129504004323.
- 724 **74** Robin Milner. Functions as Processes. In Mike Paterson, editor, *Automata, Languages and*
725 *Programming, 17th International Colloquium, ICALP90, Warwick University, England, UK,*
726 *July 16-20, 1990, Proceedings*, volume 443 of *Lecture Notes in Computer Science*, pages
727 167–180. Springer, 1990. doi:10.1007/BFb0032030.
- 728 **75** Robin Milner. *Communicating and Mobile Systems - the Pi-Calculus*. Cambridge University
729 Press, 1999.
- 730 **76** James H. Morris. *Lambda-calculus models of programming languages*. PhD thesis, Massachusetts
731 Institute of Technology, 1969. URL: <https://dspace.mit.edu/handle/1721.1/64850>.
- 732 **77** Keiko Nakata, Tarmo Uustalu, and Marc Bezem. A Proof Pearl with the Fan Theorem and
733 Bar Induction - Walking through Infinite Trees with Mixed Induction and Coinduction. In
734 Hongseok Yang, editor, *Programming Languages and Systems - 9th Asian Symposium, APLAS*
735 *2011, Kenting, Taiwan, December 5-7, 2011. Proceedings*, volume 7078 of *Lecture Notes in*
736 *Computer Science*, pages 353–368. Springer, 2011. doi:10.1007/978-3-642-25318-8_26.
- 737 **78** Rocco De Nicola and Matthew Hennessy. CCS without tau's. In Hartmut Ehrig, Robert A.
738 Kowalski, Giorgio Levi, and Ugo Montanari, editors, *TAPSOFT'87: Proceedings of the*
739 *International Joint Conference on Theory and Practice of Software Development, Pisa, Italy,*
740 *March 23-27, 1987, Volume 1: Advanced Seminar on Foundations of Innovative Software*
741 *Development I and Colloquium on Trees in Algebra and Programming (CAAP'87)*, volume
742 249 of *Lecture Notes in Computer Science*, pages 138–152. Springer, 1987. doi:10.1007/
743 3-540-17660-8_53.
- 744 **79** Catuscia Palamidessi. Comparing the Expressive Power of the Synchronous and Asynchronous
745 *pi*-calculi. *Mathematical Structures in Computer Science*, 13(5):685–719, 2003. doi:10.1017/
746 S0960129503004043.
- 747 **80** Damien Pous. Coinduction All the Way Up. In Martin Grohe, Eric Koskinen, and Natarajan
748 Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer*
749 *Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 307–316. ACM, 2016. doi:
750 10.1145/2933575.2934564.
- 751 **81** Enguerrand Prebet. Functions and References in the Pi-Calculus: Full Abstraction and
752 Proof Techniques. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors,
753 *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022,*
754 *July 4-8, 2022, Paris, France*, volume 229 of *LIPICs*, pages 130:1–130:19. Schloss Dagstuhl -
755 Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.ICALP.2022.130.
- 756 **82** R. Pugliese. *Semantic Theories for Asynchronous Languages*. PhD thesis, Università di Roma
757 "La Sapienza", 1996.
- 758 **83** Vincent Rahli, Mark Bickford, Liron Cohen, and Robert L. Constable. Bar Induction is
759 Compatible with Constructive Type Theory. *J. ACM*, 66(2):13:1–13:35, 2019. doi:10.1145/
760 3305261.
- 761 **84** António Ravara, Pedro Resende, and Vasco Thudichum Vasconcelos. An Algebra of Behavioural
762 Types. *Inf. Comput.*, 212:64–91, 2012.
- 763 **85** Arend Rensink and Walter Vogler. Fair testing. *Inf. Comput.*, 205(2):125–198, 2007. doi:
764 10.1016/j.ic.2006.06.002.

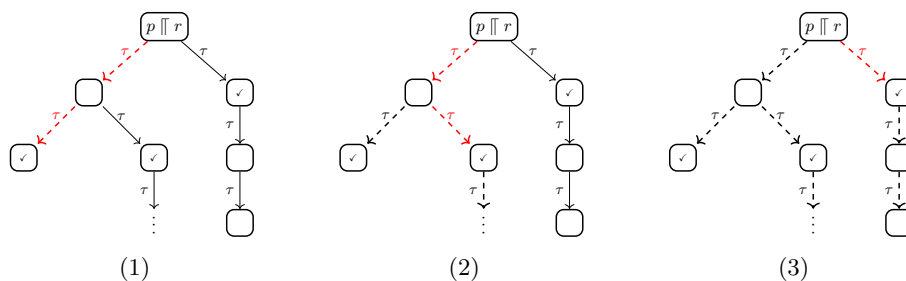
- 765 **86** David Sabel and Manfred Schmidt-Schauß. A call-by-need lambda calculus with locally
766 bottom-avoiding choice: context lemma and correctness of transformations. *Math. Struct.*
767 *Comput. Sci.*, 18(3):501–553, 2008. doi:10.1017/S0960129508006774.
- 768 **87** Davide Sangiorgi. *Introduction to Bisimulation and Coinduction*. Cambridge University Press,
769 2011.
- 770 **88** Davide Sangiorgi. Asynchronous pi-calculus at Work: The Call-by-Need Strategy. In Mário S.
771 Alvim, Kostas Chatzikokolakis, Carlos Olarte, and Frank Valencia, editors, *The Art of*
772 *Modelling Computational Systems: A Journey from Logic and Concurrency to Security and*
773 *Privacy - Essays Dedicated to Catuscia Palamidessi on the Occasion of Her 60th Birthday*,
774 volume 11760 of *Lecture Notes in Computer Science*, pages 33–49. Springer, 2019. doi:
775 10.1007/978-3-030-31175-9_3.
- 776 **89** Davide Sangiorgi and David Walker. *The Pi-Calculus - a Theory of Mobile Processes*. Cam-
777 bridge University Press, 2001.
- 778 **90** Manfred Schmidt-Schauß and David Sabel. Correctly Implementing Synchronous Message
779 Passing in the Pi-Calculus By Concurrent Haskell’s MVars. In Ornela Dardha and Jurriaan Rot,
780 editors, *Proceedings Combined 27th International Workshop on Expressiveness in Concurrency*
781 *and 17th Workshop on Structural Operational Semantics, EXPRESS/SOS 2020, and 17th*
782 *Workshop on Structural Operational SemanticsOnline, 31 August 2020*, volume 322 of *EPTCS*,
783 pages 88–105, 2020. doi:10.4204/EPTCS.322.8.
- 784 **91** Manfred Schmidt-Schauß, David Sabel, and Nils Dallmeyer. Sequential and Parallel Im-
785 improvements in a Concurrent Functional Programming Language. In David Sabel and Peter
786 Thiemann, editors, *Proceedings of the 20th International Symposium on Principles and Practice*
787 *of Declarative Programming, PPDP 2018, Frankfurt am Main, Germany, September 03-05,*
788 *2018*, pages 20:1–20:13. ACM, 2018. doi:10.1145/3236950.3236952.
- 789 **92** Peter Selinger. First-Order Axioms for Asynchrony. In Antoni W. Mazurkiewicz and Józef
790 Winkowski, editors, *CONCUR ’97: Concurrency Theory, 8th International Conference, War-*
791 *saw, Poland, July 1-4, 1997, Proceedings*, volume 1243 of *Lecture Notes in Computer Science*,
792 pages 376–390. Springer, 1997. doi:10.1007/3-540-63141-0_26.
- 793 **93** S. Taha, L. Ye, and B. Wolff. HOL-CSP Version 2.0. *Archive of Formal Proofs*, April 2019.
- 794 **94** Erica Tanti and Adrian Francalanza. Towards Sound Refactoring in Erlang. 2015. URL:
795 <https://api.semanticscholar.org/CorpusID:63046364>.
- 796 **95** Prasanna Thati. *A Theory of Testing for Asynchronous Concurrent Systems*. PhD thesis,
797 University of Illinois Urbana-Champaign, USA, 2003. URL: [https://hdl.handle.net/2142/](https://hdl.handle.net/2142/81630)
798 [81630](https://hdl.handle.net/2142/81630).
- 799 **96** Aaron Joseph Turon, Jacob Thamsborg, Amal Ahmed, Lars Birkedal, and Derek Dreyer.
800 Logical Relations for Fine-Grained Concurrency. In Roberto Giacobazzi and Radhia Cousot,
801 editors, *The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming*
802 *Languages, POPL ’13, Rome, Italy - January 23 - 25, 2013*, pages 343–356. ACM, 2013.
803 doi:10.1145/2429069.2429111.
- 804 **97** Rob van Glabbeek. Just Testing. In Orna Kupferman and Pawel Sobocinski, editors, *Founda-*
805 *tions of Software Science and Computation Structures - 26th International Conference, FoS-*
806 *SaCS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software,*
807 *ETAPS 2023, Paris, France, April 22-27, 2023, Proceedings*, volume 13992 of *Lecture Notes in*
808 *Computer Science*, pages 498–519. Springer, 2023. doi:10.1007/978-3-031-30829-1_24.

809 **A** Bar-Induction

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



■ **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.

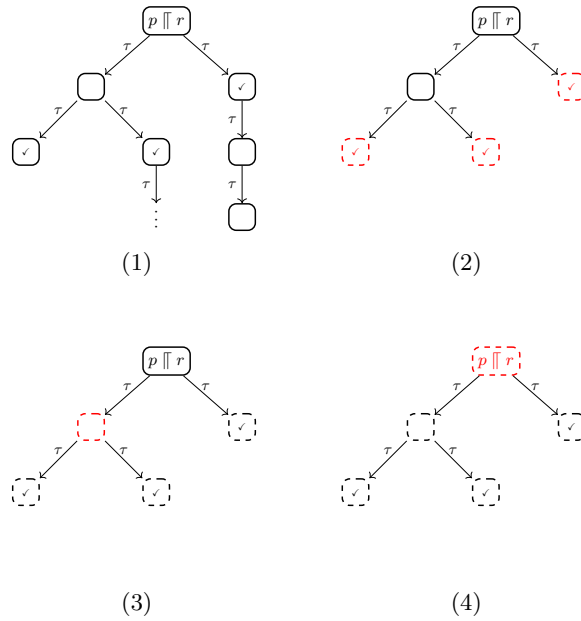
813 A.1 A visual introduction

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

816 Suppose that we have a client-server system $p \parallel r$ and that we want to prove either $p \text{ MUST } r$
817 or $p \text{ MUST}_i r$. For both proofs, what matters is the state transition system (STS) of $p \parallel r$, i.e.
818 the computation steps performed by the client-server system at issue. In fact it is customary
819 to treat this STS as a computational tree, as done for instance in the proofs of [55, Lemma
820 4.4.12] and [40, Theorem 2.3.3]. In the rest of this subsection we discuss the tree depicted
821 in Figure 7. It contains three maximal computations, the middle one being infinite. In the
822 figures of this subsection, the states in which the client is successful (i.e. in the predicate
823 GOOD) contain the symbol ✓.

824 A.1.0.1 The extensional approach

825 To prove $p \text{ MUST } r$, the extensional definition of MUST requires checking that every maximal
826 path in the tree in Figure 7 starts with a finite prefix that leads to a successful state. The
827 proof that $p \text{ MUST } r$ amounts to looking for a suitable prefix maximal path by maximal path,
828 via a loop whose iterations are suggested in Figure 8. There at every iteration a different
829 maximal path (highlighted by dashed arrows) is checked, and each time a successful prefix
830 is found (indicated by a red arrow), the loop moves on to the next maximal path. Once



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

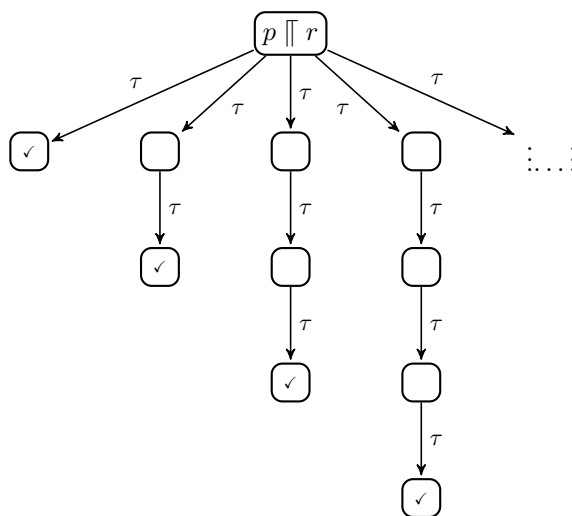
831 a maximal path is explored, it remains dashed, to denote that there a successful prefix has
 832 been found. The first iteration looks for a successful prefix in the left-most maximal path,
 833 while the last iteration looks for a successful prefix in the right-most path. In the current
 834 example the loop terminates because the tree in Figure 7 has conveniently a finite number
 835 of maximal paths, but in general the mathematical reasoning has to deal with an infinite
 836 amount of maximal path. An archetypal example is the tree in Figure 10: it has countably
 837 many maximal paths, each one starting with a successful prefix.

838 **A.1.0.2 The intensional approach**

839 Consider now the predicate $MUST_i$ - which is defined intensional ly - and a proof that
 840 $p MUST_i r$. The base case of $MUST_i$ ensures that all the nodes that contain a successful client
 841 (i.e. that satisfies the predicate Q_2 , defined on line 553 of the submission) are in $MUST_i$.
 842 Pictorially, this is the step from (1) to (2) in Figure 9, where the nodes in $MUST_i$ are drawn
 843 using dashed borders, and the freshly added ones are drawn in red. Once the base case is
 844 established, the inductive rule of $MUST_i$ ensures that any node that inevitably goes to nodes
 845 that are in $MUST_i$, is also in the predicate $MUST_i$. This leads to the step from (2) to (3) and
 846 then from (3) to (4). Note that the argument is concise, for in the tree the depth at which
 847 successful states can be found is finite. In general though is may not be the case. The tree in
 848 Figure 10 is again the archetypal example: every maximal path there contains a finite prefix
 849 that leads to a successful state, but there is no upper bound on the length on those prefixes.

850 **A.1.0.3 Do extensional and intensional predicates coincide ?**

851 Extensional and intensional definitions make us reason on computational trees in strikingly
 852 different fashions: *extensionally* we reason maximal path by maximal path, while *intensional*



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

853 *ly* we reason bottom-up, starting from the nodes in a predicate that bars the tree.⁹ It is
 854 natural to ask whether reasoning in these different manners ultimately leads to the same
 855 outcomes. In our setting this amounts to proving that the predicates MUST and MUST_i
 856 are logically equivalent, and similarly for the convergence predicates \downarrow and \downarrow_i . The proof
 857 that $p\text{MUST}_i r$ implies $p\text{MUST} r$ is - obviously - by induction on the derivation of $p\text{MUST}_i r$.
 858 Proving that the extensional predicates imply the intensional ones is, on the other hand,
 859 delicate, because we may have to deal with unbounded structures. The tree in Figure 10
 860 is once more the archetypal example: it has countably many maximal paths, and there is no
 861 upper bound on the depth at which successful states (i.e. nodes in the *bar*) are found.

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

866 A.2 Inductive definitions of predicates

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

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

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

874 $\forall \eta$ maximal execution of $S. \eta_0 = s$ implies $\exists n \in \mathbb{N}, Q(\eta_n)$

⁹ Whence the name *bar*-induction.

XX:24 Constructive characterisations of the must-preorder for asynchrony

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

$$\begin{array}{c} \text{[AXIOM]} \frac{Q(s)}{\text{int}_Q(s)} \quad \text{[IND-RULE]} \frac{s \rightarrow \quad \forall s'. s \rightarrow s' \text{ implies } \text{int}_Q(s')}{\text{int}_Q(s)} \end{array}$$

875

■

876 For instance, by letting

$$877 \quad Q_1(p) \iff p \dashv\vdash \quad Q_2(p, r) \iff \text{GOOD}(r)$$

878 we have by definition that

$$879 \quad p \downarrow \iff \text{ext}_{Q_1}(p) \quad p \text{ MUST } r \iff \text{ext}_{Q_2}(p, r) \quad (\text{ext-preds})$$

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

$$883 \quad p \downarrow_i \iff \text{int}_{Q_1}(p) \quad p \text{ MUST}_i r \iff \text{int}_{Q_2}(p, r) \quad (\text{int-preds})$$

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

886 ► **Lemma 26.** For $p \in S$,

- 887 (a) $p \downarrow_i$ implies $p \downarrow$,
888 (b) for every r . $p \text{ MUST}_i r$ implies $p \text{ MUST } r$.

889 The proofs of completeness are more delicate. To the best of our knowledge, the ones
890 about CCS [40, 13] proceed by induction on the greatest number of steps necessary to arrive
891 at termination or at a successful state. Since the STS of $\langle \text{CCS}, \xrightarrow{\tau} \rangle$ is finite branching,
892 König's lemma guarantees that such a bound exists. This technique does not work on infinite-
893 branching STSs, for example the one of CCS with infinite sums [14]. If we reason in classical
894 logic, we can prove completeness without König's lemma and also over infinite-branching
895 STSs via a proof *ad absurdum*: suppose $p \downarrow$. If $\neg(p \downarrow_i)$ no finite derivation tree exists to prove
896 $p \downarrow_i$, and then we construct an infinite sequence of τ moves starting with p , thus $\neg(p \downarrow)$.
897 Since we strive to be constructive we replace reasoning *ad absurdum* with a constructive
898 axiom: (decidable) *bar-induction*. In the rest of this section we discuss this axiom, and adapt
899 it to our client-server setting. This requires a little terminology.

900 A.2.0.1 Bar-induction

901 The axiom we want to use is traditionally stated using natural numbers. We use the
902 standard notations \mathbb{N}^* for finite sequences of natural numbers, \mathbb{N}^ω for infinite sequences,
903 and $\mathbb{N}^\infty = \mathbb{N}^* \cup \mathbb{N}^\omega$ for finite or infinite sequences. Remark that, in constructive logics, given
904 $u \in \mathbb{N}^\infty$, we cannot do a case analysis on whether u is finite or infinite. The set \mathbb{N}^∞ equipped
905 with the prefix order can be seen as a *tree*, denoted $T_{\mathbb{N}}$, in the sense of set theory: a tree is
906 an ordered set (A, \leq) such that, for each $a \in A$, the set $\{b \mid b < a\}$ is well-ordered by $<$. A
907 *path* in a tree A is a maximal element in A . In the tree \mathbb{N}^∞ , each node has ω children, and
908 the paths are exactly the infinite sequences \mathbb{N}^ω .

909 A predicate $P \subseteq \mathbb{N}^*$ over finite words is a *bar* if every infinite sequence of natural numbers
910 has a finite prefix in P . Note that a bar defines a subtree of $T_{\mathbb{N}}$ *extensionally*, because it

911 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
 912 finite prefix which is in the bar P .

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

914 $\forall w \in \mathbb{N}^*, \text{ if } \forall n \in \mathbb{N}, w \cdot n \in Q \text{ then } w \in Q.$

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

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

919 1. *for all $\pi \in \mathbb{N}^\omega$, there exists $n \in \mathbb{N}$ such that $(\pi_1, \dots, \pi_n) \in Q$;*

920 2. *for all $w \in \mathbb{N}^*$, it is decidable whether $Q(w)$ or $\neg Q(w)$;*

921 3. *for all $w \in \mathbb{N}^*$, $Q(w) \Rightarrow P_{int}(w)$;*

922 4. *P_{int} is hereditary;*

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

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

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

931 A.2.0.2 Admissibility.

932 To show that the principle is admissible, we prove that it follows from the Classical Epsilon
 933 (CE) axiom of the Coq standard library. In short, CE gives a choice function ϵ such that
 934 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
 935 implies Excluded Middle, and thus classical reasoning, because $A \vee \neg A$ is equivalent to
 936 $\exists b : \text{bool}, (b = \text{true} \wedge A) \vee (b = \text{false} \wedge \neg A)$. Since CE is guaranteed by the Coq developers
 937 to be admissible, our statement of bar-induction is also admissible.

938 A.2.0.3 Encoding states

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

$$945 \quad F(p) : \mathbb{N} \rightarrow \{q \mid p \rightarrow q\} \tag{6}$$

946 where a surjection is defined as follows.

947 ► **Definition 28.** *A map $f : A \rightarrow B$ is a surjection if it has a section $g : B \rightarrow A$, that is,*
 948 *$f \circ g = \text{Id}_B$.*

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

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

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

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

- 961 1. *for all infinite execution η , there exists $n \in \mathbb{N}$ such that $(\eta_1, \dots, \eta_n) \in Q$;*
 - 962 2. *for all finite execution ζ , $Q(\zeta)$ or $\neg Q(\zeta)$ is decidable;*
 - 963 3. *for all finite execution ζ , $Q(\zeta) \Rightarrow P_{\text{int}}(\zeta)$;*
 - 964 4. *P_{int} is hereditary, as defined above except that $\zeta \cdot q$ is a partial operation defined when ζ*
 965 *is empty or its last state is p and $p \rightarrow q$;*
- 966 *then P_{int} holds over the empty execution: that is $P_{\text{int}}(\varepsilon)$ holds.*

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

972 ► **Definition 30.** *Define $\text{Sink}(S, \rightarrow) := \langle S \cup \{\top\}, \rightarrow^\top \rangle$, where \rightarrow^\top is defined inductively as*
 973 *follows:*

$$974 \quad p \rightarrow q \implies p \rightarrow^\top q \quad p \nrightarrow \implies p \rightarrow^\top \top \quad \top \rightarrow^\top \top$$

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

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

980 Now we easily obtain completeness of the intensional predicates.

981 ► **Corollary 32.** *For every $p \in C$,*

- 982 1. *$p \downarrow$ implies $p \downarrow_i$,*
- 983 2. *for every r . $p \text{ MUST } r$ implies $p \text{ MUST}_i r$.*

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

986 As we have outlined why Corollary 32 is true, from now on we use \downarrow_i and MUST_i instead
 987 of \downarrow and MUST . We now present the properties of these predicates that we use in the rest of
 988 the paper.

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

990 ► **Lemma 33.** *In every LTS, for every $p, p' \in C$ and $s \in \text{Act}^*$ the following facts are true,*

- 991 1. *if $p \downarrow s$ and $p \xrightarrow{\tau} p'$ then $p' \downarrow s$,*
- 992 2. *for every $\mu \in \text{Act}$. $p \downarrow \mu.s$ and $p \xrightarrow{\mu} p'$ imply $p \downarrow s$.*

993 ► **Lemma 34.** For every $s \in \text{Act}^*$ and $p \in \text{ACCS}$, if $p \Downarrow s$ then $|\{q \mid p \xrightarrow{s} q\}| \in \mathbb{N}$.

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

$$p \xrightarrow{\tau} p \parallel \bar{a} \xrightarrow{\tau} p \parallel \bar{a} \parallel \bar{a} \xrightarrow{\tau} p \parallel \bar{a} \parallel \bar{a} \parallel \bar{a} \xrightarrow{\tau} \dots$$

994 In more general languages also image-finiteness may fail. An example is given on page 267 of
995 [60].

996 The predicate MUST_i is preserved by atoms freely changing their locations in systems.
997 This is coherent with the intuition that the mailbox is a global and shared one. For instance
998 the systems $a.0 \parallel \bar{d} \parallel d.1$ and $a.0 \parallel d.1 \parallel \bar{d}$, which in the mechanisation are respectively

999 `(pr_par (pr_input a pr_nil) (pr_out d), pr_input d pr_succes)`
1000

1002 and

1003 `(pr_input a pr_nil, pr_par (pr_input a pr_succes) (pr_out d))`
1004
1005

1006 have the same mailbox, namely \bar{d} .

1007 The predicate MUST_i enjoys three useful properties: it ensures convergence of servers
1008 interacting with clients that are not in a good state; it is preserved by internal computation
1009 of servers; and it is preserved also by interactions with unhappy clients. The arguments
1010 to show these facts are by rule induction on the hypothesis $p \text{MUST}_i r$. The last fact is a
1011 consequence of a crucial property of MUST_i , namely Lemma 45.

1012 ► **Lemma 35.** Let $\mathcal{L}_A \in \text{OW}$ and $\mathcal{L}_B \in \text{OF}$. For every $p \in A$, $r \in B$ we have that $p \text{MUST}_i r$
1013 implies that $p \Downarrow_i$ or $\text{GOOD}(r)$.

1014 ► **Lemma 36.** Let $\mathcal{L}_A \in \text{OW}$ and $\mathcal{L}_B \in \text{OF}$. For every $p, p' \in A$, $r \in B$ we have that
1015 $p \text{MUST}_i r$ and $p \xrightarrow{\tau} p'$ imply $q \text{MUST}_i r$.

1016 ► **Lemma 37.** For every $\mathcal{L}_B \in \text{OBA}$, $r \in B$ and name $a \in \mathcal{N}$ such that $p \xrightarrow{\bar{a}} p'$ then
1017 $\text{GOOD}(p)$ iff $\text{GOOD}(p')$.

1018 **Proof.** This is a property of Good , more specifically `good_preserved_by_lts_output` and
1019 `good_preserved_by_lts_output_converse`. ◀

1020 **Lemma 45** 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
1021 $a \in \mathcal{N}$ such that $p_1 \xrightarrow{\bar{a}} p_2$ and $r_1 \xrightarrow{\bar{a}} r_2$, if $p_1 \text{MUST}_i r_2$ then $p_2 \text{MUST}_i r_1$.

1022 **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
1023 using the rule [AXIOM] and thus $\text{GOOD}(r_2)$. Lemma 37 implies that $\text{GOOD}(r_1)$, and so we
1024 prove $p_2 \text{MUST}_i r_1$ using rule [AXIOM]. We are done with the base case.

1025 In the inductive case, the hypothesis $p_2 \text{MUST}_i r_1$ has been derived via an rule [IND-RULE],
1026 and we therefore know the following facts:

- 1027 1. $p_1 \parallel r_2 \xrightarrow{\tau} \hat{p} \parallel \hat{r}$, and
- 1028 2. For every p', r' such that $p_1 \parallel r_2 \xrightarrow{\tau} p' \parallel r'$ we have that $p' \text{MUST}_i r'$.

1029 We prove $p_2 \text{MUST}_i r_1$ by applying rule [IND-RULE]. In turn this requires us to show that

- 1030 (i) $p_2 \parallel r_1 \xrightarrow{\tau}$, and that
- 1031 (ii) for each p' and r' such that $p_2 \parallel r_1 \xrightarrow{\tau} p' \parallel r'$, we have $p' \text{MUST}_i r'$.

1032 We prove (i). The argument starts with a case analysis on how the transition (1) has
1033 been derived. There are the following three cases:

XX:28 Constructive characterisations of the must-preorder for asynchrony

1034 [S-Srv] a τ -transition performed by the server such that $p_1 \xrightarrow{\tau} \hat{p}$ and that $\hat{r} = r_2$, or
 1035 [S-Clt] a τ -transition performed by the client such that $r_2 \xrightarrow{\tau} \hat{r}$ and that $\hat{p} = p_1$, or
 1036 [S-com] an interaction between the server p_1 and the client r_2 such that $p_1 \xrightarrow{\mu} \hat{p}$ and that
 1037 $r_2 \xrightarrow{\bar{\mu}} \hat{r}$.

