Full-abstraction for Must Testing Preorders (Extended Abstract)

Giovanni Bernardi and Adrian Francalanza

[1] Université Paris-Diderot, IRIF; gio@irif.fr
[2] University of Malta, Msida; adrian.francalanza@um.edu.mt

Abstract. The client must preorder relates tests (clients) instead of processes (servers). The existing characterisation of this preorder is unsatisfactory for it relies 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 for finite-branching LTSs, and give a sound and complete decision procedure for it. We also provide a novel coinductive characterisation of the client preorder, which we use to argue that the preorder is decidable, thus positively answering the question opened in [5,3].

1 Introduction

The standard testing theory of De Nicola–Hennessy [11,14] has recently been employed to provide theoretical foundations for web-services [8,24] (where processes denote servers). To better fit that setting, in [5] this theory has been enriched with preorders for clients (tests) and peers (where both interacting parties mutually satisfy one another). Client preorders also tie testing theory with session type theory, as is outlined in [2]: they are instrumental in defining semantic models of the Gay & Hole subtyping [13] for first-order session types [3, Theorem 6.3.4] and [4, Theorem 5.2].

The testing preorders for clients and peers are *contextual* preorders, defined by comparing the capacity of either being satisfied by servers or the capacity of peers to mutually satisfy one another. This paper focuses on the client preorder due to the must testing relation [11,14]: a client r_2 is better than a client r_1, denoted $r_1 \sqsubseteq_{\mathsf{clt}} r_2$, whenever *every* server p that must pass r_1 also must pass r_2. Although this definition is easy to understand, it suffers from the endemic universal quantification over contexts (servers) and, by itself, does not give any effective proof method to determine pairs in the preorder. To solve this problem, contextual preorders usually come equipped with *behavioural characterisations* that avoid universal context quantification thereby facilitating reasoning. In [5] the authors develop such characterisations for the client and the peer must preorders; these preorders are however *not* fully-abstract, for they are defined modulo *usable* clients, *i.e.*, clients that are satisfied by *some* server.

Usability is a pivotal notion that appears frequently in the literature of process calculi and web-service foundations, *cf.* viability in [17,25] and controllability in [7,23], and has already been studied, albeit for restricted or different settings, in [17,24,6,5,25]. In general though, the characterisation of usability is problematic, for solving it requires finding the conditions under which one can either (a) construct a server p that satisfies

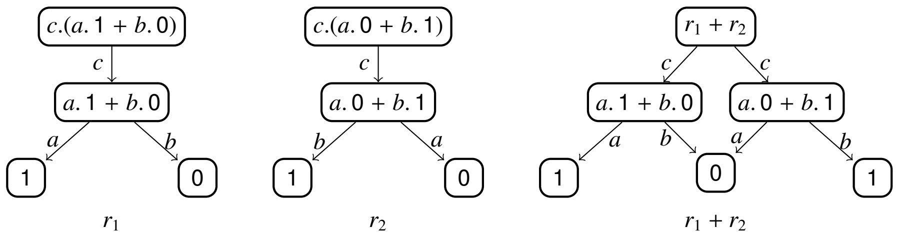

Fig. 1. LTS depictions of the behaviours described in Eq. (1)

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, the approach in (a) is complicated because client usability is *not* compositional. For instance consider the following clients, whose behaviours are depicted in Figure 1:

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

where $\mathsf{1}$ denotes satisfaction (success). Both clients are usable, since r_1 is satisfied by the server $\overline{c}.\overline{a}.\,\mathsf{0}$, and r_2 is satisfied by server $\overline{c}.\overline{b}.\,\mathsf{0}$. However, their composition $r_1 + r_2$ is *not* a usable client, *i.e.*, $p \;\mathsf{m\not{u}st}\; r_1 + r_2$ 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.*, $\overline{c}.(\overline{a}.\,\mathsf{0} + \overline{b}.\,\mathsf{0})$ does not satisfy $r_1 + r_2$). A compositional analysis is even more unwieldy for recursive tests. For instance, the client $\mu x.(c.(a.\,\mathsf{1} + b.x) + c.(a.\,\mathsf{0} + b.\,\mathsf{1}))$ is not usable because of the non-determinism analogous to $r_1 + r_2$, and the unsuccessful computations along the infinite trace of interactions $(c.b)^*$; this argument works because infinite unsuccessful computations are catastrophic *wrt.* must testing.

This paper presents a sound and complete characterisation for usable clients with finite-branching LTSs. 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. We go a step further and use this characterisation to develop a novel *coinductive* and fully-abstract characterisation of $\underset{\sim}{\sqsubset}_{\mathsf{clt}}$, which we find easier to use than the one of [5] when proving inequalities involving recursive clients. This coinductive characterisation turns out to be informed by our study on usability, and differs from related coinductive characterisations for the server preorder [17,24] in a number of respects. Finally, our inductive definition for usable clients also provides deeper insights into the original client preorder of [5]: we show that limiting contexts to servers offering only *finite* interactions preserves the discriminating power of the original preorder. Our contributions are:

- a fully-abstract characterisation of usable clients, Theorem 2;
- a coinductive, fully-abstract characterisation of the client preorder $\underset{\sim}{\sqsubset}_{\mathsf{clt}}$, Theorem 5;
- a contextual preorder $\underset{\sim}{\sqsubset}^{f}_{\mathsf{clt}}$ that is equivalent to $\underset{\sim}{\sqsubset}_{\mathsf{clt}}$ but relies only on non-recursive contexts Theorem 6;
- decidability results for usable clients and the client preorder, Theorem 7.

The solutions devised here addressing client usability are directly relevant to controllability issues in service-oriented architectures [20,29]. 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 directed the attention to non-deterministic (or *flaky*) tests. Such tests arise frequently in practice and their impact on software development has been the subject of various studies [21,19,18]. By some measures, $\approx 4.56\%$ of test failures of the TAP (Test Anything Protocol) system at Google are caused by flaky tests [18]. We believe that our concepts, models and procedures can be extended to such testing methodologies to analyse detrimental non-deterministic behaviour arising in test suites, thereby reducing the gap between empirical practices and theory.

Structure of the paper: Section 2 outlines the preliminaries for client must testing. Section 3 tackles client usability and gives a fully-abstract definition for it. Section 4 uses this result to give a coinductive characterisation for client preorders. In Section 5 we present expressiveness results for servers with finite interactions together with decidability results for client usability and the client testing preorder. Section 6 concludes.

2 Preliminaries

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* of an experiment. To emphasise their distinctness, we use $\alpha \in \mathsf{Act}_\tau$ to denote $\mathsf{Act} \cup \{\tau\}$, and similarly for $\lambda \in \mathsf{Act}_{\tau\checkmark}$. 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 $p \in \mathsf{Proc}$ and for all $\lambda \in \mathsf{Act}_{\tau\checkmark}$, the set $\{\, q \mid p \xrightarrow{\lambda} q \,\}$ is finite. For $s \in (\mathsf{Act}_\checkmark)^\star$ we also have the standard weak transitions, $p \overset{s}{\Longrightarrow} q$, defined by *ignoring* the occurrences of τs.

We limit ourselves to finite-branching LTSs. Whenever sufficient, we describe such LTSs using a version of CCS with recursion [22] and augmented with a *success* operator, denoted as $\mathsf{1}$. The syntax of this language is depicted in Figure 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 *resp.* 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. The transition relation $p \xrightarrow{\lambda} q$ between terms of the language is the least one determined by the (standard) rules in Figure 2. 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.

To model the interactions taking place between the server and the client contracts, we use the standard binary composition of contracts, $p \parallel r$, whose operational semantics

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

Semantics

$$\frac{}{\mathsf{1} \xrightarrow{\checkmark} \mathsf{0}}\ (\text{A-OK}) \qquad \frac{}{\alpha.p \xrightarrow{\alpha} p}\ (\text{A-PRE}) \qquad \frac{}{\mu x.p \xrightarrow{\tau} p\{^{\mu x.p}/_{x}\}}\ (\text{A-UNFOLD})$$

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

Contract Composition Semantics

$$\frac{p \xrightarrow{\lambda} p'}{p \parallel r \xrightarrow{\lambda} p' \parallel r}\ (\text{P-SRV}) \qquad \frac{r \xrightarrow{\lambda} r'}{p \parallel r \xrightarrow{\lambda} p \parallel r'}\ (\text{P-CLI}) \qquad \frac{p \xrightarrow{a} p' \quad r \xrightarrow{\overline{a}} r'}{p \parallel r \xrightarrow{\tau} p' \parallel r'}\ (\text{P-SYN})$$

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

is given in Figure 2. A *computation* consists of sequence of τ actions of the form

$$p \parallel r = p_0 \parallel r_0 \xrightarrow{\tau} p_1 \parallel r_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} p_k \parallel r_k \xrightarrow{\tau} \ldots \tag{2}$$

It is *maximal* if it is infinite, or whenever $p_n \parallel r_n$ is the last state then $p_n \parallel r_n \not\xrightarrow{\tau}$. We say (2) is *client-successful* if there exists some $k \geq 0$ such that $r_k \xrightarrow{\checkmark}$.

Definition 1 (Client Testing preorder [5]). *We write p must r if every maximal computation from $p \parallel r$ is* client-successful*, and write $r_1 \mathrel{\underset{\sim}{\sqsubset}}_{\mathsf{clt}} r_2$ 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 $\mathrel{\underset{\sim}{\sqsubset}}_{\mathsf{clt}}$. One way of surmounting this is by defining alternative characterisations for $\mathrel{\underset{\sim}{\sqsubset}}_{\mathsf{clt}}$ of Definition 1, that come equipped with practical proof methods.

