Science of Computer Programming 168 (2018) 94–117

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Full-abstraction for client testing preorders

Giovanni Bernardi[a,*], Adrian Francalanza[b]

[a] Université Paris-Diderot/IRIF, Paris, France
[b] University of Malta, Msida, Malta

ARTICLE INFO

Article history:
Received 30 October 2017
Received in revised form 9 July 2018
Accepted 20 August 2018
Available online 30 August 2018

Keywords:
Behavioural equivalences
Full-abstraction
Foundations of web-services

ABSTRACT

Client testing preorders relate tests (clients) instead of processes (servers), and are usually defined using either must testing or a compliance relation. Existing characterisations of these preorders are unsatisfactory for they rely on the notion of *usable* clients which, in turn, are defined using an existential quantification over the servers that ensure client satisfaction. In this paper we characterise the set of usable clients wrt must testing for finite-branching LTSs, and give a sound and complete decision procedure for it. We also provide novel coinductive characterisations of the client preorders due to must and compliance, which we use to show that these preorders are decidable, thus positively answering the question opened in [5,3].

1. Introduction

The testing theory of De Nicola–Hennessy [14,19] is a well-known approach to define preorders and equivalences for communicating processes. In this theory a process p_2 is considered as good as another process p_1 if *every* test r passed by p_1 is also passed by p_2, and two processes are equivalent if they pass the same tests. The standard notion of passing a test is formalised predominantly by the so called must testing relation: $p \; \mathsf{must} \; r$ whenever every run of the system $r \,||\, p$ leads the test r to a successful state. Concretely, the formal definition of the well-known must preorder is thus

$$p_1 \sqsubseteq\!\!\!\!\sim p_2 \text{ iff } \forall r\,.\,(p_1 \; \mathsf{must} \; r) \text{ implies } (p_2 \; \mathsf{must} \; r) \tag{1}$$

During the last decade, testing theory has been adapted and enriched to lay the theoretical foundations for web-services, where processes are seen as servers, and tests as clients (or peers). *Adapted* in that an alternative relation to must has been proposed, which fits better the setting of web-services and client/server satisfaction. This novel relation, called *compliance*, states that a client r complies with a server p, denoted $r \; \mathsf{cmp} \; p$, if whenever a computation of $r \,||\, p$ cannot go on or p diverges, the client is in a successful state [10,29]. *Enriched* in that in addition to the classical preorder for servers,[1] also preorders for clients and peers have been investigated [2,5]. Preorders for clients have a natural definition similar to (1), for example the compliance client preorder is defined by letting

$$r_1 \sqsubseteq_{\mathsf{cmp}} r_2 \text{ if } \forall p\,.\,(r_1 \; \mathsf{cmp} \; p) \text{ implies } (r_2 \; \mathsf{cmp} \; p) \tag{2}$$

* Corresponding author.
E-mail addresses: gio@irif.fr (G. Bernardi), adrian.francalanza@um.edu.mt (A. Francalanza).

[1] *Processes* according to the classic terminology.

https://doi.org/10.1016/j.scico.2018.08.004
0167-6423/© 2018 Elsevier B.V. All rights reserved.

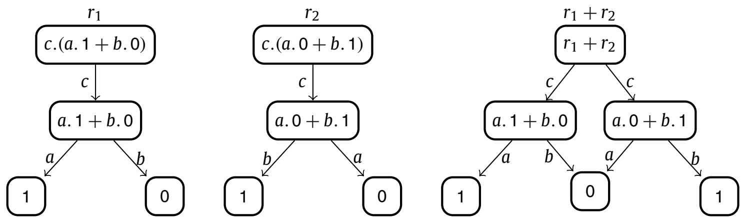

Fig. 1. LTS depictions of the behaviours described in Eq. (3).

that is a client r_2 is as good as a client r_1 whenever *every* server p that satisfies r_1 also satisfies r_2. In this paper we dwell on the client preorders due to the must relation [5] and the compliance relation [3].

Definitions such as (1) and (2) are intuitive and easy to understand, but they are hard to use in practice for they are *contextual*: they contain a universal quantification over contexts, and hence give no effective proof method to determine pairs in the preorders being defined. To overcome this problem, testing preorders are usually presented together with *alternative characterisations* that avoid universal quantification over contexts, and that are amenable to the development of proof methods and decision procedures.

In [5,3] the authors develop such characterisations for the client preorders due to must and compliance, however neither of these characterisations are fully-abstract, nor are decision procedures for the preorders discussed. In particular, the alternative preorders given in [5, Definition 3.10] and [3, Definition 5.2.16] are not fully-abstract for they are defined modulo *usable* clients, *i.e.,* clients that are satisfied by *at least one* server. In other words, the definitions of these preorders rely explicitly on the interactions that clients may have with servers.

Usability is a pivotal notion that appears frequently in the literature on process calculi as foundations for web-service: it has been called viability in [22,30] and controllability in [9,28], and has already been studied in various settings [22,7,5,30]. While usable clients wrt the compliance relation have been characterised in [29], the situation remains unclear with respect to the must testing. The characterisation of usable clients is indeed problematic, for solving it requires finding the conditions under which one can either (a) construct a server p that satisfies a given client, or (b) show that every p does *not* satisfy a given client. Whereas proving (b) is complicated by the universal quantification over *all* servers, the proof of (a) is complicated by the non-deterministic behaviour of clients. In particular, determining usability using approach (a) is complicated because client usability is *not* compositional. For instance consider the following two clients, whose behaviour is depicted in Fig. 1:

$$r_1 = c.(a.1 + b.0) \qquad \text{and} \qquad r_2 = c.(a.0 + b.1) \tag{3}$$

where 1 denotes satisfaction (success). Both clients are usable, since r_1 is satisfied by the server $\overline{c}.\overline{a}.0$, and r_2 is satisfied by the server $\overline{c}.\overline{b}.0$. However, their composition $r_1 + r_2$ is *not* a usable client, *i.e.,* $p \neg\mathsf{must}\ r_1 + r_2$ for every p; intuitively, this is because r_1 and r_2 impose opposite constraints on the processes that pass one or the other (*e.g.,* $\overline{c}.(\overline{a}.0 + \overline{b}.0)$ does not satisfy $r_1 + r_2$). A compositional analysis is even more unwieldy for recursive tests. For instance, the recursive client $\mu x.\big(c.(a.1 + b.x) + c.(a.0 + b.1)\big)$ is *not* usable wrt must because of the non-determinism analogous to $r_1 + r_2$, and the unsuccessful computations along the infinite trace $(c.b)^*$; this argument works because infinite unsuccessful computations are catastrophic in must testing settings.

This paper presents a sound and complete characterisation for usable clients wrt must within a finite-branching LTS. Through the results of [5] — in particular, the equivalence of usability for clients and peers stated on [5, pag. 11] — our characterisation directly yields a fully-abstract characterisation for the must preorder for clients and peers. These characterisations, though, are still hard to use in practice when reasoning on recursive clients. Spurred by this observation, we define a new *coinductive* and fully-abstract characterisation for the client preorders due to must and compliance, which we find easier to use than the ones of [5,3]. These coinductive characterisation are informed by our study on usability, and differs subtly from the coinductive characterisations of preorders for servers given in [22,29,6]. Finally, our inductive definition for usable clients also provides insights into the must client preorder of [5]: we show that limiting contexts to servers offering only *finite* interactions preserves the discriminating power of the original preorder. The contributions of this paper are thus:

- a fully-abstract characterisation of usable clients wrt must (Theorem 1);
- two coinductive, fully-abstract characterisations of the client preorders due to must (Theorem 2) and compliance (Theorem 5);
- a proof that non-recursive contexts are sufficient to define the client preorder due to must (Theorem 3);
- decidability results for usable clients and the client preorder due to must (Theorem 4).

We hope that this work has an impact outside of testing theory and foundations for web-services, in the following sense. Our original motivation to study usability of clients was to arrive at decision procedures for client preorders. However, it turns out that our study of usability is relevant to controllability issues in service-oriented and monitor-oriented architectures [25,34,16]. For instance, the symbolic characterisation for consistently-detecting monitors in [16] called controllability

Syntax $\quad p, q, r, o \in \mathsf{CCS}^{\mu} ::= 0 \mid 1 \mid \alpha.p \mid p + q \mid \mu x.p \mid x$

Semantics

$$\frac{}{1 \xrightarrow{\checkmark} 0}\ (\text{A-OK}) \qquad \frac{}{\alpha.p \xrightarrow{\alpha} p}\ (\text{A-PRE})$$

$$\frac{}{\mu x.p \xrightarrow{\tau} p\{^{\mu x.p}/_{x}\}}\ (\text{A-UNFOLD})$$

$$\frac{p \xrightarrow{\lambda} p'}{p + q \xrightarrow{\lambda} p'}\ (\text{R-EXT-L}) \qquad \frac{q \xrightarrow{\lambda} q'}{p + q \xrightarrow{\lambda} q'}\ (\text{R-EXT-R})$$

Semantics of process composition

$$\frac{p \xrightarrow{\lambda} p'}{p \mid\mid r \xrightarrow{\lambda} p' \mid\mid r}\ (\text{P-SRV}) \qquad \frac{r \xrightarrow{\lambda} r'}{p \mid\mid r \xrightarrow{\lambda} p \mid\mid r'}\ (\text{P-CLI})$$

$$\frac{p \xrightarrow{a} p' \quad r \xrightarrow{\overline{a}} r'}{p \mid\mid r \xrightarrow{\tau} p' \mid\mid r'}\ (\text{P-SYN})$$

Fig. 2. Syntax and Semantics of recursive CCS^{μ} with 1.

for the instrumentation relation of [15] is closely related to the behavioural characterisation of usability given in this paper. Our techniques may also be extended beyond this remit. The ever growing sizes of test suites, together with the ubiquitous reliance on testing for the increasing quality-assurance requirements in software systems, has recently directed the attention to *flaky* (*i.e.*, non-deterministic) tests. Such tests arise frequently in practice and their impact on software development and productivity has been the subject of various studies in the software engineering field [26,24,23]. By some measures, $\approx 4.56\%$ of failures of tests the TAP (Test Anything Protocol) system at Google are caused by flaky tests [23]. We believe that our concepts, models and procedures can be extended to such testing methodologies and analyse detrimental non-deterministic behaviour arising in test suites, thereby reducing the gap between empirical practices and theory. Concretely, we conjecture that our techniques can be adapted to identify and filter out flaky tests, thereby improving the quality of test suites and effectiveness of test procedures.

Structure of the paper: This paper is the full version of an extended abstract presented at COORDINATION 2017 [8], and it adheres to the general structure of that abstract, but with additional results on the compliance client preorder. Section 2 sketches the preliminaries to define must testing and compliance, and recalls an existing characterisation of the client preorder due to must. Section 3 presents a fully-abstract characterisation of usable clients wrt must testing, and Section 4 uses this result to give a fully-abstract characterisation of the must client preorder. In Section 5 we present expressiveness results for servers with finite interactions together with decidability results for client usability. This concludes our study of the client preorder due to must. Section 6 shifts the attention to compliance, and we present a fully-abstract characterisation of the compliance client preorder, with a digression on decidability. Section 7 discusses related work and concludes.

2. Preliminaries

We recall the standard definitions of a recursive variant of CCS [27], where processes are used to describe contracts. Let $a, b, c, \ldots \in \mathsf{Act}$ be a set of actions, and let τ, $\checkmark$ be two distinct actions *not* in Act; the first denotes *internal* unobservable activity whereas the second is used to *report success*. To emphasise their distinctness, we use Act_τ to denote $\mathsf{Act} \cup \{\tau\}$, and similarly for $\mathsf{Act}_\checkmark$ and $\mathsf{Act}_{\tau\checkmark}$. We range over Act_τ with α, over $\mathsf{Act}_\checkmark$ with $\alpha_\checkmark$, and with λ over $\mathsf{Act}_{\tau\checkmark}$. We assume Act has an involution function, with $\overline{a}$ being the complement to a.

A *labelled transition system*, LTS, consists of a triple $\langle \mathsf{Proc}, \mathsf{Act}_{\tau\checkmark}, \longrightarrow \rangle$, where Proc is a set of processes and $\longrightarrow \subseteq (\mathsf{Proc} \times \mathsf{Act}_{\tau\checkmark} \times \mathsf{Proc})$ is a transition relation between processes decorated with labels drawn from the set $\mathsf{Act}_{\tau\checkmark}$; we write $p \xrightarrow{\lambda} q$ in lieu of $(p, \lambda, q) \in \longrightarrow$. An LTS is *finite-branching* if for all $p \in \mathsf{Proc}$ the set $\{(p, \lambda, q) \mid \exists \lambda \in \mathsf{Act}. \exists q \in \mathsf{Proc}. p \xrightarrow{\lambda} q\}$ is finite, and *image-finite* if for all $p \in \mathsf{Proc}$ and $\lambda \in \mathsf{Act}$ the set $\{(p, \lambda, q) \mid \exists q \in \mathsf{Proc}. p \xrightarrow{\lambda} q\}$ is finite [18, Definition 2.7]. For $s \in (\mathsf{Act}_\checkmark)^\star$ we define the standard weak transitions, $p \overset{s}{\Longrightarrow} q$, by *ignoring* the occurrences of τs. We say that a state p is *successful* if $p \xrightarrow{\checkmark}$, and that it is *stable* if $p \overset{\tau}{\not\longrightarrow}$.

We limit ourselves to finite-branching LTS to express both servers and clients. Whenever sufficient, we describe such LTS using a version of CCS with recursion [27] and augmented with a *success* operator, denoted as 1. The syntax of this language is depicted in Fig. 2 and assumes a denumerable set of variables $x, y, z \ldots \in \mathsf{Var}$. For finite I, we use the notation $\sum_{i \in I} p_i$ to denote the respective sequence of summations $p_1 + \ldots + p_n$ where $I = 1..n$. Similarly, when I is a non-empty set, we define $\bigoplus_{i \in I} p_i = \sum_{i \in I} \tau.p_i$ to represent process *internal* choice [19]. As usual, $\mu x.p$ binds x in p and we identify terms up to alpha conversion of bound variables. The operation $p\{^{\mu x.p}/_{x}\}$ denotes the unfolding of the recursive process $\mu x.p$, by substituting the term $\mu x.p$ for the free occurrences of the variable x in p.

In the rest of the paper we use the LTS whose states are the *closed* terms in CCS^{μ} (i.e. terms that contain only bound variables), and where the transition relation $p \xrightarrow{\lambda} q$ between closed terms of the language is the least one determined by the (standard) rules in Fig. 2. In general though our arguments are independent of the syntax, and rely only on hypothesis over the LTS.

To model the interactions taking place between the server and the client contracts, we use the standard binary composition of contracts [10], $p \,||\, r$, whose operational semantics is given in Fig. 2. A *computation* consists of a sequence of τ actions of the form

$$p \,||\, r = p_0 \,||\, r_0 \xrightarrow{\tau} p_1 \,||\, r_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} p_k \,||\, r_k \xrightarrow{\tau} \ldots \tag{4}$$

It is *maximal* if it is infinite, or whenever $p_n \,||\, r_n$ is the last state then $p_n \,||\, r_n \overset{\tau}{\not\longrightarrow}$. A computation may be viewed as two processes p, r, one a server (p) and the other a client (r) co-operating to achieve individual goals. We say that the computation (4) is *client-successful* if there exists some $k \geq 0$ such that $r_k \xrightarrow{\checkmark}$.

Definition 1 *(*must *client preorder [5]).* We write p must r when *every* maximal computation from $p \,||\, r$ is *client-successful*, and write $r_1 \sqsubseteq_{\mathsf{must}} r_2$ if, for *every* p, p must r_1 implies p must r_2. ■

Although intuitive, the universal quantification on servers in Definition 1 complicates reasoning about $\sqsubseteq_{\mathsf{must}}$. One way of surmounting this is by defining alternative characterisations for $\sqsubseteq_{\mathsf{must}}$ of Definition 1, that come equipped with practical proof methods.

2.1. Alternative characterisation of the must client preorder

Definition 3.10 of [5] gives an alternative characterisation for the preorder $\sqsubseteq_{\mathsf{must}}$, which is proven to be sound and complete. We recall this characterisation, restating the respective notation. The alternative characterisation relies on *unsuccessful* traces: $r \overset{s}{\Longrightarrow}_{\not\checkmark} r'$ means that r may weakly perform the trace of (weak) external actions s reaching state r' *without* passing through *any* successful state; in particular neither r nor r' are successful. Formally, $r \overset{s}{\Longrightarrow}_{\not\checkmark} r'$ is the least relation satisfying

(a) $r \overset{\checkmark}{\not\longrightarrow}$ implies $r \overset{\varepsilon}{\Longrightarrow}_{\not\checkmark} r$, and
(b) if $r'' \overset{s}{\Longrightarrow}_{\not\checkmark} r'$ and $r \overset{\checkmark}{\not\longrightarrow}$ then (i) $r \xrightarrow{a} r''$ implies $r \overset{as}{\Longrightarrow}_{\not\checkmark} r'$, and (ii) $r \xrightarrow{\tau} r''$ implies $r \overset{s}{\Longrightarrow}_{\not\checkmark} r'$.

The *unsuccessful* acceptance sets of r after s, are defined as

$$\mathsf{Acc}_{\not\checkmark}(r, s) = \{\, S(r') \mid r \overset{s}{\Longrightarrow}_{\not\checkmark} r' \overset{\tau}{\not\longrightarrow} \,\} \tag{5}$$

where $S(r) = \{a \in \mathsf{Act} \mid r \xrightarrow{a}\}$ denotes the *strong actions* of r. Intuitively, for the client r, the set $\mathsf{Acc}_{\not\checkmark}(r, s)$ records all the actions that lead r out of *potentially deadlocked* (i.e. stable) states that it reaches performing *unsuccessfully* the trace s. It turns out that these abstractions are fundamental to characterise must-testing preorders and also compliance preorders [3,5,29]. In the sequel, we shall also use the judgement $r \xrightarrow{\alpha}_{\not\checkmark} r'$ whenever $r \xrightarrow{\alpha} r'$, $r \overset{\checkmark}{\not\longrightarrow}$ and $r' \overset{\checkmark}{\not\longrightarrow}$ hold.

Example 1. Consider the client $r_3 = \tau.(1 + \tau.0)$ that can perform a silent transition to reach a state where it can produce the success action $\checkmark$,

$$\tau.(1 + \tau.0) \xrightarrow{\tau} 1 + \tau.0 \xrightarrow{\checkmark}$$

but can also silently transition again to a deadlocked state 0.

$$\tau.(1 + \tau.0) \xrightarrow{\tau} 1 + \tau.0 \xrightarrow{\tau} 0$$

We have $\mathsf{Acc}_{\not\checkmark}(r_3, \varepsilon) = \emptyset$ since the deadlocked state above can only be reached by passing through the success state $1 + \tau.0$. However, for the client $r_3' = r_3 + \tau.0$ we have $\mathsf{Acc}_{\not\checkmark}(r_3', \varepsilon) = \{\emptyset\}$ since we can reach a deadlock state without passing through a success state, namely $r_3' \xrightarrow{\tau} 0$. In the case of the client $r_3'' = r_3 + \mu x.x$, we also have $\mathsf{Acc}_{\not\checkmark}(r_3'', \varepsilon) = \emptyset$ since the additional branch $\mu x.x$ never reaches a deadlocked state. ■

The example above illustrates the fact that, whenever $\mathsf{Acc}_{\not\checkmark}(r, s) = \emptyset$, then any sequence of moves with trace s from r to a *stable* reduct r' must pass through a successful state, for otherwise we would have $S(r') \in \mathsf{Acc}_{\not\checkmark}(r, s)$ for some r'.