1038 In case [S-SRV] we use the OUTPUT-TAU axiom together with the transitions $p_1 \xrightarrow{\bar{a}} p_2$
 1039 and $p_1 \xrightarrow{\tau} \hat{p}$ to obtain that either:

- 1040 ■ there exists a p_3 such that $p_2 \xrightarrow{\tau} p_3$ and $\hat{p} \xrightarrow{\bar{a}} p_3$, or
- 1041 ■ $p_2 \xrightarrow{a} p_3$.

1042 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
 1043 the second case recall that by hypothesis $r_1 \xrightarrow{\bar{a}} r_2$, and thus the transition $p_2 \xrightarrow{a} \hat{p}$ and
 1044 rule [S-COM] let us construct the desired reduction $p_2 \parallel r_1 \xrightarrow{\tau} \hat{p} \parallel r_2$.

1045 In case [S-CLT] we use the OUTPUT-COMMUTATIVITY axiom together with the transitions
 1046 $r_1 \xrightarrow{\bar{a}} r_2 \xrightarrow{\tau} \hat{r}$ to obtain a r_3 such that $r_1 \xrightarrow{\tau} r_3 \xrightarrow{\bar{a}} \hat{r}$ and it follows that there exists the
 1047 silent move $p_2 \parallel r_1 \xrightarrow{\tau} p_2 \parallel r_3$.

1048 In case [S-COM] we have that $p_1 \xrightarrow{\mu} \hat{p}$ and $r_2 \xrightarrow{\bar{\mu}} \hat{r}$. We distinguish whether $\mu = \bar{a}$ or not.
 1049 If $\mu = \bar{a}$ then observe that $r_1 \xrightarrow{\bar{a}} r_2 \xrightarrow{a} \hat{r}$. Since by hypothesis $r_1, r_2 \in B$ and $\mathcal{L}_B \in \text{OF}$ we
 1050 apply FEEDBACK axiom to these transitions and obtain $r_1 \xrightarrow{\tau} \hat{r}$. An application of [S-COM]
 1051 let us construct the desired transition $p_2 \parallel r_1 \xrightarrow{\tau} p_2 \parallel \hat{r}$.

1052 If $\mu \neq \bar{a}$ we apply the OUTPUT-CONFLUENCE axiom to the transitions $p_1 \xrightarrow{\bar{a}} p_2$ and
 1053 $p_1 \xrightarrow{\mu} \hat{p}$ to obtain a p_3 such that $p_2 \xrightarrow{\mu} p_3$ and $\hat{p} \xrightarrow{\bar{a}} p_3$. We then apply the OUTPUT-
 1054 COMMUTATIVITY axiom to obtain $r_1 \xrightarrow{\bar{\mu}} r_3 \xrightarrow{\bar{a}} \hat{r}$ for some r_3 . Finally, we have the desired
 1055 $p_2 \parallel r_1 \xrightarrow{\tau} \hat{p} \parallel r_3$ thanks to the existence of an interaction between p_2 and r_1 that follows
 1056 from $p_2 \xrightarrow{\mu} p_3$ and $r_1 \xrightarrow{\bar{\mu}} r_3$. This concludes the proof of (i).

1057 We now tackle (ii). First of all, note that the inductive hypothesis states the following
 1058 fact,

1059 For every p', r', p_0 and r_0 , such that $p_1 \parallel r_2 \xrightarrow{\tau} p' \parallel r'$, $p' \xrightarrow{\bar{a}} p_0$ and $r_0 \xrightarrow{\bar{a}} r'$ then
 1060 $p_0 \text{ MUST}_i r_0$.

Fix a transition

$$p_2 \parallel r_1 \xrightarrow{\tau} p' \parallel r',$$

1061 we must show $p' \text{ MUST}_i r'$. We proceed by case analysis on the rule used to derive the
 1062 transition at issue, and the cases are as follows,

- 1063 (a) a τ -transition performed by the server such that $p_2 \xrightarrow{\tau} p'$ and that $r' = r_1$, or
- 1064 (b) a τ -transition performed by the client such that $r_1 \xrightarrow{\tau} r'$ and that $p' = p_2$, or
- 1065 (c) an interaction between the server p_2 and the client r_1 such that $p_2 \xrightarrow{\mu} p'$ and that
 1066 $r_1 \xrightarrow{\bar{\mu}} r'$.

1067 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
 1068 the OUTPUT-COMMUTATIVITY axiom to the transitions $p_1 \xrightarrow{\bar{a}} p_2 \xrightarrow{\tau} p'$ to obtain a p_3 such
 1069 that $p_1 \xrightarrow{\tau} p_3 \xrightarrow{\bar{a}} p'$. We apply the inductive hypothesis with $p' = p_3, r' = r_2, p_0 = p'$ and
 1070 $r_0 = r_1$ and obtain $p_2 \text{ MUST}_i r_1$ as required.

1071 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
 1072 the OUTPUT-TAU axiom to the transitions $r_1 \xrightarrow{\tau} r'$ and $r_1 \xrightarrow{\bar{a}} r_2$ to obtain that

- 1073 ■ either there exists a \hat{r} such that $r_2 \xrightarrow{\tau} \hat{r}$ and $r' \xrightarrow{\bar{a}} \hat{r}$,
- 1074 ■ or $r_2 \xrightarrow{a} r'$.

1075 In the first case we apply the inductive hypothesis with $p' = p_1, r' = \hat{r}, p_0 = p_2$ and $r_0 = r'$
 1076 and obtain $p_2 \text{ MUST}_i r'$ as required. In the second case, the transitions $p_1 \xrightarrow{\bar{a}} p_2$ and $r_2 \xrightarrow{a} r'$
 1077 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
 1078 required.

1079 We now consider the case (c) in which $p_2 \xrightarrow{\mu} p'$ and $r_1 \xrightarrow{\bar{\mu}} r'$. We must show $p' \text{ MUST}_i r'$
 1080 and to do so we distinguish whether $\mu = a$ or not.

1081 If $\mu = a$ then we apply the OUTPUT-DETERMINACY axiom to the transitions $r_1 \xrightarrow{\bar{a}} r_2$
 1082 and $r_1 \xrightarrow{\bar{\mu}} r'$ to obtain that $r_2 = r'$. Since by hypothesis $p_1, p_2 \in A$ and $\mathcal{L}_A \in \text{OW}$ we apply
 1083 the FWD-FEEDBACK axiom to the transitions $p_1 \xrightarrow{\bar{a}} p_2 \xrightarrow{a} p'$ to prove that either $p_1 \xrightarrow{\tau} p'$
 1084 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)
 1085 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
 1086 $p_1 = p'$ then $p' \text{ MUST}_i r_2$ is a direct consequence of the hypothesis $p_1 \text{ MUST}_i r_2$.

1087 If $\mu \neq a$ then we are allowed to apply the OUTPUT-CONFLUENCE axiom to the transitions
 1088 $r_1 \xrightarrow{\bar{a}} r_2$ and $r_1 \xrightarrow{\bar{\mu}} r'$ to obtain a \hat{r} such that $r_2 \xrightarrow{\bar{\mu}} \hat{r}$ and $r' \xrightarrow{\bar{a}} \hat{r}$. An application of the
 1089 OUTPUT-COMMUTATIVITY axiom to the transitions $p_1 \xrightarrow{\bar{a}} p_2 \xrightarrow{\mu} p'$ provides us with a \hat{p}
 1090 such that $p_1 \xrightarrow{\mu} \hat{p} \xrightarrow{\bar{a}} p'$. We now apply the inductive hypothesis with $p' = \hat{p}, r' = \hat{r}, p_0 = p'$
 1091 and $r_0 = r'$ and obtain $p_2 \text{ MUST}_i r'$ as required. This concludes the proof of (ii), and therefore
 1092 of the lemma. ◀

1093 ▶ **Lemma 38.** *Let $\mathcal{L}_A \in \text{OW}$ and $\mathcal{L}_B \in \text{OF}$. For every $p, p' \in A, r, r' \in B$ and every*
 1094 *action $\mu \in \text{Act}$ such that $p \xrightarrow{\mu} p'$ and $r \xrightarrow{\bar{\mu}} r'$ we have that $p \text{ MUST}_i r$ and $\neg \text{GOOD}(r)$ implies*
 1095 *$p' \text{ MUST}_i r'$.*

1096 **Proof.** By case analysis on the hypothesis that $p \text{ MUST}_i r$. ◀

$$\begin{array}{c}
 \text{[L-PROC]} \quad \frac{p \xrightarrow{\alpha} p'}{p \triangleright M \xrightarrow{\alpha}_{\text{fw}} p' \triangleright M} \qquad \text{[L-COMM]} \quad \frac{p \xrightarrow{a} p'}{p \triangleright (\{\bar{a}\} \uplus M) \xrightarrow{\tau}_{\text{fw}} p' \triangleright M} \\
 \text{[L-MOUT]} \quad \frac{}{p \triangleright (\{\bar{a}\} \uplus M) \xrightarrow{\bar{a}}_{\text{fw}} p \triangleright M} \qquad \text{[L-MINP]} \quad \frac{}{p \triangleright M \xrightarrow{a}_{\text{fw}} p \triangleright (\{\bar{a}\} \uplus M)}
 \end{array}$$

■ **Figure 11** Lifting of a transition relation to transitions of forwarders.

1097 B Forwarders

1098 The intuition behind forwarders, quoting [64], is that “any message can come into the
 1099 configuration, regardless of the forms of inner receptors. [...] As the experimenter is not
 1100 synchronously interacting with the configuration [...], he may send any message as he likes.”

1101 In this appendix we give the technical results to ensure that the function $\text{FW}(-)$ builds
 1102 an LTS that satisfies the axioms of the class LtsEq .

1103 ► **Definition 39.** For any LTS $\mathcal{L} = \langle C, L, \longrightarrow \rangle \in \text{OF}$, we define the function $\text{strip} : C \rightarrow C$
 1104 by induction on $\text{mbox}(p)$ as follows: if $\text{mbox}(p) = \emptyset$ then $\text{strip}(p) = p$, while if $\exists \bar{a} \in \text{mbox}(p)$
 1105 and $p \xrightarrow{\bar{a}} p'$ then $\text{strip}(p) = \text{strip}(p')$. Note that $\text{strip}(p)$ is well-defined thanks to the
 1106 *OUTPUT-DETERMINACY* and the *OUTPUT-COMMUTATIVITY* axioms. ■

1107 We now wish to show that $\text{FW}(\mathcal{L}) \in \text{OW}$ for any LTS \mathcal{L} of output-buffered agents
 1108 with feedback. Owing to the structure of our typeclasses, we have first to construct an
 1109 equivalence \doteq over $\text{FW}(\mathcal{L})$ that is compatible with the transition relation, *i.e.* satisfies the
 1110 axiom in Figure 5. We do this in the obvious manner, *i.e.* by combining the equivalence \simeq
 1111 over the states of \mathcal{L} with an equivalence over mailboxes.

1112 ► **Definition 40.** For any LTS $\mathcal{L} \in \text{OF}$, two states $p \triangleright M$ and $q \triangleright N$ of $\text{FW}(\mathcal{L})$ are equivalent,
 1113 denoted $p \triangleright M \doteq q \triangleright N$, if $\text{strip}(p) \simeq \text{strip}(q)$ and $M \uplus \text{mbox}(p) = N \uplus \text{mbox}(q)$. ■

1114 ► **Lemma 41.** For every \mathcal{L}_A and every $p \triangleright M, q \triangleright N \in A \times \text{MO}$, and every $\alpha \in L$, if
 1115 $p \triangleright M (\doteq \cdot \xrightarrow{\alpha}_{\text{fw}}) q \triangleright N$ then $p \triangleright M (\xrightarrow{\alpha}_{\text{fw}} \cdot \doteq) q' \triangleright N'$.

1116 ► **Lemma 42.** For every $\mathcal{L}_A \in \text{OF}$ and every $p, q \in A, M \in \text{MO}$, if $p \simeq q$ then $p \triangleright M \doteq q \triangleright M$.

1117 **Proof.** This follows from the fact that if $p \simeq q$ then $\text{strip}(p) \simeq \text{strip}(q)$ and $\text{mbox}(p) =$
 1118 $\text{mbox}(q)$. ◀

1120 **Lemma 13.** For every LTS $\mathcal{L} \in \text{OF}$, $\text{FW}(\mathcal{L}) \in \text{OW}$.

1121 **Proof.** We must show that, given an LTS $\mathcal{L} = \langle C, L, \longrightarrow \rangle \in \text{OF}$, we have that $\text{FW}(\mathcal{L}) \in \text{OW}$.
 1122 To do so, we need to show that $\text{FW}(\mathcal{L})$ obeys to the axioms given in Equation (5), namely
 1123 *INPUT-BOOMERANG* and *FWD-FEEDBACK*. We first show that $\text{FW}(\mathcal{L})$ obeys to the *INPUT-*
 1124 *BOOMERANG* axiom.

We pick a process $p \in C$, a mailbox $M \in \text{MO}$ and a name $a \in \mathcal{N}$. The axiom *INPUT-BOOMERANG* requires us to exhibit a process $p' \in A$ and a mailbox $M' \in \text{MO}$ such that the following transitions hold.

$$p \triangleright M \xrightarrow{a}_{\text{fw}} p' \triangleright M' \xrightarrow{\bar{a}}_{\text{fw}} p \triangleright M$$

We choose $p' = p$ and $M' = (\{\bar{a}\} \uplus M)$. An application of the rule [L-MINP] and then the rule [L-MOUT] from Figure 11 allows us to derive the required sequence of transitions as shown below.

$$p \triangleright M \xrightarrow{a}_{\text{fw}} p \triangleright (\{\bar{a}\} \uplus M) \xrightarrow{\bar{a}}_{\text{fw}} p \triangleright M$$

1125 We now show that $\text{FW}(\mathcal{L})$ obeys to the FWD-FEEDBACK axiom. To begin we pick three
1126 processes $p_1, p_2, p_3 \in \mathcal{C}$, three mailboxes $M_1, M_2, M_3 \in \mathcal{MO}$ and a name $a \in \mathcal{N}$ such that:

$$p_1 \triangleright M_1 \xrightarrow{\bar{a}}_{\text{fw}} p_2 \triangleright M_2 \xrightarrow{a}_{\text{fw}} p_3 \triangleright M_3$$

1127 We need to show that either:

- 1128 1. $p_1 \triangleright M_1 \xrightarrow{\tau}_{\text{fw}} p_3 \triangleright M_3$, or
- 1129 2. $p_1 \triangleright M_1 \dot{=} p_3 \triangleright M_3$

1130 We proceed by case analysis on the last rule used to derive the transition $p_1 \triangleright M_1 \xrightarrow{\bar{a}}_{\text{fw}}$
1131 $p_2 \triangleright M_2$. This transition can either be derived by the rule [L-MOUT] or the rule [L-PROC].

We first consider the case where the transition has been derived using the rule [L-MOUT]. We then have that $p_1 = p_2$ and $M_1 = (\{\bar{a}\} \uplus M_2)$. We continue by case analysis on the last rule used to derive the transition $p_2 \triangleright M_2 \xrightarrow{a}_{\text{fw}} p_3 \triangleright M_3$. If this transition has been derived using the rule [L-MINP] then it must be the case that $p_2 = p_3$ and that $(\{\bar{a}\} \uplus M_2) = M_3$. This lets us conclude by the following equality to show that $p_1 \triangleright M_1 \dot{=} p_3 \triangleright M_3$.

$$p_1 \triangleright M_1 = p_2 \triangleright (\{\bar{a}\} \uplus M_2) = p_3 \triangleright M_3$$

Otherwise, this transition has been derived using the rule [L-PROC], which implies that $p_2 \xrightarrow{a} p_3$ together with $M_2 = M_3$. An application of the rule [L-COMM] ensures the following transition and allows us to conclude this case with $p_1 \triangleright M_1 \xrightarrow{\tau}_{\text{fw}} p_3 \triangleright M_3$.

$$p_1 \triangleright M_1 = p_2 \triangleright (\{\bar{a}\} \uplus M_2) \xrightarrow{\tau} p_3 \triangleright M_2 = p_3 \triangleright M_3$$

1132 We now consider the case where the transition $p_1 \triangleright M_1 \xrightarrow{\bar{a}}_{\text{fw}} p_2 \triangleright M_2$ has been derived
1133 using the rule [L-PROC] such that $p_1 \xrightarrow{\bar{a}} p_2$ and $M_1 = M_2$.

1134 Again, we continue by case analysis on the last rule used to derive the transition
1135 $p_2 \triangleright M_2 \xrightarrow{a}_{\text{fw}} p_3 \triangleright M_3$. If this transition has been derived using the rule [L-MINP] then it must
1136 be the case that $p_2 = p_3$ and $(\{\bar{a}\} \uplus M_2) = M_3$. Also, note that, as $\mathcal{L} \in \text{OF}$, the transition
1137 $p_1 \xrightarrow{\bar{a}} p_2$ implies $\text{mbox}(p_1) = \text{mbox}(p_2) \uplus \{\bar{a}\}$. In order to prove $p_1 \triangleright M_1 \dot{=} p_3 \triangleright M_3$, it
1138 suffices to show the following:

- 1139 (a) $\text{strip}(p_1) \simeq \text{strip}(p_3)$, and
- 1140 (b) $M_1 \uplus \text{mbox}(p_1) = M_3 \uplus \text{mbox}(p_3)$

1141 We show that $\text{strip}(p_1) \simeq \text{strip}(p_3)$ by definition of $\text{strip}()$ together with the transition
1142 $p_1 \xrightarrow{\bar{a}} p_2$ and the equality $p_2 = p_3$.

1143 The following ensures that $M_1 \uplus \text{mbox}(p_1) = M_3 \uplus \text{mbox}(p_3)$.

$$\begin{aligned} & M_1 \uplus \text{mbox}(p_1) \\ = & M_1 \uplus \text{mbox}(p_2) \uplus \{\bar{a}\} && \text{from } \text{mbox}(p_1) = \text{mbox}(p_2) \uplus \{\bar{a}\} \\ = & M_2 \uplus \text{mbox}(p_3) \uplus \{\bar{a}\} && \text{from } M_1 = M_2, p_2 = p_3 \\ = & (M_2 \uplus \{\bar{a}\}) \uplus \text{mbox}(p_3) \\ = & M_3 \uplus \text{mbox}(p_3) && \text{from } M_3 = M_2 \uplus \{\bar{a}\} \end{aligned}$$

1144 If the transition $p_2 \triangleright M_2 \xrightarrow{a}_{\text{fw}} p_3 \triangleright M_3$ has been derived using the rule [L-PROC] then it
1145 must be the case that $p_2 \xrightarrow{a} p_3$ and $M_2 = M_3$. As $\mathcal{L} \in \text{OF}$, we are able to call the axiom

XX:32 Constructive characterisations of the must-preorder for asynchrony

1146 FEEDBACK together with the transitions $p_1 \xrightarrow{\bar{a}} p_2$ and $p_2 \xrightarrow{a} p_3$ to obtain a process p'_3
1147 such that $p_1 \xrightarrow{\tau} p'_3$ and $p'_3 \simeq p_3$. An application of Lemma 42 and rule [L-PROC] allows us to
1148 conclude that $p_1 \triangleright M_1 (\xrightarrow{\tau}_{f_w} \cdot \dot{=}) p_3 \triangleright M_3$. ◀

-
- $\forall s \in \text{Act}^*, \forall a \in \mathcal{N}$,
(1) $\neg \text{GOOD}(f(s))$
(2) $\forall \mu \in \text{Act}, f(\mu.s) \xrightarrow{\bar{\mu}} f(s)$
(3) $f(\bar{a}.s) \xrightarrow{\tau}$
(4) $\forall r \in C, f(\bar{a}.s) \xrightarrow{\tau} r$ implies $\text{GOOD}(r)$
(5) $\forall r \in C, \mu \in \text{Act}, f(\bar{a}.s) \xrightarrow{\mu} r$ implies $\mu = a$ and $r = f(s)$

- $\forall E \subseteq \mathcal{N}$,
(t1) $ta(\varepsilon, E) \xrightarrow{\tau}$
(t2) $\forall a \in \mathcal{N}, ta(\varepsilon, E) \xrightarrow{\bar{a}}$
(t3) $\forall a \in \mathcal{N}, ta(\varepsilon, E) \xrightarrow{a}$ if and only if $a \in E$
(t4) $\forall \mu \in \text{Act}, r \in C, ta(\varepsilon, E) \xrightarrow{\mu} r$ implies $\text{GOOD}(r)$

- (c1)** $\forall \mu \in \text{Act}, tc(\varepsilon) \xrightarrow{\tau}$
(c2) $\exists r, tc(\varepsilon) \xrightarrow{\tau} r$
(c3) $\forall r, tc(\varepsilon) \xrightarrow{\tau} r$ implies $\text{GOOD}(r)$

■ **Table 1** Properties of the functions that generate clients.

1149 C Completeness

1150 This section is devoted to the proof that the alternative preorder given in Definition (9)
 1151 includes the MUST-preorder. First we present a general outline of the main technical results
 1152 to obtain the proof we are after. Afterwards, in Subsection (C) we discuss in detail on all
 1153 the technicalities.

1154 Proofs of completeness of characterisations of contextual preorders usually require using,
 1155 as the name suggests, syntactic contexts. Our calculus-independent setting, though, does not
 1156 allow us to define them. Instead we phrase our arguments using two functions $tc : \text{Act}^* \rightarrow C$,
 1157 and $ta : \text{Act}^* \times \mathcal{P}(\mathcal{N}) \rightarrow C$ where $\langle C, L, \longrightarrow \rangle$ is some LTS of OF. In Table 1 we gather all
 1158 the *properties* of tc and ta that are sufficient to give our arguments. The properties (1) - (5)
 1159 must hold for both tc and $ta(\varepsilon, -)$ for every set of names O , the properties (c1) - (c2) must
 1160 hold for tc , and (t1) - (t4) must hold for ta .

1161 We use the function tc to test the convergence of servers, and the function ta to test the
 1162 acceptance sets of servers.

1163 A natural question is whether such tc and ta can actually exist. The answer depends on
 1164 the LTS at hand. In Appendix F.2, and in particular Figure 18, we define these functions for
 1165 the standard LTS of ACCS, and it should be obvious how to adapt those definitions to the
 1166 asynchronous π -calculus [57].

1167 In short, our proofs show that \preceq_{AS} is complete with respect to \preceq_{MUST} in any LTS of
 1168 output-buffered agents with feedback wherein the functions tc and ta enjoying the properties
 1169 in Table 1 can be defined.

1170 First, converging along a finite trace s is logically equivalent to passing the client $tc(s)$.
 1171 In other words, there exists a bijection between the proofs (i.e. finite derivation trees
 1172 of $p \text{MUST}_i tc(s)$) and the ones of $p \Downarrow s$. We first give the proposition, and then discuss the
 1173 auxiliary lemmas to prove it.

XX:34 Constructive characterisations of the must-preorder for asynchrony

1174 ► **Proposition 43.** For every $\mathcal{L}_A \in OW$, $p \in A$, and $s \in \text{Act}^*$ we have that $p \text{MUST}_i tc(s)$ if
 1175 and only if $p \Downarrow s$.

1176 The *if* implication is Lemma 60 and the *only if* implication is Lemma 57. The hypothesis
 1177 that $\mathcal{L}_A \in OW$, *i.e.* the use of forwarders, is necessary to show that convergence implies
 1178 passing a client, as shown by the next example.

1179 ► **Example 44.** Consider a server p in an LTS $\mathcal{L} \in OF$ whose behaviour amounts to the
 1180 following transitions: $p \xrightarrow{b} \Omega \xrightarrow{\tau} \Omega \xrightarrow{\tau} \dots$. Note that this entails that \mathcal{L} does not *not* enjoy
 1181 the axioms of forwarders.

1182 Now let $s = a.b$. Since $p \Downarrow$ and $p \not\Rightarrow^a$ we know that $p \Downarrow a.b$. On the other hand Table 1(2)
 1183 implies that the client $tc(s)$ performs the transitions $tc(s) \xrightarrow{\bar{a}} tc(b) \xrightarrow{\bar{b}} tc(\varepsilon)$. Thanks to
 1184 the OUTPUT-COMMUTATIVITY axiom we obtain $tc(s) \xrightarrow{\bar{b}} \bar{a} \rightarrow tc(\varepsilon)$. Table 1(1) implies that
 1185 the states reached by the client are unsuccessful, and so by zipping the traces performed by
 1186 p and by $tc(s)$ we build a maximal computation of $p \parallel tc(s)$ that is unsuccessful, and thus
 1187 $p \text{MUST}_i tc(s)$. ◀

1188 This example explains why in spite of Lemma 14 output-buffered agents with feedback do
 1189 not suffice to use the standard characterisations of the MUST-preorder.

1190 We move on to the more involved technical results, *i.e.* the next three lemmas, that we
 1191 use to reason on acceptance sets of servers. We wish to stress Lemma 45: it states that, when
 1192 reasoning on MUST_i , outputs can be freely moved from the client to the server side of systems,
 1193 if servers have the forwarding ability. Its proof uses *all* the axioms for output-buffered agents
 1194 with feedback, and the lemma itself is used in the proof of the main result on acceptance
 1195 sets, namely Lemma 47.

1196 ► **Lemma 45 (Output swap).** Let $\mathcal{L}_A \in OW$ and $\mathcal{L}_B \in OF$. For every $p_1, p_2 \in A$,
 1197 every $r_1, r_2 \in B$ and name $a \in \mathcal{N}$ such that $p_1 \xrightarrow{\bar{a}} p_2$ and $r_1 \xrightarrow{\bar{a}} r_2$, if $p_1 \text{MUST}_i r_2$ then
 1198 $p_2 \text{MUST}_i r_1$.

1199 ► **Lemma 46.** Let $\mathcal{L}_A \in OW$. For every $p \in A$, $s \in \text{Act}^*$, and every $L, E \subseteq \mathcal{N}$, if
 1200 $\bar{L} \in \mathcal{A}_{\text{fw}}(p, s)$ then $p \text{MUST}_i ta(s, E \setminus L)$.

1201 ► **Lemma 47.** Let $\mathcal{L}_A \in OW$. For every $p \in A$, $s \in \text{Act}^*$, and every finite set $O \subseteq \bar{\mathcal{N}}$, if
 1202 $p \Downarrow s$ then either

- 1203 (i) $p \text{MUST}_i ta(s, \bigcup \overline{\mathcal{A}_{\text{fw}}(p, s)} \setminus O)$, or
- 1204 (ii) there exists $\hat{O} \in \mathcal{A}_{\text{fw}}(p, s)$ such that $\hat{O} \subseteq O$.