2.1 Characterising the client preorder

In [5, Def. 3.10, pg. 9], an alternative characterisation for the preorder $\mathrel{\underset{\sim}{\sqsubset}}_{\mathsf{clt}}$ is given and proven to be sound and complete. We recall this characterisation, restating the *resp.* notation. The alternative characterisation relies on *unsuccessful* traces: $r \overset{s}{\Longrightarrow}_{\not\checkmark} r'$ means that r may weakly perform the trace of 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) $r \not\xrightarrow{\checkmark}$ implies $r \overset{\varepsilon}{\Longrightarrow}_{\not\checkmark} r$, and (b) if $r'' \overset{s}{\Longrightarrow}_{\not\checkmark} r'$ and $r \not\xrightarrow{\checkmark}$ 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 set of r after s, are defined as

$$\mathsf{Acc}_{\not\checkmark}(r, s) = \{\, S(r') \mid r \overset{s}{\Longrightarrow}_{\not\checkmark} r' \not\xrightarrow{\tau} \,\} \tag{3}$$

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 [3,5,24]. In the sequel, we shall also use $r \xrightarrow{\alpha}_{\not\checkmark} r'$ whenever $r \xrightarrow{\alpha} r'$, $r \overset{\checkmark}{\not\longrightarrow}$ and $r' \overset{\checkmark}{\not\longrightarrow}$ hold.

Example 1. For client $r_3 = \tau.(\mathsf{1} + \tau.\mathsf{0})$ we have $\mathsf{Acc}_{\not\checkmark}(r_3, \epsilon) = \emptyset$, but for $r_3' = r_3 + \tau.\mathsf{0}$ we have $\mathsf{Acc}_{\not\checkmark}(r_3', \epsilon) = \{\, \emptyset \,\}$. We also have $\mathsf{Acc}_{\not\checkmark}(r_3'', \epsilon) = \emptyset$ for $r_3'' = r_3 + \mu x.x$. ■

Note 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'.

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

Example 2. Recall clients r_1 and r_2 from (1) in Section 1. We show that despite being individually usable, the sum of these clients is not: $p \mathrel{\mathsf{m\not{u}st}} r_1 + r_2$ for *every* p. Fix a process p. If p does not offer an interaction on $\overline{c}$, then, plainly, $p \mathrel{\mathsf{m\not{u}st}} r_1 + r_2$. Suppose that $p \xrightarrow{\overline{c}} p'$; to prove $p \mathrel{\mathsf{m\not{u}st}} r_1 + r_2$, it suffices to show that there exists a client r reached by $r_1 + r_2$ by performing action c (*i.e.*, $r \in \{\, a.\mathsf{1} + b.\mathsf{0}, a.\mathsf{0} + b.\mathsf{1} \,\}$) such that $p' \mathrel{\mathsf{m\not{u}st}} r$. Indeed, for $r = a.\mathsf{1} + b.\mathsf{0}$, if $p' \mathrel{\mathsf{must}} r$ implies p' has to interact on a and *not* on b, but then such a p' does not satisfy the derivative $r = a.\mathsf{0} + b.\mathsf{1}$, *i.e.*, $p' \mathrel{\mathsf{m\not{u}st}} r$ (because the composition $p' \parallel r$ is stable but *not* client-successful). Using a symmetric argument we deduce that if $p' \mathrel{\mathsf{must}} a.\mathsf{0} + b.\mathsf{1}$ then $p' \mathrel{\mathsf{m\not{u}st}} a.\mathsf{1} + b.\mathsf{0}$, and thus 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 $\mu x.(c.(a.\mathsf{1} + b.x) + c.b.\mathsf{1})$ from Section 1 is unusable for similar reasons, the analysis being more involved due to infinite computations. ■

We let $(r \mathrel{\mathsf{after}_{\not\checkmark}} s) = \{\, r' \mid r \overset{s}{\Longrightarrow}_{\not\checkmark} r' \,\}$, and call the set $(r \mathrel{\mathsf{after}_{\not\checkmark}} s)$ the *residuals* of r after the *unsuccessful* trace s. We extend the notion of usability and say that r is *usable along* an unsuccessful trace s whenever $r \mathrel{usbl_{\not\checkmark}} s$, which is the least predicate satisfying the conditions (a) $r \mathrel{usbl_{\not\checkmark}} \varepsilon$ whenever $r \in \mathcal{U}$, and (b) $r \mathrel{usbl_{\not\checkmark}} as$ whenever (i) $r \in \mathcal{U}$ and (ii) if $r \overset{a}{\Longrightarrow}_{\not\checkmark}$ then $\bigoplus(r \mathrel{\mathsf{after}_{\not\checkmark}} a) \mathrel{usbl_{\not\checkmark}} s$. If $r \mathrel{usbl_{\not\checkmark}} s$, any state reachable from r by performing any unsuccessful subsequence of s is usable [5]. Finally, let $ua_{\mathsf{clt}}(r, s) = \{\, a \in \mathsf{Act} \mid r \overset{sa}{\Longrightarrow}_{\not\checkmark}$ implies $r \mathrel{usbl_{\not\checkmark}} sa \,\}$ denote all the usable actions for a client r after the unsuccessful trace s.

Definition 3 (Semantic client-preorder). *Let* $r_1 \precsim_{\mathsf{clt}} r_2$ *if, for every* $s \in \mathsf{Act}^\star$ *such that* $r_1 \mathrel{usbl_{\not\checkmark}} s$*, we have (i)* $r_2 \mathrel{usbl_{\not\checkmark}} s$*, (ii) for every* $B \in \mathsf{Acc}_{\not\checkmark}(r_2, s)$ *there exists a* $A \in \mathsf{Acc}_{\not\checkmark}(r_1, s)$ *such that* $A \cap ua_{\mathsf{clt}}(r_1, s) \subseteq B$*, (iii)* $r_2 \overset{s}{\Longrightarrow}_{\not\checkmark}$ *implies* $r_1 \overset{s}{\Longrightarrow}_{\not\checkmark}$. ■

Theorem 1. *In any finite branching LTS,* $r_1 \sqsubseteq_{\mathsf{clt}} r_2$ *if and only* $r_1 \precsim_{\mathsf{clt}} r_2$.

Proof. Follows from [5, Theorem 3.13] and König's Infinity Lemma (see Lemma 8).

Definition 3 enjoys a few pleasing properties and, through Theorem 1, sheds light on behavioural properties of clients related by $\sqsubsim_{\mathsf{clt}}$. Concretely, it shares a similar structure to well-studied characterisations of the (standard) must-testing preorder of [11,14], where process convergence is replaced by client usability, and traces and acceptance sets are replaced by their unsuccessful counterparts (modulo usable actions). Unfortunately, Definition 3 has a major drawback: it is parametric *wrt.* the set of usable clients $\mathcal{U}$ (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 $\sqsubsim_{\mathsf{clt}}$ on it.

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 $\mathcal{U}$ (Proposition 1). We use these predicates to construct a set $\mathcal{U}_{\mathsf{bhv}}$ that coincides with $\mathcal{U}$ (Theorem 2); 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 $r_1 \xrightarrow{\checkmark}$.* ■

Recalling Eq. (3), let $\mathsf{Acc}_{\not\checkmark}(r) = \mathsf{Acc}_{\not\checkmark}(r, \varepsilon)$. Proposition 1 crystallises the characteristic properties of usable clients, providing a blue print for our alternative definition Definition 5. Instead of giving a direct proof of this proposition, we obtain it indirectly as consequence of our other results.

Proposition 1. *For every $r \in \mathsf{Proc}$, $r \in \mathcal{U}$ if and only if*

1. $r \Downarrow_{\checkmark}$, and
2. if $A \in \mathsf{Acc}_{\not\checkmark}(r)$, then there exists $a \in A. (r \overset{a}{\Longrightarrow}_{\not\checkmark}$ implies $\bigoplus(r \mathrel{\mathsf{after}_{\not\checkmark}} a) \in \mathcal{U})$. □

The proposition above states 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 another state $r''$where r'' is certainly usable.

Example 3. We use Proposition 1 to discuss the (non) usability of clients from previous example. Recall $r_3 = \tau.(1 + \tau.0)$, $r'_3 = r_3 + \tau.0$ and $r''_3 = r_3 + \mu x.x$ from Example 1. Since we have $r_3 \Downarrow_{\checkmark}$ and $\mathsf{Acc}_{\not\checkmark}(r_3) = \emptyset$, r_3 satisfies both condition of Proposition 1, with the second one being trivially true. As a consequence r_3 is usable, and indeed $0 \mathrel{\mathsf{must}} r_3$. On the contrary, we have $\mathsf{Acc}_{\not\checkmark}(r'_3) = \{\emptyset\}$, thus r'_3 violates Proposition 1(2) and thus r'_3 is unusable. Client r''_3 is unusable as well, but violates Proposition 1(1) instead. Conversely, client $r'''_3 = r_3 + \tau.(1 + \mu x.x)$ satisfies both conditions of Proposition 1, and it is usable. For instance, $0 \mathrel{\mathsf{must}} r'''_3$.

A more involved client is $r_1 + r_2$ from Example 2. There we proved that $r_1 + r_2 \notin \mathcal{U}$, and indeed $r_1 + r_2$ does not satisfy Proposition 1(2). This is true because $\mathsf{Acc}_{\not\checkmark}(r_1 + r_2) = \{\{c\}\}$, and $r' \notin \mathcal{U}$, where

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

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

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. *if $A \in \mathsf{Acc}_{\not\checkmark}(r)$, then there exists an $a \in A.(r \overset{a}{\Longrightarrow}_{\not\checkmark}$ 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 $\langle \mathcal{P}(\mathsf{Proc}), \subseteq \rangle$ (Lemma 7), thus Kleene fixed point theorem [30, Theorem 5.11] ensures that $\mu x.\mathcal{F}(x)$ (the least fix-point of $\mathcal{F}$) exists and is equal to $\bigcup_{n=0}^{\infty} \mathcal{F}^n(\emptyset)$ where $\mathcal{F}^0(S) = S$ and $\mathcal{F}^{n+1}(S) = \mathcal{F}(\mathcal{F}^n(S))$.

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

Lemma 1. *For every $n \in \mathbb{N}$ and $r \in \mathsf{Proc}$, $r \in \mathcal{F}^n(\emptyset)$ implies that there exists a non-recursive server p such that p must r.* □

An inductive argument is used to prove that $\mathcal{U}_{\mathsf{bhv}}$ is complete *wrt.* $\mathcal{U}$, 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 $r \parallel p$ 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 $r = \mu x.a.x + b.\mathsf{1}$, we have $\mathsf{itr}(r, \overline{a}.\overline{a}.\overline{b}.\mathsf{0}) = 3$, but $\mathsf{itr}(r, \mu x.\overline{a}.x + \overline{b}.\mathsf{0}) = \infty$.