The alternative characterisation for the preorder $\sqsubseteq_{\mathsf{must}}$ of [5, Def. 3.10] also relies on the notion of *usable* clients.

Definition 2 *(Usable Clients).* $\mathcal{U}_{\mathsf{must}} = \{ r \mid \exists p.\, p \;\mathsf{must}\; r \}$. ■

Example 2. Recall clients $r_1 = c.(a.1 + b.0)$ and $r_2 = c.(a.0 + b.1)$ from (3) and Fig. 1. We can show that despite being individually usable, the sum of these clients, $r_1 + r_2$, is not, *i.e.*, for *every* p we have $p \,\neg\mathsf{must}\; r_1 + r_2$.

To see why this is the case, fix a process p. If p does not offer an interaction on $\overline{c}$, then, plainly, $p \,\neg\mathsf{must}\; r_1 + r_2$. Suppose that $p \xrightarrow{\overline{c}} p'$; to prove $p \,\neg\mathsf{must}\; r_1 + r_2$, it suffices to show that there exists a client r reached by $r_1 + r_2$ by performing action c, namely $r \in \{a.1 + b.0, a.0 + b.1\}$, such that $p' \,\neg\mathsf{must}\; r$. Consider the case $r = a.1 + b.0$. If $p' \,\neg\mathsf{must}\; r$ we are done. Alternatively, if $p' \;\mathsf{must}\; r$, this implies p' has to interact on a and *not* on b: in this case we can conclude that p' does not satisfy the other derivative $r = a.0 + b.1$, *i.e.*, $p' \,\neg\mathsf{must}\; r$ since the composition $p' \,||\, r$ is stable but *not* client-successful. Using a symmetric argument we also deduce that if $p' \;\mathsf{must}\; a.0 + b.1$ then $p' \,\neg\mathsf{must}\; a.1 + b.0$. This means that no process p exists that satisfies $r_1 + r_2$; note that the argument above crucially exploits the external non-determinism of $r_1 + r_2$.

The client $Recx\big(c.(a.1 + b.x) + c.(a.0 + b.1)\big)$ from Section 1 is unusable for similar reasons, but the analysis is more complicated, because we need to consider infinite computations. ■

The alternative characterisation for the preorder $\sqsubseteq_{\mathsf{must}}$ of [5, Def. 3.10] also uses machinery that centres around the notion of an *unsuccessful trace*. We let

$$(r \;\mathsf{after}_{\not\checkmark}\; s) = \{ r' \mid r \overset{s}{\Longrightarrow}_{\not\checkmark} r' \}$$

to denote the *residuals* of r after performing the *unsuccessful* trace s. We extend the notion of usability and say that r is *usable along* an unsuccessful trace s whenever $r\; usbl_{\not\checkmark}\; s$, which is the least predicate satisfying the following conditions:

(a) $r\; usbl_{\not\checkmark}\; \varepsilon$ whenever $r \in \mathcal{U}_{\mathsf{must}}$, and
(b) $r\; usbl_{\not\checkmark}\; as$ if (i) $r \in \mathcal{U}_{\mathsf{must}}$ and (ii) if $r \overset{a}{\Longrightarrow}_{\not\checkmark}$ then $\bigoplus(r \;\mathsf{after}_{\not\checkmark}\; a)\; usbl_{\not\checkmark}\; s$.

Note that whenever $r\; usbl_{\not\checkmark}\; s$, any state reachable from r by performing any unsuccessful subsequence of s is usable [5]. Finally, the set

$$ua_{\mathsf{clt}}(r, s) = \{ a \in \mathsf{Act} \mid r \overset{sa}{\Longrightarrow}_{\not\checkmark} \text{ implies } r\; usbl_{\not\checkmark}\; sa \}$$

denotes all the *usable actions* for a client r after the unsuccessful trace s.

Definition 3 *(Semantic client-preorder [5]).* Let $r_1 \lesssim_{\mathsf{clt}} r_2$ if, for every $s \in \mathsf{Act}^\star$ such that $r_1\; usbl_{\not\checkmark}\; s$, we have

(i) $r_2\; usbl_{\not\checkmark}\; s$,
(ii) $\forall B \in \mathsf{Acc}_{\not\checkmark}(r_2, s).\, \exists A \in \mathsf{Acc}_{\not\checkmark}(r_1, s)$ such that $A \cap ua_{\mathsf{clt}}(r_1, s) \subseteq B$,
(iii) $r_2 \overset{s}{\Longrightarrow}_{\not\checkmark}$ implies $r_1 \overset{s}{\Longrightarrow}_{\not\checkmark}$. ■

Lemma 1 *(König's infinity lemma [21]). If G is a finite-branching connected graph with infinitely many nodes, then G contains an infinitely long simple path.*

Proposition 1. *In any finite-branching LTS, $r_1 \sqsubseteq_{\mathsf{must}} r_2$ if and only if $r_1 \lesssim_{\mathsf{clt}} r_2$.*

Proof. Follows from [5, Theorem 3.13] and Lemma 1. □

Thanks to Proposition 1, Definition 3 sheds light on the behavioural properties observed by clients related by $\sqsubseteq_{\mathsf{must}}$. Moreover, the definition shares a similar structure to well-studied characterisations of the (standard) must-testing preorder of [14,19], where process convergence is replaced by client usability, and traces and acceptance sets are replaced by their unsuccessful counterparts (modulo usable actions). In spite of these advantages, Definition 3 has two drawbacks. First, it is parametric *wrt.* the set of usable clients $\mathcal{U}_{\mathsf{must}}$ (Definition 2), which relies on an existential quantifications over servers. As a result, the definition is *not* fully-abstract, and this makes it hard to use as proof technique and to ground decision procedures for $\sqsubseteq_{\mathsf{must}}$ on it. Second, Definition 3 requires us to consider an infinite number of (unsuccessful) traces to establish client inequality. In the next example we show why this is a problem in practice.

Example 3. The use of $\lesssim_{\mathsf{clt}}$ is hindered by the universal quantification over traces in its definition. Consider for instance clients r_4 and r_5,

$$r_4 = a.1 + \mu y.(a.r_3'' + b.y + c.1) \qquad \text{and} \qquad r_5 = (\mu z.(b.z + c.1)) + d.1$$

where $r_3'' = (\tau.(1 + \tau.0)) + \mu x.x$ from Example 1. One way to prove $r_4 \sqsubseteq_{\mathsf{must}} r_5$ amounts to showing that $r_4 \lesssim_{\mathsf{clt}} r_5$. The definition of $\lesssim_{\mathsf{clt}}$ requires us to show that for *every* trace $s \in \mathsf{Act}^\star$ where $r_4\; usbl_{\not\checkmark}\; s$ holds, clauses (i), (ii) and (iii) of

Definition 3 also hold. In this case, there are *infinitely* many unsuccessful traces s to consider and, a priori, there is no clear way how to do this in finite time. Specifically, there are unsuccessful traces that r_4 *can* perform while remaining usable at every step, such as $s = b^n$, and also unsuccessful traces that r_4 *cannot* perform (which trivially imply $r_4 \; usbl_{\not\checkmark} \; s$ according to the definition in Section 2.1), such as $s = d(b^n)$, $s = (db)^n$ or $s = (ac)^n$. □

We conclude the section gathering a few properties of the must relation, that we will use in some of the proofs hereafter.

Lemma 2. *For every* $p, r \in \mathsf{Proc}$*, if* $p \; \mathsf{must} \; r$ *then*

(1) $r \Downarrow_{\checkmark}$*, and*
(2) *for every* $s \in \mathsf{Act}^\star$*, if* $p \overset{\bar{s}}{\Longrightarrow} p'$ *then*
  (i) $r \overset{s}{\Longrightarrow}_{\not\checkmark} r'$ *implies* $p' \; \mathsf{must} \; r'$*, and*
  (ii) $r \; usbl_{\not\checkmark} \; s$*.*

Proof. See Lemma 4.5, Lemma 4.1, and Corollary 4.2 in [5]. □

3. Characterising usability

We use the behavioural predicates of Section 2.1, together with the new predicate in Definition 4, to formulate the characterising properties of the set of usable clients $\mathcal{U}_{\mathsf{must}}$ (Definition 5). We use these predicates to construct a set $\mathcal{U}_{\mathsf{bhv}}$ that coincides with $\mathcal{U}_{\mathsf{must}}$ (Theorem 1); this gives us an inductive proof method for determining usability.

Definition 4. We write $r \Downarrow_{\checkmark}$ whenever for every infinite sequence of internal moves $r \xrightarrow{\tau} r_1 \xrightarrow{\tau} r_2 \xrightarrow{\tau} \ldots$, there exists a state r_i such that $r_i \xrightarrow{\checkmark}$. ■

Recalling Eq. (5), let $\mathsf{Acc}_{\not\checkmark}(r) = \mathsf{Acc}_{\not\checkmark}(r, \varepsilon)$.

Definition 5. Let $\mathcal{F} : \mathcal{P}(\mathsf{Proc}) \longrightarrow \mathcal{P}(\mathsf{Proc})$ be defined by letting $r \in \mathcal{F}(S)$ whenever

1. $r \Downarrow_{\checkmark}$, and
2. for every $A \in \mathsf{Acc}_{\not\checkmark}(r)$ there exists an action $a \in A$ such that

$$r \overset{a}{\Longrightarrow}_{\not\checkmark} \text{ implies } \bigoplus(r \; \mathsf{after}_{\not\checkmark} \; a) \in S.$$

We let $\mathcal{U}_{\mathsf{bhv}} = \mu x.\mathcal{F}(x)$, the *least* fix-point of $\mathcal{F}$. ■

The function $\mathcal{F}$ is continuous over the CPO $\langle \mathcal{P}(\mathsf{Proc}), \subseteq \rangle$, thus Kleene fixed point theorem [35, Theorem 5.11] ensures that $\mu x.\mathcal{F}(x)$ (the least fix-point of $\mathcal{F}$) exists and in particular

$$\mu x.\mathcal{F}(x) = \bigcup_{n=0}^{\infty} \mathcal{F}^n(\emptyset) \tag{6}$$

where $\mathcal{F}^0(S) = S$ and $\mathcal{F}^{n+1}(S) = \mathcal{F}(\mathcal{F}^n(S))$. We state now the main results of the section, but defer its proof to the end of this section.

Theorem 1 *(Full-abstraction usability). In every finite-branching LTS the sets* $\mathcal{U}_{\mathsf{must}}$ *and* $\mathcal{U}_{\mathsf{bhv}}$ *coincide.*

In view of Definition 5 the theorem above guarantees that a client r is usable if and only if, for every potentially deadlocked state r' reached via silent moves by r, there exists an action a that leads r' out of the potential deadlock, *i.e.*, into some other state r'' where r'' is certainly usable.

Example 4. Theorem 1 and Definition 5 let us discuss the (non) usability of clients from Example 1, from which we recall

$$r_3 = \tau.(1 + \tau.0) \qquad r_3' = r_3 + \tau.0 \qquad r_3'' = r_3 + \mu x.x$$

Since we have $r_3 \Downarrow_{\checkmark}$ and $\mathsf{Acc}_{\not\checkmark}(r_3) = \emptyset$, r_3 satisfies both condition of Definition 5, with the second one being trivially true. As a consequence r_3 is usable, and indeed $0 \; \mathsf{must} \; r_3$. On the contrary, we have $\mathsf{Acc}_{\not\checkmark}(r_3') = \{\emptyset\}$, which means that r_3' violates Definition 5(2) (since we cannot pick any unblocking action from the set $\emptyset$) and thus r_3' is unusable. Client r_3'' is unusable as well, but violates Definition 5(1) instead (due to the unsuccessful infinite trace $r_3'' \xrightarrow{\tau} \mu x.x \xrightarrow{\tau} \mu x.x \xrightarrow{\tau} \ldots$). Conversely, client $r_3''' = r_3 + \tau.(1 + \mu x.x)$ satisfies both conditions of Definition 5 and it is usable (note that the infinite computation $r_3''' \xrightarrow{\tau} (1 + \mu x.x) \xrightarrow{\tau} \mu x.x \xrightarrow{\tau} \ldots$ is successful because $1 + \mu x.x \xrightarrow{\checkmark}$). In fact, $0 \text{ must } r_3'''$.

A more involved client is $r_1 + r_2$ from Example 2. There we proved that $r_1 + r_2 \notin \mathcal{U}_{\text{must}}$, and indeed $r_1 + r_2$ does not satisfy Definition 5(2). This is true because $\text{Acc}_{\not\checkmark}(r_1 + r_2) = \{\{c\}\}$, and $r' \notin \mathcal{U}_{\text{must}}$, where

$$r' = \bigoplus\big((r_1 + r_2) \text{ after}_{\not\checkmark}\ c\big) = \tau.(a.1 + b.0) + \tau.(a.0 + b.1).$$

In turn, the reason why r' is not usable is that $\text{Acc}_{\not\checkmark}(r') = \{\{a, b\}\}$, and Definition 5(2) requires us to consider every set in $\{\{a, b\}\}$ – we have only $\{a, b\}$ to consider – and show that for *some* action $a' \in \{a, b\}$, we have $\bigoplus(r' \text{ after}_{\not\checkmark}\ a') \in \mathcal{U}_{\text{must}}$. It turns out that neither action in $\{a, b\}$ satisfies this condition. For instance, in the case of action b, we have $\bigoplus(r' \text{ after}_{\not\checkmark}\ b) = \tau.1 + \tau.0$ and $\text{Acc}_{\not\checkmark}(\tau.1 + \tau.0) = \{\emptyset\}$, so $\bigoplus(r' \text{ after}_{\not\checkmark}\ b)$ violates Definition 5(2) and as a result $\bigoplus(r' \text{ after}_{\not\checkmark}\ b) \notin \mathcal{U}_{\text{must}}$. The reasoning why action a is *not* a good candidate either is analogous. ■

As argued in the Introduction, usability of clients is not compositional, for instance $r_1, r_2 \in \mathcal{U}$ does not imply $r_1 + r_2 \in \mathcal{U}$. This is why Definition 5(2) requires the non-deterministic combination of all the elements of $(r \text{ after}_{\not\checkmark}\ a)$ to be in the set S.

Counterexample 1. Define the function $\mathcal{F}_{\text{bad}}$ as $\mathcal{F}$ of Definition 5 but replacing point (2) with the following condition,

$$\begin{aligned}&\text{for every } A \in \text{Acc}_{\not\checkmark}(r) \text{ there exists } a \in A \text{ such that}\\ &r \overset{a}{\Longrightarrow}_{\not\checkmark} \text{ implies that for very } r' \in (r \text{ after}_{\not\checkmark}\ a).\, r' \in S,\end{aligned} \tag{2bad}$$

and let $\mathcal{U}_{\text{bad}} = \mu x.\mathcal{F}_{\text{bad}}(x)$. We have that $\mathcal{U}_{\text{bad}} \not\subseteq \mathcal{U}_{\text{must}}$, that is $\mathcal{U}_{\text{bad}}$ is not sound wrt the set $\mathcal{U}_{\text{must}}$. To see why this is the case, consider the clients in Eq. (3), namely $r_1 = c.(a.1 + b.0)$ and $r_2 = c.(a.0 + b.1)$. We have that $1 \in \mathcal{F}^1_{\text{bad}}(\emptyset)$, from which we obtain that $a.1 + b.0 \in \mathcal{F}^2_{\text{bad}}(\emptyset)$ and $a.0 + b.1 \in \mathcal{F}^2_{\text{bad}}(\emptyset)$. These two facts and Eq. (2bad) above let us prove that $r_1 + r_2 \in \mathcal{F}^3_{\text{bad}}(\emptyset)$, and so $r_1 + r_2 \in \mathcal{U}_{\text{bad}}$. This is because (2bad) requires us to consider the two clients r_1 and r_2 in isolation. However we know that $r_1 + r_2 \notin \mathcal{U}_{\text{must}}$ because there is no server that passes $r_1 + r_2$. This is correctly captured by $\mathcal{F}$ of Definition 5 because $r_1 + r_2 \notin \mathcal{U}_{\text{bhv}}$, that is $r_1 + r_1 \notin \mathcal{F}^n(\emptyset)$ for every n. To see why, observe that $\text{Acc}_{\not\checkmark}(r_1 + r_1) = \{\{c\}\}$, and $\bigoplus(r_1 + r_1 \text{ after}_{\not\checkmark}\ c) \notin \mathcal{F}^n(\emptyset)$ for every n. This is the case because

$$\bigoplus(r_1 + r_1 \text{ after}_{\not\checkmark}\ c) = \tau.(a.1 + b.0) + \tau.(a.0 + b.1)$$

and $\text{Acc}_{\not\checkmark}(\bigoplus(r_1 + r_1 \text{ after}_{\not\checkmark}\ c)) = \{\{a, b\}\}$, and

$$\begin{aligned}&\bigoplus(r_1 + r_1 \text{ after}_{\not\checkmark}\ ca) \notin \mathcal{F}^{n-1}(\emptyset),\\ &\bigoplus(r_1 + r_1 \text{ after}_{\not\checkmark}\ cb) \notin \mathcal{F}^{n-1}(\emptyset).\end{aligned}$$

Both statements above are true because of the equalities $\bigoplus(r_1 + r_1 \text{ after}_{\not\checkmark}\ ca) = \bigoplus(r_1 + r_1 \text{ after}_{\not\checkmark}\ cb) = \tau.1 + \tau.0$, and also because $\tau.1 + \tau.0 \notin \mathcal{F}^m(\emptyset)$ for every m. This is indeed the case since $\text{Acc}_{\not\checkmark}(\tau.1 + \tau.0) = \{\emptyset\}$, and therefore $\tau.1 + \tau.0$ does *not* satisfy Definition 5(2). □

The bulk of the soundness result follows from the next lemma, which also lays bare the role of non-recursive servers in proving usability of clients.

Lemma 3. *For every $n \in \mathbb{N}$ and $r \in \text{Proc}$, $r \in \mathcal{F}^n(\emptyset)$ implies that there exists a non-recursive server p such that $p \text{ must } r$.*

Proof. We reason by numerical induction on n. For the base case, $n = 0$, the lemma is trivially true since there is no r such that $r \in \mathcal{F}^0(\emptyset) = \emptyset$. For the inductive case, we have $n = m + 1$. We know, by hypothesis, that $r \in \mathcal{F}^{m+1}(\emptyset)$; we must therefore exhibit a *non-recursive* p such that $p \text{ must } r$. The proof proceeds by case analysis on $\text{Acc}_{\not\checkmark}(r)$, where we have two subcases to consider:

$\text{Acc}_{\not\checkmark}(r) = \emptyset$**:** This means that

$$r = r_0 \xrightarrow{\tau} \ldots \xrightarrow{\tau} r_n = r' \overset{\tau}{\not\longrightarrow} \text{ implies } r_i \xrightarrow{\checkmark} \text{ for some } r_i \tag{7}$$

We prove that the non-recursive server 0 is our witness, *i.e.*, $0 \text{ must } r$. Pick a maximal computation of $0 \mid\mid r \xrightarrow{\tau} 0 \mid\mid r_1 \xrightarrow{\tau} \ldots$. The computation must exclusively be due to the silent moves $r \xrightarrow{\tau} r_1 \xrightarrow{\tau} \ldots$. If the computation is finite then $r \Longrightarrow r_n \overset{\tau}{\not\longrightarrow}$ for some r_n, and (7) above implies that the computation is successful. If the computation is infinite, then $r \Downarrow_{\checkmark}$ of Definition 5(1) ensures that the computation is successful.