1205 We can now show that the alternative preorder \preceq_{AS} includes $\sqsubseteq_{\text{MUST}}$ when used over LTSs
 1206 of forwarders.

1207 ► **Lemma 48.** 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 $p \preceq_{\text{AS}} q$.

1208 **Proof.** Let $p \sqsubseteq_{\text{MUST}} q$. To prove that $p \preceq_{\text{cnv}} q$, suppose $p \Downarrow s$ for some trace s . Proposition 43
 1209 implies $p \text{MUST}_i tc(s)$, and so by hypothesis $q \text{MUST}_i tc(s)$. Proposition 43 ensures that $q \Downarrow s$.

1210 We now show that $p \preceq_{\text{acc}} q$. Thanks to Lemma 15, it is enough to prove that $p \preceq_{\text{acc}}^{\text{fw}} q$.
 1211 So, we show that for every trace $s \in \text{Act}^*$, if $p \Downarrow s$ then $\mathcal{A}_{\text{fw}}(p, s) \ll \mathcal{A}_{\text{fw}}(q, s)$. Fix an
 1212 $O \in \mathcal{A}_{\text{fw}}(q, s)$. We have to exhibit a set $\hat{O} \in \mathcal{A}_{\text{fw}}(p, s)$ such that $\hat{O} \subseteq O$.

1213 By definition $O \in \mathcal{A}_{\text{fw}}(q, s)$ means that for some q' we have $q \xrightarrow{s} q' \xrightarrow{\tau}$ and $O(q') = O$.
 1214 Let $E = \bigcup \mathcal{A}_{\text{fw}}(p, s)$ and $X = E \setminus O$. The hypothesis that $p \Downarrow s$, and the construction of
 1215 the set X let us apply Lemma 47, which implies that either

1216 (a) $p \text{ MUST}_i ta(s, \overline{X})$, or

1217 (b) there exists a $\widehat{O} \in \mathcal{A}_{fw}(p, s)$ such that $\widehat{O} \subseteq O(q')$.

1218 Since (b) is exactly what we are after, to conclude the argument it suffices to prove that (a)

1219 is false. This follows from Lemma 46, which proves $q \text{ MUST}_i ta(s, \overline{X})$, and the hypothesis

1220 $p \sqsubset_{\text{MUST}} q$, which ensures $p \text{ MUST}_i ta(s, \overline{X})$. \blacktriangleleft

1221 The fact that the MUST-preorder can be captured via the function $\text{FW}(-)$ and \preceq_{AS} is a
1222 direct consequence of Lemma 14 and Lemma 48.

1223 **► Proposition 49 (Completeness).** *For every $\mathcal{L}_A, \mathcal{L}_B \in \text{OF}$ and servers $p \in A, q \in B$, if*
1224 *$p \sqsubset_{\text{MUST}} q$ then $\text{FW}(p) \preceq_{\text{AS}} \text{FW}(q)$.*

1225 We now gather all the technical auxiliary lemmas and then discuss the proofs of the main
1226 ones.

1227 By assumption, outputs preserve the predicate GOOD. For stable clients, they also preserve
1228 the negation of this predicate.

1229 **► Lemma 50.** *For all $r, r' \in A$ and trace $s \in \overline{\mathcal{N}}^*$, $r \xrightarrow{\tau} r'$, $\neg \text{GOOD}(r)$ and $r \xrightarrow{s} r'$ implies*
1230 *$\neg \text{GOOD}(r')$.*

1231 C.1 Testing convergence

1232 We start with preliminary facts, in particular two lemmas that follow from the properties in
1233 Table 1.

1234 A process p converges along a trace s if for every p' reached by p performing any prefix
1235 of s , the process p' converges.

1236 **► Lemma 51.** *For every $\langle A, L, \longrightarrow \rangle$, $p \in A$, and $s \in \text{Act}^*$, $p \Downarrow s$ if and only if $p \xrightarrow{s'} p'$*
1237 *implies $p' \Downarrow$ for every s' prefix of s .*

1238 Traces of output actions impact neither the stability of servers, nor their input actions.

1239 **► Lemma 52.** *For every \mathcal{L}_A , every $p, p' \in A$ and every trace $s \in \overline{\mathcal{N}}^*$,*

1240 1. $p \xrightarrow{\tau} r$ and $p \xrightarrow{s} p'$ implies $p' \xrightarrow{\tau} r$.

1241 2. $p \xrightarrow{\tau} r$ and $p \xrightarrow{s} p'$ implies $I(p) = I(p')$.

1242 **► Lemma 53.** *For every $s \in \text{Act}^*$, $tc(s) \xrightarrow{\tau} r$.*

1243 The BACKWARD-OUTPUT-DETERMINACY axiom is used in the proof of the next lemma.

1244 **► Lemma 54.** *For every $s \in \text{Act}^*$, if $tc(s) \xrightarrow{\mu} r$ then either*

1245 (a) $\text{GOOD}(r)$, or

1246 (b) $s = s_1.\bar{\mu}.s_2$ for some $s_1 \in \mathcal{N}^*$ and $s_2 \in \text{Act}^*$ such that $r \simeq tc(s_1.s_2)$.

1247 **► Lemma 55.** *For every $s \in \text{Act}^*$, if $tc(s) \xrightarrow{\tau} r$ then either:*

1248 (a) $\text{GOOD}(r)$, or

1249 (b) there exist b, s_1, s_2 and s_3 with $s_1.b.s_2 \in \mathcal{N}^*$ such that $s = s_1.b.s_2.\bar{b}.s_3$ and $r \simeq$
1250 $tc(s_1.s_2.s_3)$.

1251 **► Lemma 56.** *Let $\mathcal{L}_A \in \text{OW}$. For every server $p, p' \in A$, trace $s \in \text{Act}^*$ and action $\mu \in \text{Act}$*
1252 *such that $p \xrightarrow{\mu} p'$ we have that $p \text{ MUST}_i tc(\mu.s)$ implies $p' \text{ MUST}_i tc(s)$.*

1253 **Proof.** By rule induction on the reduction $p \xrightarrow{\mu} p'$ together with Lemma 36 and Lemma 38.

1254 \blacktriangleleft

XX:36 Constructive characterisations of the must-preorder for asynchrony

1255 ► **Lemma 57.** *Let $\mathcal{L}_A \in OW$. For every server $p \in A$, trace $s \in \text{Act}^*$ we have that*
1256 *$p \text{ MUST}_i tc(s)$ implies $p \Downarrow s$.*

1257 **Proof.** We proceed by induction on the trace s . In the base case s is ε . Table 1(1) states
1258 that $\neg \text{GOOD}(tc(\varepsilon))$ and we apply Lemma 35 to obtain $p \Downarrow_i$, and thus $p \Downarrow \varepsilon$. In the inductive
1259 case s is $\mu.s'$ for some $\mu \in \text{Act}$ and $s' \in \text{Act}^*$. We must show the following properties,

- 1260 1. $p \Downarrow_i$, and
1261 2. for every p' such that $p \xrightarrow{\mu} p'$, $p' \Downarrow s'$.

1262 We prove the first property as we did in the base case, and we apply Lemma 56 to prove the
1263 second property. ◀

1264 ► **Lemma 58.** *Let $\mathcal{L}_A \in OW$. For every $p \in A$, $s_1 \in \mathcal{N}^*$ and $s_3 \in \text{Act}^*$ we have that*

- 1265 1. *for every $\mu \in \text{Act}$, if $p \Downarrow s_1.\mu.s_3$ and $p \xrightarrow{\mu} q$ then $q \Downarrow s_1.s_3$,*
1266 2. *for every $a.s_2 \in \mathcal{N}^*$ if $p \Downarrow s_1.a.s_2.\bar{a}.s_3$ then $p \Downarrow s_1.s_2.s_3$.*

1267 ► **Lemma 59.** *For every LTS \mathcal{L}_A and every $p \in A$, $p \Downarrow_i$ implies $p \text{ MUST}_i tc(\varepsilon)$.*

1268 **Proof.** Rule induction on the derivation of $p \Downarrow_i$. ◀

1269 ► **Lemma 60.** *For every $\mathcal{L}_A \in OW$, every $p \in A$, and $s \in \text{Act}^*$, if $p \Downarrow s$ then $p \text{ MUST}_i tc(s)$.*

1270 **Proof.** The hypothesis $p \Downarrow s$ ensures $p \Downarrow_i$. We show that $p \text{ MUST}_i tc(s)$ reasoning by complete
1271 induction on the length of the trace s . The base case is Lemma 59 and here we discuss the
1272 inductive case, i.e. when $\text{len}(s) = n + 1$ for some $n \in \mathbb{N}$.

1273 We proceed by rule induction on $p \Downarrow_i$. In the base case $p \xrightarrow{\tau}$, and the reduction at hand
1274 is due to either a τ transition in $tc(s)$, or a communication between p and $tc(s)$.

1275 In the first case $tc(s) \xrightarrow{\tau} r$, and so Lemma 55 ensures that one of the following conditions
1276 holds,

- 1277 1. $\text{GOOD}(r)$, or
1278 2. there exist $a \in \mathcal{N}$, s_1, s_2 and s_3 with $s = s_1.a.s_2.\bar{a}.s_3$ and $r \simeq tc(s_1.s_2.s_3)$.

1279 If $\text{GOOD}(r)$ then we conclude via rule [AXIOM]; otherwise Lemma 58(2) and the hypothesis
1280 that $p \Downarrow s$ imply $p \Downarrow s_1.s_2.s_3$, thus prove $p \text{ MUST}_i r$ via the inductive hypothesis of the
1281 complete induction on s .

1282 We now consider the case when the transition is due to a communication, i.e. $p \xrightarrow{\mu} p'$
1283 and $tc(s) \xrightarrow{\bar{\mu}} r$. Lemma 54 tells us that either $\text{GOOD}(r)$ or there exist s_1 and s_2 such
1284 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
1285 second case we apply Lemma 58(1) to prove $p' \Downarrow s_1.s_2$, and thus $p' \text{ MUST}_i r$ follows from the
1286 inductive hypothesis of the complete induction. In the inductive case of the rule induction
1287 on $p \Downarrow_i$, we know that $p \xrightarrow{\tau} p'$ for some process p' . We reason again by case analysis on
1288 how the reduction we fixed has been derived, i.e. either via a τ transition in $tc(s)$, or via
1289 a communication between p and $tc(s)$, or via a τ transition in p . In the first two cases we
1290 reason as we did for the base case of the rule induction. In the third case $p \Downarrow s$ and $p \xrightarrow{\tau} p'$
1291 imply $p' \Downarrow s$, we thus obtain $p' \text{ MUST}_i tc(s)$ thanks to the inductive hypothesis of the rule
1292 induction which we can apply because the tree to derive $p' \Downarrow_i$ is smaller than the tree to
1293 derive that $p \Downarrow_i$. ◀

1294 C.2 Testing acceptance sets

1295 In this section we present the properties of the function $ta(-, -)$ that are sufficient to obtain
1296 completeness. To begin with, $ta(-, -)$ function enjoys a form of monotonicity with respect
1297 to its second argument.

1298 ► **Lemma 61.** *Let $\mathcal{L}_A \in OF$. For every $p \in A$, trace $s \in \text{Act}^*$, and sets of outputs O_1, O_2 ,
1299 if $p \text{ MUST}_i ta(s, O_1)$ and $O_1 \subseteq O_2$ then $p \text{ MUST}_i ta(s, O_2)$.*

1300 **Proof.** Induction on the derivation of $p \text{ MUST}_i ta(s, O_1)$. ◀

1301 Let OBA denote the set of LTS of output-buffered agents. Note that any $\mathcal{L} \in \text{OBA}$ need
1302 not enjoy the FEEDBACK axiom.

1303 ► **Lemma 62.** *Let $\mathcal{L}_A \in \text{OBA}$, and $\mathcal{L}_B \in \text{OBA}$. For every $p \in A$, trace $s \in \text{Act}^*$, set of
1304 outputs O and name $a \in \mathcal{N}$, such that*

1305 (i) $p \downarrow_i$ and,

1306 (ii) For every $p' \in A$, $p \xrightarrow{\bar{a}} p'$ implies $p' \text{ MUST}_i ta(s, \bar{O})$,

1307 we have that $p \text{ MUST}_i ta(\bar{a}.s, \bar{O})$.

1308 **Proof.** We proceed by induction on the hypothesis $p \downarrow_i$.

1309 C.2.0.1 (Base case: p is stable)

1310 We prove $p \text{ MUST}_i ta(\bar{a}.s, \bar{O})$ by applying rule [IND-RULE]. Since Table 1(3) implies that
1311 $p \parallel ta(\bar{a}.s, \bar{O}) \xrightarrow{\tau}$, all we need to prove is the following fact,

1312 $\forall p' \in A, r \in B$. if $p \parallel ta(\bar{a}.s, \bar{O}) \xrightarrow{\tau} p' \parallel r$ then $p' \text{ MUST}_i r$. (*)

1313 Fix a transition $p \parallel ta(\varepsilon, \bar{O}) \xrightarrow{\tau} p' \parallel r$. As p is stable, this transition can either be due to:

1314 1. a τ -transition performed by the client such that $ta(\bar{a}.s, \bar{O}) \xrightarrow{\tau} r$, or

1315 2. an interaction between the server p and the client $ta(\bar{a}.s, \bar{O})$.

In the first case Table 1(4) implies GOOD(r), and hence we obtain $p' \text{ MUST}_i r$ via rule [AXIOM].

In the second case there exists an action μ such that

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

1316 Table 1(5) implies μ is \bar{a} and $r = ta(s, \bar{O})$. We then have $p \xrightarrow{\bar{a}} p'$ and thus the reduction

1317 $p \xrightarrow{\bar{a}} p'$, which allows us to apply the hypothesis (ii) and obtain $p' \text{ MUST}_i r$ as required.

1318 C.2.0.2 (Inductive case: $p \xrightarrow{\tau} p'$ implies p')

1319 The argument is similar to one for the base case, except that we must also tackle the case

1320 when the transition $p \parallel ta(\bar{a}.s, \bar{O}) \xrightarrow{\tau} p' \parallel r$ is due to a τ action performed by p , i.e. $p \xrightarrow{\tau} p'$

1321 and $r = ta(\bar{a}.s, \bar{O})$. The inductive hypothesis tells us the following fact:

1322 For every p_1 and a , such that $p \xrightarrow{\tau} p_1$, for every p_2 , if $p_1 \xrightarrow{\bar{a}} p_2$ then $p_2 \text{ MUST}_i ta(s, O)$.

1323 To apply the inductive hypothesis we have to show that for every p_2 such that $p' \xrightarrow{\bar{a}} p_2$ we

1324 have that $p_2 \text{ MUST}_i ta(s, \bar{O})$. This is a consequence of the hypothesis (ii) together with the

1325 reduction $p \xrightarrow{\tau} p' \xrightarrow{\bar{a}} p_2$, and thus concludes the proof. ◀

1326 ► **Lemma 63.** *Let $\mathcal{L}_A \in OF$. For every $p \in A$ and set of outputs O , if p is stable then either*

1327 (a) $p \text{ MUST}_i ta(\varepsilon, \overline{O(p)} \setminus \bar{O})$, or

1328 (b) $O(p) \subseteq O$.

XX:38 Constructive characterisations of the must-preorder for asynchrony

1329 **Proof.** We distinguish whether $O(p) \setminus O$ is empty or not. In the first case, $O(p) \setminus O = \emptyset$
 1330 implies $O(p) \subseteq O$, and we are done.

1331 In the second case, there exists $\bar{a} \in O(p)$ such that $\bar{a} \notin O$. Note also that Table 1(1) ensures
 1332 that $\neg \text{GOOD}(ta(\varepsilon, \overline{O(p) \setminus O}))$, and thus we construct a derivation of $p \text{MUST}_i ta(\varepsilon, \overline{O(p) \setminus O})$
 1333 by applying the rule [IND-RULE]. This requires us to show the following facts,

- 1334 1. $p \parallel ta(s, \overline{O(p) \setminus O}) \longrightarrow$, and
- 1335 2. for each p', r such that $p \parallel ta(s, \overline{O(p) \setminus O}) \xrightarrow{\tau} p' \parallel r$, $p' \text{MUST}_i r$ holds.

1336 To prove (1), we show that an interaction between the server p and the test $ta(s, \overline{O(p) \setminus O})$
 1337 exists. As $\bar{a} \in O(p)$, we have that $p \xrightarrow{\bar{a}}$. Then $\bar{a} \in O(p) \setminus O$ together with (3) ensure
 1338 that $ta(s, \overline{O(p) \setminus O}) \xrightarrow{\bar{a}}$. An application of the rule [S-COM] gives us the required transition
 1339 $p \parallel ta(s, \overline{O(p) \setminus O}) \longrightarrow$.

1340 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
 1341 analysis on the rule used to derive the transition under scrutiny. Recall that the server p is
 1342 stable by hypothesis, and that $ta(s, \overline{O(p) \setminus O})$ is stable thanks to Table 1(1). This means
 1343 that the silent transition must have been derived via rule [S-COM]. Furthermore, Table 1(2)
 1344 implies that the test $ta(s, \overline{O(p) \setminus O})$ does not perform any output. As a consequence, if there
 1345 is an interaction it must be because the test performs an input and becomes r . Table 1(4)
 1346 implies that $\text{GOOD}(r)$, and hence we obtain the required $p' \text{MUST}_i r$ applying rule [AXIOM]. ◀

1347 **Lemma 46** Let $\mathcal{L}_A \in \text{OW}$. For every $p \in A$, $s \in \text{Act}^*$, and every $L, E \subseteq \mathcal{N}$, if $\bar{L} \in \mathcal{A}_{\text{fw}}(p, s)$
 1348 then $p \text{MUST}_i ta(s, E \setminus L)$.

1349 **Proof.** By hypothesis there exists a set $\bar{L} \in \mathcal{A}_{\text{fw}}(p, s)$, *i.e.* for some p' we have $p \xrightarrow{s} p' \xrightarrow{\tau}$
 1350 and $O(p') = \bar{L}$. We have to show that $p \text{MUST}_i ta(s, E \setminus L)$, *i.e.* $p \text{MUST}_i ta(s, E \setminus L)$ implies
 1351 \perp . For convenience, let $X = E \setminus L$.

1352 We proceed by induction on the derivation of the weak transitions $p \xrightarrow{s} p'$. In the base
 1353 case the derivation consists in an application of rule [WT-REFL], which implies that $p = p'$
 1354 and $s = \varepsilon$. We show that there exists no derivation of judgement $p \text{MUST}_i ta(s, X)$. By
 1355 definition, $\neg \text{GOOD}(ta(s, X))$ and thus no tree that ends with [AXIOM] can have $p \text{MUST}_i ta(s, X)$
 1356 as conclusion. The hypotheses ensure that p is stable, and $ta(\varepsilon, X)$ is stable by definition.
 1357 The set of inputs of $ta(\varepsilon, X)$ is X , which prevents an interaction between p and $ta(s, X)$, *i.e.*
 1358 an application of rule [S-COM]. This proves that $p \parallel ta(s, X)$ is stable, thus a side condition
 1359 of [IND-RULE] is false, and the rule cannot be employed to prove $p \text{MUST}_i ta(s, X)$.

1360 In the inductive cases $p \xrightarrow{s} p'$ is derived using either:

- 1361 (i) rule [WT-TAU] such that $p \xrightarrow{\tau} \hat{p} \xrightarrow{s} p'$, or
- 1362 (ii) rule [WT-MU] such that $p \xrightarrow{\mu} \hat{p} \xrightarrow{t} p'$, with $s = \mu.t$.

1363 In the first case, applying the inductive hypothesis requires us to show $\hat{p} \text{MUST}_i ta(s, X)$,
 1364 which is true since $p \text{MUST}_i ta(s, X)$ is preserved by the τ -transitions performed by the server.

1365 In the second case, applying the inductive hypothesis requires us to show $\hat{p} \text{MUST}_i ta(t, X)$.
 1366 Table 1(2) implies that $ta(\mu.t, X) \xrightarrow{\bar{\mu}} ta(t, X)$. Then we derive via [S-COM] the transition
 1367 $p \parallel ta(\mu.t, E) \xrightarrow{\tau} \hat{p} \parallel ta(t, E)$. Since $p \text{MUST}_i ta(s, X)$ is preserved by the interactions
 1368 occurring between the server and the client, which implies $\hat{p} \text{MUST}_i ta(t, X)$ as required. ◀

1369 **Lemma 47** Let $\mathcal{L}_A \in \text{OW}$. For every $p \in A$, $s \in \text{Act}^*$, and every finite set $O \subseteq \bar{\mathcal{N}}$, if $p \Downarrow s$
 1370 then either

- 1371 (i) $p \text{MUST}_i ta(s, \bigcup \overline{\mathcal{A}_{\text{fw}}(p, s) \setminus O})$, or
- 1372 (ii) there exists $\hat{O} \in \mathcal{A}_{\text{fw}}(p, s)$ such that $\hat{O} \subseteq O$.

1373 **Proof.** We proceed by induction on the trace s .

1374 **C.2.0.3 (Base case, $s = \varepsilon$)**

1375 The hypothesis $p \Downarrow \varepsilon$ implies $p \Downarrow_i$ and we continue by induction on the derivation of $p \Downarrow_i$.¹⁰
 1376 In the base case $p \Downarrow_i$ was proven using rule [AXIOM], and hence $p \xrightarrow{\tau}$. We apply Lemma 63
 1377 to obtain either:

- 1378 (i) $p \text{ MUST}_i ta(\varepsilon, \overline{O(p) \setminus O})$, or
 1379 (ii) $O(p) \subseteq O$.

1380 In case (i) we are done. In case (ii), as p is stable we have $\{p' \mid p \xrightarrow{\varepsilon} p' \xrightarrow{\tau}\} = \{p\}$ and
 1381 thus $\mathcal{A}_{fw}(p, \varepsilon) = \{O(p)\}$ and we conclude by letting $\widehat{O} = O(p)$.

1382 In the inductive case $p \Downarrow_i$ was proven using rule [IND-RULE]. We know that $p \xrightarrow{\tau}$, and the
 1383 inductive hypothesis states that for any p' such that $p \xrightarrow{\tau} p'$, either:

- 1384 (a) $p' \text{ MUST}_i ta(\varepsilon, \overline{O(p') \setminus O})$, or
 1385 (b) there exists $\widehat{O} \in \mathcal{A}_{fw}(p', s)$ such that $\widehat{O} \subseteq O$.

1386 It follows that either

- 1387 (\forall) for each $p' \in \{p' \mid p \xrightarrow{\tau} p'\}$, $p' \text{ MUST}_i ta(\varepsilon, \overline{\bigcup \mathcal{A}_{fw}(p', s) \setminus O})$, or
 1388 (\exists) there exists a $p' \in \{p' \mid p \xrightarrow{\tau} p'\}$ and a $\widehat{O} \in \mathcal{A}_{fw}(p', \varepsilon)$ such that $\widehat{O} \subseteq O$,

1389 We discuss the two cases. If (\exists) the argument is straightforward: we pick the existing p' such
 1390 that $p \xrightarrow{\tau} p'$. The definition of $\mathcal{A}_{fw}(-, -)$ ensures that and show that $\mathcal{A}_{fw}(p', \varepsilon) \subseteq \mathcal{A}_{fw}(p, \varepsilon)$,
 1391 and thus we conclude by choosing \widehat{O} .

1392 Case (\forall) requires more work. We are going to show that $p \text{ MUST}_i ta(\varepsilon, \overline{\bigcup \mathcal{A}_{fw}(p, s) \setminus O})$
 1393 holds. To do so we apply the rule [IND-RULE] and we need to show the following facts,

- 1394 (a) $p \Vdash ta(\varepsilon, \overline{\bigcup \mathcal{A}_{fw}(p, s) \setminus O}) \xrightarrow{\tau}$, and
 1395 (b) for each $p' \Vdash r'$ such that $p \Vdash ta(\varepsilon, \overline{\bigcup \mathcal{A}_{fw}(p, s) \setminus O}) \xrightarrow{\tau} p' \Vdash r'$, we have $p' \text{ MUST}_i r'$.

1396 The first requirement follows from the fact that p is not stable. To show the second
 1397 requirement we proceed by case analysis on the transition $p \Vdash ta(\varepsilon, \overline{\bigcup \mathcal{A}_{fw}(p, s) \setminus O}) \xrightarrow{\tau} p' \Vdash r'$.
 1398 As $ta(\varepsilon, \overline{O(p) \setminus O})$ is stable by (1), it can either be due to:

- 1399 1. a τ -transition performed by the server p such that $p \xrightarrow{\tau} p'$, or
 1400 2. an interaction between the server p and the client $ta(\varepsilon, \overline{\bigcup \mathcal{A}_{fw}(p, s) \setminus O})$.

In the first case we apply the first part of the inductive hypothesis to prove that
 $p' \text{ MUST}_i ta(\varepsilon, \overline{\bigcup \mathcal{A}_{fw}(p', s) \setminus O})$, and we conclude via Lemma 61 to get the required

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

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

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

1401 Thanks to Table 1(4) we apply rule [AXIOM] to prove that $p' \text{ MUST}_i r$ and we are done with
 1402 the base case of the main induction on the trace s .

1403 **C.2.0.4 (Inductive case, $s = \mu.s'$)**

1404 By induction on the set $\{p' \mid p \xrightarrow{\mu} p'\}$ and an application of the inductive hypothesis we
 1405 know that either:

- 1406 (i) there exists $p' \in \{p' \mid p \xrightarrow{\mu} p'\}$ and $\widehat{O} \in \mathcal{A}_{fw}(p', s')$ such that $\widehat{O} \subseteq O$, or
 1407 (ii) for each $p' \in \{p' \mid p \xrightarrow{\mu} p'\}$ we have that $p' \text{ MUST}_i ta(s', \overline{\bigcup \mathcal{A}_{fw}(p', s')})$.

¹⁰ Recall that the definition of \Downarrow_i is in Equation (int-preds)

$$\begin{array}{c}
 \text{[MSET-NOW]} \quad \frac{\text{GOOD}(r)}{X \text{ MUST}_{\text{aux}} r} \\
 \\
 \text{[MSET-STEP]} \quad \frac{\begin{array}{l}
 \neg \text{GOOD}(r) \quad \forall X'. X \xrightarrow{\tau} X' \text{ implies } X' \text{ MUST}_{\text{aux}} r \\
 \forall p \in X. p \parallel r \xrightarrow{\tau} \quad \forall r'. r \xrightarrow{\tau} r' \text{ implies } X \text{ MUST}_{\text{aux}} r' \\
 \forall X', \mu \in \text{Act}^*. X \xrightarrow{\bar{\mu}} X' \text{ and } r \xrightarrow{\mu} r' \text{ imply } X' \text{ MUST}_{\text{aux}} r'
 \end{array}}{X \text{ MUST}_{\text{aux}} r}
 \end{array}$$

■ **Figure 12** Rules to define inductively the predicate MUST_{aux} .