Lemma 2. *Let T be a tree with root v. If T is finite branching and it has a finite number of nodes, then the number of paths $v \longrightarrow \ldots$ is finite.* □

Lemma 3. *In a finite branching LTS, p must r implies the number $\mathsf{itr}(r, p)$ is finite.*

Proof. If p must r, *every* $c \in MC(r, p)$ reaches a successful state after a *finite* number of reductions. Since the number of interactions is not more than the number of reductions:

$$\text{for every } c \in MC(r, p).\ \#\mathsf{itr}(c) \in \mathbb{N} \tag{4}$$

A set of successful computations from $r \parallel p$, *e.g.*, $MC(r, p)$, may also be seen as a *computation tree*, where common prefixes reach the same node in the tree. In general, such a tree may have infinite depth. Consider the computation tree T obtained by *truncating* all the maximal computations of $r \parallel p$ at their *first* successful state, and let $TMC(r, p)$ be the set of all the computations obtained this way. It follows that

$$\{\#\mathsf{itr}(c) \mid c \in MC(r, p)\} = \{\#\mathsf{itr}(c) \mid c \in TMC(r, p)\} \tag{5}$$

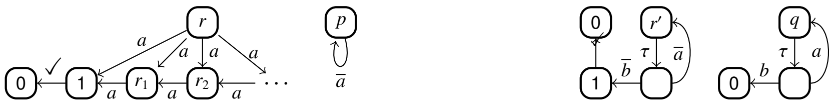

Fig. 3. Servers and clients to discuss the hypothesis in Lemma 3

From $\mathsf{itr}(r, p) = \max\{\, \#\mathsf{itr}(c) \mid c \in MC(r, p)\,\}$, (4) and (5) we know that that $\mathsf{itr}(r, p)$ is finite if the set $\{\, c \mid c \in TMC(r, p)\,\}$ is finite. This will follow from Lemma 2 if we prove that the tree T has a finite number of nodes. By the contrapositive of König's Lemma [16,15] (restated in Lemma 8), since every node in the tree T above is finitely branching, and there are no infinite paths, then T necessarily contains a *finite* number of nodes. By Lemma 2, $\{\, c \mid c \in TMC(r, p)\,\}$ must also be finite, and hence we can put a (finite) 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 3 is false. To see why, consider the infinite branching client r and the server p depicted in Figure 3. Since r engages in *finite* sequences of a actions which are unbounded in size, and the p offers any number of interactions on action $\overline{a}$, we have that p must r, but the set $MC(r, p)$ contains an infinite amount of computations, and the number $\mathsf{itr}(r, p)$ is not finite. Dually, even if the LTS of a composition $r \parallel p$ is finite branching and finite state, it is necessary that p must r for $\mathsf{itr}(r, p)$ to be finite. Lemma 3 lets us associate a rank to every usable client r, defined as $rank(r) = \min\{\, \mathsf{itr}(r, p) \mid p \ \mathsf{must}\ r\,\}$. 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,[3] where we can stratify $\mathcal{U}$ as follows:

$$\mathcal{U} = \textstyle\bigcup_{i \in \mathbb{N}} \mathcal{U}^i, \qquad \text{where } \mathcal{U}^i = \{\, r \in \mathsf{Proc} \mid rank(r) = i\,\} \tag{6}$$

Lemma 4. *For every $i \in \mathbb{N}$, $r \in \mathcal{U}^i$ implies $r \in \mathcal{F}(\mathcal{F}^j(\emptyset))$ for some $j \le i$.* □

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

Theorem 2 (Full-abstraction usability). *The sets $\mathcal{U}$ and $\mathcal{U}_{\mathsf{bhv}}$ coincide.*

Proof. To show $\mathcal{U} \subseteq \mathcal{U}_{\mathsf{bhv}}$, pick an $r \in \mathcal{U}$. By (6), $r \in \mathcal{U}^i$ for some $i \in \mathbb{N}$, and by Lemma 4 we obtain $r \in \mathcal{F}^j(\emptyset) \subseteq \mathcal{U}_{\mathsf{bhv}}$ for some $j \in \mathbb{N}^+$. To show $\mathcal{U}_{\mathsf{bhv}} \subseteq \mathcal{U}$, pick an $r \in \mathcal{U}_{\mathsf{bhv}}$. Definition 5 ensures that $\mathcal{U}_{\mathsf{bhv}} \subseteq \bigcup_{n=0}^{\infty} \mathcal{F}^n(\emptyset)$, thus $r \in \mathcal{F}^n(\emptyset)$ for some $n \in \mathbb{N}$. Lemma 1 implies that $r \in \mathcal{U}$. The reasoning applies to any $r \in \mathcal{U}_{\mathsf{bhv}}$, thus $\mathcal{U}_{\mathsf{bhv}} \subseteq \mathcal{U}$. □

4 The client preorder revisited

By combining the definition of $\precsim_{\mathsf{clt}}$ with $\mathcal{U}_{\mathsf{bhv}}$ of Definition 5, Theorem 2 yields a fully-abstract characterisation of the client preorder $\sqsubseteq_{\mathsf{clt}}$. In general, however, this characterisation still requires us to consider an infinite number of (unsuccessful) traces to

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

establish client inequality. In this section, we put forth a novel coinductive definition for the client preorder and exploit the finite-branching property of the LTS to show that this definition characterises the contextual preorder $\underset{\sim}{\sqsubset}_{\mathsf{clt}}$, Theorem 5. We also argue that this new characterisation is easier to use in practice than Definition 3, a claim that is substantiated by showing how this coinductive preorder can be used to prove the second result in this section, namely that servers offering a *finite* amount of interactions are sufficient and necessary to distinguish clients, Theorem 6. Subsequently, in Theorem 7, we also show that the coinductive preorder is decidable for our client language.

Example 4. The use of $\lesssim_{\mathsf{clt}}$ is hindered, in practice, by the universal quantification over traces in its definition. Consider, for instance, clients r_4 and r_5,

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

where $r_3'' = (\tau.(\mathsf{1} + \tau.\mathsf{0})) + \mu x.x$ from Example 1. One way to prove $r_4 \underset{\sim}{\sqsubset}_{\mathsf{clt}} r_5$ amounts in showing that $r_4 \lesssim_{\mathsf{clt}} r_5$, even though this task is far from obvious. Concretely, the definition of $\lesssim_{\mathsf{clt}}$ requires us to show that for *every* trace $s \in \mathsf{Act}^\star$ where $r_4\ \mathit{usbl}_{\not\checkmark}\ s$ holds, clauses (i), (ii) and (iii) of Definition 3 also hold. In this case, there are an *infinite* number of such 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 $s = b^n$, but also (unsuccessful) traces that r_4 *cannot* perform (which trivially imply $r_4\ \mathit{usbl}_{\not\checkmark}\ s$ according to the definition in Section 2.1), such as $s = d(b^n)$, $s = (db)^n$ or $s = (ac)^n$.

The definition of $r_4\ \mathit{usbl}_{\not\checkmark}\ s$ does however rule out a number of traces to consider, and Definition 5 helps us with this analysis. For instance, for $s = a$, we have $\neg(r_4\ \mathit{usbl}_{\not\checkmark}\ a)$ because $\bigoplus(r_4 \mathrel{\mathsf{after}_{\not\checkmark}} a) = (\tau.\mathsf{1} + \tau.r_3'' + \tau.\mathsf{0} + \tau.\mu x.x)$ and, by using similar reasoning to that in Example 3 for r_3'', we know that $\neg((r_4 \mathrel{\mathsf{after}_{\not\checkmark}} a) \Downarrow_{\checkmark})$ which implies $\bigoplus(r_4 \mathrel{\mathsf{after}_{\not\checkmark}} a) \notin \mathcal{U}_{\mathsf{bhv}}$ and, by Theorem 2, we have $\bigoplus(r_4 \mathrel{\mathsf{after}_{\not\checkmark}} a) \notin \mathcal{U}$. □

To overcome the problems outlined in Example 4, we identify three properties of the preorder $\underset{\sim}{\sqsubset}_{\mathsf{clt}}$, stated in Lemma 5, which partly motivate the conditions defining the transfer function $\mathcal{G}$ in Definition 6. Conditions (ii) and (iii) are explained in greater detail as discussions to points (2) and (3c) of Definition 6 below.

Lemma 5. *$r_1 \underset{\sim}{\sqsubset}_{\mathsf{clt}} r_2$ implies (i) if $r_2 \xrightarrow{\tau}_{\not\checkmark} r_2'$ then $r_1 \underset{\sim}{\sqsubset}_{\mathsf{clt}} r_2'$; (ii) if $r_2 \overset{\checkmark}{\not\longrightarrow}$ then $r_1 \overset{\checkmark}{\not\longrightarrow}$ (iii) if $r_2 \xrightarrow{a}_{\not\checkmark}$ then $(r_1 \overset{a}{\Longrightarrow}_{\not\checkmark}$ and $\bigoplus(r_1 \mathrel{\mathsf{after}_{\not\checkmark}} a) \underset{\sim}{\sqsubset}_{\mathsf{clt}} \bigoplus(r_2 \mathrel{\mathsf{after}_{\not\checkmark}} a))$* □

Definition 6. *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 $A \cap \mathit{ua}_{\mathsf{bhv}}(r_1) \subseteq B$*
   - *(c) if $r_2 \xrightarrow{a}_{\not\checkmark}$ then $(r_1 \overset{a}{\Longrightarrow}_{\not\checkmark}$ and $\bigoplus(r_1 \mathrel{\mathsf{after}_{\not\checkmark}} a)\ R\ \bigoplus(r_2 \mathrel{\mathsf{after}_{\not\checkmark}} a))$*

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* $\preccurlyeq_{\mathsf{clt}} = \nu x.\mathcal{G}(x)$ *where* $\nu x.\mathcal{G}(x)$ *denotes the greatest fixpoint of* $\mathcal{G}$*. The function* $\mathcal{G}$ *is monotone over the complete lattice* $\langle \mathcal{P}(\mathsf{Proc} \times \mathsf{Proc}), \subseteq \rangle$ *(Lemma 11), and thus* $\nu x.\mathcal{G}(x)$ *exists.* □

The definition of $\mathcal{G}$ follows a similar structure to that of the *resp.* definitions that coinductively characterise the must preorder for servers [17,24]. Definition 6, 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 6 is condition (2). The next example elucidates why such a condition is necessary for $\preccurlyeq_{\mathsf{clt}}$ to be sound.