$\mathsf{Acc}_{\not\checkmark}(r) \neq \emptyset$**:** Since r is finite-branching, the set $\mathsf{Acc}_{\not\checkmark}(r)$ is finite, that is $|\mathsf{Acc}_{\not\checkmark}(r)| = h$ and denote the elements of this set as A_i for $i \in 1..h$.

Definition 5(2) ensures that for every A_i there exists an action a_i such that, whenever $(r \mathsf{\ after}_{\not\checkmark}\ a_i) \neq \emptyset$, then $(r \mathsf{\ after}_{\not\checkmark}\ a_i) \in \mathcal{F}^m(\emptyset)$.

By the inductive hypothesis and $(r \mathsf{\ after}\ a_i) \in \mathcal{F}^m(\emptyset)$, for all $i \in 1..h$ such that $(r \mathsf{\ after}_{\not\checkmark}\ a_i) \neq \emptyset$, we know that there exists a non-recursive server p^i that satisfies $p^i \mathsf{\ must\ } \bigoplus(r \mathsf{\ after}_{\not\checkmark}\ a_i)$. The required witness (non-recursive) server proving that r is usable is

$$\hat{p} = \Big(\sum_{\{ a_i \mid (r \mathsf{\ after}_{\not\checkmark}\ a_i) \neq \emptyset \}} \overline{a_i}.p^i \Big) + \Big(\sum_{\{ a_i \mid (r \mathsf{\ after}_{\not\checkmark}\ a_i) = \emptyset \}} \overline{a_i}.0 \Big) \tag{8}$$

By construction one can easily see that $\hat{p}$ is non-recursive, and to conclude the proof we are only left to show that $\hat{p} \mathsf{\ must\ } r$. Pick a maximal computation

$$r \mid\mid \hat{p} = r_0 \mid\mid p_0 \xrightarrow{\tau} r_1 \mid\mid p_1 \xrightarrow{\tau} \dots \tag{9}$$

To show why this computation is necessarily successful we have to consider two subcases:

- In (9) there is *no* interaction between derivatives of r and those of $\hat{p}$. Since $\hat{p}$ is stable, the computation (9) must be due to reductions of r. If (9) is infinite, it is because r diverges, and by $r \Downarrow_{\checkmark}$ of Definition 5(1) the computation must be successful. Else (9) is finite and has the form $r \mid\mid \hat{p} \Longrightarrow r' \mid\mid \hat{p} \stackrel{\tau}{\not\longrightarrow}$. It follows that $r \Longrightarrow r' \stackrel{\tau}{\not\longrightarrow}$. Since $r' \mid\mid \hat{p} \stackrel{\tau}{\not\longrightarrow}$, it also follows that there must be a successful state amongst the silent moves $r \Longrightarrow r'$. For otherwise, we would have $r \Longrightarrow_{\not\checkmark} r' \stackrel{\tau}{\not\longrightarrow}$ and by Eq. (5) and Definition 5(2), we know that $S(r') \neq \emptyset$ (the second clause in Definition 5 requires all $A \in \mathsf{Acc}_{\not\checkmark}(r)$, one of which is $S(r')$, to contain at least one $a \in A$). In such a case, Eq. (8) would then guarantee that $r' \mid\mid \hat{p} \xrightarrow{\tau}$ (by an interaction on an action in $S(r')$), contradicting $r' \mid\mid \hat{p} \stackrel{\tau}{\not\longrightarrow}$.
- In (9) there exists at least one interaction, then assume that the first one results in the reduction $r_k \mid\mid p_k \xrightarrow{\tau} r_{k+1} \mid\mid p_{k+1}$. As this is the first reduction, $p_k = \hat{p}$, and Eq. (8) ensures that for some a_i, $r_k \xrightarrow{a_i} r_{k+1}$ and $p_k \xrightarrow{\overline{a_i}} p_{k+1}$. If one of the states between r and r_{k+1} is successful then the computation is successful. So suppose instead that in the computation at hand $r \stackrel{a_i}{\Longrightarrow}_{\not\checkmark} r_{k+1}$. This means that $r_{k+1} \in (r \mathsf{\ after}_{\not\checkmark}\ a_i)$, hence $(r \mathsf{\ after}_{\not\checkmark}\ a_i) \neq \emptyset$. Now Eq. (8) and $p_k \xrightarrow{\overline{a_i}} p_{k+1}$ imply that $p_{k+1} = p^i$. We already know that $p^i \mathsf{\ must\ } \bigoplus(r \mathsf{\ after}_{\not\checkmark}\ a_i)$, and this implies $p^i \mathsf{\ must\ } r_{k+1}$. It follows that $p_{k+1} \mathsf{\ must\ } r_{k+1}$, and thus the maximal computation at hand must contain a successful state.

We have proven that an arbitrary maximal computation of $r \mid\mid \hat{p}$ is successful, and thus $\hat{p} \mathsf{\ must\ } r$, as required. □

Corollary 1. *For every $r \in \mathsf{Proc}$, $r \in \mathcal{U}_{\mathsf{bhv}}$ implies $r \in \mathcal{U}_{\mathsf{must}}$.*

An inductive argument let us prove that $\mathcal{U}_{\mathsf{bhv}}$ is complete *wrt.* $\mathcal{U}_{\mathsf{must}}$, where we define the following measure over which to perform induction. We let $MC(r, p)$ denote the set of maximal computations of a composition $r \mid\mid p$ and, for every computation $c \in MC(r, p)$, we associate the number $\#\mathsf{itr}(c)$ denoting the number of *interactions* that take place between the initial state of c, and the *first successful state* of the computation c ($\#\mathsf{itr}(c) = \infty$ whenever c is unsuccessful). Let $\mathsf{itr}(r, p) = \max\{ \#\mathsf{itr}(c) \mid c \in MC(r, p) \}$. For instance, if $r = \mu x.a.x + b.1$, we have $\mathsf{itr}(r, \overline{a}.\overline{a}.\overline{b}.0) = 3$, but $\mathsf{itr}(r, \mu x.\overline{a}.x + \overline{b}.0) = \infty$.

Definition 6 *(Tree [13, Appendix B.5]).* A (free) *tree* is a undirected graph in which any two nodes are connected by a unique simple path.

Lemma 4. *In a finite-branching LTS, $p \mathsf{\ must\ } r$ implies $\mathsf{itr}(r, p)$ is finite.*

Proof. If $p \mathsf{\ must\ } r$, *every* $c \in MC(r, p)$ reaches the first successful state after a *finite* number of reductions. This number is an upper bound on the number of interactions that lead to the first successful state in a maximal computation, and thus:

$$\text{for every } c \in MC(r, p).\ \#\mathsf{itr}(c) \in \mathbb{N} \tag{10}$$

The hypothesis that the LTS is finite-branching ensures that a set of successful computations starting from $r \mid\mid p$, *e.g.*, $MC(r, p)$, may be seen as a *computation tree*, where common prefixes reach the same node in the tree [19, Lemma 4.4.12]. In general, paths in such a tree may have infinite length. Consider the computation tree T obtained by *truncating* all the maximal computations of $r \mid\mid p$ at their *first* successful state, and let $TMC(r, p)$ be the set of all the computations obtained this way. The definition of TMC implies that

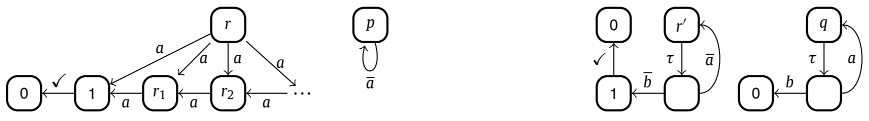

Fig. 3. Servers and clients to discuss the hypothesis in Lemma 4.

$$\{\#\mathsf{itr}(c) \mid c \in MC(r,p)\} = \{\#\mathsf{itr}(c) \mid c \in TMC(r,p)\} \tag{11}$$

From $\mathsf{itr}(r,p) = \max\{\#\mathsf{itr}(c) \mid c \in MC(r,p)\}$, (10) and (11) we know that $\mathsf{itr}(r,p)$ is finite if the set $\{c \mid c \in TMC(r,p)\}$ is finite. This will follow from Definition 6 if we prove that the tree T has a finite number of nodes. By the contrapositive of König's Lemma [21,20], since every node in the tree T above is finite-branching, and there are no paths of infinite length, then T necessarily contains a *finite* number of nodes, as required. It follows that we can put a natural number $\mathsf{itr}(r,p) \in \mathbb{N}$ as an upper bound on the number of interactions required to reach success. □

If the LTS is not *image-finite* then Lemma 4 is false. To see why, consider the infinite branching client r and the server p depicted in Fig. 3. Since r engages in *finite* sequences of a actions which are unbounded in size, and p offers any number of interactions on action $\bar{a}$, we have that p must r, but the set $MC(r,p)$ contains an infinite amount of computations, and $\mathsf{itr}(r,p)$ is not defined. Dually, even if the LTS of a composition $r \,||\, p$ is finite-branching and finite state, it is necessary that p must r for $\mathsf{itr}(r,p)$ to be finite. Lemma 4 lets us associate a rank to every usable client r, defined as $rank(r) = \min\{\mathsf{itr}(r,p) \mid p \text{ must } r\}$. The well-ordering of $\mathbb{N}$ ensures that $rank(r)$ is defined for every usable r. When defined, the rank of a client r gives us information about its usability,[2] where we stratify $\mathcal{U}_{\mathsf{must}}$ as follows:

$$\mathcal{U}_{\mathsf{must}} = \bigcup_{i \in \mathbb{N}} \mathcal{U}^i_{\mathsf{must}}\,, \text{ where } \mathcal{U}^i_{\mathsf{must}} = \{r \in \mathsf{Proc} \mid rank(r) = i\} \tag{12}$$

In what follows, we always assume that the LTS is finite-branching.

Lemma 5. *For every $n \in \mathbb{N}$ and $r \in \mathcal{U}^n_{\mathsf{must}}$, and for every $A \in \mathsf{Acc}_{\not\checkmark}(r)$, there exists an $a \in A$ such that, whenever $r \overset{a}{\Longrightarrow}_{\not\checkmark}$, then $\bigoplus(r \text{ after}_{\not\checkmark} a) \in \mathcal{U}^m_{\mathsf{must}}$ for some $m < n$.*

Proof. Fix an $r \in \mathcal{U}^n_{\mathsf{must}}$. We know by definition that $rank(r) = n$, and so there exists a server p such that $\mathsf{itr}(r,p) = rank(r)$. This implies that p must r. Now pick a set $A \in \mathsf{Acc}_{\not\checkmark}(r)$. This ensures that if $\mathsf{Acc}_{\not\checkmark}(r) \neq \emptyset$ then there exists a r' such that $r \Longrightarrow_{\not\checkmark} r' \overset{\tau}{\not\longrightarrow}$ (which, in turn, implies that $r \overset{\checkmark}{\not\longrightarrow}$). Since p must r and $r \overset{\checkmark}{\not\longrightarrow}$, the process p cannot diverge, meaning that there exists a p' such that $p \Longrightarrow p' \overset{\tau}{\not\longrightarrow}$. Lemma 2(2i) now implies that p' must r'. Since $r' \overset{\checkmark}{\not\longrightarrow}$ and both p' and r' are stable, they must interact (for otherwise $p' \neg\mathsf{must}\ r'$). This means that $r' \overset{a}{\longrightarrow}$ for some $a \in \mathsf{Act}$, and that $p' \overset{\bar{a}}{\longrightarrow} p''$ for some p''.

Assume now that $r \overset{a}{\Longrightarrow}_{\not\checkmark}$, *i.e.*, it may not succeed after weakly performing action a. Since the LTS is finite, the set $(r \text{ after}_{\not\checkmark} a)$ is also finite: we let $r_a = \bigoplus(r \text{ after}_{\not\checkmark} a)$ and proceed to show that $r_a \in \mathcal{U}^m_{\mathsf{must}}$ for some $m < n$.

First observe that p'' must r_a because p'' must r'' for every $r'' \in (r \text{ after}_{\not\checkmark} a)$: the latter is a consequence of Lemma 2(2ii), our assumption p must r, $p \overset{\bar{a}}{\Longrightarrow} p''$ and the fact that $r'' \in (r \text{ after}_{\not\checkmark} a)$ implies $r \overset{a}{\Longrightarrow}_{\not\checkmark} r''$.

Let $rank(r_a) = m$. Since p'' must r_a, the definition of rank ensures that $m \leq \mathsf{itr}(r_a, p'')$, and thus all we have to do now is to show that $\mathsf{itr}(r_a, p'') < n$, from which $m < n = rank(r)$ follows. To prove this fact, observe that every maximal computation of $r_a \,||\, p''$ can be split into an initial part of internal moves, and a suffix c that contains the interactions between (the reducts of) r_a and (the reducts of) p''. The suffix c must be a suffix of a maximal computation of $r \,||\, p \Longrightarrow_{\not\checkmark} r'' \,||\, p''$ where $r'' \in (r \text{ after}_{\not\checkmark} a)$ and r and p interact on a in $r \,||\, p \Longrightarrow_{\not\checkmark} r'' \,||\, p''$. It follows that c must contain at least one less interaction that the computation starting with $r \,||\, p \Longrightarrow_{\not\checkmark} r'' \,||\, p''$, namely the interaction on a. This implies that $\mathsf{itr}(r_a, p'') < \mathsf{itr}(r,p)$. But we have by assumption that $\mathsf{itr}(r,p) = rank(r) = n$, thus $\mathsf{itr}(r_a, p'') < n$. □

Lemma 6. *For every $i \in \mathbb{N}$, $r \in \mathcal{U}^i_{\mathsf{must}}$ implies $r \in \mathcal{F}(\mathcal{F}^j(\emptyset))$ for some $j \leq i$.*

Proof. We proceed by strong (complete) induction on i where $r \in \mathcal{U}^i_{\mathsf{must}}$. For the base case, $i = 0$, we need to show that $r \in \mathcal{F}(\emptyset)$. The first property, $r \Downarrow_{\checkmark}$, follows from the hypothesis that $r \in \mathcal{U}^0_{\mathsf{must}} \subseteq \mathcal{U}_{\mathsf{must}}$ of (12), and Lemma 2(1). For the second property,

$$\forall A \in \mathsf{Acc}_{\not\checkmark}(r).\, \exists a \in A.\, r \overset{a}{\Longrightarrow}_{\not\checkmark} \text{ implies } \bigoplus(r \text{ after}_{\not\checkmark} a) \in \emptyset \tag{13}$$

2 Function min is *not* defined for empty sets, thus $rank(r)$ is undefined whenever r is unusable.

we show that $\mathsf{Acc}_{\not\checkmark}(r) = \emptyset$, in which case (13) holds trivially. From $r \in \mathcal{U}^0_{\mathsf{must}}$ we know that there exists a p such that $p \;\mathsf{must}\; r$ and $\mathsf{itr}(r, p) = 0$, which means that p and r do not need to interact for r to reach a successful state. We have two subcases to consider:

- If p diverges then, by $p \;\mathsf{must}\; r$, it must be the case that $r \xrightarrow{\checkmark}$. But then r does not perform any unsuccessful trace, so we immediately have $\mathsf{Acc}_{\not\checkmark}(r) = \emptyset$.
- If p converges then fix a p' such that $p \Longrightarrow p' \overset{\tau}{\not\longrightarrow}$. Independently, pick an arbitrary r' such that $r \Longrightarrow r' \overset{\tau}{\not\longrightarrow}$. We need to show that for *any* such r', a success is reached along $r \Longrightarrow r' \overset{\tau}{\not\longrightarrow}$, which would mean that $r \Longrightarrow_{\not\checkmark} r'$ never holds and, as a result, $\mathsf{Acc}_{\not\checkmark}(r) = \emptyset$.
  The assumption $\mathsf{itr}(r, p) = 0$ means that *all* maximal computations are successful and, moreover, that success is *always* reached *before* the first interaction. Thus, zipping $p \Longrightarrow p' \overset{\tau}{\not\longrightarrow}$ with $r \Longrightarrow r' \overset{\tau}{\not\longrightarrow}$ as $p \,||\, r \Longrightarrow p' \,||\, r'$ necessarily forms a prefix of one of these maximal computations right up to the point of the first interaction of the respective maximal computation (in case there is no interaction in the respective maximal computation, we then have $p' \,||\, r' \overset{\tau}{\not\longrightarrow}$). This means that a success must have been reached during $p \,||\, r \Longrightarrow p' \,||\, r'$, which also means that a success is reached along $r \Longrightarrow r' \overset{\tau}{\not\longrightarrow}$ and hence $r \Longrightarrow_{\not\checkmark} r'$ is false.

For the inductive case, we have $r \in \mathcal{U}^{m+1}_{\mathsf{must}}$ and need to show that $r \in \mathcal{F}(\mathcal{F}^j(\emptyset))$ for some $j \leq m + 1$. By Definition 5, this means that we need to show that $r \Downarrow_{\checkmark}$ and that

$$\forall A \in \mathsf{Acc}_{\not\checkmark}(r).\, \exists a \in A.\, r \overset{a}{\Longrightarrow}_{\not\checkmark} \text{ implies } \bigoplus(r \;\mathsf{after}_{\not\checkmark}\; a) \in \mathcal{F}^j(\emptyset) \tag{14}$$

The proof for $r \Downarrow_{\checkmark}$ is analogous to that used for the base case. For the proof of (14), we fix an $A \in \mathsf{Acc}_{\not\checkmark}(r)$ and then exhibit an $a \in A$ such that if $r \overset{a}{\Longrightarrow}_{\not\checkmark}$ then $\bigoplus(r \;\mathsf{after}_{\not\checkmark}\; a) \in \mathcal{F}^j(\emptyset)$. By applying Lemma 5, we know that there exists an action $a \in A$, such that

$$r \overset{a}{\Longrightarrow}_{\not\checkmark} \text{ implies } \bigoplus(r \;\mathsf{after}_{\not\checkmark}\; a) \in \mathcal{U}^k_{\mathsf{must}} \text{ for some } k < m + 1 \tag{15}$$

By the above and the inductive hypothesis, we know that whenever $r \overset{a}{\Longrightarrow}_{\not\checkmark}$ then we also have that $\bigoplus(r \;\mathsf{after}_{\not\checkmark}\; a) \in \mathcal{F}(\mathcal{F}^l(\emptyset))$ for some $l \leq k$, from which (14) follows. □

We are now ready to prove the main result of this section.