1408 In the first case, the inclusion $\mathcal{A}_{\text{fw}}(p', s') \subseteq \mathcal{A}_{\text{fw}}(p, \mu.s')$ and $\widehat{O} \in \mathcal{A}_{\text{fw}}(p', s')$ imply
 1409 $\widehat{O} \in \mathcal{A}_{\text{fw}}(p, s)$ and we are done.

1410 In the second case, we show $p \text{ MUST}_i ta(s, \bigcup \overline{\mathcal{A}_{\text{fw}}(p, s)})$ by case analysis on the action μ ,
 1411 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 $p \xrightarrow{a} p' \xrightarrow{\bar{a}} p$. An application of Table 1(2) gives us the following transition,

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

By an application of Lemma 45 it is enough to show

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

1412 to obtain the required $p \text{ MUST}_i ta(a.s', \bigcup \overline{\mathcal{A}_{\text{fw}}(p, a.s') \setminus O})$.

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

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

1413 We apply Lemma 62 together with (ii) to obtain $p \text{ MUST}_i ta(\bar{a}.s', \bigcup \overline{\mathcal{A}_{\text{fw}}(p', s') \setminus O})$. Again,
 1414 Lemma 61 together with the inclusion $\mathcal{A}_{\text{fw}}(p', s') \subseteq \mathcal{A}_{\text{fw}}(p, \bar{a}.s')$ ensures the required
 1415 $p \text{ MUST}_i ta(s, \bigcup \overline{\mathcal{A}_{\text{fw}}(p, s) \setminus O})$.
 1416 ◀

1417 **D** Soundness

1418 In this section we prove the converse of Proposition 49, i.e. that \preceq_{AS} is included in $\overline{\preceq}_{\text{MUST}}$.
 1419 We remark immediately that a naïve reasoning does not work. Fix two servers p and q such
 1420 that $p \preceq_{\text{AS}} q$. We need to prove that for every client r , if $p \text{ MUST}_i r$ then $p \text{ MUST}_i r$. The
 1421 reasonable first proof attempt consisting in proceeding by induction on $p \text{ MUST}_i r$ fails, as
 1422 demonstrated by the following example.

1423 ► **Example 64.** Consider the two servers $p = \tau.(\bar{a} \parallel \bar{b}) + \tau.(\bar{a} \parallel \bar{c})$ and $q = \bar{a} \parallel (\tau.\bar{b} + \tau.\bar{c})$
 1424 of Equation (4). Fix a client r such that $p \text{ MUST}_i r$. Rule induction yields the following
 1425 inductive hypothesis:

$$1426 \quad \forall p', q'. p \parallel r \xrightarrow{\tau} p' \parallel r' \wedge p' \preceq_{\text{AS}} q' \Rightarrow q' \text{ MUST}_i r'.$$

1427 In the proof of $q \text{ MUST}_i r$ we have to consider the case where there is a communication
 1428 between q and r such that, for instance, $q \xrightarrow{\bar{a}} \tau.\bar{b} + \tau.\bar{c}$ and $r \xrightarrow{a} r'$. In that case, we need
 1429 to show that $\tau.\bar{b} + \tau.\bar{c} \text{ MUST}_i r'$. Ideally, we would like to use the inductive hypothesis. This
 1430 requires us to exhibit a p' such that $p \parallel r \xrightarrow{\tau} p' \parallel r'$ and $p' \preceq_{\text{acc}} \tau.\bar{b} + \tau.\bar{c}$. However, note
 1431 that there is no way to derive $p \parallel r \xrightarrow{\tau} p' \parallel r'$, because $p \xrightarrow{\bar{a}}$. The inductive hypothesis
 1432 thus cannot be applied, and the naïve proof does not go through. ◀

1433 This example suggests that defining an auxiliary predicate MUST_{aux} in some sense equivalent
 1434 to MUST_i , but that uses explicitly *weak* outputs of servers, should be enough to prove that \preceq_{AS}
 1435 is sound with respect to \approx_{MUST} . Unfortunately, though, there is an additional nuisance to
 1436 tackle: server nondeterminism.

1437 ▶ **Example 65.** Assume that we defined the predicate MUST_i using weak transitions on the
 1438 server side for the case of communications. Recall the argument put forward in the previous
 1439 example. The inductive hypothesis now becomes the following:

1440 For every p', q', μ such that $p \xrightarrow{\mu} p'$ and $r \xrightarrow{\mu} r'$, $p' \preceq_{\text{AS}} q'$ implies $q' \text{ MUST}_i r'$.

1441 To use the inductive hypothesis we have to choose a p' such that $p \xrightarrow{\bar{a}} p'$ and $p' \preceq_{\text{AS}} \tau.\bar{b} + \tau.\bar{c}$.
 1442 This is still not enough for the entire proof to go through, because (modulo further τ -moves)
 1443 the particular p' we pick has to be related also to either \bar{b} or \bar{c} . It is not possible to find
 1444 such a p' , because the two possible candidates are either \bar{b} or \bar{c} ; neither of which can satisfy
 1445 $p' \preceq_{\text{AS}} \tau.\bar{b} + \tau.\bar{c}$, as the right-hand side has not committed to a branch yet.

1446 If instead of a single state p in the novel definition of MUST_i we used a set of states and a
 1447 suitable transition relation, the choice of either \bar{b} or \bar{c} will be suitably delayed. It suffices for
 1448 instance to have the following states and transitions: $\{p\} \xrightarrow{\bar{a}} \{\bar{b}, \bar{c}\}$. ◀

1449 Now that we have motivated the main intuitions behind the definition of our novel
 1450 auxiliary predicate MUST_{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

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

1451 Essentially we lift the standard notion of state derivative to sets of states. We construct
 1452 the LTS $\langle \mathcal{P}^+(A), \text{Act}_\tau, \longrightarrow \rangle$ by letting $X \xrightarrow{\alpha} D(\alpha, X)$ whenever $D(\alpha, X) \neq \emptyset$. Similarly, we
 1453 have $X \xrightarrow{\alpha} WD(\alpha, X)$ whenever $WD(\alpha, X) \neq \emptyset$. This construction is standard [42, 20, 21]
 1454 and goes back to the determinisation of nondeterministic automata.

1455 Let MUST_{aux} be defined via the rules in Figure 12. This predicate let us reason on MUST_i
 1456 via sets of servers, in the following sense.

1457 ▶ **Lemma 66.** *For every LTS $\mathcal{L}_A, \mathcal{L}_B$ and every $X \in \mathcal{P}^+(A)$, we have that $X \text{ MUST}_{\text{aux}} r$ if*
 1458 *and only if for every $p \in X. p \text{ MUST}_i r$.*

1459 To lift the predicates \preceq_{cnv} and \preceq_{acc} to sets of servers, we let $\mathcal{A}_{\text{fw}}(X, s) = \{O \mid \exists p \in$
 1460 $X. O \in \mathcal{A}_{\text{fw}}(p, s)\}$, and for every finite $X \in \mathcal{P}^+(A)$, we write $X \downarrow$ to mean $\forall p \in X. p \downarrow$, we
 1461 write $X \downarrow s$ to mean $\forall p \in X. p \downarrow s$, and let

- 1462 ■ $X \preceq_{\text{cnv}}^{\text{set}} q$ to mean $\forall s \in \text{Act}^*$, if $X \downarrow s$ then $q \downarrow s$,
- 1463 ■ $X \preceq_{\text{acc}}^{\text{set}} q$ to mean $\forall s \in \text{Act}^*$, $X \downarrow s$ implies $\mathcal{A}_{\text{fw}}(X, s) \ll \mathcal{A}_{\text{fw}}(q, s)$,
- 1464 ■ $X \preceq_{\text{AS}}^{\text{set}} q$ to mean $X \preceq_{\text{cnv}}^{\text{set}} q \wedge X \preceq_{\text{acc}}^{\text{set}} q$.

XX:42 Constructive characterisations of the must-preorder for asynchrony

1465 These definitions imply immediately the following equivalences, $\{p\} \preceq_{\text{acc}}^{\text{set}} q \iff p \preceq_{\text{cnv}} q$,
 1466 $\{p\} \preceq_{\text{cnv}}^{\text{set}} q \iff p \preceq_{\text{acc}} q$ and thereby the following lemma.

1467 ► **Lemma 67.** *For every LTS $\mathcal{L}_A, \mathcal{L}_B, p \in A, q \in B, p \preceq_{\text{AS}} q$ if and only if $\{p\} \preceq_{\text{AS}}^{\text{set}} q$.*

1468 The preorder $\preceq_{\text{AS}}^{\text{set}}$ is preserved by τ -transitions on its right-hand side, and by visible
 1469 transitions on both sides. We reason separately on the two auxiliary preorders $\preceq_{\text{cnv}}^{\text{set}}$ and $\preceq_{\text{acc}}^{\text{set}}$.
 1470 We need one further notion.

1471 ► **Lemma 68.** *Let $\mathcal{L}_A, \mathcal{L}_B \in \text{OW}$. For every set $X \in \mathcal{P}^+(A)$, and $q \in B$, such that $X \preceq_{\text{cnv}}^{\text{set}} q$,*

- 1472 1. $q \xrightarrow{\tau} q'$ implies $X \preceq_{\text{cnv}}^{\text{set}} q'$,
- 1473 2. $X \downarrow_i, X \xrightarrow{\mu} X'$ and $q \xrightarrow{\mu} q'$ imply $X' \preceq_{\text{cnv}}^{\text{set}} q'$.

1474 ► **Lemma 69.** *Let $\mathcal{L}_A, \mathcal{L}_B \in \text{OW}$. For every $X, X' \in \mathcal{P}^+(A)$ and $q \in B$, such that $X \preceq_{\text{acc}}^{\text{set}} q$,
 1475 then*

- 1476 1. $q \xrightarrow{\tau} q'$ implies $X \preceq_{\text{acc}}^{\text{set}} q'$,
- 1477 2. if $X \downarrow_i$ then for every $\mu \in \text{Act}$, every q' and X' such that $q \xrightarrow{\mu} q'$ and $X \xrightarrow{\mu} X'$ we have
 1478 $X' \preceq_{\text{acc}}^{\text{set}} q'$.

1479 The main technical work for the proof of soundness is carried out by the next lemma.

1480 ► **Lemma 70.** *Let $\mathcal{L}_A, \mathcal{L}_B \in \text{OW}$ and $\mathcal{L}_C \in \text{OF}$. For every set of servers $X \in \mathcal{P}^+(A)$,
 1481 server $q \in B$ and client $r \in C$, if $X \text{ MUST}_{\text{aux}} r$ and $X \preceq_{\text{AS}}^{\text{set}} q$ then $q \text{ MUST}_i r$.*

1482 ► **Proposition 71 (Soundness).** *For every $\mathcal{L}_A, \mathcal{L}_B \in \text{OF}$ and servers $p \in A, q \in B$, if
 1483 $\text{FW}(p) \preceq_{\text{AS}} \text{FW}(q)$ then $p \sqsubset_{\text{MUST}} q$.*

1484 **Proof.** Lemma 14 ensures that the result follows if we prove that $\text{FW}(p) \sqsubset_{\text{MUST}} \text{FW}(q)$. Fix
 1485 a client r such that $\text{FW}(p) \text{ MUST}_i r$. Lemma 70 implies the required $\text{FW}(q) \text{ MUST}_i r$, if we
 1486 show that

- 1487 (i) $\{\text{FW}(p)\} \text{ MUST}_{\text{aux}} r$, and that
- 1488 (ii) $\{\text{FW}(p)\} \preceq_{\text{AS}}^{\text{set}} \text{FW}(q)$.

1489 The first fact follows from the assumption that $\text{FW}(p) \text{ MUST}_i r$ and Lemma 66 applied to the
 1490 singleton $\{\text{FW}(p)\}$. The second fact follows from the hypothesis that $\text{FW}(p) \preceq_{\text{AS}} \text{FW}(q)$
 1491 and Lemma 67. ◀

1492 D.1 Technical results to prove soundness

1493 We now discuss the proofs of the main technical results behind Proposition 71. The predicate
 1494 MUST_{aux} is monotonically decreasing with respect to its first argument, and it enjoys properties
 1495 analogous to the ones of MUST_i that have been shown in Lemma 35 and Lemma 36.

1496 ► **Lemma 72.** *For every LTS $\mathcal{L}_A, \mathcal{L}_B$ and every set $X_1 \subseteq X_2 \subseteq A$, client $r \in B$, if
 1497 $X_2 \text{ MUST}_{\text{aux}} r$ then $X_1 \text{ MUST}_{\text{aux}} r$.*

1498 ► **Lemma 73.** *Let $\mathcal{L}_A \in \text{OW}$ and $\mathcal{L}_B \in \text{OF}$. For every set $X \in \mathcal{P}^+(A)$, client $r \in B$, if
 1499 $\neg \text{GOOD}(r)$ and $X \text{ MUST}_{\text{aux}} r$ then $X \downarrow_i$.*

1500 ► **Lemma 74.** *For every $\mathcal{L}_A, \mathcal{L}_B$, every set $X_1, X_2 \in \mathcal{P}^+(A)$, and client $r \in B$, if $X_1 \text{ MUST}_{\text{aux}}$
 1501 r and $X_1 \xrightarrow{\varepsilon} X_2$ then $X_2 \text{ MUST}_{\text{aux}} r$.*

1502 ► **Lemma 75.** *For every LTS $\mathcal{L}_A, \mathcal{L}_B$ and every $X \in \mathcal{P}^+(A)$ and $r \in B$, if $X \text{ MUST}_{\text{aux}} r$
 1503 then for every X' such that*

- 1504 (a) If $X \xrightarrow{\varepsilon} X'$ then $X' \text{ MUST}_{\text{aux}} r$,

1505 (b) For any $\mu \in \text{Act}$ and client r' , if $X \xrightarrow{\mu} X'$, $r \xrightarrow{\bar{\mu}} r'$ and $\neg \text{GOOD}(r)$, then $X' \text{ MUST}_{\text{aux}} r'$.

1506 ▶ **Lemma 76.** 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
1507 $p \in X$ we have that $p \text{ MUST}_i r$, then $X \text{ MUST}_{\text{aux}} r$.

1508 ▶ **Lemma 77.** Let $\mathcal{L}_A, \mathcal{L}_B \in \text{OW}$ and $\mathcal{L}_C \in \text{OF}$. For every $X \in \mathcal{P}^+(A)$ and $q \in B$ such
1509 that $X \preceq_{\text{AS}}^{\text{set}} q$, for every $r \in C$ if $\neg \text{GOOD}(r)$ and $X \text{ MUST}_{\text{aux}} r$ 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

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

1510 The hypotheses $\neg \text{GOOD}(r)$ and $X \text{ MUST}_{\text{aux}} r$ together with Lemma 73 imply $X \downarrow_i$ and thus
1511 $X \downarrow \varepsilon$. The hypothesis $X \preceq_{\text{acc}}^{\text{set}} q$ with $s = \varepsilon$, gives us a p' such that $p \xrightarrow{\varepsilon} p' \xrightarrow{\tau}$ and
1512 $O(p') \subseteq O(q)$. By definition there exists the weak silent trace $X \Longrightarrow X'$ for some set X'
1513 such that $\{p'\} \subseteq X'$. The hypothesis $X \text{ MUST}_{\text{aux}} r$ together with Lemma 74 and Lemma 72
1514 ensure that $\{p'\} \text{ MUST}_{\text{aux}} r$.

1515 As $\neg \text{GOOD}(r)$, $\{p'\} \text{ MUST}_{\text{aux}} r$ must have been derived using rule [IND-RULE] which implies
1516 that $p' \parallel r \xrightarrow{\tau}$. As both r is stable by assumption, and p' is stable by definition, this
1517 τ -transition must have been derived using [S-COM], and so $p' \xrightarrow{\mu}$ and $r \xrightarrow{\bar{\mu}}$ for some $\mu \in \text{Act}$.
1518 Now we distinguish whether μ is an input or an output. In the first case μ is an input. Since
1519 $\mathcal{L}_B \in \text{OW}$ we use the INPUT-BOOMERANG axiom to prove $q \xrightarrow{\mu}$, and thus $q \parallel r \xrightarrow{\tau}$ via
1520 rule [S-COM]. In the second case μ is an output, and so the inclusion $O(p') \subseteq O(q)$ implies
1521 that $q \xrightarrow{\mu}$, and so we conclude again applying rule [S-COM]. ◀

1522 ▶ **Lemma 78.** Let $\mathcal{L}_A, \mathcal{L}_B \in \text{OW}$. For every $X \in \mathcal{P}^+(A)$ and $q, q' \in B$, such that $X \preceq_{\text{AS}}^{\text{set}} q$,
1523 then for every $\mu \in \text{Act}$, if $X \downarrow \mu$ and $q \xrightarrow{\mu} q'$ then $X \xrightarrow{\mu}$.

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

$$q \xrightarrow{\mu} q' \xrightarrow{\varepsilon} q'' \xrightarrow{\tau}$$

1524 and so $\mathcal{A}_{\text{fw}}(q, \mu, \rightarrow_B) \neq \emptyset$. An application of the hypothesis $X \preceq_{\text{acc}}^{\text{set}} q$ implies that there
1525 exists a set $\tilde{O} \in \mathcal{A}_{\text{fw}}(X, \mu, \rightarrow_A)$, and thus there exist two servers $p' \in X$ and p'' such that
1526 $p' \xrightarrow{\mu} p'' \xrightarrow{\tau}$. Since $p' \in X$ it follows that $X \xrightarrow{\mu}$. ◀

1527 **Lemma 68** Let $\mathcal{L}_A, \mathcal{L}_B \in \text{OW}$. For every set $X \in \mathcal{P}^+(A)$, and $q \in B$, such that $X \preceq_{\text{cnv}}^{\text{set}} q$
1528 then

- 1529 1. $q \xrightarrow{\tau} q'$ implies $X \preceq_{\text{cnv}}^{\text{set}} q'$,
- 1530 2. if $X \downarrow_i$ and $q \xrightarrow{\mu} q'$ then for every set $X \xrightarrow{\mu} X'$ we have that $X' \preceq_{\text{cnv}}^{\text{set}} q'$.

1531 **Proof.** We first prove part (1). Let us fix a trace s such that $X \downarrow s$. We must show $q' \downarrow s$.
1532 An application of the hypothesis $X \preceq_{\text{cnv}}^{\text{set}} q$ ensures $q \downarrow s$. From the transition $q \xrightarrow{\tau} q'$ and
1533 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 $q \xrightarrow{\mu} q'$, the required $q' \downarrow s$
follows from $q \downarrow \mu.s$. Thanks to the hypothesis $X \preceq_{\text{cnv}}^{\text{set}} q$ it suffices to show that $X \downarrow \mu.s'$,
i.e. that

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

1534 Fix a server $p \in X$. We must show that

- 1535 1. $p \downarrow_i$ and that
- 1536 2. for any p' such that $p \xrightarrow{\mu} p'$ we have $p' \downarrow s$.

XX:44 Constructive characterisations of the must-preorder for asynchrony

1537 The first requirement follows from the hypothesis $X \downarrow_i$. The second requirement follows from
 1538 the transition $p \xrightarrow{\mu} p'$, from the assumption $X' \downarrow s$, and the hypothesis that $X \xrightarrow{\mu} X'$,
 1539 which ensures that $p' \in X'$ and thus by definition of $X' \downarrow s$ that $p' \downarrow s$. \blacktriangleleft

1540 **Lemma 69** Let $\mathcal{L}_A, \mathcal{L}_B \in \text{OW}$. For every $X, X' \in \mathcal{P}^+(A)$ and $q \in B$, such that $X \preceq_{\text{acc}}^{\text{set}} q$,
 1541 then

- 1542 1. $q \xrightarrow{\tau} q'$ implies $X \preceq_{\text{acc}}^{\text{set}} q'$,
- 1543 2. for every $\mu \in \text{Act}$, if $X \downarrow_i$, then for every $q \xrightarrow{\mu} q'$ and set $X \xrightarrow{\mu} X'$ we have $X' \preceq_{\text{acc}}^{\text{set}} q'$.

1544 **Proof.** To prove part (1) fix a trace $s \in \text{Act}^*$ such that $X \downarrow s$. We have to explain why
 1545 $\mathcal{A}_{\text{fw}}(X, s) \ll \mathcal{A}_{\text{fw}}(q', s)$. By unfolding the definitions, this amounts to showing that

$$1546 \quad \forall O \in \mathcal{A}_{\text{fw}}(q', s). \exists p_{\text{attaboy}} \in X. \exists \widehat{O} \in \mathcal{A}_{\text{fw}}(p_{\text{attaboy}}, s). \widehat{O} \subseteq O \quad (\star)$$

1547 Fix a set $O \in \mathcal{A}_{\text{fw}}(q', s)$. By definition there exists some q'' such that $q' \xrightarrow{s} q'' \xrightarrow{\tau}$,
 1548 and that $O = O(q'')$. The definition of $\mathcal{A}_{\text{fw}}(-, -)$ and the silent move $q \xrightarrow{\tau} q'$ ensures that
 1549 $O \in \mathcal{A}_{\text{fw}}(q, s)$. The hypothesis $X \preceq_{\text{acc}}^{\text{set}} q$ and that $X \downarrow s$ now imply that $\mathcal{A}_{\text{fw}}(X, s) \ll \mathcal{A}_{\text{fw}}(q, s)$,
 1550 which together with $O \in \mathcal{A}_{\text{fw}}(q, s)$ implies exactly Equation (\star) .

1551 We now prove part (2). To show $X' \preceq_{\text{acc}}^{\text{set}} q'$ fix a trace $s \in \text{Act}^*$ such that $X' \downarrow s$.

1552 We have to explain why $\mathcal{A}_{\text{fw}}(X, s) \ll \mathcal{A}_{\text{fw}}(q', s)$. By unfolding the definitions we obtain
 1553 our aim,

$$1554 \quad \forall O \in \mathcal{A}_{\text{fw}}(q', s). \exists p_{\text{attaboy}} \in X'. \exists \widehat{O} \in \mathcal{A}_{\text{fw}}(p_{\text{attaboy}}, s). \widehat{O} \subseteq O \quad (\star\star)$$

1555 To begin with, we prove that $X \downarrow \mu.s$. Since $X \xrightarrow{\mu} X'$ we know that $X \xrightarrow{\mu} X'$. This,
 1556 together with $X \downarrow_i$ and $X' \downarrow s$ implies the convergence property we are after.

1557 Now fix a set $O \in \mathcal{A}_{\text{fw}}(q', s)$. Thanks to the transition $q \xrightarrow{\mu} q'$, we know that $O \in$
 1558 $\mathcal{A}_{\text{fw}}(q, \mu.s)$. The hypothesis $X \preceq_{\text{acc}}^{\text{set}} q$ together with $X \downarrow \mu.s$ implies that there exists
 1559 a server $p_{\text{attaboy}} \in X$ such that there exists an $\widehat{O} \in \mathcal{A}_{\text{fw}}(p_{\text{attaboy}}, \mu.s)$. This means that
 1560 $p_{\text{attaboy}} \xrightarrow{\mu} p'_{\text{attaboy}}$ and that $\widehat{O} \in \mathcal{A}_{\text{fw}}(p'_{\text{attaboy}}, s)$. Since $X \xrightarrow{\mu} X'$ we know that $p'_{\text{attaboy}} \in X'$
 1561 and this concludes the argument. \blacktriangleleft

1562

1563 **► Lemma 79.** For every $\mathcal{L}_A \in \text{OW}$, $\mathcal{L}_B \in \text{OF}$, every set of processes $X \in \mathcal{P}^+(A)$, every
 1564 $r \in B$, and every $\mu \in \text{Act}$, if $X \text{ MUST}_{\text{aux}} r$, $\neg \text{GOOD}(r)$ and $r \xrightarrow{\mu}$ then $X \downarrow \bar{\mu}$.

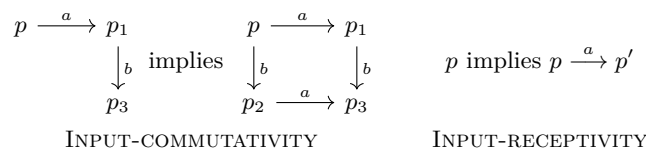
1565 **Lemma 70** Let $\mathcal{L}_A, \mathcal{L}_B \in \text{OW}$ and $\mathcal{L}_C \in \text{OF}$. For every set of processes $X \in \mathcal{P}^+(A)$, server
 1566 $q \in B$ and client $r \in C$, if $X \text{ MUST}_{\text{aux}} r$ and $X \preceq_{\text{AS}}^{\text{set}} q$ then $q \text{ MUST}_i r$.

1567 **Proof.** We proceed by induction on the derivation of $X \text{ MUST}_{\text{aux}} r$. In the base case, $\text{GOOD}(r)$
 1568 so we trivially derive $q \text{ MUST}_i r$. In the inductive case the proof of the hypothesis $X \text{ MUST}_{\text{aux}} r$
 1569 terminates with an application of [MSET-STEP]. Since $\neg \text{GOOD}(r)$, we show the result applying
 1570 [IND-RULE]. This requires us to prove that

- 1571 (1) $q \parallel r \xrightarrow{\tau}$, and that
- 1572 (2) for all q', r' such that $q \parallel r \xrightarrow{\tau} q' \parallel r'$ we have $q' \text{ MUST}_i r'$.

1573 The first fact is a consequence of Lemma 77, which we can apply because $\neg \text{GOOD}(r)$ and
 1574 thanks to the hypothesis $X \preceq_{\text{acc}}^{\text{set}} q$ and $X \text{ MUST}_{\text{aux}} r$. To prove the second fact, fix a transition
 1575 $q \parallel r \xrightarrow{\tau} q' \parallel r'$. We have to explain why the following properties are true,

- 1576 (a) for every $q'. q \xrightarrow{\tau} q'$ implies $q' \text{ MUST}_i r$,
- 1577 (b) for every $r. r \xrightarrow{\tau} r'$ implies $q \text{ MUST}_i r'$,
- 1578 (c) for every q', r' and $\mu \in \text{Act}$, $q \xrightarrow{\mu} q'$ and $r \xrightarrow{\bar{\mu}} r'$ imply $q' \text{ MUST}_i r'$.



■ **Figure 13** The INPUT-RECEPTIVITY axiom of [92], and our version of input-commutativity, which allows swapping only consecutive inputs.

1579 First, note that $X \text{ MUST}_{\text{aux}} r$, $\neg \text{GOOD}(r)$, and Lemma 73 imply $X \downarrow$. Second, the inductive
1580 hypotheses state that for every r' , non-empty set X' , and q the following facts hold,

1581 (i) $X \xrightarrow{\tau} X'$ and $X' \preceq_{\text{AS}}^{\text{set}} q$ implies $q \text{ MUST}_i r$,

1582 (ii) $r \xrightarrow{\tau} r'$ and $X \preceq_{\text{AS}}^{\text{set}} q$ implies $q \text{ MUST}_i r'$,

1583 (iii) for every $\mu \in \text{Act}$, $X \xrightarrow{\mu} X'$ and $r \xrightarrow{\mu} r'$, and $X' \preceq_{\text{AS}}^{\text{set}} q$ implies $q \text{ MUST}_i r'$.