Counterexample 3 *Let* $\mathcal{G}_{\mathrm{bad}}$ *be defined as* $\mathcal{G}$ *in Definition 6, but without part (2). In this case, we prove that the pair of clients* $(\mathit{1}, \tau.\,\mathit{1})$ *is contained in the greatest fixed point of* $\mathcal{G}_{\mathrm{bad}}$*, and then proceed to show that this pair is* not *contained in* $\underset{\sim}{\sqsubset}_{\mathsf{clt}}$*. Let* $R = \{\, (\mathit{1}, \tau.\,\mathit{1}) \,\}$*. It follows that* $R \subseteq \mathcal{G}_{\mathrm{bad}}(R)$ *if all the conditions for* $\mathcal{G}_{\mathrm{bad}}$ *are satisfied: condition (1) in is trivially true, condition (3a) is true because* $\mathit{0}\ \mathsf{must}\ \mathit{1}$ *and* $\mathit{0}\ \mathsf{must}\ \tau.\,\mathit{1}$*, condition (3b) holds trivially because* $\mathsf{Acc}_{\not\checkmark}(\tau.\,\mathit{1}) = \emptyset$*, whereas condition (3c) is satisfied because* $\tau.\,\mathit{1}$ *does not perform any strong actions. It therefore follows that* $(\mathit{1}, \tau.\,\mathit{1}) \in \mu x.\mathcal{G}_{\mathrm{bad}}(x)$*. Contrarily,* $\mathit{1} \not\underset{\sim}{\sqsubset}_{\mathsf{clt}} \tau.\,\mathit{1}$ *because the divergent server* τ^{∞} *distinguishes between the two clients: whereas* $\tau^{\infty}\ \mathsf{must}\ \mathit{1}$ *since the client succeeds immediately, we have* $\tau^{\infty}\ \mathsf{m\not{u}st}\ \tau.\,\mathit{1}$ *because the composition* $\tau.\,\mathit{1} \parallel \tau^{\infty}$ *has an infinite unsuccessful computation due to the divergence of* τ^{∞}*.* ■

A more fundamental difference between Definition 6 and the coinductive server preorders in [17,24] is that, in Definition 6(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 (1) from Section 1). Using the standard weaker condition makes the preorder $\preccurlyeq_{\mathsf{clt}}$ unsound *wrt.* $\underset{\sim}{\sqsubset}_{\mathsf{clt}}$, as we proceed to show in the next example.

Counterexample 4 *Let* $\mathcal{G}_{\mathrm{bad}}$ *be defined as* $\mathcal{G}$ *in Definition 6, 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* $\mathcal{G}_{\mathrm{bad}}$*,* $\preccurlyeq^{\mathrm{bad}}_{\mathsf{clt}}$*, contains client pairs that are not in* $\underset{\sim}{\sqsubset}_{\mathsf{clt}}$*.*

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

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

$$r_6' = \tau.r_6^a + \tau.r_6^b \qquad\qquad r_6^a = a.\,\mathit{0} + \tau.\,\mathit{1} \qquad\qquad r_6^b = b.\,\mathit{0} + \tau.\,\mathit{1}$$

and r_1 *and* r_2 *are the clients defined in (1) above. On the one hand, we have that* $r_6 \not\underset{\sim}{\sqsubset}_{\mathsf{clt}} r_7$*, because* $\overline{c}.\,\mathit{0}\ \mathsf{must}\ r_6$ *whereas* $\overline{c}.\,\mathit{0}\ \mathsf{m\not{u}st}\ r_7$*. On the other hand, we now show that* $r_6 \preccurlyeq^{\mathrm{bad}}_{\mathsf{clt}} r_7$*. Focusing on condition Definition 6(3), we start by deducing that* $r_6 \in \mathcal{U}_{\mathsf{bhv}}$ *(either directly using Definition 5 or indirectly through* $\overline{c}.\,\mathit{0}\ \mathsf{must}\ r_6$*, recalling Theorem 2). Now, Definition 6(3a) is true because* $\mathit{0}\ \mathsf{must}\ r_7$*, thus* r_7 *is usable,* *and thanks to Theorem 2 we have $r_7 \in \mathcal{U}_{\mathsf{bhv}}$. Also point (3b) is satisfied, because* $\mathsf{Acc}_{\not\checkmark}(r_7) = \mathsf{Acc}_{\not\checkmark}(r_6) = \{\{a\}\}$.[4] *To prove that the (relaxed) condition* (3bad) *holds, we have to show that*

$$r_6^c \preccurlyeq_{\mathsf{clt}}^{\mathrm{bad}} a.\,\mathit{1} + b.\,\mathit{0} \quad \textit{and} \quad r_6^c \preccurlyeq_{\mathsf{clt}}^{\mathrm{bad}} a.\,\mathit{0} + b.\,\mathit{1}, \qquad \textit{with } r_6^c = r_6' + \tau.r_6^a + \tau.r_6^b \tag{7}$$

Let $r_7' = a.\,\mathit{1} + b.\,\mathit{0}$. We only show the proof for the inequality $r_6^c \preccurlyeq_{\mathsf{clt}}^{\mathrm{bad}} r_7'$, since the proof for the other inequality is analogous. We focus again on conditions (3a), (3b), and (3bad). Condition (3a) is true because $\mathit{0}$ 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., $\overline{a}.\,\mathit{0}$ 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 $r_7' \xrightarrow{b}_{\not\checkmark} \mathit{0}$, which requires us to show that $\tau.\,\mathit{0} \preccurlyeq_{\mathsf{clt}}^{\mathrm{bad}} \mathit{0}$; this latter check is routine. As a result, we have $r_6^c \preccurlyeq_{\mathsf{clt}}^{\mathrm{bad}} r_7'$. Since we can also show that $r_6^c \preccurlyeq_{\mathsf{clt}}^{\mathrm{bad}} a.\,\mathit{0} + b.\,\mathit{1}$ holds, we obtain (7), and consequently $r_6 \preccurlyeq_{\mathsf{clt}}^{\mathrm{bad}} r_7$.* ■

Theorem 5. *In any finite branching LTS $r_1 \underset{\sim}{\sqsubset}_{\mathsf{clt}} r_2$ if and only if $r_1 \preccurlyeq_{\mathsf{clt}} r_2$.*

Proof. We have to show the set inclusions, $\underset{\sim}{\sqsubset}_{\mathsf{clt}} \subseteq \preccurlyeq_{\mathsf{clt}}$ and $\preccurlyeq_{\mathsf{clt}} \subseteq \underset{\sim}{\sqsubset}_{\mathsf{clt}}$. Lemma 5 and Theorem 1 imply that $\underset{\sim}{\sqsubset}_{\mathsf{clt}} \subseteq \mathcal{G}(\underset{\sim}{\sqsubset}_{\mathsf{clt}})$, and thus, by the Knaster-Tarski theorem, we obtain the first inclusion. The second set inclusion follows from a series of smaller results, namely Lemma 13, Lemma 14, Lemma 15, and Theorem 1, which we give in Appendix B. □

Example 5. Recall clients $r_4 = a.\,1 + \mu y.(a.r_3'' + b.y + c.\,1)$ and $r_5 = (\mu z.(b.z + c.\,1)) + d.\,1$ from Example 4, used to argue that the alternative relation $\lesssim_{\mathsf{clt}}$ is still a burdensome method for reasoning on $\underset{\sim}{\sqsubset}_{\mathsf{clt}}$. By contrast, We now contend that it is simpler to show $r_4 \underset{\sim}{\sqsubset}_{\mathsf{clt}} r_5$ by proving $r_4 \preccurlyeq_{\mathsf{clt}} r_5$, thanks to Theorem 5 and the Knaster-Tarski theorem. By Definition 6, 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 6 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 $\preccurlyeq_{\mathsf{clt}}$ may also be used to prove that two clients are *not* in the contextual preorder $\underset{\sim}{\sqsubset}_{\mathsf{clt}}$: by iteratively following the conditions of Definition 6 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 a such 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 4. By virtue of the full-abstraction result, we can show directly that $r_6 \not\underset{\sim}{\sqsubset}_{\mathsf{clt}} r_7$ by following the requirements of Definition 6 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,

[4] The restriction of the left hand side of the inclusion of Definition 6(3b) by $ua_{\mathsf{bhv}}(r_6)$ is superfluous.

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 6(3c) requires that we show that

$$r_6^c \; R \; \tau.r_7' + \tau.r_7'' \;\text{ where } r_6^c = \bigoplus(r_6 \,\mathsf{after}_{\not\checkmark}\, c) \text{ and } (\tau.r_7' + \tau.r_7'') = \bigoplus(r_7 \,\mathsf{after}_{\not\checkmark}\, c) \quad (8)$$

and r_6^c, r_7' and r_7'' are the clients defined earlier in Counterexample 4. Since we want to show that $R \nsubseteq \mathcal{G}(R)$, the condition Definition 6(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 6(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 (7) of Counterexample 4 resulted in $r_6 \preccurlyeq^{\mathsf{bad}}_{\mathsf{clt}} r_7$, (8) is instrumental to conclude that $r_6 \not\preccurlyeq_{\mathsf{clt}} r_7$. Note also that the path along c leading to a violation of the requirements of Definition 6 is related to the discriminating server $\overline{c}.\,\mathsf{0}$ used in Counterexample 4 to justify $r_6 \not\underset{\sim}{\sqsubset}_{\mathsf{clt}} r_7$. ■

5 Expressiveness and Decidability

We show that servers with finite interactions suffice to preserve the discriminating power of the contextual preorder $\underset{\sim}{\sqsubset}_{\mathsf{clt}}$ in Definition 1, which has ramifications on standard verification techniques for the preorder, such as counter-example generation [10]. We also show that, for finite-state LTSs, the set of usable clients is decidable. Using standard techniques [26] we then argue that, in such cases, there exists a procedure to decide whether two finite-state clients are related by $\underset{\sim}{\sqsubset}_{\mathsf{clt}}$.