Proof. [Theorem 1] We have to show the set inclusions $\mathcal{U}_{\mathsf{must}} \subseteq \mathcal{U}_{\mathsf{bhv}}$ and $\mathcal{U}_{\mathsf{bhv}} \subseteq \mathcal{U}_{\mathsf{must}}$, within a finite-branching LTS. To show $\mathcal{U}_{\mathsf{must}} \subseteq \mathcal{U}_{\mathsf{bhv}}$, pick an $r \in \mathcal{U}_{\mathsf{must}}$. By (12), $r \in \mathcal{U}^i_{\mathsf{must}}$ for some $i \in \mathbb{N}$, and by Lemma 6 we obtain $r \in \mathcal{F}^j(\emptyset)$ for some $j \in \mathbb{N}^+$. Eq. (6) implies that $r \in \mathcal{U}_{\mathsf{bhv}}$. To show $\mathcal{U}_{\mathsf{bhv}} \subseteq \mathcal{U}_{\mathsf{must}}$, pick an $r \in \mathcal{U}_{\mathsf{bhv}}$. Eq. (6) ensures that $r \in \bigcup_{n=0}^{\infty} \mathcal{F}^n(\emptyset)$, thus $r \in \mathcal{F}^n(\emptyset)$ for some $n \in \mathbb{N}$. Lemma 3 implies that $r \in \mathcal{U}_{\mathsf{must}}$. The reasoning applies to any $r \in \mathcal{U}_{\mathsf{bhv}}$, and so $\mathcal{U}_{\mathsf{bhv}} \subseteq \mathcal{U}_{\mathsf{must}}$. □

Theorem 1 is a full-abstraction result in the following sense. Clients have a natural denotational semantics defined by letting $[\![r]\!] = \{ p \in \mathsf{LTS} \mid p \;\mathsf{must}\; r \}$, that is the denotation of a client r is the set of servers that pass r. According to this definition (a) usable clients are the only ones whose denotation is not empty, and (b) $r_1 \sqsubseteq_{\mathsf{must}} r_2$ means that the denotation of r_1 is smaller than the denotation of r_2. Theorem 1 implies that the operational characterisation of usability for a client r let us check the non-emptiness of its denotation $[\![r]\!]$, and thanks to Theorem 1, also Proposition 1 becomes a full-abstraction result, as it characterises the set inclusion $[\![r_1]\!] \subseteq [\![r_2]\!]$ entirely via the operational semantics of r_1 and r_2.

4. The client preorder revisited

In this section we put forth a novel coinductive definition for the must client preorder and exploit the finite-branching property of the LTS to show that the new coinductive preorder characterises the contextual preorder $\sqsubseteq_{\mathsf{must}}$ (Theorem 2). We also argue that the coinductive preorder is easier to use in practice than Definition 3. We do so by using it in the proof of the second result in this section, namely that servers offering a *finite* amount of interactions are sufficient and necessary to distinguish clients (Theorem 3). Subsequently, we also show that the coinductive preorder is decidable for our client language (Theorem 4).

We start by identifying three characteristic properties of the preorder $\sqsubseteq_{\mathsf{must}}$, stated in Lemma 7. At an intuitive level, $r_1 \sqsubseteq_{\mathsf{must}} r_2$ means that r_2 is better than r_1. Thus, the first condition in Lemma 7 states that the preorder is preserved by any unsuccessful reductions performed by the better client. The second condition states that if the better client is unsuccessful the worse one must be unsuccessful too (this is due to divergent servers). The third condition states that if the better client may interact unsuccessfully with the environment (via a visible action) then the worse server must engage unsuccessfully on the same action as well (up-to internal actions) and the non-deterministic continuations of the two servers must still be in the preorder.

Lemma 7. *For every* $r_1, r_2 \in \mathsf{Proc}$. $r_1 \sqsubseteq_{\mathsf{must}} r_2$ *implies*

1. *if* $r_2 \xrightarrow{\tau}_{\not\checkmark} r_2'$ *then* $r_1 \sqsubseteq_{\mathsf{must}} r_2'$*;*
2. *if* $r_2 \overset{\checkmark}{\not\longrightarrow}$ *then* $r_1 \overset{\checkmark}{\not\longrightarrow}$*;*
3. *if* $r_2 \xrightarrow{a}_{\not\checkmark}$ *then* $(r_1 \overset{a}{\Longrightarrow}_{\not\checkmark}$ *and* $\bigoplus(r_1 \mathsf{\ after}_{\not\checkmark}\ a) \sqsubseteq_{\mathsf{must}} \bigoplus(r_2 \mathsf{\ after}_{\not\checkmark}\ a))$*.* □

Lemma 7 partly motivates the conditions defining the transfer function $\mathcal{G}$ in Definition 7. Conditions (2) and (3) are explained in greater detail as discussions to points (2) and (3c) of Definition 7.

Definition 7. Let $\mathcal{G} : \mathcal{P}(\mathsf{Proc} \times \mathsf{Proc}) \longrightarrow \mathcal{P}(\mathsf{Proc} \times \mathsf{Proc})$ be the function such that $(r_1, r_2) \in \mathcal{G}(R)$ whenever all the following conditions hold:

1. if $r_2 \xrightarrow{\tau}_{\not\checkmark} r_2'$ then $r_1\ R\ r_2'$
2. if $r_2 \overset{\checkmark}{\not\longrightarrow}$ then $r_1 \overset{\checkmark}{\not\longrightarrow}$
3. if $r_1 \in \mathcal{U}_{\mathsf{bhv}}$ then
   (a) $r_2 \in \mathcal{U}_{\mathsf{bhv}}$
   (b) if $B \in \mathsf{Acc}_{\not\checkmark}(r_2)$ then there exists an $A \in \mathsf{Acc}_{\not\checkmark}(r_1)$ such that $A \cap ua_{\mathsf{bhv}}(r_1) \subseteq B$
   (c) if $r_2 \xrightarrow{a}_{\not\checkmark}$ then $(r_1 \overset{a}{\Longrightarrow}_{\not\checkmark}$ and $\bigoplus(r_1 \mathsf{\ after}_{\not\checkmark}\ a)\ R\ \bigoplus(r_2 \mathsf{\ after}_{\not\checkmark}\ a))$

where $ua_{\mathsf{bhv}}(r) = \{\, a \mid r \overset{a}{\Longrightarrow}_{\not\checkmark}$ implies $\bigoplus(r \mathsf{\ after}_{\not\checkmark}\ a) \in \mathcal{U}_{\mathsf{bhv}}\,\}$. Let $\precsim_{\mathsf{must}} = \nu x.\mathcal{G}(x)$ where $\nu x.\mathcal{G}(x)$ denotes the greatest fix-point of $\mathcal{G}$. The function $\mathcal{G}$ is monotone over the complete lattice $\langle \mathcal{P}(\mathsf{Proc} \times \mathsf{Proc}), \subseteq \rangle$ and thus $\nu x.\mathcal{G}(x)$ exists. □

The definition of $\mathcal{G}$ follows a similar structure to that of the respective definitions that coinductively characterise the must preorder for servers [10, Definition 3.1], [6, Definition 3.36]. Definition 7, however, uses predicates for clients, *i.e.*, unsuccessful traces and usability, in place of the predicates for servers, *i.e.*, traces and convergence. Note, in particular, that we use the *fully-abstract* version of usability, $\mathcal{U}_{\mathsf{bhv}}$, from Definition 5 and adapt the definition of usable actions accordingly, $ua_{\mathsf{bhv}}(r)$. Another subtle but crucial difference in Definition 7 is how unsuccessful states are related by condition (2). The next example elucidates why such a condition is necessary for $\precsim_{\mathsf{must}}$ to be sound.

Counterexample 2. Let $\mathcal{G}_{\mathsf{bad}}$ be defined as $\mathcal{G}$ in Definition 7, but without part (2). We prove that the pair of clients $(1, \tau.1)$ is contained in the greatest fixed point of $\mathcal{G}_{\mathsf{bad}}$, and *not* in $\sqsubseteq_{\mathsf{must}}$. Let $R = \{(1, \tau.1)\}$. We have that $R \subseteq \mathcal{G}_{\mathsf{bad}}(R)$ if the pair $(1, \tau.1)$ satisfies all the conditions for $\mathcal{G}_{\mathsf{bad}}$: condition (1) is trivially true, condition (3a) is true because $0 \mathsf{\ must\ } 1$ and $0 \mathsf{\ must\ } \tau.1$, condition (3b) holds trivially because $\mathsf{Acc}_{\not\checkmark}(\tau.1) = \emptyset$, whereas condition (3c) is satisfied because $\tau.1$ does not perform any strong actions. It therefore follows that $(1, \tau.1) \in \mu x.\mathcal{G}_{\mathsf{bad}}(x)$. Contrarily, $1 \not\sqsubseteq_{\mathsf{must}} \tau.1$ because the divergent server τ^∞ distinguishes between the two clients: whereas $\tau^\infty \mathsf{\ must\ } 1$ since the client succeeds immediately, we have $\tau^\infty \ \neg\mathsf{must}\ \ \tau.1$ because the composition $\tau.1 \,||\, \tau^\infty$ has an infinite unsuccessful computation due to the divergence of τ^∞. ■

A more fundamental difference between Definition 7 and the coinductive server preorders in [22,29] is that, in Definition 7(3c), the relation R has to relate internal sums of derivative clients on *both* sides. Although non-standard, this condition is sufficient to compensate for the lack of compositionality of usable clients (see clients r_1 and r_2 from (3) in Section 1, and Counterexample 1). Using the standard weaker condition makes the preorder $\precsim_{\mathsf{must}}$ unsound *wrt.* $\sqsubseteq_{\mathsf{must}}$, as we proceed to show in the next example.

Counterexample 3. Let $\mathcal{G}_{\mathsf{bad}}$ be defined as $\mathcal{G}$ in Definition 7, but replacing the condition (3c) with the relaxed condition in (3bad) below, which requires each derivative r_2' to be analysed in isolation. We show that the greatest fixpoint of $\mathcal{G}_{\mathsf{bad}}$, namely $\preccurlyeq^{\mathsf{bad}}_{\mathsf{must}}$, contains client pairs that are not in $\sqsubseteq_{\mathsf{must}}$.

$$\text{if } r_2 \xrightarrow{a}_{\not\checkmark} r_2' \text{ then } \Big(r_1 \overset{a}{\Longrightarrow}_{\not\checkmark} \text{ and } \bigoplus(r_1 \mathsf{\ after}_{\not\checkmark}\ a)\ R\ r_2'\Big) \tag{3bad}$$

Consider the clients $r_6 = c.r_6'$ and $r_7 = (r_1 + r_2) + \tau.1$ where

$$r_6' = \tau.r_6^a + \tau.r_6^b \qquad r_6^a = a.0 + \tau.1 \qquad r_6^b = b.0 + \tau.1$$

and r_1 and r_2 are the clients defined in (3) above. On the one hand, we have that $r_6 \not\sqsubseteq_{\mathsf{must}} r_7$, because $\overline{c}.0 \mathsf{\ must\ } r_6$ whereas $\overline{c}.0\ \neg\mathsf{must}\ r_7$. On the other hand, we now show that $r_6 \preccurlyeq^{\mathsf{bad}}_{\mathsf{must}} r_7$. Focusing on condition Definition 7(3), we start by deducing that $r_6 \in \mathcal{U}_{\mathsf{bhv}}$ (either directly using Definition 5 or indirectly through $\overline{c}.0 \mathsf{\ must\ } r_6$, recalling Theorem 1). Now, Definition 7(3a) is true because $0 \text{ must } r_7$, thus r_7 is usable, and thanks to Theorem 1 we have $r_7 \in \mathcal{U}_{\mathsf{bhv}}$. Also point (3b) is satisfied, because $\mathsf{Acc}_{\not\checkmark}(r_7) = \mathsf{Acc}_{\not\checkmark}(r_6) = \{\{a\}\}$.[3] To prove that the (relaxed) condition (3bad) holds, we have to show that

$$r_6^c \preccurlyeq^{\mathsf{bad}}_{\mathsf{must}} a.1 + b.0 \quad \text{and} \quad r_6^c \preccurlyeq^{\mathsf{bad}}_{\mathsf{must}} a.0 + b.1, \qquad \text{with } r_6^c = \tau.r_6' + \tau.r_6^a + \tau.r_6^b \tag{16}$$

Let $r_7' = a.1 + b.0$. We only show the proof for the inequality $r_6^c \preccurlyeq^{\mathsf{bad}}_{\mathsf{must}} r_7'$, since the proof for the other inequality is analogous. We focus again on conditions (3a), (3b), and (3bad). Condition (3a) is true because $0 \text{ must } r_6^c$, and thus $r_6^c \in \mathcal{U} = \mathcal{U}_{\mathsf{bhv}}$, and because $r_7' \in \mathcal{U} = \mathcal{U}_{\mathsf{bhv}}$ as well (*e.g.*, $\overline{a}.0 \text{ must } r_7'$). Condition (3b) holds because $\mathsf{Acc}_{\not\checkmark}(r_7') = \{\{c\}\}$ and $\mathsf{Acc}_{\not\checkmark}(r_6^c) = \{\{b\}, \{c\}\}$. Finally for (3bad) we only have to check the case for $r_7' \xrightarrow{b}_{\not\checkmark} 0$, which requires us to show that $\tau.0 \preccurlyeq^{\mathsf{bad}}_{\mathsf{must}} 0$; this latter check is routine. As a result, we have $r_6^c \preccurlyeq^{\mathsf{bad}}_{\mathsf{must}} r_7'$. Since we can also show that $r_6^c \preccurlyeq^{\mathsf{bad}}_{\mathsf{must}} a.0 + b.1$ holds, we obtain (16), and consequently $r_6 \preccurlyeq^{\mathsf{bad}}_{\mathsf{must}} r_7$. ∎

After our digression on Definition 7, we outline why $\precsim_{\mathsf{must}}$ coincides with $\sqsubseteq_{\mathsf{must}}$. A detailed proof can be found in Appendix A.

Lemma 8. *Whenever $r_1 \precsim_{\mathsf{must}} r_2$, for every $s \in \mathsf{Act}^\star$, r_1 usbl$_{\not\checkmark}$ s implies r_2 usbl$_{\not\checkmark}$ s and also that for every $B \in \mathsf{Acc}_{\not\checkmark}(r_2, s)$, there exists a set $A \in \mathsf{Acc}_{\not\checkmark}(r_1, s)$ such that $A \cap ua_{\mathsf{clt}}(r_2, s) \subseteq B$; and that if $r_2 \overset{s}{\Longrightarrow}_{\not\checkmark}$ then $r_1 \overset{s}{\Longrightarrow}_{\not\checkmark}$.* □

Theorem 2. *In any finite-branching LTS $r_1 \sqsubseteq_{\mathsf{must}} r_2$ if and only if $r_1 \precsim_{\mathsf{must}} r_2$.*

Proof. We have to show the set inclusions, $\sqsubseteq_{\mathsf{must}} \subseteq \precsim_{\mathsf{must}}$ and $\precsim_{\mathsf{must}} \subseteq \sqsubseteq_{\mathsf{must}}$. The hypothesis that the LTS is finite-branching let us apply Proposition 1, which together with Lemma 7 implies that $\sqsubseteq_{\mathsf{must}} \subseteq \mathcal{G}(\sqsubseteq_{\mathsf{must}})$, and thus, by the Knaster–Tarski theorem, we obtain the first inclusion. The second set inclusion follows from Proposition 1 and Lemma 8. □

Example 5. Recall from Example 3 clients $r_4 = a.1 + \mu y.(a.r_3'' + b.y + c.1)$ and $r_5 = (\mu z.(b.z + c.1)) + d.1$. In that example we argued that the alternative relation $\lesssim_{\mathsf{clt}}$ is still a burdensome method for reasoning on $\sqsubseteq_{\mathsf{must}}$. We now contend that it is simpler to show $r_4 \sqsubseteq_{\mathsf{must}} r_5$ by proving $r_4 \precsim_{\mathsf{must}} r_5$, thanks to Theorem 2 and the Knaster–Tarski theorem. By Definition 7, it suffices to provide a witness relation R such that $(r_4, r_5) \in R$ and $R \subseteq \mathcal{G}(R)$. Let $R = \{(r_4, r_5), (r_4', r_5')\}$ where $r_3'' = (\tau.(1 + \tau.0)) + \mu x.x$ from Example 1, $r_4' = \mu y.(a.r_3'' + b.y + c.1)$, and $r_5' = \mu z.(b.z + c.1)$. Checking that R satisfies the conditions in Definition 7 is routine work. To prove condition (3b), though, note that $\mathsf{Acc}_{\not\checkmark}(r_5) = \mathsf{Acc}_{\not\checkmark}(r_5') = \{\{b, c\}\}$ and that $\mathsf{Acc}_{\not\checkmark}(r_4) = \{\{a, b, c\}\}$. However $ua_{\mathsf{bhv}}(r_4) = \{b, c\}$ and thus the required set inclusion $(\{a, b, c\} \cap \{b, c\}) \subseteq \{b, c\}$ holds. ∎

The coinductive preorder of $\precsim_{\mathsf{must}}$ may also be used to prove that two clients are *not* in the contextual preorder $\sqsubseteq_{\mathsf{must}}$: by iteratively following the conditions of Definition 7 one can determine whether a relation including the pair of clients exists. This approach is useful when guessing a discriminating server is not straightforward; in failing to define such a relation R one obtains information on how to construct the discriminating server.

Example 6. Recall the clients r_6 and r_7 considered in Counterexample 3. By virtue of the full-abstraction result, we can show directly that $r_6 \not\sqsubseteq_{\mathsf{must}} r_7$ by following the requirements of Definition 7 and arguing that no relation exists that contains the pair (r_6, r_7) while satisfying the conditions of the coinductive preorder. Without loss of generality, pick a relation R such that $r_6 \, R \, r_7$: we have to show that $R \subseteq \mathcal{G}(R)$. Since $r_6 \in \mathcal{U}_{\mathsf{bhv}}$, $r_7 \xrightarrow{c}_{\not\checkmark}$ and $r_6 \overset{c}{\Longrightarrow}_{\not\checkmark}$, Definition 7(3c) requires that we show that

$$r_6^c \; R \; \tau.r_7' + \tau.r_7'' \text{ where } r_6^c = \bigoplus(r_6 \text{ after}_{\not\checkmark} c) \text{ and } (\tau.r_7' + \tau.r_7'') = \bigoplus(r_7 \text{ after}_{\not\checkmark} c) \tag{17}$$

and r_6^c, r_7' and r_7'' are the clients defined earlier in Counterexample 3. Since we want to show that $R \not\subseteq \mathcal{G}(R)$, the condition Definition 7(3a) requires that, if $r_6^c \in \mathcal{U}_{\mathsf{bhv}}$, then $(\tau.r_7' + \tau.r_7'') \in \mathcal{U}_{\mathsf{bhv}}$. However, even though $r_6^c \in \mathcal{U}_{\mathsf{bhv}}$, we have $(\tau.r_7' + \tau.r_7'') \notin \mathcal{U}_{\mathsf{bhv}}$, violating Definition 7(3a) and thus showing that no such R satisfying both $(r_6, r_7) \in R$ and $R \subseteq \mathcal{G}(R)$ can exist. We highlight the fact that whereas (16) of Counterexample 3 resulted in $r_6 \preccurlyeq^{\mathsf{bad}}_{\mathsf{must}} r_7$, (17) is instrumental to conclude that $r_6 \not\precsim_{\mathsf{must}} r_7$. Note also that the path along c leading to a violation of the requirements of Definition 7 is related to the discriminating server $\overline{c}.0$ used in Counterexample 3 to justify $r_6 \not\sqsubseteq_{\mathsf{must}} r_7$. ∎

[3] The restriction of the left hand side of the inclusion of Definition 7(3b) by $ua_{\mathsf{bhv}}(r_6)$ is superfluous.