1584 To prove (a) we use $X \downarrow$ and the hypothesis $X \preceq_{\text{cnv}}^{\text{set}} q$ to obtain $q \downarrow_i$. A rule induction on
1585 $q \downarrow_i$ now suffices: in the base case (a) is trivially true and in the inductive case (a) follows
1586 from Lemma 68(1) and Lemma 69(1), and the inductive hypothesis.

1587 The requirement (b) follows directly from the hypothesis $X \preceq_{\text{AS}}^{\text{set}} q$ and part (ii) of the
1588 inductive hypothesis.

1589 To see why (c) holds, fix an action μ such that $q \xrightarrow{\mu} q'$ and $r \xrightarrow{\bar{\mu}} r'$. Since $\neg \text{GOOD}(r)$
1590 Lemma 79 implies that $X \downarrow \mu$, and so Lemma 78 proves that $X \xrightarrow{\mu}$. In turn this implies
1591 that there exists a X' such that $X \xrightarrow{\mu} X'$, and thus Lemma 68(2) and Lemma 69(2) prove
1592 that $X' \preceq_{\text{AS}}^{\text{set}} q'$ holds, and (iii) ensures the result, *i.e.* that $q' \text{ MUST}_i r'$. ◀

1593 E Traces in normal form and further alternative characterisations

1594 As hinted at in the main body of the paper, we characterise the MUST-preorder using only
1595 the causal order of actions on traces. In this appendix we outline the necessary constructions
1596 and our reasoning. All the results are mechanised.

Let $\text{nf} : \text{Act}^* \rightarrow (MI \times MO)^*$ be the function

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

which is defined inductively in Figure 14. 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

$$\text{nf}(\overline{cabddae\bar{f}e}) = (\{c, a\}, \{\bar{b}, \bar{d}, \bar{d}\}), (\{a\}, \{\bar{e}, \bar{f}\}), (\{e\}, \emptyset)$$

1597 Let σ range over the set $(MI \times MO)^*$. We say that σ is a trace in *normal form*, and we
1598 write $p \xrightarrow{\sigma} q$, whenever there exists $s \in \text{Act}^*$ such that $p \xrightarrow{s} q$ and $\text{nf}(s) = \sigma$.

1599 We lift in the obvious way the predicates \preceq_{cnv} , \preceq_{acc} , and $\preceq_{\text{acc}}^{\text{fw}}$ to traces in nformal forms.

1600 For every $\mathcal{L}_A, \mathcal{L}_B$ and $p \in A, q \in B$ let

1601 ■ $p \preceq_{\text{asyn}}^{\text{cnv}} q$ to mean $\forall \sigma \in (MI \times MO)^*. p \downarrow \sigma$ implies $q \downarrow \sigma$,

1602 ■ $p \preceq_{\text{asyn}}^{\text{acc}} q$ to mean $\forall \sigma \in (MI \times MO)^*. p \downarrow \sigma$ implies $\mathcal{A}_{\text{fw}}(p, \sigma) \ll \mathcal{A}_{\text{fw}}(q, \sigma)$,

$$\begin{aligned}
 \text{nf}(\varepsilon) &= \varepsilon \\
 \text{nf}(s) &= \text{nf}'(s, \emptyset, \emptyset) \\
 \text{nf}'(\varepsilon, I, M) &= (I, M) \\
 \text{nf}'(\bar{a}.b.s, I, M) &= (I, \{a\} \uplus M), \text{nf}'(s, \{b\}, \emptyset) \\
 \text{nf}'(a.s, I, M) &= \text{nf}'(s, \{a\} \uplus I, M) \\
 \text{nf}'(\bar{a}.s, I, M) &= \text{nf}'(s, I, \{b\} \uplus M)
 \end{aligned}$$

■ **Figure 14** Definition of the trace normalization function nf

1603 ■ $p \preceq_{\text{MS}}^{\text{asyn}} q$ to mean $\forall \sigma \in (MI \times MO)^*. p \Downarrow \sigma$ implies that if $\forall L. (p \text{ after } \sigma, \longrightarrow_A) \text{ MUST } L$
 1604 then $(q \text{ after } \sigma, \longrightarrow_B) \text{ MUST } L$

1605 ► **Definition 80.** *Let*

1606 ■ $p \preceq_{\text{AS}}^{\text{NF}} q$ whenever $q \preceq_{\text{asyn}}^{\text{cnv}} p \wedge p \preceq_{\text{asyn}}^{\text{acc}} q$,
 1607 ■ $p \preceq_{\text{MS}}^{\text{NF}} q$ whenever $q \preceq_{\text{asyn}}^{\text{cnv}} p \wedge p \preceq_{\text{MS}}^{\text{asyn}} q$. ■

1608 If an LTS is of forwarders, *i.e.* $\mathcal{L} \in \text{OW}$, the transition relation \longrightarrow is *input-receptive*
 1609 (Axiom (IB4), Table 2 of [92]), and in Lemma 81 we prove that it enjoys a restricted version
 1610 of INPUT-COMMUTATIVITY, and that so does its weak version. Sequences of input actions
 1611 $s \in \mathcal{N}^*$ enjoy a form of diamond property in \Longrightarrow . The crucial fact pertains consecutive input
 1612 actions.

1613 ► **Lemma 81.** *For every $\mathcal{L}_A \in \text{OW}$, every $p, q \in A$ and every $a, b \in \mathcal{N}$, if $p \xrightarrow{a.b} q$ then*
 1614 $p \xrightarrow{b.a} \cdot \simeq q$.

1615 Lemma 81, together with an induction on traces, allows us to prove that nf preserves
 1616 convergence and acceptance sets.

1617 ► **Lemma 82.** *For every $\mathcal{L}_A \in \text{OW}$, every $p \in A$ and every $s \in \text{Act}^*$ and we have that*

- 1618 1. $p \xrightarrow{s} q$ iff $p \xrightarrow{\text{nf}(s)} \cdot \simeq q$, and if the first trace does not pass through a successful state then
 1619 the normal form does not either,
- 1620 2. $p \Downarrow \text{nf}(s)$ iff $p \Downarrow s$,
- 1621 3. $\mathcal{A}_{\text{fw}}(p, s) = \mathcal{A}_{\text{fw}}(p, \text{nf}(s))$.

1622 We thereby obtain two other characterisations of the contextual preorder $\sqsubseteq_{\text{MUST}}$: Theo-
 1623 rem 17 and Lemma 82 ensure that the preorders $\sqsubseteq_{\text{MUST}}$, $\preceq_{\text{AS}}^{\text{NF}}$, and $\preceq_{\text{MS}}^{\text{NF}}$ coincide.

1624 ► **Corollary 83.** *For every $\mathcal{L}_A, \mathcal{L}_B \in \text{OF}$, every $p \in A$ and $q \in B$, the following facts are*
 1625 *equivalent:*

- 1626 1. $p \sqsubseteq_{\text{MUST}} q$,
- 1627 2. $\text{FW}(p) \preceq_{\text{AS}}^{\text{NF}} \text{FW}(q)$,
- 1628 3. $\text{FW}(p) \preceq_{\text{MS}}^{\text{NF}} \text{FW}(q)$.

1629 **F** Asynchronous CCS

1630 Here we recall the syntax and the LTS of asynchronous CCS, or **ACCS** for short, a version of
 1631 CCS where outputs have no continuation and sum is restricted to input- and τ -guards. This

$$\begin{array}{ll}
\text{[INPUT]} & \frac{}{a.p \xrightarrow{a} p} & \text{[TAU]} & \frac{}{\tau.p \xrightarrow{\tau} p} \\
\text{[MB-OUT]} & \frac{}{\bar{a} \xrightarrow{\bar{a}} 0} & \text{[UNF]} & \frac{}{\text{rec } x.p \xrightarrow{\tau} p[\text{rec } x.p/x]} \\
\text{[SUM-L]} & \frac{p \xrightarrow{\alpha} p'}{p + q \xrightarrow{\alpha} p'} & \text{[SUM-R]} & \frac{q \xrightarrow{\alpha} q'}{p + q \xrightarrow{\alpha} q'} \\
\text{[PAR-L]} & \frac{p \xrightarrow{\alpha} p'}{p \parallel q \xrightarrow{\alpha} p' \parallel q} & \text{[PAR-R]} & \frac{q \xrightarrow{\alpha} q'}{p \parallel q \xrightarrow{\alpha} p \parallel q'} \\
\text{[COM]} & \frac{p \xrightarrow{\mu} p' \quad q \xrightarrow{\bar{\mu}} q'}{p \parallel p' \xrightarrow{\tau} q \parallel q'} & &
\end{array}$$

■ **Figure 15** The LTS of processes. The meta-variables are $a \in \mathcal{N}$, $\mu \in \text{Act}$, $\alpha \in \text{Act}_\tau$.

1632 calculus, which is inspired by the variant of the asynchronous π -calculus considered by [4, 5]
 1633 for their study of asynchronous bisimulation, was first investigated by [92], and subsequently
 1634 resumed by other authors such as [24]. Different asynchronous variants of CCS were studied
 1635 in the same frame of time by [82], whose calculus included output prefixing and operators
 1636 from ACP, and by [39], whose calculus TACCS included asynchronous output prefixing and
 1637 featured two forms of choice, internal and external, in line with previous work on testing
 1638 semantics [78].

1639 The syntax of terms is given in Equation (3). As usual, $\text{rec } x.p$ binds the variable x in p ,
 1640 and we use standard notions of free variables, open and closed terms. Processes, ranged over
 1641 by p, q, r, \dots are *closed* terms. The operational semantics of processes is given by the LTS
 1642 $\langle \text{ACCS}, \text{Act}_\tau, \longrightarrow \rangle$ specified by the rules in Figure 15.

1643 The prefix $a.p$ represents a *blocked* process, which waits to perform the input a , *i.e.*
 1644 to interact with the atom \bar{a} , and then becomes p ; and atoms \bar{a}, \bar{b}, \dots represent output
 1645 messages. We will discuss in detail the role played by atoms in the calculus, but we first
 1646 overview the rest of the syntax. We include 1 to syntactically denote successful states. The
 1647 prefix $\tau.p$ represents a process that does one step of internal computation and then becomes p .
 1648 The sum $g_1 + g_2$ is a process that can behave as g_1 or g_2 , but not both. Thus, for example
 1649 $\tau.p + \tau.q$ models an *if ... then ... else*, while $a.p + b.q$ models a *match ... with*. Note
 1650 that the sum operator is only defined on *guards*, namely it can only take as summands $0, 1$ or
 1651 input-prefixed and τ -prefixed processes. While the restriction to guarded sums is a standard
 1652 one, widely adopted in process calculi, the restriction to input and τ guards is specific to
 1653 asynchronous calculi. We will come back to this point after discussing atoms and mailboxes.
 1654 Parallel composition $p \parallel q$ runs p and q concurrently, allowing them also to interact with
 1655 each other, thanks to rule [COM]. For example

$$1656 \quad b.a.0 \parallel b.c.0 \parallel (\bar{a} \parallel \bar{b} \parallel \bar{c}) \tag{7}$$

1657 represents a system in which two concurrent processes, namely $b.a.0$ and $b.c.0$, are both
 1658 ready to consume the message \bar{b} from a third process, namely $\bar{a} \parallel \bar{b} \parallel \bar{c}$. This last process is

1659 a parallel product of atoms, and it is not guarded, hence it is best viewed as an unordered
 1660 mailbox shared by *all* the processes running in parallel with it. For instance in (7) the terms
 1661 $b.a.0$ and $b.c.0$ share the mailbox $\bar{a} \parallel \bar{b} \parallel \bar{c}$. Then, depending on which process consumes \bar{b} ,
 1662 the overall process will evolve to either $b.c.0 \parallel \bar{c}$ or $b.a.0 \parallel \bar{a}$, which are both stuck.¹¹

1663 Concerning the sum construct, we follow previous work on asynchronous calculi ([4, 5, 89,
 1664 24]) and only allow input-prefixed or τ -prefixed terms as summands. The reason for forbidding
 1665 atoms in sums is that the nondeterministic sum is essentially a synchronising operator: the
 1666 choice is solved by executing an action in one of the summands and *simultaneously* discarding
 1667 all the other summands. Then, if an atom were allowed to be a summand, this atom could
 1668 be discarded by performing an action in another branch of the choice. This would mean
 1669 that a process would have the ability to withdraw a message from the mailbox without
 1670 consuming it, thus contradicting the intuition that the mailbox is a shared entity which is
 1671 out of the control of any given process, and with which processes can only interact by feeding
 1672 a message into it or by consuming a message from it. In other words, this restriction on the
 1673 sum operator ensures that atoms indeed represent messages in a global mailbox. For further
 1674 details see the discussion on page 191 of [89].

1675 A structural induction on the syntax ensures that processes perform only a finite number
 1676 of outputs:

1677 ► **Lemma 84.** *For every $p \in \text{ACCS}$. $|O(p)| \in \mathbb{N}$.*

1678 Together with Lemma 87, this means that at any point of every execution the global mailbox
 1679 contains a finite number of messages. Since the LTS is image-finite under any visible action,
 1680 a consequence of Lemma 84 is that the number of reducts of a program is finite.

1681 ► **Lemma 85.** *For every $p \in \text{ACCS}$. $|\{p' \in \text{ACCS} \mid p \xrightarrow{\tau} p'\}| \in \mathbb{N}$.*

1682 **Proof.** Structural induction on p . The only non-trivial case is if $p = p_1 \parallel p_2$. In this case the
 1683 result is a consequence of the inductive hypothesis, of Lemma 84 and of the following fact:
 1684 $p \xrightarrow{\tau} q$ iff

- 1685 1. $p_1 \xrightarrow{\tau} p'_1$ and $q = p'_1 \parallel p_2$,
- 1686 2. $p_2 \xrightarrow{\tau} p'_2$ and $q = p_1 \parallel p'_2$,
- 1687 3. $p_1 \xrightarrow{a} p'_1$ and $p_2 \xrightarrow{\bar{a}} p'_2$ and $q = p'_1 \parallel p'_2$,
- 1688 4. $p_1 \xrightarrow{\bar{a}} p'_1$ and $p_2 \xrightarrow{a} p'_2$ and $q = p'_1 \parallel p'_2$.

1689 In the third case the number of possible output actions \bar{a} is finite thanks to Lemma 84, and
 1690 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
 1691 argument works for the fourth case. ◀

1692 Thanks to Lemma 84 and Lemma 85, Lemma 85 holds also for the LTS modulo structural
 1693 congruence, i.e. $\langle \text{ACCS}_{\equiv}, \longrightarrow_{\equiv}, \text{Act}_{\tau} \rangle$.

1694 F.1 Structural equivalence and its properties

1695 To manipulate the syntax of processes we use a standard structural congruence denoted \equiv ,
 1696 stating that ACCS is a commutative monoid with identity 0 with respect to both sum and
 1697 parallel composition.

¹¹The global shared mailbox that we treat is reminiscent but less general than the chemical “soup” of [18].
 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 16** A typeclass for LTSs where a structural congruence exists over states.

1698 A first fact is the following one.

1699 ▶ **Lemma 86.** *For every $\mu \in \overline{\mathcal{N}}$ and $\alpha \in \text{Act}_\tau$, if $p \xrightarrow{\mu, \alpha} q$ then $p \xrightarrow{\alpha, \mu} \cdot \equiv q$.*

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

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

This notation is useful to treat the global shared mailbox. In particular, if $\{\mu_0, \mu_1, \dots, \mu_n\}$ is a multiset of output actions, then the syntax $\Pi\{\mu_0, \mu_1, \dots, \mu_n\}$ represents the shared mailbox that contains the messages μ_i ; for instance $\Pi\{\bar{a}, \bar{a}, \bar{c}\} = \bar{a} \parallel \bar{a} \parallel \bar{c}$. We use the colour — 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

$$\bar{c} \parallel a.(\bar{b} \parallel c.d.1) \parallel \bar{d} \parallel \tau.\bar{e}$$

1700 the mailbox is $\bar{c} \parallel \bar{d}$. The global mailbox that we denote with — is exactly the buffer B in
 1701 the *configurations* of [95], and reminiscent of the ω used by [28]. The difference is that ω
 1702 represents an unbounded *ordered* queue, while our mailbox is an unbounded *unordered* buffer.

1703 As for the relation between output actions in the LTS and the global mailbox, an output
 1704 \bar{a} can take place if and only if the message \bar{a} appears in the mailbox:

1705 ▶ **Lemma 87.** *For every $p \in \text{ACCS}$,*

- 1706 1. *for every $a \in \mathcal{N}$. $p \xrightarrow{\bar{a}} p'$ implies $p \equiv p' \parallel \bar{a}$,*
- 1707 2. *there exists p' such that $p \equiv p' \parallel \Pi M$, and p' performs no output action.*

1708 This lemma and Lemma 86 essentially hold, because, as already pointed out in Section 2,
 1709 the syntax enforces outputs to have no continuation.

1710 The following lemma states a fundamental fact ([58, Lemma 2.13], [75, Proposition 5.2],
 1711 [89, Lemma 1.4.15]). Its proof is so tedious that even the references we have given only
 1712 sketch it. In this paper we follow the masters example, and give merely a sketch. However,
 1713 we have a complete machine-checked proof.

$$\begin{array}{l}
\text{[S-SZERO]} \quad p + \mathbf{0} \equiv p \\
\text{[S-SCOM]} \quad p + q \equiv q + p \\
\text{[S-SASS]} \quad (p + q) + r \equiv p + (q + r) \\
\\
\text{[S-PZERO]} \quad p \parallel \mathbf{0} \equiv p \\
\text{[S-PCOM]} \quad p \parallel q \equiv q \parallel p \\
\text{[S-PASS]} \quad (p \parallel q) \parallel r \equiv p \parallel (q \parallel r) \\
\\
\text{[S-REFL]} \quad p \equiv p \\
\text{[S-SYMM]} \quad p \equiv q \quad \text{if } q \equiv p \\
\text{[S-TRANS]} \quad p \equiv q \quad \text{if } p \equiv p' \text{ and } p' \equiv q \\
\\
\text{[S-PREFIX]} \quad \alpha.p \equiv \alpha.q \quad \text{if } p \equiv q \\
\text{[S-SUM]} \quad p + q \equiv p' + q \quad \text{if } p \equiv p' \\
\text{[S-PPAR]} \quad p \parallel q \equiv p' \parallel q \quad \text{if } p \equiv p'
\end{array}$$

■ **Figure 17** Rules to define structural congruence on ACCS.

1714 ► **Lemma 88.** For every $p, q \in \text{ACCS}$ and $\alpha \in \text{Act}_\tau$, $p \equiv \cdot \xrightarrow{\alpha} q$ implies $p \xrightarrow{\alpha} \cdot \equiv q$.

1715 **Proof sketch.** We need to show that if there exists a process p' such that $p \equiv p'$ and $p' \xrightarrow{\alpha} q$
1716 then there exists a process q' such that $p \xrightarrow{\alpha} p'$ and $p' \equiv q$. The proof is by induction on
1717 the derivation $p \equiv p'$.

1718 We illustrate one case with the rule [S-TRANS]. The hypotheses tell us that there exists \hat{p}
1719 such that $p \equiv \hat{p}$ and $\hat{p} \equiv p'$, that $p' \xrightarrow{\alpha} q$, and the inductive hypotheses that

1720 (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 $\hat{q} \equiv q'$

1721 (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 $\hat{q} \equiv q'$

1722 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
1723 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 $\hat{q}_2 \equiv \hat{q}_1$.

1724 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
1725 property of the structural congruence relation to show that $\hat{q}_2 \equiv \hat{q}_1$ and $\hat{q}_1 \equiv q$ imply $\hat{q}_2 \equiv q$
1726 as required and we are done with this case. ◀

1727 Time is a finite resource. The one spent to machine check Lemma 88 would have been
1728 best invested into bibliographical research. Months after having implemented the lemma
1729 we realised that [3] already had an analogous result for a mechanisation of the π -calculus.
1730 Lemma 88 is crucial to prove the Harmony Lemma, which states that τ -transitions coincide
1731 with the standard reduction relation of ACCS. This is out of the scope of our discussion, and
1732 we point the interested reader to Lemma 1.4.15 of [89], and to the list of problems presented
1733 on the web-page of THE CONCURRENT CALCULI FORMALISATION BENCHMARK.¹²

1734 We give a corollary that is useful to prove Lemma 90.

1735 ► **Corollary 89.** For every $p, q \in \text{ACCS}$, $\alpha \in \text{Act}_\tau$, $p \equiv q$ implies that $p \xrightarrow{\alpha} \cdot \equiv r$ if and only if
1736 $q \xrightarrow{\alpha} \cdot \equiv r$.

¹²<https://concurrentbenchmark.github.io/>

1737 **Proof.** Since $q \equiv p \xrightarrow{\alpha} p' \equiv r$ Lemma 88 implies $q \xrightarrow{\alpha} \cdot \equiv p'$, thus $q \xrightarrow{\alpha} \cdot \equiv r$ by transitivity
1738 of \equiv . The other implication follows from the same argument and the symmetry of \equiv . ◀

1739 A consequence of Lemma 87 is that the LTS $\langle \text{ACCS}_{\equiv}, \xrightarrow{\equiv}, \text{Act}_{\tau} \rangle$ enjoys the axioms in
1740 Figure 2, and thus it is OF. [92, Theorem 4.3] proves it reasoning modulo bisimilarity, while
1741 we reason modulo structural equivalence.

1742 ▶ **Lemma 90.** For every $p \in \text{ACCS}$, and $a \in \mathcal{N}$ the following properties are true,

- 1743 ■ for every $\alpha \in \text{Act}_{\tau}$. $p \xrightarrow{\bar{a}} \xrightarrow{\alpha} p_3$ implies $p \xrightarrow{\alpha} \xrightarrow{\bar{a}} \cdot \equiv p_3$;
- 1744 ■ for every $\alpha \in \text{Act}_{\tau}$. $\alpha \notin \{\tau, \bar{a}\}$. $p \xrightarrow{\bar{a}} p'$ and $p \xrightarrow{\alpha} p''$ imply that $p'' \xrightarrow{\bar{a}} q$ and $p' \xrightarrow{\alpha} q$
1745 for some q ;
- 1746 ■ $p \xrightarrow{\bar{a}} p'$ and $p \xrightarrow{\bar{a}} p''$ imply $p' \equiv p''$;
- 1747 ■ $p \xrightarrow{\bar{a}} p' \xrightarrow{a} q$ implies $p \xrightarrow{\tau} \cdot \equiv q$;
- 1748 ■ $p \xrightarrow{\bar{a}} p'$ and $p \xrightarrow{\tau} p''$ imply that $p' \xrightarrow{\tau} q$ and $p'' \xrightarrow{\bar{a}} q$; or that $p' \xrightarrow{\tau} p''$.
- 1749 ■ for every p' if there exists a \hat{p} such that $p \xrightarrow{\bar{a}} \hat{p}$ and $p' \xrightarrow{\bar{a}} \hat{p}$ then $p \equiv p'$

1750 **Proof.** To show FEEDBACK we begin via Lemma 87 which proves $p \equiv p' \parallel \boxed{\bar{a}}$. We derive
1751 $p' \parallel \boxed{\bar{a}} \xrightarrow{\tau} q'$ and apply Corollary 89 to obtain $p \xrightarrow{\tau} \cdot \equiv q$.

1752 We prove OUTPUT-TAU. The hypothesis and Lemma 87 imply that $p \equiv p' \parallel \boxed{a}$. Since
1753 $p \xrightarrow{\tau} p''$ 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
1754 that $p'' \xrightarrow{a} \hat{p} \parallel \mathbf{0} \equiv q$. ◀

1755 Processes that enjoy OUTPUT-TAU are called *non-preemptive* in [41, Definition 10].

1756 Each time a process p reduces to a *stable* process p' , it does so by consuming at least
1757 part of the mailbox, for instance a multiset of outputs N , thereby arriving in a state q whose
1758 inputs cannot interact with what remains of the mailbox, i.e. $M \setminus N$, where M is the original
1759 mailbox.

1760 ▶ **Lemma 91.** For every $M \in \text{MO}$, $p, p' \in \text{ACCS}$, if $p \parallel \boxed{\Pi M} \xrightarrow{\varepsilon} p' \xrightarrow{\tau}$ then there exist an
1761 $N \subseteq M$ and some $q \in \text{ACCS}$ such that $p \xrightarrow{N} q \xrightarrow{\tau}$, $O(q) \subseteq O(p')$, and $\overline{I(q)} \# (M \setminus N)$.

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

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

1762 from which we obtain $p \parallel \boxed{\Pi M} \xrightarrow{\tau}$. This ensures that $\overline{I(p)} \# M$. We pick as q and N
1763 respectively $p \parallel \boxed{\Pi M}$ and \emptyset as $p \parallel \boxed{\Pi M} \xrightarrow{\emptyset} p \parallel \boxed{\Pi M}$ by reflexivity, and $O(p \parallel \boxed{\Pi M}) =$
1764 $O(p')$.

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

$$\frac{\begin{array}{c} \vdots \\ p \parallel \boxed{\Pi M} \xrightarrow{\tau} p' \quad \overline{p' \xrightarrow{\varepsilon} p'} \end{array}}{p \parallel \boxed{\Pi M} \xrightarrow{\varepsilon} p'}$$

1765 We continue by case analysis on the rule used to infer the transition $p \parallel \boxed{\Pi M} \xrightarrow{\tau} p'$.
1766 As by definition $\boxed{\Pi M} \xrightarrow{\tau}$, the rule is either [PAR-L], i.e. a τ -transition performed by p , or
1767 [COM], i.e. an interaction between p and $\boxed{\Pi M}$.

1768 **F.1.0.1 Rule [Par-L]:**

1769 In this case $p \xrightarrow{\tau} p''$ for some p'' , thus $p'' \parallel \Pi M \xRightarrow{\varepsilon} p'$ and the result follows from the
1770 inductive hypothesis.

1771 **F.1.0.2 Rule [Com]:**

1772 The hypothesis of the rule ensure that $p \xrightarrow{a} p''$ and $\Pi M \xrightarrow{\bar{a}} p'$, and as the process ΠM
1773 does not perform any input, it must be the case that $a \in \mathcal{N}$, that $\bar{a} \in M$, and that
1774 $q \equiv \Pi(M \setminus \{\bar{a}\})$.¹³ Note that $p' \equiv p'' \parallel \Pi(M \setminus \{\bar{a}\})$.