5.1 On the power of finite interactions

We employ the coinductive characterisation of the client preorder, Theorem 5, 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 $\underset{\sim}{\sqsubset}_{\mathsf{clt}}$ of Definition 1. Let $\mathsf{CCS}^f ::= \mathsf{0} \;\mid\; \mathsf{1} \;\mid\; \alpha.p \;\mid\; p + q \;\mid\; \tau^\infty$, and

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

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 powers (see Counterexample 3). Accordingly, $\underset{\sim}{\sqsubset}^f_{\mathsf{clt}}$ and $\mathcal{U}^f$ restrict the *resp.* sets to the syntactic class CCS^f.

Corollary 1 *The sets $\mathcal{U}$ and $\mathcal{U}^f$ coincide.*

Proof. The inclusion $\mathcal{U}^f \subseteq \mathcal{U}$ is immediate. Suppose that $r \in \mathcal{U}$. By Theorem 2 we have $r \in \mathcal{U}_{\mathsf{bhv}}$. By Lemma 1, there exists a non-recursive $p \in \mathsf{CCS}^f$ such that $p \;\mathsf{must}\; r$, thus $r \in \mathcal{U}^f$ follows. □

Theorem 6. *In any finite-branching LTS $r_1 \underset{\sim}{\sqsubset}^f_{\mathsf{clt}} r_2$ if and only if $r_1 \underset{\sim}{\sqsubset}_{\mathsf{clt}} r_2$.*

Proof. The inclusion $\underset{\sim}{\sqsubset}_{\mathsf{clt}} \subseteq \underset{\sim}{\sqsubset}^{f}_{\mathsf{clt}}$ follows immediately from the *resp.* definitions. On the other hand, Theorem 5 provides us with a proof technique for showing the inclusion $\underset{\sim}{\sqsubset}^{f}_{\mathsf{clt}} \subseteq \underset{\sim}{\sqsubset}_{\mathsf{clt}}$: if we show that $\underset{\sim}{\sqsubset}^{f}_{\mathsf{clt}} \subseteq \mathcal{G}(\underset{\sim}{\sqsubset}^{f}_{\mathsf{clt}})$ then $\underset{\sim}{\sqsubset}^{f}_{\mathsf{clt}} \subseteq \preccurlyeq_{\mathsf{clt}} = \underset{\sim}{\sqsubset}_{\mathsf{clt}}$. In view of the Knaster-Tarski theorem it suffices to show that $\underset{\sim}{\sqsubset}^{f}_{\mathsf{clt}} \subseteq \mathcal{G}(\underset{\sim}{\sqsubset}^{f}_{\mathsf{clt}})$. In turn, this requires us to prove the three conditions stated in Definition 6. The argument for the first two conditions is virtually the same to that of Lemma 5. Similarly, the arguments for the third condition follow closely those used in Theorem 1 (albeit in a simpler setting of unsuccessful traces of length 1). The only new reasoning required is that servers that exists because of $r_1 \in \mathcal{U}$ also belong to CCS^f, which we know from Corollary 1. □

An analogous result should also hold for the server-preorder, for the proofs of completeness in [5, Theorem 3.1] rely on clients that can be written in the language CCS^f.

5.2 Deciding the client preorder

Figure 4 describes the pseudo-code for the eponymous function $\mathsf{isUsable}(r, acm)$, which is meant to determine 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 4-9). In particular, line 4 checks that infinite sequences of internal moves are always successful (using function $\mathsf{convtick}$ defined on lines 11-17) and that partially deadlocked clients reached through a finite number of unsuccessful internal moves, $\mathsf{Acc}_{\not\checkmark}(r) \neq \emptyset$, contain at least one action that unblocks them to some other usable client (lines 7-8). This latter check employs the function $\mathsf{existsUnblockAction}$ (defined on lines 19-26) which recursively calls $\mathsf{isUsable}$ to determine whether the client reached after an action is indeed usable. $\mathsf{isUsable}(r, acm)$ of Figure 4 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 Figure 2. Concretely, we define the set of internal-sums for the derivatives that a client r reaches via all the finite traces $\in \mathsf{Act}^{\star}$, 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 6. *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 Lemma 4.2.11 of [28] for the language serial-CCS, which is homologous to CCS^{μ} of Figure 2 modulo the satisfaction construct $\mathsf{1}$. □

```
isUsable ($r$, $acm$) =
  if $r$ in $acm$
    then false
    else if convtick ($\emptyset$, $r$)
      then if  $\mathsf{Acc}_{\not\checkmark}(r)$ == empty
        then true
        else $BoolSet$ = map ( existsUnblockAction $acm$ $r$) $\mathsf{Acc}_{\not\checkmark}(r)$
               conjunction $BoolSet$
      else false
where
  convtick($acm$, $r$) =
      if $r$ in $acm$
        then false
        else if $r \xrightarrow{\checkmark}$
          then true
          else $BoolSet$ = map (convtick ($acm$ $\cup\{r\}$)) $\{r' \mid r \xrightarrow{\tau} r'\}$
                 conjunction $BoolSet$
and
  existsUnblockAction ($acm$, $r$, $A$) =
    case $A$ of
      empty -> false
      $\{a\} \uplus A'$ ->
        if $r \overset{a}{\Longrightarrow}_{\not\checkmark}$
          then if isUsable ($\bigoplus(r\ \mathsf{after}_{\not\checkmark}\ a)$, $acm$ $\cup$ { $r$ })
            then true else existsUnblockAction ($r$, A', $acm$)
        else true
```

Fig. 4. An algorithm for deciding inclusion in the set $\mathcal{U}$

Theorem 7. *For every $r \in \mathsf{Proc}$ we have that*

(i) $r \in \mathcal{U}$ *iff* $\mathsf{isUsable}(r, \emptyset) = \mathsf{true}$*,*
(ii) $r \notin \mathcal{U}$ *iff* $\mathsf{isUsable}(r, \emptyset) = \mathsf{false}$*.*

Proof. For the *only-if* case of clause (*i*), we use Theorem 2 and show instead that $r \in \mathcal{U}_{\mathsf{bhv}}$ implies $\mathsf{isUsable}(r, \emptyset) = \mathsf{true}$; 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 $\mathsf{isUsable}(r, \emptyset) = \mathsf{true}$ implies $r \in \mathcal{U}_{\mathsf{bhv}}$, by numerical induction on the *least* number $n \in \mathbb{N}^+$ of (recursive) calls to isUsable that yield the outcome true. We note that in either direction of clause (*i*), 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 $\mathsf{Acc}_{\not\checkmark}(r) = \emptyset$).

For the second clause (*ii*), the statements ($r \notin \mathcal{U}$ implies $\mathsf{isUsable}(r, \emptyset) = \mathsf{true}$) and ($\mathsf{isUsable}(r, \emptyset) = \mathsf{false}$ implies $r \in \mathcal{U}$) contradict the first clause (*i*) which we just proved. The required result thus holds if we ensure that $\mathsf{isUsable}(r, \emptyset)$ is defined for any $r \in \mathsf{Proc}$. This follows from Lemma 6. □

From Theorem 5, Theorem 7 and Lemma 6, we conclude that Definition 6 can be used to decide $\sqsubseteq_{\mathsf{clt}}$ for languages such as CCS^{μ} of Figure 2.We can do this by adapting the algorithm of [26, Chapter 21.5], and proving that in our setting [26, Theorem 21.5.9 and Theorem 21.5.12] are true. In particular, using the terminology of [26] we have that $reachable_{\mathcal{G}}(X)$ is finite, essentially because the *resp.* LTS is finite-state, and thus the decidability of $\preccurlyeq_{\mathsf{clt}}$ follows from Theorem 21.5.12.

6 Conclusion

We present a study that revolves around the notion of usability and preorders for clients (tests). Preorders for clients first appeared for compliance testing [2], and were subsequently investigated in [3,5] for must testing [11] and extended to include peers. The characterisations given in [5] relied fundamentally on the set of usable terms $\mathcal{U}$ which made 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, but the very nature of must testing — which regards infinite unsuccessful computations as catastrophic — let us treat recursive terms in a finite manner (see Definition 5).

We focus on the client preorder, even though [5] presents preorders for both client and peers; note however that [5, Theorem 3.20] and Theorem 2 imply full-abstraction for the peer preorder as well. Our investigations and the *resp.* proofs for Theorem 2, Theorem 5 and Theorem 6 are conducted in terms of finitely-branching LTSs, which cover the semantics used by numerous other work describing client and server contracts [7,17,8,5] — we only rely on an internal choice construct to economise on our presentation, but this can be replaced by tweaking the *resp.* definitions so as 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 7 relies on a stronger property, namely that the language is finite-state. In [28], 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.

6.1 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 [13] do not contain unsuccessful termination, $\mathsf{0}$, restrict internal (*resp.* external) choices to contain only pair-wise distinct outputs (*resp.* inputs) and are, by definition, strongly convergent [24] (*i.e.*, no infinite sequences of τ- transitions). *E.g.*, $\tau.!a.\,\mathsf{1}\ +\ \tau.!b.?c.\,\mathsf{1}$ corresponds to a session type in our language (modulo syntactic transformations such as those for internal choices), whereas $\tau.!a.\,\mathsf{0}\ +\ \tau.!b.?c.\,\mathsf{1}$, $\tau.!a.\,\mathsf{1}\ +\ \tau.!a.?b.\,\mathsf{1}$ and $?a.\,\mathsf{1}\ +\ ?a.!b.\,\mathsf{1}$ do *not*. Since they are mostly deterministic — only internal choices on outputs are permitted — usability is relatively easy to characterise. In fact [6, Section 5] shows that every session type is usable *wrt.* compliance testing (even in the presence of higher-order communication) whereas, in [25, 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 2 is enough to (positively) characterise usable session types *wrt.* must testing; we leave the axiomatisation of $\mathcal{U}$ in this setting as future work.