5. Expressiveness and decidability

We show that servers with finite interactions suffice to preserve the discriminating power of the contextual preorder $\sqsubseteq_{\mathsf{must}}$ in Definition 1, which has ramifications on standard verification techniques for the preorder, such as counter-example generation [12]. We also show that for any finite-state LTS, the set of usable clients is decidable. Using standard techniques [31] we then argue that, in such cases, there exists a procedure to decide whether two finite-state clients are related by $\sqsubseteq_{\mathsf{must}}$.

5.1. On the power of finite interactions

We employ the coinductive characterisation of the client preorder, Theorem 2, to prove an important property of the client preorder of Definition 1, namely that servers that only offer a *finite amount of interactions* to clients are necessary and sufficient to distinguish all the clients according to our touchstone preorder $\sqsubseteq_{\mathsf{must}}$ of Definition 1. Let $\mathsf{CCS}^f ::= 0 \mid 1 \mid \alpha.p \mid p+q \mid \tau^\infty$, and

$$\begin{aligned}\sqsubseteq^f_{\mathsf{must}} &= \{(r_1,r_2) \mid \text{ for every } p \in \mathsf{CCS}^f.\ p \;\mathsf{must}\; r_1 \text{ implies } p \;\mathsf{must}\; r_2\}\\ \mathcal{U}^f &= \{r \mid \text{there exists } p \in \mathsf{CCS}^f.\ p \;\mathsf{must}\; r\}\end{aligned}$$

In what follows, we find it convenient to use the definitions above: CCS^f excludes recursively-defined processes, but explicitly adds the divergent process τ^∞ because of its discriminating power (see Counterexample 2). Accordingly, $\sqsubseteq^f_{\mathsf{must}}$ and $\mathcal{U}^f$ restrict the respective sets to the syntactic class CCS^f.

Corollary 2. *The sets $\mathcal{U}_{\mathsf{must}}$ and $\mathcal{U}^f$ coincide.*

Proof. The inclusion $\mathcal{U}^f \subseteq \mathcal{U}_{\mathsf{must}}$ is immediate. Suppose that $r \in \mathcal{U}_{\mathsf{must}}$. By Theorem 1 we have $r \in \mathcal{U}_{\mathsf{bhv}}$. By Lemma 3, there exists a non-recursive $p \in \mathsf{CCS}^f$ such that $p \;\mathsf{must}\; r$, thus $r \in \mathcal{U}^f$ follows. □

Theorem 3. *In any finite-branching LTS $r_1 \sqsubseteq^f_{\mathsf{must}} r_2$ if and only if $r_1 \sqsubseteq_{\mathsf{must}} r_2$.*

Proof. The inclusion $\sqsubseteq_{\mathsf{must}} \subseteq \sqsubseteq^f_{\mathsf{must}}$ follows immediately from the respective definitions. On the other hand, Theorem 2 provides us with a proof technique for showing the inclusion $\sqsubseteq^f_{\mathsf{must}} \subseteq \sqsubseteq_{\mathsf{must}}$: if we show that $\sqsubseteq^f_{\mathsf{must}} \subseteq \mathcal{G}(\sqsubseteq^f_{\mathsf{must}})$ then thanks to the Knaster–Tarski theorem we obtain the set inclusion $\sqsubseteq^f_{\mathsf{must}} \subseteq \precsim_{\mathsf{must}}$ and so thanks to Theorem 2 $\sqsubseteq^f_{\mathsf{must}} = \sqsubseteq_{\mathsf{must}}$. In turn, this requires us to prove the three conditions stated in Definition 7. The argument for the first two conditions is virtually the same to that of Lemma 7. Similarly, the arguments for the third condition follow closely those used in Proposition 1 (albeit in a simpler setting of unsuccessful traces of length 1). The only new reasoning required is that servers that exists because of $r_1 \in \mathcal{U}_{\mathsf{must}}$ also belong to CCS^f, which we know from Corollary 2. □

We claim that an analogous result holds for the server-preorder, for the proofs of completeness in [5, Theorem 3.1] rely on clients written in the language CCS^f.

5.2. Deciding the must client preorder

Fig. 5 describes the code for the function `is_usable(r,acm)`, which decides whether a client `r` is usable. It adheres closely to the conditions of Definition 5 for $\mathcal{U}_{\mathsf{bhv}}$, using `acm` as an *accumulator* to keep track of all the terms that have already been explored. Thus, if an `r` is revisited, the algorithm rejects it on the basis that a loop of unsuccessful interactions (leading to an infinite sequence of unsuccessful interactions that makes the client unusable) is detected (lines 2–3). If not, the algorithm checks for the conditions in Definition 5 (lines 4–6).

Line 4 checks that infinite sequences of internal moves are always successful (using the function `conv`✓ of Fig. 4), while line 5 checks that partially deadlocked clients reached through a finite number of unsuccessful internal moves contain at least one action that unblocks them to some other usable client. This latter check employs the function `unblock_action` (lines 8–11) which recursively calls `is_usable` to determine whether the client reached after an action is indeed usable.

Lemma 9. *For every finite-branching r we have that if `is_usable(r,[])` terminates then `is_usable(r,[]) = true` if and only if $r \in \mathcal{U}$.*

Proof. To prove the *only-if* case we use Theorem 1 and show that $r \in \mathcal{U}_{\mathsf{bhv}}$ implies `is_usable(r,[]) = true`; we do so by numerical induction on $n \in \mathbb{N}^+$ where $r \in \mathcal{F}^n(\emptyset)$. For the *if* case we dually show that `is_usable(r,[]) = true` implies $r \in \mathcal{U}_{\mathsf{bhv}}$, by numerical induction on the *least* number $n \in \mathbb{N}^+$ of (recursive) calls to `is_usable` that yield the outcome true.

```
let rec conv✓ acm r =
  if r ∈ acm
  then false
  else if r ─✓→
       then true
       else List.for_all (conv✓ (r::acm)) {r̂ | r ─τ→ r̂}
```

Fig. 4. An algorithm to decide the predicate $\Downarrow_{\checkmark}$.

```
let rec is_usable r acm =
  if r ∈ acm
  then false
  else if conv✓ acm r
       then List.for_all (List.exists (unblock_action r acm)) Acc_✓̸(r)
       else false
and
  unblock_action r acm a =
    if r ═a═>_✓̸
    then is_usable (⨁(r after_✓̸ a)) (r::acm)
    else true
```

Fig. 5. An algorithm for deciding inclusion in the set $\mathcal{U}^{\mathsf{bhv}}$.

We note that in either direction there is a direct correspondence between the respective inductive indices (*e.g.*, for the base case $n = 1$, $r \in \mathcal{F}^1(\emptyset) = \mathcal{F}(\emptyset)$ implies that $r \Downarrow_{\checkmark}$ and that $\mathsf{Acc}_{\not\checkmark}(r) = \emptyset$).

The function `is_usable(r,acm)` of Fig. 5 relies on the LTS of `r` being *finite-state* in order to guarantee termination via the state accumulation held in `acm`. This is indeed the case for our expository language CCS^{μ} of Fig. 2. Concretely, we define the set of internal-sums for the derivatives that a client r reaches via all the finite traces $s \in \mathsf{Act}^{\star}$, and show that this set is finite. Let

$$\mathsf{sumsRdx}(r) = \{ \bigoplus(r \ \mathsf{after}_{\not\checkmark}\ s) \mid \text{ for some } s \in \mathsf{Act}^{\star} \}$$

Lemma 10. *For every $r \in \mathsf{CCS}^{\mu}$, the set $\mathsf{sumsRdx}(r)$ is finite.* □

Proof. Let $Reach_r = \{ r' \mid r \overset{s}{\Longrightarrow} r' \text{ for some } s \in \mathsf{Act}^{\star} \}$ denote the set of reachable terms from client r, and $PwrR_r = \{ \bigoplus B \mid B \in \mathcal{P}(Reach_r) \}$ denote the elements of the powerset of $Reach_r$, expressed as internal summations of the elements of $\mathcal{P}(Reach_r)$. By definition, we have that $\mathsf{sumsRdx}(r) \subseteq PwrR_r$. Hence, it suffices to prove that $Reach_r$ is finite to show that $PwrR_r$ is finite, from which the finiteness of $\mathsf{sumsRdx}(r)$ follows. The proof of the finiteness of $Reach_r$ is the same as that of [33, Lemma 4.2.11] for the language serial-CCS, which is homologous to CCS^{μ} of Fig. 2 modulo the satisfaction construct 1. □

The previous lemma and the monotone growth of the accumulator `acm` are enough to prove that the function `is_usable` terminates.

Lemma 11. *For every $r \in \mathsf{CCS}^{\mu}$ the algorithm `is_usable(r,[])` terminates.*

Theorem 4. *For every $r \in \mathsf{CCS}^{\mu}$ we have `is_usable(r,[]) = true` if and only if $r \in \mathcal{U}$.*

Proof. Follow Lemma 9 and Lemma 11. □

Thanks to Theorem 4 and Theorem 2, Definition 7 let us decide $\sqsubseteq_{\mathsf{must}}$ for languages such as CCS^{μ} of Fig. 2. In particular, using the terminology of [31] we have that $reachable_{\mathcal{G}}(X)$ is finite, essentially because the respective LTS is finite-state and Lemma 10., and thus the decidability of $\sqsubseteq_{\mathsf{must}}$ follows from a result analogous to [31, Theorem 21.5.9 and Theorem 21.5.12].

6. The compliance preorder for clients

Thus far we have studied the client preorder due to must testing. The theory of web-services [10], though, relies on a notion of client/server satisfaction formalised by a different testing relation, called compliance. Intuitively, a client r complies with a server p if whenever r and p cannot interact or p diverges the client is in a satisfied state.

In this section we present the preorder for clients due to the compliance relation, and we give a behavioural characterisation of the preorder. The proofs omitted here are in Appendix B.

Despite the difference between must and compliance preorders [4], it is known that preorders for servers have similar alternative characterisations [6, Theorems 3.38, 3.50]. The main result of this section (Theorem 5) shows that also the characterisations for client preorders share a common structure. To define compliance we need a predicate for divergence. A process p *diverges*, written $p \Uparrow$, whenever p engages in an infinite sequence of τ actions.

Definition 8 *(Compliance).* Let $\mathcal{C} : \mathcal{P}(\mathsf{Proc} \times \mathsf{Proc}) \to \mathcal{P}(\mathsf{Proc} \times \mathsf{Proc})$ be defined by letting $(r, p) \in \mathcal{C}(R)$ if all the following conditions are true:

1. if $p \Uparrow$ then $r \xrightarrow{\checkmark}$
2. if $p \mid\mid r \overset{\tau}{\not\longrightarrow}$ then $r \xrightarrow{\checkmark}$
3. if $p \mid\mid r \xrightarrow{\tau} p' \mid\mid r'$ then $(r', p') \in R$

A pre-fixed point of $\mathcal{C}$ is a *coinductive compliance*, and we define the *compliance* relation as $\mathsf{cmp} = \nu X.\mathcal{C}(X)$. If $r \,\mathsf{cmp}\, p$ we say that r complies with p. ■

The relation cmp and must are not comparable, that is $r \,\mathsf{cmp}\, p$ does not imply $p \,\mathsf{must}\, r$, and vice-versa. One reason is the treatment of divergence, another reason is the meaning of successful states. This is laid bare in [4, Example 2.6, Example 2.7], which we recall here.

Example 7. We have that $\mu x.a.x \,\mathsf{cmp}\, \mu y.\bar{a}.y$, while $\mu y.\bar{a}.y \,\neg\mathsf{must}\, \mu x.a.x$. Intuitively, the two processes are compliant because they engage in an infinite interaction sequence without ever deadlocking; note also that $\mu y.\bar{a}.y$ does not diverge by itself, even though the composite system $\mu y.\bar{a}.y \mid\mid \mu x.a.x$ does. On the other hand, the pair is not contained in the must relation because the system $\mu y.\bar{a}.y \mid\mid \mu x.a.x$ performs an infinite computation in which the client never reaches a successful state (and according to must unsuccessful divergences are catastrophic). ■

Example 8. We have that $0 \,\mathsf{must}\, 1 + \tau.0$, while $1 + \tau.0 \,\neg\mathsf{cmp}\, 0$. The key insight here is the difference between the success conditions imposed by two relations. We have that $0 \,\mathsf{must}\, 1 + \tau.0$ because $1 + \tau.0 \xrightarrow{\checkmark}$ ensures that every computation starting from $0 \mid\mid 1 + \tau.0$ is (immediately) successful. On the other hand, we have the transition $0 \mid\mid 1 + \tau.0 \xrightarrow{\tau} 0 \mid\mid 0$ and the system $0 \mid\mid 0$ violates Definition 8(2) since $0 \overset{\checkmark}{\not\longrightarrow}$, and hence no coinductive compliance contains the pair $(1 + \tau.0, 0)$. In other terms, the compliance requires stable clients to be successful, while must does not. ■

As for the treatment of divergence, according to compliance ever divergent clients are satisfied by the trivial server 0, as we show in the next lemma.

Lemma 12. *For every $r \in \mathsf{Proc}$ such that $r \Longrightarrow r'$ implies $r' \Uparrow$ we have r cmp 0.*

Proof. Let $R = \{ (r, 0) \mid r \in \mathsf{Proc}.\, r \Longrightarrow r' \text{ implies } r' \Uparrow \}$. It suffices to show that the relation R is a coinductive compliance. Let $(r, p) \in R$. By definition $p = 0$, thus p converges (trivially satisfying Definition 8(1)). Since $r \Longrightarrow r$ we know that r diverges, and thus we have $p \mid\mid r \xrightarrow{\tau}$ (trivially satisfying Definition 8(2)). Let $p \mid\mid r \xrightarrow{\tau} p' \mid\mid r'$. We have that $p' = p = 0$ and $r \Longrightarrow r'$, and hence for every $r' \Longrightarrow r''$ implies r'' diverges (satisfying Definition 8(3)), and thus $(r', p') \in R$. □

Let us now move on from compliance to the preorder for clients due to the compliance relation.

Definition 9. We write $r_1 \sqsubseteq_{\mathsf{cmp}} r_2$ if for every p, $r_1 \,\mathsf{cmp}\, p$ implies $r_2 \,\mathsf{cmp}\, p$. ■

In view of Example 7 and Example 8, it is not surprising that the preorders due to cmp and must, namely $\sqsubseteq_{\mathsf{cmp}}$ and $\underset{\sim}{\sqsubset}_{\mathsf{must}}$, are not comparable [4, Section 3.1] ($\sqsubseteq_{\mathsf{cmp}} \not\subseteq \underset{\sim}{\sqsubset}_{\mathsf{must}}$ and $\underset{\sim}{\sqsubset}_{\mathsf{must}} \not\subseteq \sqsubseteq_{\mathsf{cmp}}$). Despite the set-theoretic differences, though, we can characterise the compliance preorder for clients using behavioural predicates similar to the ones we have seen thus far, and coinductive definition analogous to Definition 7 and [6, Definition 3.36]. While usable clients are defined as expected,

$$\mathcal{U}_{\mathsf{cmp}} = \{ r \in \mathsf{Proc} \mid \text{ there exists } p \in \mathsf{Proc}.\, r \,\mathsf{cmp}\, p \}$$

we need to change the definition of acceptance sets to account for every weak reduction instead of just the unsuccessful ones, and the $\checkmark$ action:

$$\mathsf{Acc}_{\checkmark}(r) = \{\, S_{\checkmark}(r') \mid r \Longrightarrow r' \overset{\tau}{\not\longrightarrow} \,\}$$

where $S_{\checkmark}(r) = \{\, \alpha_{\checkmark} \mid r \xrightarrow{\alpha_{\checkmark}} \,\}$.

To amend the definition of usable action we have first to provide a fully-abstract characterisation of the set $\mathcal{U}_{\mathsf{cmp}}$. It turns out that this set has already been characterised in [29]. Although the compliance relation used in [29] differs slightly from the one given in Definition 8, the characterisation of usable clients is essentially the same (see [29, Proposition 4.3]).

Definition 10 *(Behavioural usability).* Let $\mathcal{U}^{\mathsf{bhv}}_{\mathsf{cmp}}$ be the greatest set such that $r \in \mathcal{U}^{\mathsf{bhv}}_{\mathsf{cmp}}$ if for every $A \in \mathsf{Acc}_{\checkmark}(r)$ either (i) $\checkmark \in A$ or, (ii) there exists $\alpha \in A$ such that $\bigoplus(r \,\mathsf{after}\, \alpha) \in \mathcal{U}^{\mathsf{bhv}}_{\mathsf{cmp}}$. ■

Lemma 13. *The sets $\mathcal{U}_{cmp}$ and $\mathcal{U}^{bhv}_{cmp}$ coincide.*

Note that whereas Definition 5 is an inductive definition, Definition 10 is coinductive. This is so because compliance allows infinite unsuccessful computations, while must disallows them.

To define our alternative characterisation of $\sqsubseteq_{\mathsf{cmp}}$ we need to present a notion that in Section 4 we left implicit: *fallible* clients. Let

$$\mathcal{F}_{\mathsf{cmp}} = \{\, r \in \mathsf{Proc} \mid \text{there exists } p \in \mathsf{Proc} \text{ such that } r \neg\mathsf{cmp}\; p \,\}$$

Fallible clients are easy to characterise in terms of their LTS.

Lemma 14. *$r \in \mathcal{F}_{cmp}$ iff there exists $s \in \mathsf{Act}^{\star}$ st $r \overset{s}{\Longrightarrow} r'$ and $r' \overset{\checkmark}{\not\longrightarrow}$.*

Proof. Pick a client $r \in \mathcal{F}_{\mathsf{cmp}}$. There exists a p_{bad} such that $r \neg\mathsf{cmp}\; p_{bad}$. As cmp is the greatest coinductive compliance this means that for every R such that $(r, p_{bad}) \in R$ we have $R \subseteq\!\!\!\!/\;\, \mathcal{C}(R)$. Let $R = \{\, (r', p') \mid p_{bad} \,||\, r \Longrightarrow p' \,||\, r' \,\}$. Plainly $(r, p_{bad}) \in R$ and R satisfies Definition 8(3), but $R \not\subseteq \mathcal{C}(R)$. It follows that there exists a pair (r', p') in R that does not satisfy one of the implications in point (1) and point (2) of Definition 8 is not satisfied, and hence $r' \overset{\checkmark}{\not\longrightarrow}$. In turns this ensures that for some $s \in \mathsf{Act}^{\star}$ we have the desired $r \overset{s}{\Longrightarrow} r'$.