1775 The inductive hypothesis ensures that for some $N' \subseteq M \setminus \{\bar{a}\}$ and some $q_3 \in \text{ACCS}$ we
1776 have

- 1777 (a) $p' \xRightarrow{N'} q_3 \xrightarrow{\tau}$,
1778 (b) $O(q_3) \subseteq O(p')$, and
1779 (c) $\overline{I(q_3)} \# ((M \setminus \{\bar{a}\}) \setminus N')$

1780 We conclude by letting $q = q_3$, and $N = \{a\} \uplus N'$. The trace $p \xrightarrow{a} p' \xRightarrow{N'} q_3$ proves that
1781 $p \xRightarrow{\{a\} \uplus N'} q_3$, moreover we already know that q_3 is stable. The set inclusion $O(q_3) \subseteq O(p')$
1782 follows from b, and lastly $\overline{I(q)} \# (M \setminus (\{\bar{a}\} \uplus N'))$ is a consequence of $\overline{I(q_3)} \# ((M \setminus \{\bar{a}\}) \setminus N')$
1783 and of $(M \setminus \{\bar{a}\}) \setminus N' = (I \setminus (\{\bar{a}\} \uplus N'))$. ◀

We define the predicate GOOD,

$$\begin{aligned} & \text{GOOD(1)} \\ & \text{GOOD}(p \parallel q) \quad \text{if } \text{GOOD}(p) \text{ or } \text{GOOD}(q) \\ & \text{GOOD}(p + q) \quad \text{if } \text{GOOD}(p) \text{ or } \text{GOOD}(q) \end{aligned}$$

1784 This predicate is preserved by structural congruence.

1785 ▶ **Lemma 92.** For every $p, q \in \text{ACCS}$. $p \equiv q$ and $\text{GOOD}(p)$ imply $\text{GOOD}(q)$.

1786 ▶ **Lemma 93.** For every $p, q \in \text{ACCS}$. $p \equiv q$ and $p \downarrow_i$ imply $q \downarrow_i$.

1787 ▶ **Lemma 94.** For every $p, q \in \text{ACCS}$ and $s \in \text{Act}^*$, we have that $p \equiv q$ and $\text{FW}(p) \downarrow s$ imply
1788 $\text{FW}(q) \downarrow s$.

1789 ▶ **Lemma 95.** For every $p, r, r' \in \text{ACCS}$. $r \equiv r'$ and $p \text{MUST}_i r$ then $p \text{MUST}_i r'$.

1790 ▶ **Lemma 96.** For every $p, q, r \in \text{ACCS}$. $p \equiv q$ and $p \text{MUST}_i r$ then $q \text{MUST}_i r$.

1791 A typical technique to reason on the LTS of concurrent processes, and so also of client-
1792 server systems, is trace zipping: if $p \xRightarrow{s} p'$ and $q \xRightarrow{\bar{s}} q'$, an induction on s ensures that
1793 $p \parallel q \xRightarrow{s} p' \parallel q'$. Zipping together different LTS is slightly more delicate: we can zip weak
1794 transitions $\xRightarrow{s}_{\text{fw}}$ together with the co-transitions $\xRightarrow{\bar{s}}$, but possibly moving inside equivalence
1795 classes of \equiv instead of performing actual transitions in \longrightarrow .

1796 ▶ **Lemma 97 (Zipping).** For every $p, q \in \text{ACCS}$

- 1797 1. for every $\mu \in \text{Act}$. if $p \xrightarrow{\mu}_{\text{fw}} p'$ and $q \xrightarrow{\bar{\mu}} q'$ then $p \parallel q \xrightarrow{\tau} p' \parallel q'$ or $p \parallel q \equiv p' \parallel q'$;
1798 2. for every $s \in \text{Act}^*$. if $p \xRightarrow{s}_{\text{fw}} p'$ and $q \xRightarrow{\bar{s}} q'$ then $p \parallel q \xRightarrow{\varepsilon} \cdot \equiv p' \parallel q'$.

¹³In terms of LTS with mailboxes, $p' = (M \setminus \{\bar{a}\})$.

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

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

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

■ **Figure 18** Functions to generate clients.

1799 Obviously, for every $p, q \in A$ and output $a \in \mathcal{N}$ we have

$$1800 \quad p \xrightarrow{\tau}_{fw} q \text{ if and only if } p \xrightarrow{\tau} q \tag{8}$$

$$1801 \quad p \xrightarrow{\varepsilon}_{fw} q \text{ if and only if } p \xrightarrow{\varepsilon} q \tag{9}$$

$$1802 \quad p \xrightarrow{\bar{a}}_{fw} q \text{ if and only if } p \xrightarrow{\bar{a}} q \tag{10}$$

1804 together with the expected properties of finiteness, the first one amounting to the finiteness
1805 of the global mailbox in any state:

$$1806 \quad |\{\bar{a} \in \bar{\mathcal{N}} \mid p \xrightarrow{\bar{a}}_{fw} q\}| \in \mathbb{N} \tag{11}$$

$$1807 \quad \text{For every } \mu \in \text{Act}. |\{q \mid p \xrightarrow{\mu}_{fw} q\}| \in \mathbb{N} \tag{12}$$

$$1808 \quad |\{q \in A \mid p \xrightarrow{\tau}_{fw} q\}| \in \mathbb{N} \tag{13}$$

1810 F.2 Client generators and their properties

1811 This subsection is devoted to the study of the semantic properties of the clients produced
1812 by the function g . In general these are the properties sufficient to obtain our completeness
1813 result.

1814 ► **Lemma 98.** *For every $p \in \text{ACCS}$ and $s \in \text{Act}^*$, if $p \xrightarrow{\tau}$ then $g(p, s) \xrightarrow{\tau}$.*

1815 **Proof.** By induction on the sequence s . In the base case $s = \varepsilon$. The test generated by g is p ,
1816 which reduces by hypothesis, and so does $g(\varepsilon, p)$. In the inductive case $s = \alpha.s_2$, and we
1817 proceed by case-analysis on α . If α is an output then $g(\alpha.s_2, p) = \bar{\alpha}.(g(s_2, p)) + \tau.1$ which
1818 reduces to 1 using the transition rule [SUM-R]. If μ is an input then $g(\alpha.s_2, p) = \boxed{\bar{\alpha}} \parallel g(s_2, p)$
1819 which reduces using the transition rule [PAR-R] and the inductive hypothesis, which ensures
1820 that $g(s_2, p)$ reduces. ◀

1821 ► **Lemma 99.** *For every p and s , if $\neg \text{GOOD}(p)$ then for every s , $\neg \text{GOOD}(g(s, p))$.*

1822 **Proof.** The argument is essentially the same of Lemma 98 ◀

1823 ► **Lemma 100.** *For every $s \in \text{Act}^*$, if $g(s, q) \xrightarrow{\mu} o$ then either*

1824 (a) $q \xrightarrow{\mu} q'$, $s \in \mathcal{N}^*$, and $o = \boxed{\Pi \bar{s}} \parallel q'$, or

1825 (b) $s = s_1.\boxed{\bar{\mu}}.s_2$ for some $s_1 \in \mathcal{N}^*$ and $s_2 \in \text{Act}^*$, and $o \equiv \boxed{\Pi \bar{s}_1} \parallel g(s_2, q)$, and

1826 (i) $\mu \in \mathcal{N}$ implies $g(s, q) \equiv \boxed{\Pi \bar{s}_1} \parallel (\tau.1 + \mu.g(s_2, q))$,

1827 (ii) $\mu \in \boxed{\bar{\mathcal{N}}}$ implies $g(s, q) \equiv \boxed{\Pi \bar{s}_1 \parallel \mu} \parallel (\tau.1 + \mu.g(s_2, q))$.

XX:54 Constructive characterisations of the must-preorder for asynchrony

1828 **Proof.** The proof is by induction on s .

1829 In the case, $s = \varepsilon$, and hence by definition $g(s, q) = q$. The hypotheses $g(s, q) \xrightarrow{\mu} o$
 1830 implies $q \xrightarrow{\mu} o$, and $o \equiv 0 \parallel o \equiv \Pi \varepsilon \parallel o$.

1831 In the inductive case, $s = \nu.s'$. We have two cases, depending on whether ν is an output
 1832 action or an input action.

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

$$g(s, q) = \tau.1 + \mu.g(s_2, q) \equiv \Pi \bar{s}_1 \parallel (\tau.1 + \mu.g(s_2, q))$$

1833 and $o \equiv \Pi \bar{s}_1 \parallel g(s_2, q)$.

1834 Now suppose that ν is an input action. By definition

$$1835 \quad g(s, q) = \bar{\nu} \parallel g(s', q) \tag{16}$$

1836 and the inductive hypothesis ensures that either

- 1837 (1) $q \xrightarrow{\mu} q'$, $s' \in \mathcal{N}^*$, and $o' = \Pi \bar{s}' \parallel q'$, or
 1838 (2) $s' = s'_1.\bar{\mu}.s'_2$, for some $s'_1 \in \mathcal{N}^*$ and $s_2 \in \text{Act}^*$, and

$$1839 \quad o' \equiv \Pi \bar{s}'_1 \parallel g(s'_2, q) \tag{17}$$

1840 and

$$1841 \quad \mu \in \mathcal{N} \text{ implies } g(s', q) \equiv \Pi \bar{s}'_1 \parallel (\tau.1 + \mu.g(s'_2, q)) \tag{18}$$

$$1842 \quad \mu \in \bar{\mathcal{N}} \text{ implies } g(s', q) \equiv \Pi \bar{s}'_1 \parallel \mu \parallel (\tau.1 + \mu.g(s'_2, q)) \tag{19}$$

1844 The action μ is either an input or an output, and we organise the proof accordingly.

1845 Suppose μ is an input. Since $\bar{\nu}$ is an output, the transition $g(s, q) \xrightarrow{\mu} o$ must be due to
 1846 a transition $g(s', q) \xrightarrow{\mu} o'$, thus Equation (16) implies

$$1847 \quad o = \bar{\nu} \parallel o' \tag{20}$$

1848 In case 1, then $s' \in \mathcal{N}^*$ and $\nu \in \mathcal{N}$ ensure $s \in \mathcal{N}^*$ and the equality $o \equiv \Pi \bar{s} \parallel q'$ follows
 1849 from $o' = \Pi \bar{s}' \parallel q'$ and Equation (20).

1850 In case 2, let $s_1 = \nu.s'_1$, $s_2 = s'_2$. Since ν is an input we have $s_1 \in \mathcal{N}^*$. The equalities
 1851 $s = \nu.s'$ and $s' = s'_1.\bar{\mu}.s'_2$ imply that $s = s_1.\bar{\mu}.s_2$. The required $o \equiv \Pi \bar{s}_1 \parallel g(s'_2, q)$ follows
 1852 from $o' \equiv \Pi \bar{s}'_1 \parallel g(s'_2, q)$ and Equation (20).

Now we proceed as follows,

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

1853 Now suppose that μ is an output. Then either $\bar{\nu} = \mu$ or $\bar{\nu} \neq \mu$.

1854 In the first case we let $s_1 = \varepsilon$ and $s_2 = s'$. Equation (16) and $\bar{\nu} = \mu$ imply $g(s, q) = \mu \parallel$
 1855 $g(s_2, q)$ from which we obtain the required $g(s, q) \equiv \Pi \bar{s}_1 \parallel \mu \parallel g(s_2, q)$, and $o \equiv \Pi \bar{s}_1 \parallel$
 1856 $g(s_2, q)$.

1857 If $\bar{\nu} \neq \mu$ then Equation (16) ensures that the transition $g(s, q) \xrightarrow{\mu} o$ must be due to
 1858 $g(s', q) \xrightarrow{\mu} o'$ 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 $s_1 \in \mathcal{N}^*$. Let $s_2 = s'_2$, now we have that

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

1859

1860 ► **Lemma 101.** For every $s \in \text{Act}^*$, if $g(s, q) \xrightarrow{\mu} p$ then either:

- 1861 (a) there exists q' such that $q \xrightarrow{\mu} q'$, $s \in \mathcal{N}^*$ with $p \equiv \Pi \bar{s} \parallel q'$, or
 1862 (b) $s = s_1. \boxed{\bar{\mu}} .s_2$ for some $s_1 \in \mathcal{N}^*$ and $s_2 \in \text{Act}^*$ with $p \equiv g(s_1.s_2, q)$.

1863 **Proof.** The proof is by induction over the sequence s .

1864 In the base case $s = \varepsilon$ and we have $g(\varepsilon, q) = q$. We show part (a) and choose $q' = p$. We
 1865 have $g(\varepsilon, q) = q \xrightarrow{\mu} p$, $p \equiv \Pi \bar{\varepsilon} \parallel q'$ and $\varepsilon \in \mathcal{N}^*$ as required.

1866 In the inductive case $s = \nu.s'$. We proceed by case-analysis on ν . If ν is an input, then
 1867 $g(\nu.s', q) = \bar{\nu} \parallel g(s', q)$. The hypothesis $g(\nu.s', q) \xrightarrow{\mu} p$ implies that either:

- 1868 (i) $\bar{\nu} \xrightarrow{\mu} 0$ with $p = 0 \parallel g(s', q)$ and $\bar{\nu} = \mu$, or
 1869 (ii) $g(s', q) \xrightarrow{\mu} \hat{p}$ with $p = \bar{\nu} \parallel \hat{p}$.

1870 In the first case we show part (b). We choose $s_1 = \varepsilon$, $s_2 = s'$. We have $s = \mu.s' = \bar{\nu}.s'$,
 1871 $p = 0 \parallel g(s', q) \equiv g(\varepsilon.s', q)$ and $\varepsilon \in \mathcal{N}^*$ as required.

1872 In the second case the inductive the hypothesis tells us that either:

- 1873 (H-a) there exists q' such that $q \xrightarrow{\mu} q'$, $s' \in \mathcal{N}^*$ with $\hat{p} \equiv \Pi \bar{s}' \parallel q'$, or
 1874 (H-b) $s = s_1. \boxed{\bar{\mu}} .s_2$ for some $s_1 \in \mathcal{N}^*$ and $s_2 \in \text{Act}^*$ with $\hat{p} \equiv g(s_1.s_2, q)$.

1875 part (a) or part (b) is true.

1876 If part (a) is true then $s' \in \mathcal{N}^*$ and there exists q'' such that $q \xrightarrow{\mu} q''$ with $\hat{p} \equiv \Pi \bar{s}' \parallel q''$.
 1877 We prove part (a). We choose $q' = q''$ and $s = \nu.s'$. We have $p \equiv \bar{\nu} \parallel \hat{p} \equiv \bar{\nu} \parallel \Pi \bar{s}' \parallel q'' \equiv$
 1878 $\Pi \bar{\nu.s'} \parallel q''$ and $\nu.s' \in \mathcal{N}^*$ as required.

1879 If part (b) is true then $s' = s'_1. \boxed{\bar{\mu}} .s'_2$ for some $s'_1 \in \mathcal{N}^*$ and $s'_2 \in \text{Act}^*$ with $\hat{p} \equiv g(s'_1.s'_2, q)$.
 1880 We prove part (a). We choose $s_1 = \nu.s'_1$ and $s_2 = s'_2$. We have $p \equiv \bar{\nu} \parallel \hat{p} \equiv \bar{\nu} \parallel g(s'_1.s'_2, q) \equiv$
 1881 $g(\nu.s'_1.s'_2, q)$ as required.

1882 If ν is an output, then $g(\nu.s', q) = \bar{\nu}.(g(s', q)) + \tau.1$. We prove part (b) and choose $s_1 = \varepsilon$,
 1883 $s_2 = s'$. The hypothesis $g(\nu.s', q) \xrightarrow{\mu} p$ implies that $\mu = \bar{\nu}$ and $p \equiv g(s', q) \equiv g(\varepsilon.s', q)$ as
 1884 required. ◀

1885 ► **Lemma 102.** For every $s \in \text{Act}^*$, if $g(s, q) \xrightarrow{\tau} o$ then either

- 1886 (a) $\text{GOOD}(o)$, or
 1887 (b) $s \in \mathcal{N}^*$, $q \xrightarrow{\tau} q'$, and $o = \Pi \bar{s} \parallel q'$, or
 1888 (c) $s \in \mathcal{N}^*$, $q \xrightarrow{\nu} q'$, and $s = s_1.\nu.s_2$, and $o \equiv \Pi \bar{s}_1.\bar{s}_2 \parallel q'$, or
 1889 (d) $o \equiv g(s_1.s_2.s_3, q)$ where $s = s_1.\mu.s_2.\bar{\mu}.s_3$ with $s_1.\mu.s_2 \in \mathcal{N}^*$.

1890 **Proof.** By induction on the structure of s .

1891 In the base case $s = \varepsilon$. We prove b. Trivially $s \in \mathcal{N}^*$, and by definition $g(\varepsilon, q) = q$, the
 1892 hypothesis implies therefore that $q \xrightarrow{\tau} o$. The q' we are after is o itself, for $o \equiv 0 \parallel o = \Pi \bar{\varepsilon} \parallel o$.

XX:56 Constructive characterisations of the must-preorder for asynchrony

1893 In the inductive case $s = \nu.s'$. We proceed by case analysis on whether $\nu \in \overline{\mathcal{N}}$ or $\nu \in \mathcal{N}$.
 1894 If ν is an output, by definition $g(s, q) = \tau.1 + \bar{\nu}.gens'q$. Since $\bar{\nu}.g(s', q) \xrightarrow{\tau}$, the silent
 1895 move $g(s, q) \xrightarrow{\tau} o$ is due to rule [SUM-L], thus $o = 1 + \bar{\nu}.g(s', q)$, and thus $\text{GOOD}(o)$. We
 1896 have proven a.

1897 Suppose now that ν is an input, by definition

$$1898 \quad g(s, q) = \bar{\nu} \parallel g(s', q) \tag{21}$$

1899 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

$$\frac{\bar{\nu} \xrightarrow{\tau} 0 \quad \frac{\vdots}{g(s', q) \xrightarrow{\nu} o'}}{\bar{\nu} \parallel g(s', q) \xrightarrow{\tau} 0 \parallel o'}$$

1900 and thus $o \equiv o'$. Since ν is an input and $g(s', q) \xrightarrow{\nu} o'$, Lemma 100 ensures that either

- 1901 **(1)** $q \xrightarrow{\nu} q'$, $s' \in \mathcal{N}^*$, and $o' = \Pi \bar{s}' \parallel q'$, or
 1902 **(2)** $s' = s'_1.\bar{\nu}.s'_2$ for some $s'_1 \in \mathcal{N}^*$ and $s'_2 \in \text{Act}^*$, and

$$1903 \quad o' \equiv \Pi \bar{s}'_1 \parallel g(s'_2, q) \tag{22}$$

$$1904 \quad g(s, q) \equiv \Pi \bar{s}'_1 \parallel (\tau.1 + \nu.g(s'_2, q)) \tag{23}$$

1905

1906 In case 1 we prove part (c). Since ν is an input, $s' \in \mathcal{N}^*$ ensures that $s \in \mathcal{N}^*$. By letting
 1907 $s_1 = \varepsilon$ and $s_2 = s'$ we obtain $s = s_1.\nu.s_2$. We have to explain why $o \equiv \Pi \bar{s}_1.\bar{s}_2 \parallel q'$. This
 1908 follows from the definitions of s_1 and s_2 , from $o \equiv 0 \parallel o'$ and from $o' \equiv \Pi \bar{s}' \parallel q'$.

In case 2 we prove part (d). Let $s_1 = \varepsilon$, $s_2 = s'_1$ and $s_3 = s'_2$.

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

1909 and $s_1.\nu.s_2 \in \mathcal{N}^*$ as required. Moreover $o' = \Pi \bar{s}'_1 \parallel g(s'_2, q) = \Pi \bar{s}_2 \parallel g(s_3, q) =$
 1910 $g(s_2.s_3, q) = g(s_1.s_2.s_3, q)$ as required. This concludes the argument due to an applicaton of
 1911 [COM].

If [PAR-R] was employed we know that

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

1912 thus $g(s', q) \xrightarrow{\tau} o'$ and

$$1913 \quad o = \bar{\nu} \parallel o' \tag{24}$$

1914 Since s' is smaller than s , thanks to $g(s', q) \xrightarrow{\tau} o'$ we apply the inductive hypothesis to
 1915 obtain either

- 1916 **(i)** $\text{GOOD}(o')$, or

- 1917 (ii) $s' \in \mathcal{N}^*$, $q \xrightarrow{\tau} q'$, and $o' = \boxed{\Pi s'} \parallel q'$, or
 1918 (iii) $s' \in \mathcal{N}^*$, $q \xrightarrow{\mu} q'$, and $s' = s'_1 \cdot \mu \cdot s'_2$, or
 1919 (iv) $o' \equiv g(s'_1 \cdot s'_2 \cdot s'_3, q)$ where $s' = s'_1 \cdot \mu \cdot s'_2 \cdot \bar{\mu} \cdot s'_3$ with $s'_1 \cdot \mu \cdot s'_2 \in \mathcal{N}^*$,

1920 If i then Equation (24) implies a. If ii then $s = \nu \cdot s'$ and the assumption that ν is input
 1921 imply that $s \in \mathcal{N}^*$. Equation (24) and $o' = \boxed{\Pi s'} \parallel q'$ imply that $o = \Pi \bar{s} \parallel q'$. We have
 1922 proven b. If ii we prove b, because $s' \in \mathcal{N}^*$ ensures $s \in \mathcal{N}^*$ and $s' = s'_1 \cdot \mu \cdot s'_2$ let us prove
 1923 $s = s_1 \cdot \nu \cdot s_2$ by letting $s_1 = \nu \cdot s'_1$ and $s_2 = s'_2$.

1924 If iv we prove d. We have $o' \equiv g(s'_1 \cdot s'_2 \cdot s'_3, q)$ and $s' = s'_1 \cdot \lambda \cdot s'_2 \cdot \bar{\lambda} \cdot s'_3$ with $s'_1 \cdot \lambda \cdot s'_2 \in \mathcal{N}^*$.

Let $s_1 = \nu \cdot s'_1$, $s_2 = s'_2$ and $s_3 = s'_3$. Since $s'_1 \cdot \mu \cdot s'_2 \in \mathcal{N}^*$, we have $s_1 \cdot \mu \cdot s_2 \in \mathcal{N}^*$. We also have

$$\begin{aligned} s' &= s'_1 \cdot \mu \cdot s'_2 \cdot \bar{\mu} \cdot s'_3 && \text{By inductive hypothesis} \\ \nu \cdot s' &= \nu \cdot s'_1 \cdot \mu \cdot s'_2 \cdot \bar{\mu} \cdot s'_3 \\ s &= \nu \cdot s'_1 \cdot \mu \cdot s'_2 \cdot \bar{\mu} \cdot s'_3 && \text{Because } s = \nu \cdot s' \\ s &= s_1 \cdot \mu \cdot s_2 \cdot \bar{\mu} \cdot s_3 && \text{By definition} \end{aligned}$$

1925 It remains to prove that $o \equiv g(s_1 \cdot s_2 \cdot s_3, q)$. This is a consequence of Equation (24), of
 1926 $o' \equiv g(s'_1 \cdot s'_2 \cdot s'_3, q)$, and of the definitions of s_1 , s_2 , and s_3 . ◀

1927 ▶ **Lemma 103.** For every $s \in \text{Act}^*$, and process q such that o or $q \xrightarrow{\tau} q'$ implies $\text{GOOD}(q')$,
 1928 and for every $\mu \in \mathcal{N} \cdot q \xrightarrow{\mu} q'$ implies $\text{GOOD}(q')$, if $g(s, q) = o_0 \xrightarrow{\tau} o_1 \xrightarrow{\tau} o_2 \xrightarrow{\tau} \dots o_n \xrightarrow{\tau}$
 1929 and $n > 0$ then $\text{GOOD}(o_i)$ for some $i \in [1, n]$.

1930 **Proof.** Lemma 102 implies that one of the following is true,

- 1931 (a) $\text{GOOD}(o_1)$, or
 1932 (b) $s \in \mathcal{N}^*$, $q \xrightarrow{\tau} q'$, and $o_1 = \boxed{\Pi \bar{s}} \parallel q'$, or
 1933 (c) $s \in \mathcal{N}^*$, $q \xrightarrow{\mu} q'$, and $s = s_1 \cdot \mu \cdot s_2$, and $o_1 \equiv \Pi \boxed{s_1 \cdot s_2} \parallel q'$, or
 1934 (d) $o_1 \equiv g(s_1 \cdot s_2 \cdot s_3, q)$ where $s = s_1 \cdot \mu \cdot s_2 \cdot \bar{\mu} \cdot s_3$ with $s_1 \cdot \mu \cdot s_2 \in \mathcal{N}$.

1935 If a we are done. If b or c then $\text{GOOD}(q')$, and thus $\text{GOOD}(o_1)$.

1936 In the base case $\text{len}(s) = 0$, thus d is false. It follows that $\text{GOOD}(o_1)$.

1937 In the inductive case $s = \nu \cdot s'$. We have to discuss only the case in which d is true. The
 1938 inductive hypothesis ensures that

1939 For every $s' \in \bigcup_{i=0}^{\text{len}(s)-1}$, if $g(s', q) \xrightarrow{\tau} o'_1 \xrightarrow{\tau} o'_2 \xrightarrow{\tau} \dots o'_m \xrightarrow{\tau}$ and $m > 0$ then $\text{GOOD}(o'_j)$
 1940 for some j .

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

$$g(s_1 \cdot s_2 \cdot s_3, q) \xrightarrow{\tau} o'_1 \xrightarrow{\tau} o'_2 \xrightarrow{\tau} \dots o'_m \xrightarrow{\tau} \text{ implies } o'_j \text{ for some } j.$$

1941 We conclude the argument via Lemma 88 and because \equiv preserves success. ◀

1942 ▶ **Lemma 104.** For every $s \in \text{Act}^*$ and process q , if $g(s, q) \xrightarrow{\tau}$ then

- 1943 1. $s \in \mathcal{N}^*$,
 1944 2. $q \xrightarrow{\tau}$,
 1945 3. $I(q) \cap \bar{s} = \emptyset$,
 1946 4. $R(g(s, q)) = \bar{s} \cup R(q)$.

1947 **Proof.** By induction on s . In the base case $\varepsilon \in \mathcal{N}^*$, and $g(\varepsilon, q) = q$, thus $q \xrightarrow{\tau}$. The last
 1948 two points follow from this equality and from ε containing no actions.

1949 In the inductive case $s = \mu \cdot s'$. The hypothesis $g(\mu \cdot s', q) \xrightarrow{\tau}$ and the definition of g imply
 1950 that $g(\mu \cdot s', q) = \bar{\mu} \parallel g(s', q)$, thus $\mu \in \mathcal{N}$. The inductive hypothesis ensures that

XX:58 Constructive characterisations of the must-preorder for asynchrony

- 1951 1. $s' \in \mathcal{N}^*$,
1952 2. $q \xrightarrow{\tau}$,
1953 3. for every $I(q) \cap \bar{s}' = \emptyset$,
1954 4. for every $R(g(s', q)) = \bar{s}' \cup R(q)$
1955 Since $\bar{\mu} \parallel g(s', q) \xrightarrow{\tau}$ rule [COM] cannot be applied, thus $q \xrightarrow{\mu}$, and so $I(q) \cap \bar{s} = \emptyset$. From
1956 $R(g(s', q)) = \bar{s}' \cup R(q)$ we obtain $R(g(s, q)) = \bar{s} \cup R(q)$. ◀

1957 ▶ **Lemma 105.** For every $\mu \in \text{Act}$, s and p , $g(\mu.s, p) \xrightarrow{\bar{\mu}} g(s, p)$.