Contracts [24] are usually formalised as (mild variants of) our language CCS^{μ}. In the case of must testing, the authors in [5, Theorem 6.9, Lemma 7.8(2)] characterise *non*-usable clients (and peers) for the sublanguage CCS^{f} as the terms that can be rewritten into $\mathsf{0}$ via equational reasoning. Full-abstraction for usable clients *wrt. compliance* testing has been solved for *strongly convergent terms* in [24, Proposition 4.3] by giving a coinductive characterisation for 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 $r = \mathsf{1} + \tau.\mathsf{0}$; this client is usable *wrt.* must because, for arbitrary p, any computation of $p \parallel r$ is successful (since we have $r \xrightarrow{\checkmark}$ immediately). On the other hand, r is *not* viable *wrt.* compliance testing of [24] (where every server is strongly convergent), because for any server p we observe the computation starting with the reduction $p \parallel r \xrightarrow{\tau} p \parallel \mathsf{0}$, and once p stabilises to some p', the final state $p' \parallel \mathsf{0}$ 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 [24, Definition 4.2], the two definitions have strikingly different meanings: we are forced to reason *wrt. unsuccessful* actions and *unsuccessful* acceptance sets whereas [24, 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 [24]). We note also that our Definition 5 is inductive whereas [24, 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 6 (and consequently Theorem 5). We are unaware of any full-abstraction results for contract usability in the case of should-testing [27,7,23].

Future work: In the line of [9], 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 [12], 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.

References

1. J. R. Andersen, N. Andersen, S. Enevoldsen, M. M. Hansen, K. G. Larsen, S. R. Olesen, J. Srba, and J. Wortmann. CAAL: concurrency workbench, aalborg edition. In *ICTAC*, 2015.

2. F. Barbanera and 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 and M. Hennessy. Modelling session types using contracts. In *SAC*, 2012.
5. G. Bernardi and M. Hennessy. Mutually testing processes. *LMCS*, 11(2), 2015.
6. G. Bernardi and M. Hennessy. Using higher-order contracts to model session types. *LMCS*, 12(2), 2016.
7. M. Bravetti and G. Zavattaro. A foundational theory of contracts for multi-party service composition. *Fundam. Inf.*, 89(4), 2008.
8. G. Castagna, N. Gesbert, and L. Padovani. A theory of contracts for web services. *ACM Trans. Program. Lang. Syst.*, 31(5), 2009.
9. A. Cerone and M. Hennessy. Process behaviour: Formulae vs. tests. In *EXPRESS*, 2010.
10. E. M. Clarke and H. Veith. Counterexamples revisited: Principles, algorithms, applications. In *Verification: Theory and Practice*, 2003.
11. R. De Nicola and M. Hennessy. Testing equivalences for processes. *TCS*, 34(1–2), 1984.
12. A. Francalanza. A Theory of Monitors. In *FoSSaCS*, LNCS, 2016.
13. S. J. Gay and M. Hole. Subtyping for session types in the pi calculus. *Acta Inf.*, 42(2-3), 2005.
14. M. Hennessy. *Algebraic Theory of Processes*. 1988.
15. D. E. Knuth. *The Art of Computer Programming, Volume 1 (3rd Ed.): Fundamental Algorithms*. Addison Wesley Longman Publishing Co., Inc., 1997.
16. D. König. Über eine schlussweise aus dem endlichen ins unendliche. *Acta Litt. ac. sci. Szeged*, 3, 1927.
17. C. Laneve and L. Padovani. The must preorder revisited. In *CONCUR*, 2007.
18. Q. Luo, F. Hariri, L. Eloussi, and D. Marinov. An empirical analysis of flaky tests. In *FSE*, 2014.
19. P. Marinescu, P. Hosek, and C. Cadar. Covrig: A framework for the analysis of code, test, and coverage evolution in real software. In *ISSTA*, 2014.
20. A. Martens. Analyzing Web Service Based Business Processes. In *FASE*, 2005.
21. A. M. Memon and M. B. Cohen. Automated testing of gui applications: Models, tools, and controlling flakiness. In *ICSE*, 2013.
22. R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.
23. A. J. Mooij, C. Stahl, and M. Voorhoeve. Relating fair testing and accordance for service replaceability. *J. Log. Algebr. Program.*, 79(3-5), 2010.
24. L. Padovani. Contract-based discovery of web services modulo simple orchestrators. *TCS*, 411(37), 2010.
25. L. Padovani. Fair subtyping for multi-party session types. *MSCS*, 26(3), 2016.
26. B. Pierce. *Types and programming languages*. 2002.
27. A. Rensink and W. Vogler. Fair testing. *Information and Computation*, 205(2), 2007.
28. C. Spaccasassi. *Language Support for Communicating Transactions*. PhD thesis, TCD, 2015.
29. D. Weinberg. Efficient controllability analysis of open nets. In *WS-FM*, 2009.
30. G. Winskel. *The Formal Semantics of Programming Languages: An Introduction*. 1993.

A Technical results

We start by proving a few basic results ensuring that the least fixpoint of $\mathcal{F}$, *i.e.*, $\mathcal{U}_{\mathsf{bhv}} = \mu x.\mathcal{F}(x)$ exists.

Lemma 7 (Continuity).

(i) For any $S, S' \subseteq \mathsf{CCS}$. $S \subseteq S'$ implies $\mathcal{F}(S) \subseteq \mathcal{F}(S')$.

(ii) For every chain $S_0 \subseteq S_2 \subseteq S_3 \subseteq \ldots$ of sets in $\mathcal{P}(\mathsf{CCS})$, the following equality is true,

$$\mathcal{F}(\bigcup_{i=0}^{\omega} S_i) = \bigcup_{i=0}^{\omega} \mathcal{F}(S_i)$$

Proof. For clause (i), fix an $r \in \mathcal{F}(S)$, we have to prove that $r \in \mathcal{F}(S')$. Definition 5 requires us to prove that $r \Downarrow_{\checkmark}$, and that for every r', $r \Longrightarrow_{\not\checkmark} r' \overset{\tau}{\not\longrightarrow}$ implies that there exists an action $a \in \mathsf{Act}$ such that (1) $r' \xrightarrow{a}$ and that (2) if $(r\ \mathsf{after}_{\not\checkmark}\ a) \neq \emptyset$ then $(r\ \mathsf{after}_{\not\checkmark}\ a) \in S'$. We know that $r \Downarrow_{\checkmark}$ because of Definition 5(1). Now fix an r' such that $r \Longrightarrow_{\not\checkmark} r' \overset{\tau}{\not\longrightarrow}$, Definition 5(2) implies that for some action a we have $r' \xrightarrow{a}$ and that if $(r\ \mathsf{after}_{\not\checkmark}\ a) \neq \emptyset$ then $(r\ \mathsf{after}_{\not\checkmark}\ a) \in S$. The hypothesis $S \subseteq S'$ implies that if $(r\ \mathsf{after}_{\not\checkmark}\ a) \neq \emptyset$ then $(r\ \mathsf{after}_{\not\checkmark}\ a) \in S'$, as required.

For clause (ii), we have to show two set inclusions, namely

1. $\mathcal{F}(\bigcup_{i=0}^{\omega} S_i) \subseteq \bigcup_{i=0}^{\omega} \mathcal{F}(S_i)$
2. $\bigcup_{i=0}^{\omega} \mathcal{F}(S_i) \subseteq \mathcal{F}(\bigcup_{i=0}^{\omega} S_i)$

To prove the first inclusion fix an $r \in \mathcal{F}(\bigcup\{ S \mid S \in \sigma \})$. The bulk of the work amounts to building a set $\hat{S}$ such that $\hat{S} \in \sigma$ and $r \in \mathcal{F}(\hat{S})$. We gather the enough material to define the set $\hat{S}$. Consider the set

$$\{ r_i \mid r \Longrightarrow_{\not\checkmark} r_i \overset{\tau}{\not\longrightarrow} \text{ and } i \in I \}$$

where I is an index set. Definition 5(2) and the definition of $\bigcup_{i=0}^{\omega} S_i$ imply that for every $i \in I$, there exists an action a_i such that $r_i \xrightarrow{a_i}$ and if $(r\ \mathsf{after}_{\not\checkmark}\ a_i) \neq \emptyset$ then $\bigoplus(r\ \mathsf{after}_{\not\checkmark}\ a_i) \in S_j$ for some $j \in \mathbb{N}$. Pick the greatest such $\hat{j}$, as by definition we have a chain of S_i's, for every $j' \leq \hat{j}$ we have $S'_j \subseteq S_{\hat{j}}$.

$$\text{for every } i \in I. (r\ \mathsf{after}_{\not\checkmark}\ a_i) \neq \emptyset \text{ implies } \bigoplus(r\ \mathsf{after}_{\not\checkmark}\ a_i) \in S_{\hat{j}} \tag{9}$$

The desired $r \in \mathcal{F}(S_{\hat{j}})$ is now an easy consequence of Definition 5(1), which implies that $r \Downarrow_{\checkmark}$ - and (9) above.

The proof of the converse inclusion, that is $\bigcup_{i=0}^{\omega} \mathcal{F}(S_i) \subseteq \mathcal{F}(\bigcup_{i=0}^{\omega} S_i)$, is straightforward. Pick a $r \in \bigcup \bigcup_{i=0}^{\omega} \mathcal{F}(S_i)$, clearly this means that for some $S \in \sigma$ we have $r \in \mathcal{F}(S)$. Definition 5 ensures that $r \Downarrow_{\checkmark}$ and that for every r' such that $r \Longrightarrow_{\not\checkmark} r' \overset{\tau}{\not\longrightarrow}$ we have $r' \xrightarrow{a}$ for some a, and if $(r\ \mathsf{after}_{\not\checkmark}\ a) \neq \emptyset$ then $\bigoplus(r\ \mathsf{after}_{\not\checkmark}\ a) \in S$. It follows that if $(r\ \mathsf{after}_{\not\checkmark}\ a) \neq \emptyset$ then $\bigoplus(r\ \mathsf{after}_{\not\checkmark}\ a) \in \bigcup \bigcup_{i=0}^{\omega} S_i$. But then we obtain immediately the desired $r \in \mathcal{F}(\bigcup_{i=0}^{\omega} S_i)$.