Fix a client r such that for some $s \in \mathsf{Act}^{\star}$ we have $r \overset{s}{\Longrightarrow} r'$ and $r' \overset{\checkmark}{\not\longrightarrow}$, and let $p_{bad} = \overline{s}.\tau^{\infty}$. The divergence of τ^{∞} and the hypothesis $r' \overset{\checkmark}{\not\longrightarrow}$ let us prove that $r' \neg\mathsf{cmp}\; \tau^{\infty}$. The computation $p_{bad} \,||\, r \Longrightarrow \tau^{\infty} \,||\, r'$ now implies $r \neg\mathsf{cmp}\; p_{bad}$. □

In passing, note that the notion of fallible client is dual to usable clients, and that while both notions may seem ad-hoc, there exists in fact a canonical order theoretic explanation for them, which we will give at the end of this section.

Definition 11. Let $\mathcal{G} : \mathcal{P}(\mathsf{Proc} \times \mathsf{Proc}) \longrightarrow \mathcal{P}(\mathsf{Proc} \times \mathsf{Proc})$ be the function such that $(r_1, r_2) \in \mathcal{G}(R)$ whenever all the following conditions hold:

1. if $r_2 \xrightarrow{\tau} r_2'$ then $r_1 \; R \; r_2'$
2. if $r_1 \in \mathcal{U}^{\mathsf{bhv}}_{\mathsf{cmp}}$ then
   (a) $r_2 \in \mathcal{U}^{\mathsf{bhv}}_{\mathsf{cmp}}$
   (b) if ($r_1 \Longrightarrow r_1'$ implies $r_1' \xrightarrow{\checkmark}$) then ($r_2 \Longrightarrow r_2'$ implies $r_2' \xrightarrow{\checkmark}$)
   (c) for every $B \in \mathsf{Acc}_{\checkmark}(r_2)$ if $\checkmark \not\in B$ then there exists an $A \in \mathsf{Acc}_{\checkmark}(r_1)$ such that $A \cap ua_{\mathsf{bhv}}(r_1) \subseteq B$
   (d) if $r_2 \xrightarrow{a} r_2'$ and $r_2' \in \mathcal{F}_{\mathsf{cmp}}$ then ($r_1 \overset{a}{\Longrightarrow}$ and $\bigoplus(r_1 \,\mathsf{after}\, a) \; R \; \bigoplus(r_2 \,\mathsf{after}\, a)$)

where $ua_{\mathsf{bhv}}(r) = \{\, a \mid r \overset{a}{\Longrightarrow} \text{ implies } \bigoplus(r \,\mathsf{after}\, a) \in \mathcal{U}^{\mathsf{bhv}}_{\mathsf{cmp}} \,\}$. Let $\preceq_{\mathsf{clt}} = \nu x.\mathcal{G}(x)$ where $\nu x.\mathcal{G}(x)$ denotes the greatest fixpoint of $\mathcal{G}$. The function $\mathcal{G}$ is monotone over the complete lattice $\langle \mathcal{P}(\mathsf{Proc} \times \mathsf{Proc}), \subseteq \rangle$ and thus $\nu x.\mathcal{G}(x)$ exists. ■

Lemma 15. *For every $r_1, r_2 \in \mathsf{Proc}$, if $r_1 \sqsubseteq_{cmp} r_2$ and $r_2 \xrightarrow{\tau} r_2'$ then $r_1 \sqsubseteq_{cmp} r_2'$.*

Proof. Fix two clients r_1 and r_2 satisfying the hypotheses, and let $r_1 \;\mathsf{cmp}\; p$. We have to show that $r_2' \;\mathsf{cmp}\; p$. Hypothesis $r_1 \sqsubseteq_{\mathsf{cmp}} r_2$ ensures that $r_2 \;\mathsf{cmp}\; p$. By Definition 8(3) and the reduction $p \,||\, r_2 \xrightarrow{\tau} p \,||\, r_2'$ we obtain $r_2' \;\mathsf{cmp}\; p$. □

Lemma 16. *If $r_1 \sqsubseteq_{cmp} r_2$, $r_1 \in \mathcal{U}^{bhv}_{cmp}$, and $r_2 \xrightarrow{a} r_2'$ with $r_2' \in \mathcal{F}_{cmp}$, then*

1. *$r_1 \overset{a}{\Longrightarrow}$, and*
2. *$\bigoplus(r_1 \,\mathsf{after}\, a) \; R \; \bigoplus(r_2 \,\mathsf{after}\, a)$.*

Proof. The hypothesis that $r_2' \in \mathcal{F}_{\mathsf{cmp}}$ ensures that there exists a p_{bad} such that $r_2' \neg\mathsf{cmp}\; p_{bad}$, and the hypothesis that $r_1 \in \mathcal{U}^{\mathsf{bhv}}_{\mathsf{cmp}}$ implies that there exists a p such that $r_1 \;\mathsf{cmp}\; p$. Let $\hat{p} = p + \overline{a}.p_{bad}$. Thanks to an interaction on the action a we have the computation $\hat{p} \parallel r_2 \Longrightarrow p_{bad} \parallel r_2'$, and thus the definition of $r_2' \neg\mathsf{cmp}\; p_{bad}$ and Definition 8(3) ensures that $r_2 \;\neg\mathsf{cmp}\; \hat{p}$.

It follows that $r_1 \neg\mathsf{cmp}\; \hat{p}$. To prove that $r_1 \overset{a}{\Longrightarrow}$ consider the following relation,

$$R = \{ (r, p + \overline{a}.q) \mid r, p, q \in \mathsf{Proc}, r \;\mathsf{cmp}\; p, r \overset{a}{\not\Longrightarrow} \} \cup \mathsf{cmp}$$

The relation R is a coinductive compliance, i.e. $(r, p + \overline{a}.q) \in \mathcal{C}(R)$ for the $\mathcal{C}$ given in Definition 8. To see why this is the case, pick a pair $r \; R \; p + \overline{a}.q$, we have that either ($r \;\mathsf{cmp}\; p + \overline{a}.q$) or $r \;\mathsf{cmp}\; p$ and r performs no a action (not even weakly). In both cases it is routine work to check that $(r, p + \overline{a}.q) \in \mathcal{C}(R)$. Now $r_1 \neg\mathsf{cmp}\; \hat{p}$ ensures that $r_1 \not R\; \hat{p}$, but $r_1 \;\mathsf{cmp}\; p$, thus it must be the case that $r \overset{a}{\Longrightarrow}$.

To prove point (2), let $\hat{r}_i = \bigoplus(r_i \;\mathsf{after}\; a)$ for $i \in 1, 2$, and fix a process p such that $\hat{r}_1 \;\mathsf{cmp}\; p$. We have to explain why $\hat{r}_2 \;\mathsf{cmp}\; p$. By hypothesis $r_1 \in \mathcal{U}_{\mathsf{cmp}}$, thus there exists a process p_{good} such that $r_1 \;\mathsf{cmp}\; p_{good}$. Let $\hat{p} = p_{good} + \overline{a}.p$. The assumptions on p and p_{good} ensure that $r_1 \;\mathsf{cmp}\; \hat{p}$, as witnessed by the coinductive compliance $R = \{ (r', p + \overline{a}.q) \mid r_1 \Longrightarrow r', r_1 \;\mathsf{cmp}\; p, \hat{r}_1 \;\mathsf{cmp}\; q \} \cup \mathsf{cmp}$. The hypothesis that $r_1 \sqsubseteq_{\mathsf{cmp}} r_2$ implies $r_2 \;\mathsf{cmp}\; \hat{p}$. For every $r' \in (r_2 \;\mathsf{after}\; a)$ there exists the computation $\hat{p} \parallel \hat{r}_2 \Longrightarrow p \parallel r'$, and thus $r' \;\mathsf{cmp}\; p$. This suffices to show that $\hat{r}_2 \;\mathsf{cmp}\; p$ as desired. □

Lemma 17. *If $r_1 \sqsubseteq_{cmp} r_2$ and $r_1 \in \mathcal{U}^{bhv}_{cmp}$ then for every $B \in \mathsf{Acc}_{\checkmark}(r_2)$ such that $\checkmark \notin B$ there exists an $A \in \mathsf{Acc}_{\not\checkmark}(r_1)$ such that $A \cap ua_{\mathsf{bhv}}(r_1) \subseteq B$.*

Theorem 5. *The preorders $\sqsubseteq_{cmp}$ and $\preceq_{\mathsf{clt}}$ coincide.*

Proof. See Appendix B. □

6.1. Order theoretic meaning of usable and fallible clients

To begin with, observe that also Definition 7 can be formulated using fallible clients wrt must testing. Indeed, if we let

$$\mathcal{F}_{\mathsf{must}} = \{ r \in \mathsf{Proc} \mid \exists p \in \mathsf{Proc} \text{ such that } p \neg\mathsf{must}\; r \}$$

then thanks to divergent servers we have that

$$r \in \mathcal{F}_{\mathsf{must}} \text{ if and only if } r \overset{\checkmark}{\not\longrightarrow} \tag{18}$$

and thus Definition 7(2) is equivalent to

$$\text{if } r_2 \in \mathcal{F}_{\mathsf{must}} \text{ then } r_1 \in \mathcal{F}_{\mathsf{must}} \tag{2F}$$

and unsuccessful traces $\overset{s}{\Longrightarrow}_{\not\checkmark}$ are equivalent to traces that cross only fallible clients. In Section 4 we omitted the discussion of fallible clients merely to simplify the presentation.

While usable and fallible clients have somewhat intuitive definitions respectively in terms of being satisfied by at least one server and of being not satisfied by every server, they also have alternative less ad-hoc characterisations as clients that are not bottom and not top wrt the preorder at issue.

Proposition 1. *For every $r \in \mathsf{Proc}$,*

1. *$r \in \mathcal{U}_{\mathsf{must}}$ if and only if r not bottom in $\sqsubseteq_{\mathsf{must}}$;*
2. *$r \in \mathcal{U}_{cmp}$ if and only if r not bottom in $\sqsubseteq_{cmp}$.*

Proof. We prove the two parts of the proposition in order. For point (1), let $r \in \mathcal{U}_{\mathsf{must}}$. By definition there exists a p_{good} such that $p_{good} \;\mathsf{must}\; r$, but $p_{good} \neg\mathsf{must}\; 0$, and so $r \not\sqsubseteq_{\mathsf{must}} 0$. That is r is not a bottom element of $\sqsubseteq_{\mathsf{must}}$. Now fix a client r that is not bottom in $\sqsubseteq_{\mathsf{must}}$. By definition for some r' we have $r \not\sqsubseteq_{\mathsf{must}} r'$, that is there exists a server p such that $p \;\mathsf{must}\; r$ and $p \neg\mathsf{must}\; r'$. But then by definition $r \in \mathcal{U}_{\mathsf{must}}$.

The second part of the proposition can be shown using the proof of point (1), but with cmp in place of must. □

Proposition 2. *For every $r \in \mathsf{Proc}$,*

1. *$r \in \mathcal{F}_{\mathsf{must}}$ if and only if r is not top in $\sqsubseteq$;*
2. *$r \in \mathcal{F}_{cmp}$ if and only if r is not top in $\sqsubseteq_{cmp}$.*

Proof. Fix a $r \in \mathcal{F}_{\mathsf{must}}$. By definition p_{bad} $\neg$must r for some p_{bad}. It follows that $1 \not\precsim_{\mathsf{must}} r$, because p_{bad} must 1, and thus r is not top in $\precsim_{\mathsf{must}}$. Now pick a client r that is not top in $\precsim_{\mathsf{must}}$. By definition there exists some r' such that $r' \not\precsim_{\mathsf{must}} r$, and hence for some p_{bad} we have p_{bad} must r' while p_{bad} $\neg$must r, that is $r \in \mathcal{F}_{\mathsf{must}}$.

The proof for the second part of the proposition is the same but with cmp in place of must. □

Interestingly, the arguments to show Proposition 1 and Proposition 2 depend respectively only on one property of the relation of satisfaction employed (must, cmp, ...), namely that there exists one client that is satisfied by no server, in our case 0, and that there exists one client satisfied by every server, in our case 1. We believe therefore that the order-theoretic justification of usable and fallible clients extend to other notions of passing a test/satisfaction, such as fair testing, and also that our results reveal a possible approach to characterising testing preorders: first study the elements that are not bottom and not top, and afterwards define an alternative characterisation.

7. Conclusion

We present a study that revolves around the notion of usability and preorders for clients (tests). Preorders for clients and peers first appeared for compliance [2], to tie testing theory with session type theory: these preorders are instrumental in defining semantic models of the Gay & Hole subtyping [17] for first-order session types [3, Theorem 6.3.4] and [6, Theorem 5.7]. Client and peer preorders have subsequently been investigated in [3,5] for must testing [14]. The characterisations given in [5,3] rely fundamentally on the sets $\mathcal{U}_{\mathsf{must}}$, $\mathcal{U}_{\mathsf{cmp}}$ of usable clients, which make them not fully-abstract and hard to automate. This provided the main impetus for our study. In general, recursion poses obstacles when characterising usable terms wrt must, but the very nature of must testing — which regards infinite unsuccessful computations as catastrophic — let us treat recursive terms in a finite (inductive) manner (see Definition 5).

We focus on the client preorders, even though [5,3] presents preorders for both client and peers; note however that [5, Theorem 3.20] and Theorem 1 imply full-abstraction for the peer preorders as well. Our investigations and the respective proofs for Theorem 1, Theorem 2 and Theorem 3 are conducted in terms of finitely-branching LTS, which cover the semantics used by numerous other work describing client and server contracts [9,22,10,5] — we only rely on an internal choice construct to economise on our presentation, but this can be replaced by amending the respective definitions to work on sets of processes instead. As a consequence, the results obtained should also extend to arbitrary languages enjoying the finite-branching property. Theorem 4 relies on a stronger property, namely that the language is finite-state. In [33], it is shown that this property is also enjoyed by larger CCS fragments, and we therefore expect our results to extend to these fragments as well.

7.1. Related work

Client usability depends both on language expressiveness and on the notion of testing employed. Our comparison with the related work is organised accordingly.

Session types [17] do not contain unsuccessful termination, 0, restrict internal (respective external) choices to contain only pair-wise distinct outputs (respective inputs) and are, by definition, strongly convergent [29] (*i.e.*, no infinite sequences of τ-transitions). *E.g.*, $\tau.!a.1 + \tau.!b.?c.1$ corresponds to a session type in our language (modulo syntactic transformations such as those for internal choices), whereas $\tau.!a.0 + \tau.!b.?c.1$, $\tau.!a.1 \ + \ \tau.!a.?b.1$ and $?a.1 \ + \ ?a.!b.1$ do *not*. Since they are mostly deterministic — only internal choices on outputs are permitted — usability is relatively easy to characterise. In fact [7, Section 5] shows that every session type is usable *wrt.* compliance testing (even in the presence of higher-order communication) whereas, in [30, Theorem 4.3], *non-usable* session types are characterised *wrt.* fair testing. First-order session types are a subset of our language, and hence, Theorem 1 is enough to (positively) characterise usable session types *wrt.* must testing; we leave the axiomatisation of $\mathcal{U}$ in this setting as future work. Testing theories have proven fit to lay the semantic foundations of session types, for instance [7] shows that the compliance preorder for peers is a semantic model of subtyping [17], and the compliance relation itself is a semantic definition of type duality. This suggests that testing relations should be seen as semantic dualities, and the preorders that they induce as semantic subtypings.

Contracts [29] are usually formalised as (mild variants of) our language CCS^{μ}. In the case of must testing and the sublanguage CCS^{f}, the pre-congruences associated to the client, peer and server preorders have been equationally characterised in [5]. In particular Theorem 6.9 and Lemma 7.8(2) there characterise *non*-usable clients (and peers) as the terms that can be re-written into 0 via equational reasoning. Full-abstraction for usable clients *wrt. compliance* testing has been achieved for *strongly convergent terms* in [29, Proposition 4.3] via a coinductive characterisation of viable (*i.e.*, usable *wrt.* compliance) contracts. If we restrict our language to strongly convergent terms, that characterisation is neither sound nor complete *wrt.* must testing. It is unsound because clients such as $\mu x.a.x$ are viable but *not* usable. It is incomplete because of clients such as $r = 1 + \tau.0$; this client is usable *wrt.* must because, for arbitrary p, any computation of $p \mid\mid r$ is successful (since we have $r \xrightarrow{\checkmark}$ immediately). On the other hand, r is *not* viable *wrt.* compliance testing of [29] (where every server is strongly convergent), because for any server p we observe the computation starting with the reduction $p \mid\mid r \xrightarrow{\tau} p \mid\mid 0$, and once p stabilises to some p', the final state $p' \mid\mid 0$ contains an unsuccessful client. This argument relies on subtle discrepancies in the definitions of the testing relations: in must testing it suffices for maximal computations to *pass through* a successful state, whereas in compliance testing the *final state* of the computation (if any) is required to be successful. This aspect impinges on the technical development: although our Definition 5(2) resembles [29, Definition 4.2], the two definitions have strikingly different meanings: we are forced to reason *wrt. unsuccessful* actions and *unsuccessful* acceptance sets whereas [29, Definition 4.2] is defined in terms of (standard) weak actions and acceptance sets (note that Definition 5(1) holds trivially in the strongly convergent setting of [29]). We note also that our Definition 5 is inductive whereas [29, Definition 4.2] is coinductive. More importantly, our work lays bare the *non-compositionality* of usable terms and how it affects other notions that depend on it, such as Definition 7 (and consequently Theorem 2). We are unaware of any full-abstraction results for contract usability in the case of should-testing [32,9,28].

Controllable, Usable, Viable: [28, Definition 5] states that a process x is *controllable* if and only if there exists a partner for x. This is the essential idea behind usability, as in our setting we say that a client r is usable if and only if "there exists a partner for r" means "there exists a server that must pass r". Viable [29] is yet another term to express the same idea, which we think should be phrased using the notion of satisfaction as parameter, for instance must-controllable, cmp-controllable, etc... We hope that this observation will encourage practitioners to adopt a uniform terminology.

Future work: In the line of [11], we plan to show a logical characterisation of the client and peer preorder. We also intend to investigate coinductive characterisations for the peer preorder of [5] and subsequently implement decision procedures for the server, client, and peer preorders in CAAL [1]. Usability is not limited to tests. We expect it to extend naturally to runtime monitoring [15,16], where it can be used as a means of lowering runtime overhead by not instrumenting unusable monitors.

Acknowledgements

This research was supported by the COST Action STSMs IC1201-130216-067787 and IC1201-170214-038253. The first author was supported by the EU FP7 ADVENT project. The second author is partly supported by the RANNIS THEOFOMON project 163406-051. The authors acknowledge the Dagstuhl seminar 17051 and thank L. Aceto, M. Bravetti, A. Gorla, M. Hennessy, C. Spaccasassi and anonymous reviewers for their help and suggestions.