1958 **Proof.** We proceed by case-analysis on μ . If μ is an input then $g(\mu.s, p) = \bar{\mu} \parallel g(s, p)$. We
1959 have $\bar{\mu} \parallel g(s, p) \xrightarrow{\bar{\mu}} 0 \parallel g(s, p) \equiv g(s, p)$ as required. If μ is an output then $g(\mu.s, p) =$
1960 $\bar{\mu}.g(s, p) + \tau.1$. We have $\bar{\mu}.g(s, p) + \tau.1 \xrightarrow{\bar{\mu}} g(s, p)$ as required. ◀

1961 ▶ **Lemma 106.** For every $s \in \text{Act}^*$, $q \in \text{ACCS}$. $c(s) \xrightarrow{\tau}_{\text{fw}} q$ either

- 1962 (a) $\text{GOOD}(q)$, or
1963 (b) there exist b, s_1, s_2 and s_3 with $s_1.b.s_2 \in \mathcal{N}^*$ such that $s = s_1.b.s_2.\bar{b}.s_3$ and $q \equiv$
1964 $c(s_1.s_2.s_3)$.

1965 **Proof.** The proof is by induction on s .

1966 In the base case $s = \varepsilon$, $c(\varepsilon) = \tau.1$ and then $q = 1$. We prove a with $\text{GOOD}(1)$.

1967 In the inductive case $s = \mu.s'$. We proceed by case-analysis over μ .

1968 If μ is an input then $c(\mu.s') = \bar{\mu} \parallel c(s')$. We continue by case-analysis over the reduction
1969 $\bar{\mu} \parallel c(s') \xrightarrow{\tau} q$. It is either due to:

- 1970 (i) a communication between $\bar{\mu}$ and $c(s')$ such that $\bar{\mu} \xrightarrow{\bar{\mu}} 0$ and $c(s') \xrightarrow{\mu} q'$ with $q = 0 \parallel q'$,
1971 or
1972 (ii) a reduction of $c(s')$ such that $c(s') \xrightarrow{\tau} q'$ with $q = \bar{\mu} \parallel q'$.

1973 If i is true then Lemma 101 tells us that there exist s'_1 and s'_2 such that $s' = s'_1.\bar{\mu}.s'_2$ and
1974 $q' \equiv c(s'_1.s'_2)$ 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
1975 show the first requirement by $s = \mu.s' = \mu.s'_1.\bar{\mu}.s'_2 = \varepsilon.\mu.s'_1.\bar{\mu}.s'_2 = s_1.b.s_2.\bar{b}.s_3$. The second
1976 requirement is $q = 0 \parallel q' \equiv c(s'_1.s'_2) = c(\varepsilon.s'_1.s'_2) = c(s_1.s_2.s_3)$.

1977 We now consider the case (ii). The inductive hypothesis tells us that either:

- 1978 1. $\text{GOOD}(q')$, or
1979 2. there exist ι, s'_1, s'_2 and s'_3 with $s'_1.\iota.s'_2 \in \mathcal{N}^*$ such that $s' = s'_1.\iota.s'_2.\bar{\iota}.s'_3$ and $q' \equiv$
1980 $c(s'_1.s'_2.s'_3)$.

1981 If (1) is true then we prove a with $q = \bar{\mu} \parallel q'$ and $\bar{\mu} \parallel q'$ and $\text{GOOD}(q')$. If (2) is
1982 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
1983 first requirement with $s = \mu.s' = \mu.s'_1.\iota.s'_2.\bar{\iota}.s'_3 = s_1.b.s_2.\bar{b}.s_3$. The second requirement is
1984 $q = \bar{\mu} \parallel q' \equiv \bar{\mu} \parallel c(s'_1.s'_2.s'_3) = c(\mu.s'_1.s'_2.s'_3) = c(s_1.s_2.s_3)$.

1985 If μ is an output then $c(\mu.s') = \bar{\mu}.c(s') + \tau.1$. The hypothesis $c(\mu.s') \xrightarrow{\tau} q$ implies $q = 1$.
1986 We prove (a) with $\text{GOOD}(1)$. ◀

1987 **G** Further related works

1988 **Contextual preorders in functional languages.** Morris preorder is actively studied in
1989 the pure λ -calculus [30, 31, 67, 11], λ -calculus with references [81, 62], in PCF [69] as well as
1990 in languages supporting shared memory concurrency [96], and mutable references [47]. The
1991 more sophisticated the languages, the more intricate and larger the proofs. The need for
1992 mechanisation became thus apparent, in particular to prove that complex logical relations
1993 defined in the framework Iris (implemented in Coq) are sound, *i.e.* included in the preorder

1994 [70, 52]. [7] provide a framework to study contextual equivalences in the setting of process
1995 calculi. It is worth noting, though, that as argued by [26] (in Section 3 of that paper), Morris
1996 equivalence coincides with MAY-equivalence, at least if the operational semantics at hand
1997 enjoys the Church-Rosser property. In fact [27] define Morris preorder literally as a testing
1998 one, via tests for convergence. The studies of the MUST-preorder in process calculi can thus
1999 be seen as providing proof methods to adapt Morris equivalence to nondeterministic settings,
2000 and using contexts that are really external observers. To sum up, one may say that Morris
2001 equivalence coincides with MAY-equivalence when nondeterminism is confluent and all states
2002 are viewed as accepting states, while it coincides with MUST equivalence in the presence of
2003 true nondeterminism and when only successful states are viewed as accepting states.

2004 In the setting of nondeterministic and possibly concurrent applicative programming
2005 languages [91, 90, 19], also a contextual preorder based on may and must-termination has
2006 been studied [86, 19]. Our preorder \preceq_{cnv} is essentially a generalisation of the must-termination
2007 preorder of [86] to traces of visible actions.

2008 **Models of asynchrony.** While synchronous (binary) communication requires the
2009 simultaneous occurrence of a send and a receive action, asynchronous communication allows
2010 a delay between a send action and the corresponding receive action. Different models of
2011 asynchrony exist, depending on which medium is assumed for storing messages in transit. In
2012 this paper, following the early work on the asynchronous π -calculus [64, 25, 4], we assume the
2013 medium to be an unbounded unordered mailbox, shared by all processes. Thus, no process
2014 needs to wait to send a message, namely the send action is non-blocking. This model of
2015 communication is best captured via the output-buffered agents with feedback of [92]. The
2016 early style LTS of the asynchronous π -calculus is a concrete example of this kind of LTSs.
2017 A similar global unordered mailbox is used also in Chapter 5 of [95], by [33], which relies
2018 explicitly on a mutable global state, and by [79], which manipulates it via two functions *get*
2019 and *set*.

2020 More deterministic models of asynchrony are obtained assigning a data structure to every
2021 channel. For example [65, 66] use an even more deterministic model in which each ordered
2022 pair of processes is assigned a dedicated channel, equipped with an *ordered* queue. Hence,
2023 messages along such channels are received in the same order in which they were sent. This
2024 model is used for asynchronous session calculi, and mimics the communication mode of the
2025 TCP/IP protocol. The obvious research question here is how to adapt our results to the
2026 different communication mechanisms and different classes of LTSs. For instance, both [94]
2027 and [35] define LTSs for ERLANG. We will study whether at least one of these LTSs is an
2028 instance of output-buffered agents with feedback. If this is not the case, we will first try to
2029 adapt our results to ERLANG LTSs.

2030 **Mutable state.** Prebet [81] has recently shown an encoding of the asynchronous π -
2031 calculus into a λ -calculus with references, which captures Morris equivalence via a bisimulation.
2032 This renders vividly the intuition that output-buffered agents manipulate a shared common
2033 state. We therefore see our work also as an analysis of the MUST-preorder for a language in
2034 which programs manipulate a global mutable store. Since the store is what contains output
2035 messages, and our formal development shows that only outputs are observable, our results
2036 suggest that characterisations of testing preorders for impure programming languages should
2037 predicate over the content of the mutable store, *i.e.* the values written by programs. Another
2038 account of π -calculus synchronisation via a functional programming language is provided in
2039 [90], that explains how to use Haskell M-VARS to implement π -calculus message passing.

2040 **Theories for synchronous semantics.** Both [72] and [37] employed LTSs as a model
2041 of contracts for web-services (*i.e.* WSCL), and the MUST-preorder as refinement for contracts.

2042 The idea is that a search engine asked to look for a service described by a contract p_1 can
 2043 actually return a service that has a contract p_2 , provided that $p_1 \sqsubseteq_{\text{MUST}} p_2$.

2044 The MUST-preorder for *clients* proposed by [14] has partly informed the theory of monitors
 2045 by [1], in particular the study of preorders for monitors by [50]. Our results concern LTSs
 2046 that are more general than those of monitors, and thus our code could provide the basis to
 2047 mechanise the results of [1].

2048 The first subtyping relation for binary session types was presented in [53] using a syntax-
 2049 oriented definition. The semantic model of that subtyping is a refinement very similar to the
 2050 MUST-preorder. The idea is to treat types as CCS terms, assign them an LTS [36, 10, 84, 17],
 2051 and use the resulting testing preorders as semantic models of the subtyping. In the setting of
 2052 coinductively defined higher-order session types, the correspondence is implicitly addressed
 2053 in [36]. In the setting of recursive higher-order session types, it is given by Theorem 4.10
 2054 of [15].

2055 We would like to mechanise in our framework these results, in particular the ones about
 2056 asynchronous semantics, and contrast and compare the various testing preorders used in the
 2057 literature. More in general, given the practical relevance of asynchronous communication, it
 2058 seems crucial not only to adapt the large body of theory outlined above to the asynchronous
 2059 setting but also to resort to machine supported reasoning to do it. This paper is meant to
 2060 be a step forward in this direction.

2061 **Must-preorder and asynchrony.** The first investigation on the MUST-preorder in an
 2062 asynchronous setting was put forth by [39]. While their very clear examples shed light on
 2063 the preorder, their alternative preorder (Definition 6 in that paper) is more complicated than
 2064 necessary: it uses the standard LTS of ACCS, the LTS of forwarders, a somewhat ad-hoc
 2065 predicate $\overset{I}{\rightsquigarrow}$, and a condition on multisets of inputs, that we do not use. Moreover that
 2066 preorder is not complete because of a glitch in the treatment of divergence. The details of
 2067 the counter-example we found to that completeness result are in Appendix I.

2068 In [57] Hennessy outlines how to adapt the approach of [39] to a typed asynchronous
 2069 π -calculus. While the LTS of forwarders is replaced by a Context LTS, the predicates to define
 2070 the alternative preorder are essentially the same used in the preceding work with Castellani.
 2071 Acceptance sets are given in Definition 3.19 there, and the predicate \rightsquigarrow is denoted \searrow , while
 2072 the generalised acceptance sets of [39] are given in Definition 3.20. Owing to the glitch in
 2073 the completeness of [39], it is not clear that Theorem 3.28 of [57] is correct either.

2074 Also the authors of [24] the MUST-preorder in ACCS. There is a major difference between
 2075 their approach and ours. When studying theories for asynchronous programs, one can either
 2076 (1) keep the definitions used for synchronous programs, and enhance the LTS with forwarders;
 2077 or
 2078 (2) adapt the definitions, and keep the standard LTS.

2079 In the first case, the complexity is moved into the LTS, which becomes infinite-branching
 2080 and infinite-state. In the second case, the complexity is moved into the definitions used to
 2081 reason on the LTS (i.e. in the meta-language), and in particular in the definition of the
 2082 alternative preorder, which deviates from the standard one. The authors of [24] follow the
 2083 second approach. This essentially explains why they employ the standard LTS of CCS and
 2084 to tackle asynchrony they reason on traces via

- 2085 (i) a preorder \preceq (Table 2 of that paper) that defines on *input* actions the phenomena due to
 2086 asynchrony, for instance their *annihilation* rule (i.e. TO3) is analogous to the FEEDBACK
 2087 axiom, and their *postponement* (i.e. TO2) is analogous to the OUTPUT-COMMUTATIVITY
 2088 axiom; and
- 2089 (ii) a rather technical operation on traces, namely $s \ominus s' = (\{s\}_i \setminus \{s'\}_i) \setminus \overline{(\{s\}_o \setminus \{s'\}_o)}$.

2090 We favour instead the first approach, for, as we already argued, it helps us achieve a modular
2091 mechanisation.

2092 The authors of [46] give yet another account of the MUST-preorder. Even though non-
2093 blocking outputs can be written in their calculus, they use a left-merge operator that allows
2094 writing *blocking* outputs. The contexts that they use to prove the completeness of their
2095 alternative preorder use such blocking outputs, consequently their arguments need not tackle
2096 the asymmetric treatment of input and output actions. This explains why they can use
2097 smoothly a standard LTS, while [39] and [24] have to resort to more complicated structures.

2098 Theorem 5.3 of the PhD thesis by [95] states an alternative characterisation of the
2099 MUST-preorder, but it is given with no proof. The alternative preorder given in Definition 5.8
2100 of that thesis turns out to be a mix of the ones by [39] and [24]. In particular, the definition
2101 of the alternative preorder relies on the LTS of forwarders, there denoted \longrightarrow_A (Point 1. in
2102 Definition 5.1 defines exactly the input transitions that forward messages into the global
2103 buffer). The condition that compares convergence of processes is the same as in [39], while
2104 server actions are compared using MUST-sets, and not acceptance sets. In fact, Definition 5.7
2105 there is titled “acceptance sets” but it actually defines MUST-sets.

2106 **MAY-preorder.** MAY testing and the MAY-preorder, have been widely studied in asyn-
2107 chronous settings. The first characterisation for ACCS appeared in [39] and relies on comparing
2108 traces and asynchronous traces of servers. Shortly after [24] presented a characterisation
2109 based on operation on traces. A third characterisation appeared in [8], where the saturated
2110 LTS \longrightarrow_s is essentially out \longrightarrow_{fw} . That characterisation supports our claim that results
2111 about synchronous semantics are true also for asynchronous ones, modulo forwarding. Com-
2112 positionality of trace inclusion, i.e. the alternative characterisation of the MAY-preorder, has
2113 been partly investigated in Coq by [6] in the setting of IO-automata. The MAY-preorder has
2114 also been studied in the setting of actor languages by [35, 94].

2115 **Fairness.** Van Glabbeek [97] argues that by amending the semantics of parallel composi-
2116 tion (i.e. the scheduler) different notions of fairness can be embedded in the MUST-preorder.
2117 We would like to investigate which notion of fairness makes the MUST-preorder coincide with
2118 the FAIR-preorder of [85].

2119 **Bar-induction.** A mainstay in the literature on the MUST-preorder is König’s lemma,
2120 see for example Theorem 2.3.3 in [40], and Theorem 1 in [16]. [48], though, explains in
2121 detail why König’s lemma is not constructive. Instead, we use in this paper the constructive
2122 bar-induction principle, whose fundamental use is to prove that if every path in a tree T
2123 is finite, then T is well-founded, as discussed by [77, 29] and [68]. Unfortunately, while it
2124 is a constructive principle, mainstream proof assistants do not support it, which is why
2125 we had to postulate it as a proof principle that we proved using the Excluded Middle
2126 axiom. One consequence of using an axiom is that they do not have computational content.
2127 Developing a type theory with a principle of bar-induction is the subject of recent and
2128 ongoing works [51, 83].

2129 **H** Co-inductive characterisation of the must-preorder

2130 **► Definition 107** (Co-inductive characterisation of the MUST-preorder). *For every finite LTS \mathcal{L}_A ,
2131 \mathcal{L}_B and every $X \in \mathcal{P}^+(A), q \in B$, the coinductive characterisation \triangleleft of the MUST-preorder
2132 is defined as the greatest relation such that whenever $X \triangleleft q$, the following requirements hold:*

- 2133 1. $X \downarrow$ implies $q \downarrow$,
- 2134 2. For each q' such that $q \xrightarrow{\tau} q'$, we have that $X \triangleleft q'$,

XX:62 Constructive characterisations of the must-preorder for asynchrony

- 2135 3. $X \downarrow$ and $q \xrightarrow{\tau}$ imply that there exist $p \in X$ and $p' \in A$ such that $p \Longrightarrow p' \xrightarrow{\tau}$ and
 2136 $O(p') \subseteq O(q)$,
- 2137 4. For any $\mu \in \text{Act}$, $X' \in \mathcal{P}^+(A)$ and $q \in B$ such that $X \Downarrow \mu$, $X \xrightarrow{\mu} X'$ and $q \xrightarrow{\mu} q'$, we
 2138 have that $X' \leq q'$

2139 ► **Theorem 108.** For every LTS $\mathcal{L}_A, \mathcal{L}_B \in \text{OBA}$, every $p \in A$ and $q \in B$, we have that
 2140 $p \sqsubseteq_{\text{MUST}} q$ if and only if $\text{FW}(\{p\}) \leq \text{FW}(q)$.

2141 I Counter-example to existing completeness result

2142 In this section we recall the definition of the alternative preorder \ll_{ch} by [39], and show that
 2143 it is not complete with respect to $\sqsubseteq_{\text{MUST}}$, i.e. $\sqsubseteq_{\text{MUST}} \not\subseteq \ll_{\text{ch}}$. We start with some auxiliary
 2144 definitions.

2145 The predicate $\overset{I}{\rightsquigarrow}$ is defined by the following two rules:

- 2146 ■ $p \overset{I}{\rightsquigarrow} p$ if $p \xrightarrow{\tau}$ and $I(p) \cap I = \emptyset$,
- 2147 ■ $p \overset{I \uplus \{a\}}{\rightsquigarrow} p''$ if $p \xrightarrow{a} p'$ 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

$$\mathcal{GA}(p, s, I) = \{O(p'') \mid p \xrightarrow{s}_{\text{fw}} p' \overset{I}{\rightsquigarrow} p''\}$$

The set of *input multisets* of a process p after a trace s is defined by

$$\text{IM}(p, s) = \{\{a_1, \dots, a_n\} \mid a_i \in \mathcal{N}, p \xrightarrow{s}_{\text{fw}} \xrightarrow{a_1} \dots \xrightarrow{a_n}\}$$

2148 The convergence predicate over traces performed by forwarders is denoted \Downarrow_a , and defined
 2149 as \Downarrow , but over the LTS given in Example 11.

2150 The preorder \ll_{ch} is now defined as follows:

2151 ► **Definition 109** (Alternative preorder \ll_{ch} [39]). Let $p \ll_{\text{ch}} q$ if for every $s \in \text{Act}^*$. $p \Downarrow_a s$
 2152 implies

- 2153 1. $q \Downarrow_a s$,
- 2154 2. for every $R \in \mathcal{A}(q, s)$ and every $I \in \text{IM}(p, s)$ such that $I \cap R = \emptyset$ there exists some
 2155 $O \in \mathcal{GA}(p, s, I)$ such that $O \setminus \bar{I} \subseteq R$. ■

2156 We illustrate the three auxiliary definitions using the process $\text{Pierre} = b.(\tau.\Omega + c.\bar{d})$
 2157 introduced in Example 8. We may infer that

$$2158 \text{Pierre} \overset{\{b,c\}}{\rightsquigarrow} \boxed{\bar{d}} \tag{25}$$

thanks to the following derivation tree

$$\frac{\frac{\frac{\boxed{\bar{d}} \xrightarrow{\emptyset} \boxed{\bar{d}}}{\boxed{\bar{d}} \xrightarrow{\tau} \text{and } I(\boxed{\bar{d}}) \cap \emptyset = \emptyset}}{\tau.\Omega + c.\bar{d} \xrightarrow{c} \boxed{\bar{d}}}}{\tau.\Omega + c.\bar{d} \overset{\{c\}}{\rightsquigarrow} \boxed{\bar{d}}}}{\text{Pierre} \overset{\{b,c\}}{\rightsquigarrow} \boxed{\bar{d}}} \text{Pierre} \xrightarrow{b} \tau.\Omega + c.\bar{d}$$

2159 Let us now consider the generalised acceptance set of *Pierre* after the trace ε with respect
2160 to the multiset $\{\{b, c\}\}$. We prove that

$$2161 \quad \mathcal{GA}(Pierre, \varepsilon, \{\{b, c\}\}) = \{\{\bar{d}\}\} \quad (26)$$

2162 By definition $\mathcal{GA}(Pierre, \varepsilon, \{\{b, c\}\}) = \{O(p'') \mid Pierre \xRightarrow{\varepsilon}_{fw} p' \xrightarrow{\{\{b, c\}\}} p''\}$. Since $Pierre \xrightarrow{\tau}$,
2163 we have

$$2164 \quad \mathcal{GA}(Pierre, \varepsilon, \{\{b, c\}\}) = \{O(p'') \mid Pierre \xrightarrow{\{\{b, c\}\}} p''\} \quad (27)$$

2165 Then, thanks to Equation (25) we get $O(\bar{d}) = \{\bar{d}\} \in \mathcal{GA}(Pierre, \varepsilon, \{\{b, c\}\})$. We show now
2166 that $\{\bar{d}\}$ is the only element of this acceptance set. By (27) above, it is enough to show that

2167 $Pierre \xrightarrow{\{\{b, c\}\}} p''$ implies $p'' = \bar{d}$. Observe that

- 2168 1. $I(Pierre) \cap \{\{b, c\}\} \neq \emptyset$,
- 2169 2. $Pierre \xrightarrow{a} p'$ implies $a = b$, and
- 2170 3. There are two different states p' such that $Pierre \xrightarrow{b} p'$, but the only one that can do
2171 the input c is $p' = \tau.\Omega + c.\bar{d}$.

2172 This implies that the only way to infer $Pierre \xrightarrow{\{\{b, c\}\}} p''$ is via the derivation tree that proves
2173 Equation (25) above. Thus $p'' = \bar{d}$.

2174 **► Counterexample 110.** *The alternative preorder \ll_{ch} is not complete for \sqsubseteq_{MUST} , namely*
2175 *$p \sqsubseteq_{MUST} q$ does not imply $p \ll_{ch} q$.*

2176 **Proof.** The cornerstone of the proof is the process $Pierre = b.(\tau.\Omega + c.\bar{d})$ discussed above. In
2177 Example 8 we have shown that $Pierre \sqsubseteq_{MUST} 0$. Here we show that $Pierre \not\ll_{ch} 0$, because the
2178 pair $(Pierre, 0)$ does not satisfy Condition 2 of Definition (109).

2179 Since $Pierre \xrightarrow{\tau}$, we know that $Pierre \downarrow$, and thus by definition $Pierre \downarrow_a \varepsilon$. We also
2180 have by definition $\mathcal{A}(0, \varepsilon) = \{\emptyset\}$, and $IM(Pierre, \varepsilon) = \{\emptyset, \{\{b\}\}, \{\{b, c\}\}\}$.

2181 Let us check Condition 2 of Definition (109) for $p = Pierre$ and $q = 0$. Since there is a
2182 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
2183 $I \in IM(Pierre, \varepsilon)$ there exists some $O \in \mathcal{GA}(Pierre, \varepsilon, I)$ such that $O \setminus \bar{I} \subseteq \emptyset$.

2184 Let $I = \{\{b, c\}\}$. By Equation (26) it must be $O = \{\bar{d}\}$. Since $\{\bar{d}\} \setminus \bar{I} = \{\bar{d}\} \setminus \overline{\{\{b, c\}\}} =$
2185 $\{\bar{d}\} \not\subseteq \emptyset$, the condition is not satisfied. Thus $Pierre \not\ll_{ch} 0$. ◀

2186 **J Highlights of the Coq mechanisation**

2187 **J.1 Preliminaries**

2188 We begin this section recalling the definition of MUST, which is given in Definition (2). It is
2189 noteworthy that the mechanised definition, i.e. `must_extensional`, depends on the typeclass
2190 `Sts` (Figure J.1.1), and *not* the type class `Lts`. This lays bare what stated in Section 1: to
2191 define MUST a reduction semantics (i.e. a state transition system), and a predicate GOOD
2192 over clients suffice.

2193 **J.1.1 State Transition Systems**

2194 The typeclass for state transition systems (Sts) is defined as follows, where **A** is the set of
2195 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;
}.

```

2196 J.1.2 Maximal computations

2197 A computation is maximal if it is infinite or if its last state is stable. Given a state s , the
 2198 type `max_exec_from s` contains all the maximal traces that start from s . Note the use of a
 2199 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.

```

2200 J.2 The must-preorder

2201 J.2.1 Client satisfaction

2202 The predicate `GOOD` is defined as any predicate over the states of an LTS that satisfies
 2203 certain properties: it is preserved by structural congruence, by outputs in both directions (if
 2204 $p \xrightarrow{\bar{a}} p'$ then $\text{GOOD}(p) \Leftrightarrow \text{GOOD}(p')$).

2205 It is defined as a typeclass indexed over the type of states and labels, because we expect
 2206 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;
}.

```

2207 J.2.2 Must testing

2208 Definition (2): We write $p \text{ MUST } r$ if every maximal computation of $p \parallel r$ is successful.

2209 Given an integer n and a maximal execution η , the function `mex_take_from n` applied to
 2210 η returns `None` if η is shorter than n and `Some p`, where p is a finite execution corresponding
 2211 to the first n steps of η .

2212 Then, we define the extensional version of $p\text{MUST}e$ by stating that, for all maximal
 2213 executions η starting from (p, e) , there exists an integer n such that the n -th element of η is
 2214 good. The n th element is obtained by taking the last element of the finite prefix of length n
 2215 computed using the function above.

```

Context `{good : B -> Prop}.

Fixpoint mex_take_from (n: nat) {x} ( $\eta$ : max_exec_from x) :
  option (finexec_from x) :=
  match n with
  | 0 => Some $ FExSingl x
  | S n => match  $\eta$  with
    | MExStop x Hstable => None
    | MExStep x x' Hstep  $\eta'$  =>
      let p' := mex_take_from n  $\eta'$  in
      ( $\lambda$  p', FExStep x x' (bool_decide_pack _ Hstep) p') <$> p'
    end
  end.

Definition must_extensional (p : A) (e : B) : Prop :=
  forall  $\eta$  : max_exec_from (p, e), exists n fex,
    mex_take_from n  $\eta$  = Some fex /\ good (fex_from_last fex).2.

```

2216 J.2.3 The preorder

2217 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  $\sqsubseteq_e$  q" := (pre_extensional p q).