We next prove Lemma 1, which implies that $\mathcal{U}_{\mathsf{bhv}}$ of Definition 5 is a sound characterisation for the set of usable clients $\mathcal{U}$.

Lemma 1. *For every $n \in \mathbb{N}$ and $r \in \mathsf{CCS}$, $r \in \mathcal{F}^n(\emptyset)$ implies that there exists a non-recursive server p such that p must r.*

Proof. We reason by numerical induction on n. In the base case, $n = 0$, and the lemma is trivially true, for there is no r such that $r \in \mathcal{F}^0(\emptyset) = \emptyset$. In the inductive case, $n = m+1$, and we know, by hypothesis, that $r \in \mathcal{F}^{m+1}(\emptyset)$ and must exhibit a *non-recursive* p such that p must r. We proceed by case analysis on $\mathsf{Acc}_{\not\checkmark}(r)$, and have two subcases:

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

$$r = r_0 \xrightarrow{\tau} \ldots \xrightarrow{\tau} r_n = r' \overset{\tau}{\not\longrightarrow} \text{ implies } r_i \xrightarrow{\checkmark} \text{ for some } r_i \tag{10}$$

We prove that the non-recursive server $\mathsf{0}$ is our witness, *i.e.*, $\mathsf{0}$ must r. Pick a maximal computation of $\mathsf{0} \parallel r \xrightarrow{\tau} \mathsf{0} \parallel r_1 \xrightarrow{\tau} \ldots$. 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 (10) 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, and so the set $A = \mathsf{Acc}_{\not\checkmark}(r) \cap ua(r)$ is finite as well. Assume that $|\mathsf{Acc}_{\not\checkmark}(r)| = n$ and denote the elements of this set as A_i for $i \in 1..n$. 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 $i \in 1..n$ 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 must $\bigoplus(r \;\mathsf{after}_{\not\checkmark}\; a_i)$. The required witness (non-recursive) server proving that r is usable is

$$\hat{p} = \Big(\sum_{\{\, a_i \,|\, (r \,\mathsf{after}_{\not\checkmark} a_i) \neq \emptyset \,\}} \overline{a_i}.p^i\Big) + \Big(\sum_{\{\, a_i \,|\, (r \,\mathsf{after}_{\not\checkmark} a_i) = \emptyset \,\}} \overline{a_i}.\mathsf{0}\Big) \tag{11}$$

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

$$r \parallel \hat{p} = r_0 \parallel p_0 \xrightarrow{\tau} r_1 \parallel p_1 \xrightarrow{\tau} \ldots \tag{12}$$

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

- In (12) there is *no* interaction between derivatives of r and those of $\hat{p}$. Since $\hat{p}$ is stable, the computation (12) must be due to reductions of r. If (12) is infinite, it is because r diverges, and by $r \Downarrow_{\checkmark}$ of Definition 5(1) the computation must be successful. Else (12) is finite and has the form $r \parallel \hat{p} \Longrightarrow r' \parallel \hat{p} \overset{\tau}{\not\longrightarrow}$. It follows that $r \Longrightarrow r' \overset{\tau}{\not\longrightarrow}$. Since $r' \parallel \hat{p} \overset{\tau}{\not\longrightarrow}$, it also follows that there must be a successful state amongst the silent moves $r \Longrightarrow r'$. For otherwise, we would have $r \Longrightarrow_{\not\checkmark} r' \overset{\tau}{\not\longrightarrow}$ and by Eq. (3) 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. (11) would then guarantee that $r' \parallel \hat{p} \xrightarrow{\tau}$ (by an interaction on an action in $S(r')$), contradicting $r' \parallel \hat{p} \overset{\tau}{\not\longrightarrow}$.

- In (12) there exists at least one interaction, then assume that the first one results in the reduction $r_k \parallel p_k \xrightarrow{\tau} r_{k+1} \parallel p_{k+1}$. As this is the first reduction, $p_k = \hat{p}$, and Eq. (11) 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 $r \overset{a_i}{\Longrightarrow}_{\not\checkmark} r_{k+1}$. 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. (11) 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 $r \parallel \hat{p}$ is successful, and thus $\hat{p} \ \mathsf{must}\ r$, as required.

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

The following are lemmata leading up to the completeness of $\mathcal{U}_{\mathsf{bhv}}$ *wrt.* $\mathcal{U}$. We start by restating König's infinity lemma.

Lemma 8 (König's infinity lemma [16]). *If G is a finite-branching connected graph with infinitely many nodes, then G contains an infinitely long simple path.*

The must relation enjoys a few of properties that will be used in the following proofs. We restate them here for completeness.

Lemma 9. *For every $p, r \in \mathsf{CCS}$, if $p \ \mathsf{must}\ r$ then (1) $r \Downarrow_{\checkmark}$, and (2) for every $s \in \mathsf{Act}^{\star}$, if $p \overset{\overline{s}}{\Longrightarrow} p'$ 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].

Lemma 2. *Let T be a tree with root v. If T is finite-branching and it has a finite number of nodes, then the number of paths $v \longrightarrow \ldots$ is finite.*

Proof. Let m be the number of paths that start at v, and let n be the number of nodes of T. The hypothesis that n is finite lets us reason by induction on n. As T must have at least one node, namely the root v, the base case is $n = 1$. Since trees do not contain loops, the number of paths from v is 0, thus it is finite. In the inductive case, the hypothesis that T is finite-branching ensures that the root v has a finite number of outgoing edges, $v \longrightarrow v_0, \ldots v \longrightarrow v_k$. Every node v_j is the root of a tree T_j that has at most $n - 1$ nodes, that is finite state and finite-branching. The inductive hypothesis thus ensures that every T_j contains a finite number m_j of paths starting at v_j. Now note that $m = \sum_{i \in [0;k]} m_j$. Since this sum contains a finite amounts of summands, namely k, and every m_j is finite, the whole sum m is finite.

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

Lemma 10. *For every $n \in \mathbb{N}$ and $r \in \mathcal{U}^n$, 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 $\bigoplus(r \ \mathsf{after}_{\not\checkmark}\ a) \in \mathcal{U}^m$ for some $m < n$.*

Proof. Fix an $r \in \mathcal{U}^n$. 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 $p \ \mathsf{must}\ r$. 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 $r \Longrightarrow_{\not\checkmark} r' \not\xrightarrow{\tau}$ (which, in turn, implies that $r \not\xrightarrow{\checkmark}$). Since $p \ \mathsf{must}\ r$ and $r \not\xrightarrow{\checkmark}$, the process p cannot diverge, meaning that there exists a p' such that $p \Longrightarrow p' \not\xrightarrow{\tau}$. Lemma 9(2i) now implies that $p' \ \mathsf{must}\ r'$. Since $r' \not\xrightarrow{\checkmark}$ and both p' and r' are stable, they must interact (for otherwise $p' \not{\mathsf{m}}\mathsf{ust}\ r'$). This means that $r' \xrightarrow{a}$ for some $a \in \mathsf{Act}$, and that $p' \xrightarrow{\bar{a}} p''$ 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 $(r \ \mathsf{after}_{\not\checkmark}\ a)$ is also finite: we let $r_a = \bigoplus(r \ \mathsf{after}_{\not\checkmark}\ a)$ and proceed to show that $r_a \in \mathcal{U}^m$ for some $m < n$.

First observe that $p'' \ \mathsf{must}\ r_a$ because $p'' \ \mathsf{must}\ r''$ for every $r'' \in (r \ \mathsf{after}_{\not\checkmark}\ a)$: the latter is a consequence of Lemma 9(2i), our assumption $p \ \mathsf{must}\ r$, $p \overset{\bar{a}}{\Longrightarrow} p''$ and the fact that $r'' \in (r \ \mathsf{after}_{\not\checkmark}\ a)$ implies $r \overset{a}{\Longrightarrow}_{\not\checkmark} r''$.

Let $rank(r_a) = m$. Since $p'' \ \mathsf{must}\ r_a$, 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 $r_a \parallel p''$ 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''. Now this suffix c must be a suffix of a maximal computation of $r \parallel p \Longrightarrow_{\not\checkmark} r'' \parallel p''$ where r and p interact on a in $r \parallel p \Longrightarrow_{\not\checkmark} r'' \parallel p''$ and $r'' \in (r \ \mathsf{after}_{\not\checkmark}\ a)$. It follows that c must contain at least one less interaction that the computation starting with $r \parallel p \Longrightarrow_{\not\checkmark} r'' \parallel p''$, 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 4. *For every $r \in \mathcal{U}$, we have $r \in \mathcal{U}_{\mathsf{bhv}}$.*

Proof. We proceed by strong (complete) induction on i where $r \in \mathcal{U}^i$. 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 $r \in \mathcal{U}^0 \subseteq \mathcal{U}$ of (6), and Lemma 9(1). For the second property,

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

we show that $\mathsf{Acc}_{\not\checkmark}(r) = \emptyset$, in which case (13) holds trivially. From $r \in \mathcal{U}^0$ 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 $p \parallel r \Longrightarrow p' \parallel r'$ necessarily forms a prefix of one of these maximal computations right up to the point of the first interaction of the *resp.* maximal computation (in case there is no interaction in the *resp.* maximal computation, we then have $p' \parallel r' \overset{\tau}{\not\longrightarrow}$). This means that a success must have been reached during $p \parallel r \Longrightarrow p' \parallel r'$, which also means that a 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 $r \in \mathcal{U}^{m+1}$ 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

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

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 10, we know that there exists an action $a \in A$, such that

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

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.