Appendix A. Proofs for the coinductive characterisation of must preorder

Lemma 7. $r_1 \sqsubseteq_{\mathsf{must}} r_2$ *implies*

(i) *if* $r_2 \xrightarrow{\tau}_{\not\checkmark} r_2'$ *then* $r_1 \sqsubseteq_{\mathsf{must}} r_2'$;
(ii) *if* $r_2 \overset{\checkmark}{\not\longrightarrow}$ *then* $r_1 \overset{\checkmark}{\not\longrightarrow}$
(iii) *if* $r_1 \in \mathcal{U}$ *and* $r_2 \xrightarrow{a}_{\not\checkmark}$ *then* ($r_1 \overset{a}{\Longrightarrow}_{\not\checkmark}$ *and* $\bigoplus(r_1 \;\mathsf{after}_{\not\checkmark}\; a)\; R \bigoplus(r_2 \;\mathsf{after}_{\not\checkmark}\; a)$).

Proof. To show point (i), suppose that $r_1 \sqsubseteq_{\mathsf{must}} r_2$ and that $r_2 \xrightarrow{\tau}_{\not\checkmark} r_2'$. Pick a p such that $p \;\mathsf{must}\; r_1$. The hypotheses imply that $p \;\mathsf{must}\; r_2$. Every maximal computation of $r_2' \,||\, p$ is a suffix of a maximal computation of $r_2 \,||\, p$. Since $r_2 \overset{\checkmark}{\not\longrightarrow}$ and $r_2' \overset{\checkmark}{\not\longrightarrow}$ it must be the case that every maximal computation of $r_2' \,||\, p$ contains a successful state, thus $p \;\mathsf{must}\; r_2'$. It follows that $r_1 \sqsubseteq_{\mathsf{must}} r_2'$.

To show point (ii), suppose that $r_1 \sqsubseteq_{\mathsf{must}} r_2$ and that $r_2 \overset{\checkmark}{\not\longrightarrow}$. It follows that

$$\tau^\infty \neg\mathsf{must}\; r_2,$$

(where τ^∞ is a divergent server performing an infinite number of τ transitions) thus $\tau^\infty \neg\mathsf{must}\; r_1$. In turn, this implies that $r_1 \overset{\checkmark}{\not\longrightarrow}$.

To show point (iii), assume that $r_2 \xrightarrow{a}_{\not\checkmark}$. This implies that $r_2 \overset{a}{\Longrightarrow}_{\not\checkmark}$ and thus, by Definition 3 and Proposition 1, we obtain $r_1 \overset{a}{\Longrightarrow}_{\not\checkmark}$.

Let $r_1^a = \bigoplus(r_1 \;\mathsf{after}_{\not\checkmark}\; a)$ and $r_2^a = \bigoplus(r_2 \;\mathsf{after}_{\not\checkmark}\; a)$. We have to prove that $r_1^a \sqsubseteq_{\mathsf{must}} r_2^a$, and Definition 1 requires us to show that whenever $p \;\mathsf{must}\; r_1^a$ then $p \;\mathsf{must}\; r_2^a$. Pick a p such that $p \;\mathsf{must}\; r_1^a$.

To prove that $p \;\mathsf{must}\; r_2^a$, we first state an ancillary fact: The hypothesis that $r_1 \in \mathcal{U}$ ensure that there exists a q such that $q \;\mathsf{must}\; r_1$. Thus, the assumption that $p \;\mathsf{must}\; r_1^a$ ensures that $q + \overline{a}.p \;\mathsf{must}\; r_1$. The hypothesis $r_1 \sqsubseteq_{\mathsf{must}} r_2$ now implies that

$$q + \overline{a}.p \;\mathsf{must}\; r_2 \tag{A.1}$$

Back to the main argument, without loss of generality pick a maximal computation c of

$$p \,||\, r_2^a = p^0 \,||\, r^0 \xrightarrow{\tau} p^1 \,||\, r^1 \ldots \tag{A.2}$$

Note that since $p \;\mathsf{must}\; r_2^a$ and $r_2^a \overset{\checkmark}{\not\longrightarrow}$, then the server p converges (otherwise we could construct an unsuccessful computation contradicting $p \;\mathsf{must}\; r_2^a$). This, in turn, ensures that the maximal computation c contains a prefix of τ-transitions whose last τ-action is due to an internal choice of r_2^a, and whose other τ-transitions are due internal choices of p. In other terms, the computation in Eq. (A.2) above contains a prefix

$$p^0 \,||\, r^0 \Longrightarrow p^i \,||\, r^i \Longrightarrow \ldots$$

such that $r^i \in (r_2 \text{ after}_{\not\checkmark}\, a)$, and that $p \overset{\tau}{\Longrightarrow} p_i$. Observe now that

$$q + \bar{a}.p \,||\, r_2 \overset{\tau}{\Longrightarrow} p \,||\, r^i \Longrightarrow p^i \,||\, r^i \Longrightarrow \ldots \tag{A.3}$$

is a maximal computation of $q + \bar{a}.p \,||\, r_2$, whose suffix $p^i \,||\, r^i \Longrightarrow \ldots$ is a suffix of the computation in Eq. (A.2), and whose first τ is due a weak synchronisation on the action a. It follows that to show a successful state in Eq. (A.2), it is sufficient to prove that the successful state in the computation in Eq. (A.3) appears after the state $p^i \,||\, r^i$. But this is true because by assumption $r_i \in (r_2 \text{ after}_{\not\checkmark}\, a)$, thus (A.2) is successful. Since this argument applies for any maximal computation of $p \,||\, r_2^a$, we also have that p must r_2^a as required. $\square$

Lemma 18 *(Monotonicity). Let $R, R' \subseteq (\mathsf{Proc} \times \mathsf{Proc})$. If $R \subseteq R'$ then $\mathcal{G}(R) \subseteq \mathcal{G}(R')$.*

Proof. Fix a pair $(r_1, r_2) \in \mathcal{G}(R)$. To prove that $(r_1, r_2) \in \mathcal{G}(R')$, Definition 7 requires us to show that the pair enjoys the following properties,

1. if $r_2 \xrightarrow{\tau}_{\not\checkmark} r_2'$ then $r_1 \, R \, r_2'$
2. if $r_2 \overset{\checkmark}{\not\longrightarrow}$ then $r_1 \overset{\checkmark}{\not\longrightarrow}$
3. if $r_1 \in \mathcal{U}_{\mathsf{bhv}}$ then
   (a) $r_2 \in \mathcal{U}_{\mathsf{bhv}}$
   (b) if $B \in \mathsf{Acc}_{\not\checkmark}(r_2)$ then there exists a $A \in \mathsf{Acc}_{\not\checkmark}(r_1)$ such that $A \cap ua(r_1) \subseteq B$
   (c) if $r_2 \xrightarrow{a}_{\not\checkmark}$ then $(r_1 \overset{a}{\Longrightarrow}_{\not\checkmark}$ and $\bigoplus(r_1 \text{ after}_{\not\checkmark}\, a) \, R \, \bigoplus(r_2 \text{ after}_{\not\checkmark}\, a))$.

The only property worth discussing is the last one, which follows from the assumption that $(r_1, r_2) \in \mathcal{G}(R)$, from Definition 7(3c), and the hypothesis $R \subseteq R'$. $\square$

We begin by proving some ancillary technical results, which we spell out in Lemma 19.

Lemma 19. *For every $as \in \mathsf{Act}^\star$, and every $r \in \mathsf{CCS}$, we have that*

1. *$r \overset{as}{\Longrightarrow}_{\not\checkmark} r'$ if and only if $(r \text{ after}_{\not\checkmark}\, a) \overset{s}{\Longrightarrow}_{\not\checkmark} r'$*
2. *$(r \text{ after}_{\not\checkmark}\, as) = (\bigoplus(r \text{ after}_{\not\checkmark}\, a) \text{ after}_{\not\checkmark}\, s)$,*
3. *$\mathsf{Acc}_{\not\checkmark}(r, as) = \mathsf{Acc}_{\not\checkmark}(\bigoplus(r \text{ after}_{\not\checkmark}\, a), s)$,*
4. *$ua_{\mathsf{clt}}(r, as) = ua_{\mathsf{clt}}(\bigoplus(r \text{ after}_{\not\checkmark}\, a), s)$.*

Proof. Point (1) follows easily from the definition of $\text{after}_{\not\checkmark}$. Moreover, point (2) is a consequence of point (1), and similarly for point (3). To prove point (4) we have to show two set inclusions, namely

1. $ua_{\mathsf{clt}}(r, as) \subseteq ua_{\mathsf{clt}}(\bigoplus(r \text{ after}_{\not\checkmark}\, a), s)$
2. $ua_{\mathsf{clt}}(\bigoplus(r \text{ after}_{\not\checkmark}\, a), s) \subseteq ua_{\mathsf{clt}}(r, as)$

For the first inclusion, let $\hat{r} = \bigoplus(r \text{ after}_{\not\checkmark}\, a)$ and pick an action $b \in ua_{\mathsf{clt}}(r, as)$; we have to show that $b \in ua_{\mathsf{clt}}(\hat{r}, s)$. From the definition of $ua_{\mathsf{clt}}(-,-)$ in Section 2.1, we have to prove that if $\hat{r} \overset{sb}{\Longrightarrow}_{\not\checkmark}$ then $(\bigoplus \hat{r} \text{ after}_{\not\checkmark}\, sb) \in \mathcal{U}$. Thus, suppose that $\hat{r} \overset{sb}{\Longrightarrow}_{\not\checkmark}$; by $\hat{r} = \bigoplus(r \text{ after}_{\not\checkmark}\, a)$ we deduce that $r \overset{asb}{\Longrightarrow}_{\not\checkmark}$, thus the definition of $ua_{\mathsf{clt}}(-,-)$ ensures that then $r \; usbl_{\not\checkmark} \; asb$. Now observe that

$$\forall s' \in \mathsf{Act}^\star . \, r \; usbl \; s' \text{ if and only if } \forall s'' \text{ prefix of } s'. \bigoplus(r \text{ after}_{\not\checkmark}\, s'') \in \mathcal{U}.$$

It follows that $\bigoplus(r \text{ after}_{\not\checkmark}\, asb) \in \mathcal{U}$, and the required result, $(\bigoplus \hat{r} \text{ after}_{\not\checkmark}\, sb) \in \mathcal{U}$, follows by point (2) of the lemma. The argument to prove the second set inclusion is analogous to the one above. $\square$

Lemma 20. *For every $r_1 \precsim_{\mathsf{must}} r_2$, if for every $s \in \mathsf{Act}^\star$, if $r_1 \; usbl_{\not\checkmark} \; s$ then $r_2 \; usbl_{\not\checkmark} \; s$.*

Proof. As preliminary observation, note that for every $s \in \mathsf{Act}^\star$, $r_1 \; usbl_{\not\checkmark} \; s$ implies that $r_1 \in \mathcal{U}$, thus $r_1 \precsim_{\mathsf{must}} r_2$ and Definition 7(3a) imply that $r_2 \in \mathcal{U}$.

We continue our reasoning by structural induction on the string s. For the base case, $s=\varepsilon$, we have to prove that $r_2\ \mathit{usbl}_{\not\checkmark}\ \varepsilon$. This is equivalent to showing that $r_2 \in \mathcal{U}$, which we already know.

For the inductive case we have $s = as'$ for some a and $s' \in \mathsf{Act}^\star$. To prove that $r_2\ \mathit{usbl}_{\not\checkmark}\ s$ we have to show that

1. $r_2 \in \mathcal{U}$, and
2. if $r_2 \overset{a}{\Longrightarrow}_{\not\checkmark}$ then $\bigoplus(r_2 \text{ after}_{\not\checkmark}\ a)\ \mathit{usbl}_{\not\checkmark}\ s'$.

We have already shown (1). To prove (2) suppose that $r_2 \overset{a}{\Longrightarrow}_{\not\checkmark}$. Since $r_1 \in \mathcal{U}$ and $r_1 \precsim_{\mathsf{must}} r_2$, Definition 7(3c) implies that $r_1 \overset{a}{\Longrightarrow}_{\not\checkmark}$ and that $\bigoplus(r_1 \text{ after}_{\not\checkmark}\ a) \precsim_{\mathsf{must}} \bigoplus(r_2 \text{ after}_{\not\checkmark}\ a)$. The hypothesis $r_1\ \mathit{usbl}_{\not\checkmark}\ as'$ together with $r_1 \overset{a}{\Longrightarrow}_{\not\checkmark}$ ensures that $\bigoplus(r_1 \text{ after}_{\not\checkmark}\ a)\ \mathit{usbl}_{\not\checkmark}\ s'$, and so the inductive hypothesis guarantees that $\bigoplus(r_1 \text{ after}_{\not\checkmark}\ a)\ \mathit{usbl}_{\not\checkmark}\ s'$, which proves (2). □

Lemma 21. *For every $r_1 \precsim_{\mathsf{must}} r_2$, if for every $s \in \mathsf{Act}^\star$, whenever $r_1\ \mathit{usbl}_{\not\checkmark}\ s$ then for every $B \in \mathsf{Acc}_{\not\checkmark}(r_2, s)$, there exists an $A \in \mathsf{Acc}_{\not\checkmark}(r_1, s)$ such that $A \cap \mathit{ua}_{\mathsf{clt}}(r_2, s) \subseteq B$.*

Proof. We proceed by structural induction on the string s.

For the base case we have $s = \varepsilon$. Fix a $B \in \mathsf{Acc}_{\not\checkmark}(r_2, \varepsilon)$, while recalling that $\mathsf{Acc}_{\not\checkmark}(r_2) = \mathsf{Acc}_{\not\checkmark}(r_2, \varepsilon)$. The hypothesis $r_1\ \mathit{usbl}_{\not\checkmark}\ \varepsilon$ implies that $r_1 \in \mathcal{U} = \mathcal{U}_{\mathsf{bhv}}$. Thus, by $r_1 \precsim_{\mathsf{must}} r_2$ and Definition 7(3b), we know that there exists an $A \in \mathsf{Acc}_{\not\checkmark}(r_1)$, such that $A \cap \mathit{ua}(r_1) \subseteq B$. The required condition follows since $\mathsf{Acc}_{\not\checkmark}(r_1) = \mathsf{Acc}_{\not\checkmark}(r_1, \varepsilon)$.

For the inductive case we have $s = as'$ for some a and s'. Pick a set $B \in \mathsf{Acc}_{\not\checkmark}(r_2, as')$: we have to show that there exists a set $A \in \mathsf{Acc}_{\not\checkmark}(r_1, as')$ such that $A \cap \mathit{ua}_{\mathsf{clt}}(r_1, as') \subseteq B$. The definition of $\mathsf{Acc}_{\not\checkmark}(r_1, as')$ implies that $r_2 \overset{a}{\Longrightarrow}_{\not\checkmark} r_2' \overset{s'}{\Longrightarrow}_{\not\checkmark}$, thus point (3c) and point (1) of Definition 7 let us deduce that

$$r_1 \overset{a}{\Longrightarrow}_{\not\checkmark} \text{ and that } r_1^a \precsim_{\mathsf{must}} r_2' \quad \text{where } r_1^a = \bigoplus(r_1 \text{ after}_{\not\checkmark}\ a). \tag{A.4}$$

The hypothesis $r_1\ \mathit{usbl}_{\not\checkmark}\ as'$ ensures that $r_1^a\ \mathit{usbl}_{\not\checkmark}\ s'$ and, moreover, $B \in \mathsf{Acc}_{\not\checkmark}(r_2, as')$ implies that $B \in \mathsf{Acc}_{\not\checkmark}(r_2', s')$. Thus, by (A.4) and the inductive hypothesis, we obtain

$$\exists A \in \mathsf{Acc}_{\not\checkmark}(r_1^a, s').\ A \cap \mathit{ua}_{\mathsf{clt}}(r_1^a, s') \subseteq B \tag{A.5}$$

By point (3) and point (4) of Lemma 19, we have the equalities $\mathit{ua}_{\mathsf{clt}}(r_1, as') = \mathit{ua}_{\mathsf{clt}}(r_1^a, s')$ and $\mathsf{Acc}_{\not\checkmark}(r_1, as') = \mathsf{Acc}_{\not\checkmark}(r_1^a, s')$, and thus from (A.5) we obtain

$$\exists A \in \mathsf{Acc}_{\not\checkmark}(r_1, as').\ A \cap \mathit{ua}_{\mathsf{clt}}(r_1, as') \subseteq B$$

as required. □

Lemma 22. *For every $r_1 \precsim_{\mathsf{must}} r_2$, if for every $s \in \mathsf{Act}^\star$, if $r_1\ \mathit{usbl}_{\not\checkmark}\ s$ and $r_2 \overset{s}{\Longrightarrow}_{\not\checkmark}$, then $r_1 \overset{s}{\Longrightarrow}_{\not\checkmark}$.*

Proof. We proceed by structural induction on the string s. In the base case we have $s = \varepsilon$, and we must show that $r_1 \Longrightarrow_{\not\checkmark}$. Reflexivity ensures that it suffices to show that $r_1 \overset{\checkmark}{\not\longrightarrow}$. This follows from the hypothesis $r_1 \precsim_{\mathsf{must}} r_2$, the hypothesis $r_2 \Longrightarrow_{\not\checkmark}$ which ensures that $r_2 \overset{\checkmark}{\not\longrightarrow}$, and Definition 7(2).

For the inductive case we have $s = as'$ for some a and s'. The hypotheses ensure that $r_1\ \mathit{usbl}_{\not\checkmark}\ as'$ and that $r_2 \overset{a}{\Longrightarrow}_{\not\checkmark} r_2' \overset{s'}{\Longrightarrow}_{\not\checkmark}$ for some r_2'. We have to show that $r_1 \overset{as'}{\Longrightarrow}_{\not\checkmark}$. By the definition of $r_1\ \mathit{usbl}_{\not\checkmark}\ as'$, we know $r_1 \in \mathcal{U}$. Thus by $r_1 \precsim_{\mathsf{must}} r_2$, $r_2 \overset{a}{\Longrightarrow}_{\not\checkmark} r_2'$, and point (3c) and point (1) of Definition 7 let us deduce that

$$r_1 \overset{a}{\Longrightarrow}_{\not\checkmark} \quad \text{and} \quad \bigoplus(r_1 \text{ after}_{\not\checkmark}\ a) \precsim_{\mathsf{must}} r_2' \tag{A.6}$$

From $r_1\ \mathit{usbl}_{\not\checkmark}\ as'$ we also know that $\bigoplus(r_1 \text{ after}_{\not\checkmark}\ a)\ \mathit{usbl}_{\not\checkmark}\ s'$. Thus, by (A.6), $r_2' \overset{s'}{\Longrightarrow}_{\not\checkmark}$, and the inductive hypothesis we obtain that $\bigoplus(r_1 \text{ after}_{\not\checkmark}\ a) \overset{s'}{\Longrightarrow}_{\not\checkmark}$. This ensures that for some $r_1' \in (r_1 \text{ after}_{\not\checkmark}\ a)$ we have $r_1' \overset{s'}{\Longrightarrow}_{\not\checkmark}$. The definition of $(r_1 \text{ after}_{\not\checkmark}\ a)$ implies that $r_1 \overset{a}{\Longrightarrow}_{\not\checkmark} r_1'$, thus we can construct $r_1 \overset{as'}{\Longrightarrow}_{\not\checkmark}$ as required. □

Appendix B. Proof for the coinductive characterisation of cmp preorder

Lemma 17. *If $r_1 \sqsubseteq_{\mathit{cmp}} r_2$ and $r_1 \in \mathcal{U}^{\mathit{bhv}}_{\mathit{cmp}}$ then for every $B \in \mathsf{Acc}_{\checkmark}(r_2)$ such that $\checkmark \notin B$ there exists an $A \in \mathsf{Acc}_{\not\checkmark}(r_1)$ such that $A \cap \mathit{ua}_{\mathsf{bhv}}(r_1) \subseteq B$.*

Proof. Fix a $B \in \mathsf{Acc}_{\checkmark}(r_2)$. To begin with, we show that there exists some $A \in \mathsf{Acc}_{\not\checkmark}(r_1)$. Let r_2' be the state such that $r_2 \Longrightarrow r_2' \overset{\tau}{\not\longrightarrow}$ and $S_{\checkmark}(r_2') = B$. From the computation $0 \,||\, r_2 \Longrightarrow 0 \,||\, r_2' \overset{\tau}{\not\longrightarrow}$ and the hypothesis that $\checkmark \notin B$ we have that $r_2' \neg\mathsf{cmp}\, 0$, and thus $r_2 \neg\mathsf{cmp}\, 0$. It follows that $r_1 \neg\mathsf{cmp}\, 0$, and by Lemma 12 $r_1 \Longrightarrow r_1' \overset{\tau}{\not\longrightarrow}$ for some r_1'. Since r_1' is stable we have $\mathsf{Acc}_{\not\checkmark}(r_1) = \{\, A_i \mid i \in I \,\}$ for some non-empty finite I. Now we proceed by contradiction: suppose that for every $i \in I$ there exists an action $\alpha_i \in A_i \cap ua(r_1)$ such that $\alpha_i \notin B$. We use this assumption to provide a witness server p such that (a) $r_2 \neg\mathsf{cmp}\, p$ and (b) $r_1 \,\mathsf{cmp}\, p$, thereby contradicting the hypothesis $r_1 \sqsubseteq_{\mathsf{cmp}} r_2$.