```

2218 J.3 Behavioural characterizations

2219 J.3.1 Labeled Transition Systems

2220 An LTS is a typeclass indexed by the type of states and the type of labels. The type of
 2221 labels must be equipped with decidable equality and be countable, as enforced by the `Label`
 2222 typeclass. An action $a : \text{Act } L$ is either an internal action τ or an external action: an input
 2223 or an output of a label in L .

```

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

```

2224

XX:66 Constructive characterisations of the must-preorder for asynchrony

```

Inductive Act (A: Type) := ActExt ( $\mu$ : ExtAct A) |  $\tau$ .

Class Lts (A L : Type) `{Label L} := {
  lts_step: A  $\rightarrow$  Act L  $\rightarrow$  A  $\rightarrow$  Prop;
  lts_state_eqdec: EqDecision A;

  lts_step_decidable a  $\alpha$  b : Decision (lts_step a  $\alpha$  b);

  lts_outputs : A  $\rightarrow$  gset L;
  lts_outputs_spec1 p1 x p2 :
    lts_step p1 (ActExt (ActOut x)) p2  $\rightarrow$  x  $\in$  lts_outputs p1;
  lts_outputs_spec2 p1 x :
    x  $\in$  lts_outputs p1  $\rightarrow$  {p2 | lts_step p1 (ActExt (ActOut x)) p2};

  lts_stable: A  $\rightarrow$  Act L  $\rightarrow$  Prop;
  lts_stable_decidable p  $\alpha$  : Decision (lts_stable p  $\alpha$ );
  lts_stable_spec1 p  $\alpha$  :  $\neg$  lts_stable p  $\alpha$   $\rightarrow$  { q | lts_step p  $\alpha$  q };
  lts_stable_spec2 p  $\alpha$  : { q | lts_step p  $\alpha$  q }  $\rightarrow$   $\neg$  lts_stable p  $\alpha$ ;
}.

Notation "p  $\longrightarrow$  q" := (lts_step p  $\tau$  q).
Notation "p  $\longrightarrow$ {  $\alpha$  } q" := (lts_step p  $\alpha$  q).
Notation "p  $\longrightarrow$ [  $\alpha$  ] q" := (lts_step p (ActExt  $\mu$ ) q).

```

2225

2226 An LTS L is cast into an STS by taking only the τ -transitions, as formalised by the
 2227 following instance, which says that A can be equipped with an STS structure when, together
 2228 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 := lts_step p  $\tau$  q;
    sts_stable s := lts_stable s  $\tau$ ;
  | }.

```

2229 J.3.2 Weak transitions

2230 Let $\Longrightarrow \subseteq A \times \text{Act}^* \times A$ denote the least relation such that:

```

2231 [wt-refl]  $p \xrightarrow{\varepsilon} p'$ ,
2232 [wt-tau]  $p \xrightarrow{s} q$  if  $p \xrightarrow{\tau} p'$ , and  $p' \xrightarrow{s} q$ 
2233 [wt-mu]  $p \xrightarrow{\mu, s} q$  if  $p \xrightarrow{\mu} p'$  and  $p' \xrightarrow{s} q$ .

```

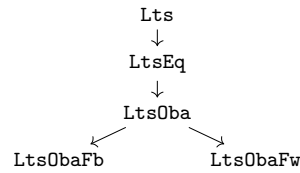
```

Definition trace L := list (ExtAct L).

Inductive wt : A  $\rightarrow$  trace L  $\rightarrow$  A  $\rightarrow$  Prop :=
  | wt_nil p : wt p [] p
  | wt_tau s p q t (l : p  $\longrightarrow$  q) (w : wt q s t) : wt p s t
  | wt_act  $\mu$  s p q t (l : p  $\longrightarrow$ [ $\mu$ ] q) (w : wt q s t) : wt p ( $\mu :: s$ ) t.

```

2234



■ **Figure 19** Typeclasses to formalise LTSs.

2235 **Notation** `"p \implies [s] q"` := `(wt p s q)`.

2236 J.3.3 Product of LTS

2237 The characteristic function of the transition relation of the LTS resulting from the parallel
 2238 composition of two LTS. States of the parallel product of L_1 and L_2 are pairs $(a, b) \in L_1 \times L_2$.
 2239 The first two cases correspond to unsynchronized steps from either LTS, and the third case
 2240 corresponds to the LTS taking steps with dual actions. The predicate `act_match l1 l2` states
 2241 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)
.

```

2242 J.4 Typeclasses for LTS

2243 The Selinger axioms for LTSs are represented as three typeclasses in our Coq development.

```

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;
  }.
2245

```

2246 J.4.1 Termination

2247 We write $p \downarrow$ and say that p *converges* if every sequence of τ -transitions performed by p
 2248 is finite. This is expressed extensionally by the property that all maximal computations
 2249 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.

```

2250 J.4.2 Convergence along a trace

2251 To define the behavioural characterisation of the preorder, we first define $\Downarrow \subseteq A \times \text{Act}^*$ as
 2252 the least relation such that,

2253 [cnv-epsilon] $p \Downarrow \varepsilon$ if $p \downarrow$,
 2254 [cnv-mu] $p \Downarrow \mu.s$ if $p \downarrow$ and for each p' , $p \xrightarrow{\mu} p'$ implies $p' \Downarrow s$.

2255 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).

```

2256 **J.5 Forwarders**2257 We define a mailbox MO as a multiset of names.

Definition mb ($L : \text{Type}$) $\{Label\ L\} := gmultiset\ L.$

2258 Definition (10) and Figure 6. Lifting of a transition relation to transitions of forwarders.

Inductive lts_fw_step $\{A\ L : \text{Type}\} \{Lts\ A\ L\} :$
 $A * mb\ L \rightarrow Act\ L \rightarrow A * mb\ L \rightarrow Prop :=$
 $| lts_fw_p\ p\ q\ m\ \alpha :$
 $lts_step\ p\ \alpha\ q \rightarrow lts_fw_step\ (p \triangleright m)\ \alpha\ (q \triangleright m)$
 $| lts_fw_out_mb\ m\ p\ a :$
 $lts_fw_step\ (p \triangleright \{[+ a +]\} \uplus m)\ (ActExt\ \$\ ActOut\ a)\ (p \triangleright m)$
 $| lts_fw_inp_mb\ m\ p\ a :$
 $lts_fw_step\ (p \triangleright m)\ (ActExt\ \$\ ActIn\ a)\ (p \triangleright \{[+ a +]\} \uplus m)$
 $| lts_fw_com\ m\ p\ a\ q :$
 $lts_step\ p\ (ActExt\ \$\ ActIn\ a)\ q \rightarrow$
 $lts_fw_step\ (p \triangleright \{[+ a +]\} \uplus m)\ \tau\ (q \triangleright m).$

2259 Definition (39) and Definition (40). For any LTS \mathcal{L} , two states of $FW(\mathcal{L})$ are equivalent,
2260 denoted $p \triangleright M \doteq q \triangleright N$, if $strip(p) \simeq strip(q)$ and $M \uplus mbox(p) = N \uplus mbox(q)$.

Inductive $strip$ $\{Lts\ A\ L\} : A \rightarrow gmultiset\ L \rightarrow A \rightarrow Prop :=$
 $| strip_nil\ p : p \rightsquigarrow\{\emptyset\} p$
 $| strip_step\ p1\ p2\ p3\ a\ m :$
 $p1 \xrightarrow{[ActOut\ a]}\ p2 \rightarrow p2 \rightsquigarrow\{m\} p3 \rightarrow p1 \rightsquigarrow\{\{[+ a +]\} \uplus m\} p3$

 where $"p \rightsquigarrow\{m\} q" := (strip\ p\ m\ q).$

Definition fw_eq $\{LtsOba\ A\ L\} (p : A * mb\ L) (q : A * mb\ L) :=$
 $forall\ (p'\ q' : A),$
 $p.1 \rightsquigarrow\{lts_oba_mo\ p.1\} p' \rightarrow$
 $q.1 \rightsquigarrow\{lts_oba_mo\ q.1\} q' \rightarrow$
 $p' \simeq q' \wedge lts_oba_mo\ p.1 \uplus p.2 = lts_oba_mo\ q.1 \uplus q.2.$

Infix $"\doteq"$ $:= fw_eq$ (at level 70).

2261 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
2262 $p \triangleright M (\doteq \cdot \xrightarrow{\alpha}_{fw}) q \triangleright N$ then $p \triangleright M (\xrightarrow{\alpha}_{fw} \cdot \doteq) q' \triangleright N'$.

Lemma $lts_fw_eq_spec$ $\{LtsObaFB\ A\ L\} p\ q\ t\ mp\ mq\ mt\ \alpha :$
 $p \triangleright mp \doteq t \triangleright mt \rightarrow (t \triangleright mt) \xrightarrow{\{\alpha\}} (q \triangleright mq) \rightarrow p \triangleright mp \xrightarrow{\doteq\{\alpha\}} q \triangleright mq.$

2263 Lemma 13. For every LTS $\mathcal{L} \in OF$, $FW(\mathcal{L}) \in OW$.

Program Instance $LtsMBObaFW$ $\{LtsObaFB\ A\ L\} : LtsObaFW\ (A * mb\ L)\ L.$

2264 Lemma 14. For every $\mathcal{L}_A, \mathcal{L}_B \in OF$, $p \in A, r \in B$, $p \text{MUST}_i r$ if and only if $FW(p) \text{MUST}_i r$.

XX:70 Constructive characterisations of the must-preorder for asynchrony

```
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.
```

2265 J.6 The Acceptance Set Characterisation

2266 The behavioural characterisation with acceptance sets (Definition 9) is formalised as follows.
2267 Note that `lts_outputs`, used in the second part of the definition, is part of the definition of
2268 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).
```

2269 Given an LTS that satisfies the right conditions, MUST-equivalence coincides with the
2270 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.
```

2271 J.7 The Must Set characterisation

2272 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.

2273 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  $\overset{\sim}{\sim}_2$  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  $\preceq_1$  q /\ p  $\overset{\sim}{\sim}_2$  q.
```

```
2274 Notation "p  $\overset{\sim}{\sim}$  q" := (bhv_pre_ms p q).
```

2275 Lemma 20. Let $\mathcal{L}_A, \mathcal{L}_B \in \text{OF}$. For every $p \in A$ and $q \in B$ such that $\text{FW}(p) \preceq_{\text{cnv}} \text{FW}(q)$,
2276 we have that $\text{FW}(p) \preceq_{\text{M}} \text{FW}(q)$ if and only if $\text{FW}(p) \preceq_{\text{acc}}^{\text{fw}} \text{FW}(q)$.

```
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,  $\emptyset$ )  $\preceq_1$  (q,  $\emptyset$ ) -> (p,  $\emptyset$ )  $\overset{\sim}{\sim}_2$  (q,  $\emptyset$ ) <-> (p,  $\emptyset$ )  $\preceq_2$  (q,  $\emptyset$ ).
```

2277 Given an LTS that satisfies the right conditions, MUST-equivalence coincides with the
2278 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  $\sqsubseteq_e$  q <-> (p,  $\emptyset$ )  $\overset{\sim}{\sim}$  (q,  $\emptyset$ ).
```

```
End correctness.
```

2279 J.8 From extensional to intensional definitions

2280 Proposition 31. Given a countably branching STS $\langle S, \rightarrow \rangle$, and a decidable predicate Q on S ,
2281 for all $s \in S$, $\text{ext}_Q(s)$ implies $\text{int}_Q(s)$.

```
Context `{@Hsts : Sts A, @CountableSts A Hsts}.
```

```
Context `{@Bar A Hsts}.
```

```
Theorem extensional_implies_intensional x :
  extensional_pred x -> intensional_pred x.
```

XX:72 Constructive characterisations of the must-preorder for asynchrony

2282 Corollary 24. For every $p \in A$,

2283 1. $p \downarrow$ if and only if $p \downarrow_i$,

2284 2. for every r we have that $p \text{ MUST } r$ if and only if $p \text{ MUST}_i r$.

```
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.
```

2285 Equivalence between the inductive definitions of MUST defined using Sts and MUST defined
2286 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.
```

2287 J.9 Completeness

2288 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 0 : gen_spec (gen_acc 0) ;
    gen_spec_acc_nil_stable_tau 0 : gen_acc 0 [] → ;
    gen_spec_acc_nil_stable_out 0 a : gen_acc 0 [] →[ActOut a] ;
    gen_spec_acc_nil_mu_inv 0 a e : gen_acc 0 [] →[ActIn a] e -> a ∈ 0 ;
    gen_spec_acc_nil_mem_lts_inp 0 a :
    a ∈ 0 -> ∃ r, gen_acc 0 [] →[ActIn a] r ;
    gen_spec_acc_nil_lts_inp_good μ e' 0 :
    gen_acc 0 [] →[μ] e' -> good e' ;
  }.

```

2289 Proposition 43. For every $\mathcal{L}_A \in \text{OW}$, $p \in A$, and $s \in \text{Act}^*$ we have that $p \text{MUST}_i tc(s)$ if
 2290 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.

```

2291 Lemma 45. 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
 2292 name $a \in \mathcal{N}$ such that $p_1 \xrightarrow{\bar{a}} p_2$ and $r_1 \xrightarrow{\bar{a}} r_2$, if $p_1 \text{MUST}_i r_2$ then $p_2 \text{MUST}_i r_1$.

```

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.

```

XX:74 Constructive characterisations of the must-preorder for asynchrony

2293 Lemma 46. Let $\mathcal{L}_A \in \text{OW}$. For every $p \in A$, $s \in \text{Act}^*$, and every $L, E \subseteq \mathcal{N}$, if
 2294 $\bar{L} \in \mathcal{A}_{\text{fw}}(p, s)$ then $p \text{MUST}_i \text{ta}(s, E \setminus L)$.

Lemma not_must_gen_a_without_required_output
 $\{ \text{@LtsObaFW } A \ L \ \text{IL } \text{LA } \text{LOA } \ V, \ \text{@LtsObaFB } B \ L \ \text{IL } \text{LB } \ \text{LOB } \ W, \\ \text{!Good } B \ L \ \text{good}, \ \text{!gen_spec_acc } \text{gen_acc} \} \ (q \ q' : A) \ s \ 0 : \\ q \implies [s] \ q' \ \rightarrow \ q' \ \nrightarrow \ \rightarrow \ \neg \ \text{must } q \ (\text{gen_acc } (0 \setminus \text{lts_outputs } q') \ s).$

2295 Lemma 47. Let $\mathcal{L}_A \in \text{OW}$. For every $p \in A$, $s \in \text{Act}^*$, and every finite set $O \subseteq \bar{\mathcal{N}}$, if
 2296 $p \Downarrow s$ then either

- 2297 (i) $p \text{MUST}_i \text{ta}(s, \bigcup \overline{\mathcal{A}_{\text{fw}}(p, s)} \setminus O)$, or
 2298 (ii) there exists $\hat{O} \in \mathcal{A}_{\text{fw}}(p, s)$ such that $\hat{O} \subseteq O$.

Lemma must_gen_a_with_s
 $\{ \text{@LtsObaFW } A \ L \ \text{IL } \text{LA } \ \text{LOA } \ V, \ \text{@LtsObaFB } B \ L \ \text{IL } \text{LB } \ \text{LOB } \ W, \\ \text{!FiniteLts } A \ L, \ \text{!Good } B \ L \ \text{good}, \ \text{!gen_spec_acc } \text{gen_acc} \} \\ s \ (p : A) \ (\text{hcnv} : p \Downarrow s) \ 0 : \\ (\text{exists } p', p \implies [s] \ p' \ \wedge \ \text{lts_stable } p' \ \tau \ \wedge \ \text{lts_outputs } p' \subseteq O) \\ \vee \ \text{must } p \ (\text{gen_acc } (\text{oas } p \ s \ \text{hcnv} \setminus 0) \ s).$

2299 Lemma 48. For every $\mathcal{L}_A, \mathcal{L}_B \in \text{OW}$ and servers $p \in A, q \in B$, if $p \sqsubseteq_{\text{MUST}} q$ then $p \preceq_{\text{AS}} q$.

Lemma completeness_fw
 $\{ \text{@LtsObaFW } A \ L \ \text{IL } \text{LA } \ \text{LOA } \ V, \ \text{@LtsObaFB } B \ L \ \text{IL } \text{LB } \ \text{LOB } \ W, \\ \text{@LtsObaFW } C \ L \ \text{IL } \text{LC } \ \text{LOC } \ \text{VC}, \ \text{!FiniteLts } A \ L, \ \text{!FiniteLts } C \ L, \\ \text{!FiniteLts } B \ L, \ \text{!Good } B \ L \ \text{good}, \\ \text{!gen_spec_conv } \text{gen_conv}, \ \text{!gen_spec_acc } \text{gen_acc} \} \\ (p : A) \ (q : C) : p \sqsubseteq q \ \rightarrow \ p \preceq q.$

2300 Proposition 49. 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
 2301 $\text{FW}(p) \preceq_{\text{AS}} \text{FW}(q)$.

Lemma completeness
 $\{ \text{@LtsObaFB } A \ L \ \text{IL } \text{LA } \ \text{LOA } \ V, \ \text{@LtsObaFB } B \ L \ \text{IL } \text{LB } \ \text{LOB } \ W, \\ \text{@LtsObaFB } C \ L \ \text{IL } \text{LC } \ \text{LOC } \ \text{VC}, \\ \text{!FiniteLts } A \ L, \ \text{!FiniteLts } B \ L, \ \text{!FiniteLts } C \ L, \ \text{!Good } C \ L \ \text{good}, \\ \text{!gen_spec_conv } \text{gen_conv}, \ \text{!gen_spec_acc } \text{gen_acc} \} \\ (p : A) \ (q : B) : p \sqsubseteq q \ \rightarrow \ p \triangleright \emptyset \preceq q \triangleright \emptyset.$

2302 J.10 Soundness

2303 Figure 12. Rules to define inductively the predicate MUST_{aux} .

Inductive mustx
 $\{ \text{Lts } A \ L, \ \text{!FiniteLts } A \ L, \ \text{!Lts } B \ L, \ \text{!LtsEq } B \ L, \ \text{!Good } B \ L \ \text{good} \} \\ (\text{ps} : \text{gset } A) \ (e : B) : \text{Prop} := \\ | \text{mx_now } (\text{hh} : \text{good } e) : \text{mustx } \text{ps } e \\ | \text{mx_step}$

```

2305   (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.

```

2306 Lemma 66. For every LTS $\mathcal{L}_A, \mathcal{L}_B$ and every $X \in \mathcal{P}^+(A)$, we have that $X \text{ MUST}_{\text{aux}} r$ if
2307 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.

```

2308 Lifting of the predicates \preceq_{cnv} and \preceq_{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).

```

2309 Lemma 67. For every LTS $\mathcal{L}_A, \mathcal{L}_B$ and servers $p \in A, q \in B$, $p \preceq_{\text{AS}} q$ if and only if
2310 $\{p\} \preceq_{\text{AS}}^{\text{set}} q$.

```

Lemma alt_set_singleton_iff
  `{@FiniteLts P L HL LtsP, @FiniteLts Q L HL LtsQ}
  (p : P) (q : Q) : p ≼ q <-> {[ p ]} ≼x q.

```

2311 Lemma 68. Let $\mathcal{L}_A, \mathcal{L}_B \in \text{OW}$. For every set $X \in \mathcal{P}^+(A)$, and $q \in B$, such that $X \preceq_{\text{cnv}}^{\text{set}} q$
2312 then

XX:76 Constructive characterisations of the must-preorder for asynchrony

- 2313 1. $q \xrightarrow{\tau} q'$ implies $X \preceq_{\text{cnv}}^{\text{set}} q'$,
 2314 2. $X \downarrow_i$, $X \xRightarrow{\mu} X'$ and $q \xrightarrow{\mu} q'$ imply $X' \preceq_{\text{cnv}}^{\text{set}} q'$.

2315 Lemma 69. Let $\mathcal{L}_A, \mathcal{L}_B \in \text{OW}$. For every $X, X' \in \mathcal{P}^+(A)$ and $q \in B$, such that $X \preceq_{\text{acc}}^{\text{set}} q$,
 2316 then

- 2317 1. $q \xrightarrow{\tau} q'$ implies $X \preceq_{\text{acc}}^{\text{set}} q'$,
 2318 2. for every $\mu \in \text{Act}$, if $X \downarrow_i$, then for every $q \xrightarrow{\mu} q'$ and set $X \xRightarrow{\mu} X'$ we have $X' \preceq_{\text{acc}}^{\text{set}} q'$.

Lemma bhvx_preserved_by_tau
 $\{ \text{@FiniteLts P L HL LtsP, @FiniteLts Q L HL LtsQ} \}$
 $(\text{ps} : \text{gset P}) (q \ q' : \text{Q}) : q \xrightarrow{\tau} q' \rightarrow \text{ps} \preceq_x q \rightarrow \text{ps} \preceq_x q'$.

Lemma bhvx_preserved_by_mu
 $\{ \text{@FiniteLts P L HL LtsP, @FiniteLts Q L HL LtsQ} \}$
 $(\text{ps0} : \text{gset P}) (q : \text{Q}) \mu \text{ ps1 } q'$
 $(\text{htp} : \text{forall } p, p \in \text{ps0} \rightarrow \text{terminate } p) :$
 $q \xrightarrow{[\mu]} q' \rightarrow \text{wt_set_from_pset_spec } \text{ps0 } [\mu] \text{ ps1} \rightarrow$
 $\text{ps0} \preceq_x q \rightarrow \text{ps1} \preceq_x q'$.

2319 Lemma 77 Let $\mathcal{L}_A, \mathcal{L}_B \in \text{OW}$ and $\mathcal{L}_C \in \text{OF}$. For every $X \in \mathcal{P}^+(A)$ and $q \in B$ such that
 2320 $X \preceq_{\text{AS}}^{\text{set}} q$, for every $r \in C$ if $\neg \text{GOOD}(r)$ and $X \text{ MUST}_{\text{aux}} r$ then $q \parallel r \xrightarrow{\tau}$.

Lemma stability_nbhvleqtwo
 $\{ \text{@LtsObaFW P L Lbl LtsP LtsEqP LtsObaP,}$
 $\text{@LtsObaFW Q L Lbl LtsQ LtsEqQ LtsObaQ,}$
 $!\text{FiniteLts P L, !FiniteLts Q L, !Lts B L, !LtsEq B L, !Good B L good} \}$
 $(X : \text{gset P}) (q : \text{Q}) e :$
 $\neg \text{good } e \rightarrow \text{mustx } X \ e \rightarrow X \preceq_x 2 \ q \rightarrow \text{exists } t, (q, e) \xrightarrow{\{\tau\}} t$.

2321 Lemma 78 Let $\mathcal{L}_A, \mathcal{L}_B \in \text{OW}$. For every $X \in \mathcal{P}^+(A)$ and $q, q' \in B$, such that $X \preceq_{\text{AS}}^{\text{set}} q$,
 2322 then for every $\mu \in \text{Act}$, if $X \downarrow \mu$ and $q \xrightarrow{\mu} q'$ then $X \xRightarrow{\mu}$.

Lemma bhvx_mu_ex $\{ \text{@FiniteLts P L HL LtsP, @FiniteLts Q L HL LtsQ} \}$
 $(\text{ps} : \text{gset P}) (q \ q' : \text{Q}) \mu$
 $: \text{ps} \preceq_x q \rightarrow (\text{forall } p, p \in \text{ps} \rightarrow p \downarrow [\mu]) \rightarrow$
 $q \xrightarrow{[\mu]} q' \rightarrow \text{exists } p', \text{wt_set_from_pset_spec1 } \text{ps } [\mu] \{ [p'] \}$.

2323 Lemma 79 For every $\mathcal{L}_A \in \text{OW}$, $\mathcal{L}_B \in \text{OF}$, every set of processes $X \in \mathcal{P}^+(A)$, every
 2324 $r \in B$, and every $\mu \in \text{Act}$, if $X \text{ MUST}_{\text{aux}} r$, $\neg \text{GOOD}(r)$ and $r \xrightarrow{\mu}$ then $X \downarrow \bar{\mu}$.

Lemma ungood_acnv_mu $\{ \text{LtsOba A L, !FiniteLts A L, !Lts B L, !LtsEq B L,}$
 $!\text{Good B L good} \}$ $\text{ps } e \ e' : \text{mustx } \text{ps } e \rightarrow$
 $\text{forall } \mu \ p, p \in \text{ps} \rightarrow e \xrightarrow{[\text{co } \mu]} e' \rightarrow \neg \text{good } e \rightarrow p \downarrow [\mu]$.

2325 Lemma 70. Let $\mathcal{L}_A, \mathcal{L}_B \in \text{OW}$ and $\mathcal{L}_C \in \text{OF}$. For every set of processes $X \in \mathcal{P}^+(A)$,
 2326 server $q \in B$ and client $r \in C$, if $X \text{ MUST}_{\text{aux}} r$ and $X \preceq_{\text{AS}}^{\text{set}} q$ then $q \text{ MUST}_i r$.

```

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.

```

2327 Proposition 71. For every $\mathcal{L}_A, \mathcal{L}_B \in \text{OF}$ and servers $p \in A, q \in B$, if $\text{FW}(p) \preceq_{\text{AS}} \text{FW}(q)$
 2328 then $p \sqsubseteq_{\text{MUST}} q$.

```

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.

```

2329 ► **Corollary 111.** Let $\mathcal{L}_A, \mathcal{L}_B \in \text{OF}$. For every $p \in A$ and $q \in B$, we have that $p \sqsubseteq_{\text{MUST}} q$ if
 2330 and only if $p \leq_{\text{fail}} q$.

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.

2331 **K** 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
$p \xrightarrow{s} p'$	TransitionSystems.v	wt
$p \downarrow$	Equivalence.v	terminate_extensional
$p \downarrow s$	TransitionSystems.v	cnv
Lemma 51	TransitionSystems.v	cnv_iff_prefix_terminate
Lemma 52	TransitionSystems.v	stable_tau_preserved_by_wt_output, stable_tau_input_preserved_by_wt_output
Lemma 50	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
2332 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
$p \text{ MUST}_i q$	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 43	Completeness.v	must_iff_cnv
Lemma 45	Lift.v	must_output_swap_l_fw
Lemma 46	Completeness.v	not_must_gen_a_without_required_output
Lemma 47	Completeness.v	must_gen_a_with_s
Lemma 48	Completeness.v	completeness_fw
Proposition 49	Completeness.v	completeness
Lemma 66	Soundness.v	must_set_iff_must_for_all
Figure 12	Soundness.v	mustx
$X \xrightarrow{\text{set}_{\text{cnv}}} q$ and $X \xrightarrow{\text{set}_{\text{acc}}} q$	Soundness.v	bhv_pre_cond1 _x and bhv_pre_cond2 _x
Lemma 67	Soundness.v	must_set_iff_must
Lemma 68, Lemma 69	Soundness.v	bhvx_preserved_by_tau, bhvx_preserved_by_mu
Lemma 77	Soundness.v	stability_nbhvleqtwo
Lemma 78	Soundness.v	bhvx_mu_ex
Lemma 70	Soundness.v	soundnessx
Proposition 71	Soundness.v	soundness