Observe that α_i may be $\checkmark$. In order to reason only on visible actions, let J be the indexes of the actions $\hat{\alpha}_j \in \mathsf{Act}$, that is the indexes of the actions α_i that are not $\checkmark$. The definition of $ua_{\mathsf{bhv}}(r_1)$ ensures that for every $j \in J$ there exists a p_j such that $\bigoplus(r_1 \;\mathsf{after}\; \hat{\alpha}_j) \;\mathsf{cmp}\; p_j$. Let $p = \sum_{j \in J} \overline{\hat{\alpha}_j}.p_j$.

To prove (a), recall the client r_2' and consider the state $p \,||\, r_2'$. By construction p offers no action that can interact with the actions of r_2', and both p and r_2' are stable, thus $p \,||\, r_2'$ is stable. As $\checkmark \notin B$, we have that $r_2' \neg\mathsf{cmp}\, p$ and hence $r_2 \neg\mathsf{cmp}\, p$.

To prove (b), we define a suitable coinductive compliance. Let

$$R = R' \cup \mathsf{cmp} \;\text{ where }\qquad R' = \{\, (r', \sum_{j \in J} \overline{\hat{\alpha}}_j.p_j) \mid r_1 \Longrightarrow r', \bigoplus(r_1 \;\mathsf{after}\; \overline{\alpha_j}) \;\mathsf{cmp}\; p_j \,\}$$

By construction $r_1 \,R\, p$ and so it suffices to show that $R \subseteq \mathcal{C}(R)$. Fix a pair $r \,R\, p$, Definition 8 requires us to show that

1. if $p \Uparrow$ then $r \xrightarrow{\checkmark}$
2. if $p \,||\, r \overset{\tau}{\not\longrightarrow}$ then $r \xrightarrow{\checkmark}$
3. if $p \,||\, r \xrightarrow{\tau} p' \,||\, r'$ then $r' \,R\, p'$

Since $r \,R\, p$, either $(r \;\mathsf{cmp}\; p)$ or $r \,R'\, p$. In the first case, Definition 8 and $\mathsf{cmp} \subseteq R$ ensure that the pair at hand enjoys the three properties above. Now suppose that $r \,R'\, p$. By the definition of R', $r_1 \Longrightarrow r$ and $p = \sum_{j \in J} \overline{\hat{\alpha}_j}.p_j$, point (1) holds trivially for p converges. To prove point (2), suppose that $p \,||\, r \overset{\tau}{\not\longrightarrow}$. This means that $r \overset{\tau}{\not\longrightarrow}$ and that it does not engage in any visible actions a_j, meaning that $S_{\checkmark}(r) \in \mathsf{Acc}_{\not\checkmark}(r_1)$. Since none of the actions in $S_{\checkmark}(r)$ is one of the a_j for $j \in J$, this ensures that $\checkmark \in S_{\checkmark}(r)$, and thus $r \xrightarrow{\checkmark}$. Now we prove point (3). Suppose that $r \,||\, p \xrightarrow{\tau} r' \,||\, p'$. The argument depends on the rule used to infer the silent move at hand. As p is stable we have only two cases to discuss. If rule (P-CLI) was used then $r \xrightarrow{\tau} r'$, thus $p' = p$, and since $r_1 \Longrightarrow r'$ we have $r \,R'\, p$, and thus $r \,R\, p'$. If rule (P-SYNCH) was applied than note that the construction of p ensures that for some $j \in J$ we have $p' = p_j$ and $r' \in (r_1 \;\mathsf{after}\; a_j)$. We already know that $\bigoplus(r_1 \;\mathsf{after}\; a_j) \;\mathsf{cmp}\; p_j$, thus $r' \;\mathsf{cmp}\; p'$, and thus $r' \,R\, p'$. $\square$

Lemma 23. *Whenever r cmp p, $r \xrightarrow{a}$, and $p \xrightarrow{\bar{a}} p'$ then $a \in ua_{\mathsf{bhv}}(r)$.*

Proof. Since, hypothesis $r \xrightarrow{a}$ means that $r \overset{a}{\Longrightarrow}$, we must show that $\bigoplus(r \;\mathsf{after}\; a) \in \mathcal{U}^{\mathsf{bhv}}_{\mathsf{cmp}}$. By Lemma 13, this is equivalent to showing that $\bigoplus(r \;\mathsf{after}\; a) \in \mathcal{U}_{\mathsf{cmp}}$. Observe that from $r \;\mathsf{cmp}\; p$ for every $r' \in (r \;\mathsf{after}\; a)$ we have that $r' \;\mathsf{cmp}\; p'$. We have to exhibit some p'' such that $\bigoplus(r \;\mathsf{after}\; a) \;\mathsf{cmp}\; p''$.

If p' does not diverge, we choose it as our witness p''. We let

$$R = \{\, (\bigoplus(r \;\mathsf{after}\; a), p') \,\} \cup \mathsf{cmp}$$

Thanks to our observation above and the assumption that p' converges, it is routine work to check that the relation R is a coinductive compliance, and thus $\bigoplus(r \;\mathsf{after}\; a) \;\mathsf{cmp}\; p'$.

If p' diverges, then our observation above implies that for every r'' such that for some $r' \in (r \;\mathsf{after}\; a)$ $r' \Longrightarrow r''$ we have $r'' \xrightarrow{\checkmark}$. In this case our candidate p'' is 0, and we define

$$R = \{\, (\bigoplus(r \;\mathsf{after}\; a), 0) \,\} \cup \{\, (r'', 0) \mid \bigoplus(r \;\mathsf{after}\; a) \Longrightarrow r'' \,\} \cup \mathsf{cmp}$$

Also in this case it is routine work to check that R is a coinductive compliance. $\square$

Theorem 5. *The preorders $\sqsubseteq_{cmp}$ and $\preceq_{\mathsf{clt}}$ coincide.*

Proof. The set inclusion $\sqsubseteq_{\mathsf{cmp}} \subseteq \preceq_{\mathsf{clt}}$ follows from Lemma 15, Lemma 16, and Lemma 17. Thus we only have to prove the converse inclusion, $\preceq_{\mathsf{clt}} \subseteq \sqsubseteq_{\mathsf{cmp}}$. It is enough to show that the relation

$$R = \{\, (r, p) \in \mathsf{Proc} \times \mathsf{Proc} \mid \exists r_1 \in \mathsf{Proc}.\, r_1 \preceq_{\mathsf{clt}} r \text{ and } r_1 \;\mathsf{cmp}\; p \,\}$$

is a pre-fixed point of $\mathcal{G}$, that is $R \subseteq \mathcal{G}(R)$. Fix a pair $r \,R\, p$, Definition 8 requires us to show that

1. if $p \Uparrow$ then $r \xrightarrow{\checkmark}$
2. if $p \,||\, r \stackrel{\tau}{\not\longrightarrow}$ then $r \xrightarrow{\checkmark}$
3. if $p \,||\, r \xrightarrow{\tau} p' \,||\, r'$ then $(r', p') \in R$

By the definition of R, there exists a $r_1 \in \mathsf{Proc}$ such that $r_1 \preceq_{\mathsf{clt}} r$ and $r_1 \,\mathsf{cmp}\, p$. We have immediately that $r_1 \in \mathcal{U}_{\mathsf{cmp}}$.

To prove Point (1), suppose that $p \Uparrow$. For every r_1' such that $r_1 \Longrightarrow r_1'$ there exists the computation $p \,||\, r_1 \Longrightarrow p \,||\, r_1'$ and since $r_1 \,\mathsf{cmp}\, p$ we have that $r_1' \,\mathsf{cmp}\, p$. The assumption $p \Uparrow$ implies that $r_1' \xrightarrow{\checkmark}$. Since $r \Longrightarrow r$, Definition 11(2b) implies that $r \xrightarrow{\checkmark}$ as required.

To prove Point (2) suppose that $p \,||\, r \stackrel{\tau}{\not\longrightarrow}$; our aim is to show that $r \xrightarrow{\checkmark}$. As the composition $p \,||\, r$ is stable we have that $r \stackrel{\tau}{\not\longrightarrow}$, thus $\mathsf{Acc}_{\not\checkmark}(r) = \{ S_{\checkmark}(r) \}$. To prove that $\checkmark \in S_{\checkmark}(r)$ we argue by contradiction and assume that $\checkmark \not\in S_{\checkmark}(r)$. By $r_1 \in \mathcal{U}_{\mathsf{cmp}}$ and Definition 11(2c), we know that for some $A \in \mathsf{Acc}_{\not\checkmark}(r_1)$ we have $A \cap ua_{\mathsf{bhv}}(r_1) \subseteq S_{\checkmark}(r)$, that is there exists a r_1' such that $r_1 \Longrightarrow r_1' \stackrel{\tau}{\not\longrightarrow}$ and $A = S_{\checkmark}(r_1')$, so

$$S_{\checkmark}(r_1') \cap ua_{\mathsf{bhv}}(r_1) \subseteq S_{\checkmark}(r) \tag{B.1}$$

We prove that $p \,||\, r_1' \stackrel{\tau}{\not\longrightarrow}$. Suppose that $p \xrightarrow{a}$ for some action a. Since $p \,||\, r \stackrel{\tau}{\not\longrightarrow}$, it must be the case that $\overline{a} \not\in S_{\checkmark}(r)$, and by Eq. (B.1) this ensures that $\overline{a} \not\in S_{\checkmark}(r_1') \cap ua_{\mathsf{bhv}}(r_1')$. But we know already that $r_1' \,\mathsf{cmp}\, p$, and thus Lemma 23 ensures that if $r_1' \xrightarrow{\overline{a}}$ it must be the case that $\overline{a} \in ua_{\mathsf{bhv}}(r_1')$. It follows that $\overline{a} \not\in S_{\checkmark}(r_1')$, that is $r_1' \stackrel{\overline{a}}{\not\longrightarrow}$. It follows that $p \,||\, r_1' \stackrel{\tau}{\not\longrightarrow}$.

Now $r_1' \,\mathsf{cmp}\, p$ and $p \,||\, r_1' \stackrel{\tau}{\not\longrightarrow}$ implies that $r_1' \xrightarrow{\checkmark}$, that is $\checkmark \in S_{\checkmark}(r_1')$, and (B.1) above implies $\checkmark \in S_{\checkmark}(r)$, contradicting our assumption that $\checkmark \not\in S_{\checkmark}(r)$.

Point (3). Suppose that $p \,||\, r \xrightarrow{\tau} p' \,||\, r'$. Either $r' \not\in \mathcal{F}_{\mathsf{cmp}}$ or $r' \in \mathcal{F}_{\mathsf{cmp}}$. If $r' \not\in \mathcal{F}_{\mathsf{cmp}}$ then by definition $r' \,\mathsf{cmp}\, p$, and since $r' \preceq_{\mathsf{clt}} r'$ we have $r' \,R\, p'$. If $r' \in \mathcal{F}_{\mathsf{cmp}}$, the argument proceeds by case analysis on the rule used to infer the reduction at issue. If the reduction is due to (P-SVR) then $p \xrightarrow{\tau} p'$ and $r' = r$. On the one hand $r_1 \preceq_{\mathsf{clt}} r'$, on the other hand $r_1 \,\mathsf{cmp}\, p$ ensures $r_1 \,\mathsf{cmp}\, p'$. Thus, by definition of R, $r' \,R\, p'$. If the rule used is (P-CLT) then $r \xrightarrow{\tau} r'$ and $p = p'$. As $r_1 \preceq_{\mathsf{clt}} r$ Definition 11(1) implies $r_1 \preceq_{\mathsf{clt}} r'$, and hence $r_1 \,\mathsf{cmp}\, p$ implies that by definition $r' \,R\, p'$. If the reduction is due to (P-SYN) then $p \xrightarrow{a} p'$ and $r \xrightarrow{\overline{a}} r'$ for some $a \in \mathsf{Act}$. Since $r_1 \in \mathcal{U}_{\mathsf{cmp}}$ and by assumption $r' \in \mathcal{F}_{\mathsf{cmp}}$ Definition 11(2d) implies $\bigoplus(r_1 \,\mathsf{after}\, \overline{a}) \preceq_{\mathsf{clt}} \bigoplus(r \,\mathsf{after}\, \overline{a})$. Plainly $\bigoplus(r \,\mathsf{after}\, \overline{a}) \xrightarrow{\tau} r'$, Definition 11(1) ensures that $\bigoplus(r_1 \,\mathsf{after}\, \overline{a}) \preceq_{\mathsf{clt}} r'$. To show that $r' \,R\, p'$ it is enough to prove that $\bigoplus(r_1 \,\mathsf{after}\, \overline{a}) \,\mathsf{cmp}\, p'$. A coinductive compliance that witnesses this is the following relation,

$$R = R' \cup \mathsf{cmp} \quad \text{where} \qquad R' = \{ (r', p') \mid \bigoplus(r_1 \,\mathsf{after}\, \overline{a}) \Longrightarrow r', p \Longrightarrow p' \} \qquad \square$$

References

[1] J.R. Andersen, N. Andersen, S. Enevoldsen, M.M. Hansen, K.G. Larsen, S.R. Olesen, J. Srba, J. Wortmann, CAAL: concurrency workbench, Aalborg edition, in: ICTAC, 2015.
[2] F. Barbanera, F. de'Liguoro, Two notions of sub-behaviour for session-based client/server systems, in: PPDP, 2010.
[3] G. Bernardi, Behavioural Equivalences for Web Services, PhD Thesis, TCD, 2013.
[4] G. Bernardi, M. Hennessy, Compliance and testing preorders differ, in: BEAT2, 2013.
[5] G. Bernardi, M. Hennessy, Mutually testing processes, Log. Methods Comput. Sci. 11 (2) (2015).
[6] G. Bernardi, M. Hennessy, Modelling session types using contracts, Math. Struct. Comput. Sci. 26 (2016) 3.
[7] G. Bernardi, M. Hennessy, Using higher-order contracts to model session types, Log. Methods Comput. Sci. 12 (2) (2016).
[8] Giovanni Bernardi, Adrian Francalanza, Full-abstraction for must testing preorders, in: COORDINATION, 2017, extended abstract.
[9] M. Bravetti, G. Zavattaro, A foundational theory of contracts for multi-party service composition, Fundam. Inform. 89 (4) (2008).
[10] G. Castagna, N. Gesbert, L. Padovani, A theory of contracts for web services, ACM Trans. Program. Lang. Syst. 31 (5) (2009).
[11] A. Cerone, M. Hennessy, Process behaviour: formulae vs. tests, in: EXPRESS, 2010.
[12] E.M. Clarke, H. Veith, Counterexamples revisited: principles, algorithms, applications, in: Verification: Theory and Practice, 2003.
[13] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, 3rd edition, The MIT Press, 2009.
[14] R. De Nicola, M. Hennessy, Testing equivalences for processes, Theor. Comput. Sci. 34 (1–2) (1984).
[15] A. Francalanza, A theory of monitors, in: FoSSaCS, in: LNCS, vol. 9634, 2016, pp. 145–161.
[16] A. Francalanza, Consistently-Detecting Monitors, in: CONCUR 2017, vol. 85, 2017, pp. 8:1–8:19.
[17] S.J. Gay, M. Hole, Subtyping for session types in the pi calculus, Acta Inform. 42 (2–3) (2005).
[18] R. Gorrieri, C. Versari, Introduction to Concurrency Theory – Transition Systems and CCS, Springer, 2015.
[19] M. Hennessy, Algebraic Theory of Processes, 1988.
[20] D.E. Knuth, The Art of Computer Programming, Vol. 1: Fundamental Algorithms, 3rd. edition, Addison Wesley Longman Publishing Co., Inc., 1997.
[21] D. König, Über eine Schlussweise aus dem Endlichen ins Unendliche, Acta Litter. Sci. Szeged 3 (1927).
[22] C. Laneve, L. Padovani, The must preorder revisited, in: CONCUR, 2007.
[23] Q. Luo, F. Hariri, L. Eloussi, D. Marinov, An empirical analysis of flaky tests, in: FSE, 2014.
[24] P. Marinescu, P. Hosek, C. Cadar, Covrig: a framework for the analysis of code, test, and coverage evolution in real software, in: ISSTA, 2014.
[25] A. Martens, Analyzing Web service based business processes, in: FASE, 2005.
[26] A.M. Memon, M.B. Cohen, Automated testing of GUI applications: models, tools, and controlling flakiness, in: ICSE, 2013.

[27] R. Milner, Communication and Concurrency, Prentice–Hall, 1989.
[28] A.J. Mooij, C. Stahl, M. Voorhoeve, Relating fair testing and accordance for service replaceability, J. Log. Algebraic Program. 79 (3–5) (2010).
[29] L. Padovani, Contract-based discovery of web services modulo simple orchestrators, Theor. Comput. Sci. 411 (37) (2010).
[30] L. Padovani, Fair subtyping for multi-party session types, Math. Struct. Comput. Sci. 26 (3) (2016).
[31] B. Pierce, Types and Programming Languages, 2002.
[32] A. Rensink, W. Vogler, Fair testing, Inf. Comput. 205 (2) (2007).
[33] C. Spaccasassi, Language Support for Communicating Transactions, PhD Thesis, TCD, 2015.
[34] D. Weinberg, Efficient controllability analysis of open nets, in: WS-FM, 2009.
[35] G. Winskel, The Formal Semantics of Programming Languages: An Introduction, 1993.