B Proofs for the coinductive characterisation

Lemma 5. *$r_1 \underset{\sim}{\sqsubset}_{\mathsf{clt}} r_2$ implies*

(i) if $r_2 \xrightarrow{\tau}_{\not\checkmark} r_2'$ then $r_1 \underset{\sim}{\sqsubset}_{\mathsf{clt}} r_2'$;
(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 $r_1 \underset{\sim}{\sqsubset}_{\mathsf{clt}} r_2$ 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 $r_2' \parallel p$ is a suffix of a maximal computation of $r_2 \parallel p$. Since $r_2 \overset{\checkmark}{\not\longrightarrow}$ and $r_2' \overset{\checkmark}{\not\longrightarrow}$ it must be the case that every maximal computation of $r_2' \parallel p$ contains a successful state, thus $p \ \mathsf{must}\ r_2'$. It follows that $r_1 \underset{\sim}{\sqsubset}_{\mathsf{clt}} r_2'$.

To show point (ii), suppose that $r_1 \underset{\sim}{\sqsubset}_{\mathsf{clt}} r_2$ and that $r_2 \overset{\checkmark}{\not\longrightarrow}$. It follows that

$$\tau^\infty \ \mathsf{m}\not{\mathsf{u}}\mathsf{st}\ r_2,$$

(where τ^∞ is a divergent server performing an infinite number of τ transitions) thus τ^∞ $\mathsf{m\not{u}st}$ 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 Theorem 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 $r_1^a \sqsubseteq_{\mathsf{clt}} r_2^a$, and Definition 1 requires us to show that whenever p must r_1^a then p must r_2^a. Pick a p such that p must r_1^a.

To prove that p 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 must r_1. Thus, the assumption that p must r_1^a ensures that $q + \overline{a}.p$ must r_1. The hypothesis $r_1 \sqsubseteq_{\mathsf{clt}} r_2$ now implies that

$$q + \overline{a}.p\ \mathsf{must}\ r_2 \tag{16}$$

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

$$p \parallel r_2^a = p^0 \parallel r^0 \xrightarrow{\tau} p^1 \parallel r^1 \ldots \tag{17}$$

Note that since p 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 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-transitionss are due internal choices of p. In other terms, the computation in Eq. (17) above contains a prefix

$$p^0 \parallel r^0 \Longrightarrow p^i \parallel r^i \Longrightarrow \ldots$$

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

$$q + \overline{a}.p \parallel r_2 \overset{\tau}{\Longrightarrow} p \parallel r^i \Longrightarrow p^i \parallel r^i \Longrightarrow \ldots \tag{18}$$

is a maximal computation of $q + \overline{a}.p \parallel r_2$, whose suffix $p^i \parallel r^i \Longrightarrow \ldots$ is a suffix of the computation in Eq. (17), and whose first τ is due a weak synchronisation on the action a. It follows that to show a successful state in Eq. (17), it is sufficient to prove that the successful state in the computation in Eq. (18) appears after the state $p^i \parallel r^i$. But this is true because by assumption $r_i \in (r_2\ \mathsf{after}_{\not\checkmark}\ a)$, thus (17) is successful. Since this argument applies for any maximal computation of $p \parallel r_2^a$, we also have that p must r_2^a as required.

Lemma 11 (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 6 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 $B \in \mathsf{Acc}_{\not\tau}(r_2)$ then there exists a $A \in \mathsf{Acc}_{\not\tau}(r_1)$ such that $A \cap ua(r_1) \subseteq B$

(c) if $r_2 \xrightarrow{a}_{\not\tau}$ then $(r_1 \xRightarrow{a}_{\not\tau}$ and $\bigoplus(r_1 \ \mathsf{after}_{\not\tau}\ a)\ R \bigoplus(r_2\ \mathsf{after}_{\not\tau}\ a))$

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 6(3c), and the hypothesis $R \subseteq R'$.

We begin by proving some ancillary technical results, which we spell out in Lemma 12.

Lemma 12. *For every $as \in \mathsf{Act}^{\star}$, and every $r \in \mathsf{CCS}$, we have that*

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

Proof. Point (1) follows easily from the definition of $\mathsf{after}_{\not\tau}$. 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. $ua_{\mathsf{clt}}(r, as) \subseteq ua_{\mathsf{clt}}(\bigoplus(r\ \mathsf{after}_{\not\tau}\ a), s)$
2. $ua_{\mathsf{clt}}(\bigoplus(r\ \mathsf{after}_{\not\tau}\ a), s) \subseteq ua_{\mathsf{clt}}(r, as)$

For the first inclusion, let $\hat{r} = \bigoplus(r\ \mathsf{after}_{\not\tau}\ a)$ 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 $\hat{r} \xRightarrow{sb}_{\not\tau}$ then $(\bigoplus \hat{r}\ \mathsf{after}_{\not\tau}\ sb) \in \mathcal{U}$. Thus, suppose that $\hat{r} \xRightarrow{sb}_{\not\tau}$; by $\hat{r} = \bigoplus(r\ \mathsf{after}_{\not\tau}\ a)$ we deduce that $r \xRightarrow{asb}_{\not\tau}$, thus the definition of $ua_{\mathsf{clt}}(-,-)$ ensures that then $r\ usbl_{\not\tau}\ asb$. Now observe that

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

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

Lemma 13. *For every $r_1 \preccurlyeq_{\mathsf{clt}} r_2$, if for every $s \in \mathsf{Act}^{\star}$, if $r_1\ usbl_{\not\tau}\ s$ then $r_2\ usbl_{\not\tau}\ s$.*

Proof. As preliminary observation, note that for every $s \in \mathsf{Act}^{\star}$, $r_1\ usbl_{\not\tau}\ s$ implies that $r_1 \in \mathcal{U}$, thus $r_1 \preccurlyeq_{\mathsf{clt}} r_2$ and Definition 6(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 $r_2\ usbl_{\not\tau}\ \varepsilon$. 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 $r_2\ usbl_{\not\tau}\ s$ we have to show that

1. $r_2 \in \mathcal{U}$, and
2. if $r_2 \xRightarrow{a}_{\not\tau}$ then $\bigoplus(r_2\ \mathsf{after}_{\not\tau}\ a)\ usbl_{\not\tau}\ s'$.

We have already shown (1). To prove (2) suppose that $r_2 \overset{a}{\Longrightarrow}_{\not\checkmark}$. Since $r_1 \in \mathcal{U}$ and $r_1 \preccurlyeq_{\mathsf{clt}} r_2$, Definition 6(3c) implies that $r_1 \overset{a}{\Longrightarrow}_{\not\checkmark}$ and that $\bigoplus(r_1 \ \mathsf{after}_{\not\checkmark}\ a) \preccurlyeq_{\mathsf{clt}} \bigoplus(r_2 \ \mathsf{after}_{\not\checkmark}\ a)$. The hypothesis $r_1 \ \mathit{usbl}_{\not\checkmark}\ as'$ together with $r_1 \overset{a}{\Longrightarrow}_{\not\checkmark}$ ensures that $\bigoplus(r_1 \ \mathsf{after}_{\not\checkmark}\ a) \ \mathit{usbl}_{\not\checkmark}\ s'$, and so the inductive hypothesis guarantees that $\bigoplus(r_1 \ \mathsf{after}_{\not\checkmark}\ a) \ \mathit{usbl}_{\not\checkmark}\ s'$, which proves (2).

Lemma 14. *For every $r_1 \preccurlyeq_{\mathsf{clt}} r_2$, 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 $r_1 \preccurlyeq_{\mathsf{clt}} r_2$ and Definition 6(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 6 let us deduce that

$$r_1 \overset{a}{\Longrightarrow}_{\not\checkmark} \text{ and that } r_1^a \preccurlyeq_{\mathsf{clt}} r_2' \quad \text{where } r_1^a = \bigoplus(r_1 \ \mathsf{after}_{\not\checkmark}\ a). \tag{19}$$

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 (19) and the inductive hypothesis, we obtain

$$\exists A \in \mathsf{Acc}_{\not\checkmark}(r_1^a, s').\, A \cap \mathit{ua}_{\mathsf{clt}}(r_1^a, s') \subseteq B \tag{20}$$

By point point (3) and point point (4) of Lemma 12, 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 (20) 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 15. *For every $r_1 \preccurlyeq_{\mathsf{clt}} r_2$, 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 $r_1 \preccurlyeq_{\mathsf{clt}} r_2$, the hypothesis $r_2 \Longrightarrow_{\not\checkmark}$ which ensures that $r_2 \overset{\checkmark}{\not\longrightarrow}$, and Definition 6(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 $r_1 \preccurlyeq_{\mathsf{clt}} r_2$, $r_2 \overset{a}{\Longrightarrow}_{\not\checkmark} r_2'$, and point (3c) and point (1) of Definition 6 let us deduce that

$$r_1 \overset{a}{\Longrightarrow}_{\not\checkmark} \quad \text{and} \quad \bigoplus(r_1 \ \mathsf{after}_{\not\checkmark}\ a) \preccurlyeq_{\mathsf{clt}} r_2' \tag{21}$$

From r_1 $\mathit{usbl}_{\not\checkmark}$ as' we also know that $\bigoplus(r_1 \ \mathsf{after}_{\not\checkmark} \ a) \ \mathit{usbl}_{\not\checkmark} \ s'$. Thus, by (21), $r'_2 \overset{s'}{\Longrightarrow}_{\not\checkmark}$, and the inductive hypothesis we obtain that $\bigoplus(r_1 \ \mathsf{after}_{\not\checkmark} \ a) \overset{s'}{\Longrightarrow}_{\not\checkmark}$. This ensures that for some $r'_1 \in (r_1 \ \mathsf{after}_{\not\checkmark} \ a)$ we have $r'_1 \overset{s'}{\Longrightarrow}_{\not\checkmark}$. The definition of $(r_1 \ \mathsf{after}_{\not\checkmark} \ a)$ implies that $r_1 \overset{a}{\Longrightarrow}_{\not\checkmark} r'_1$, thus we can construct $r_1 \overset{as'}{\Longrightarrow}_{\not\checkmark}$ as required.

C Decision procedures and their properties

Lemma 16. *For every $r \in \mathsf{CCS}$, $\mathsf{isUsable}(r, \emptyset)$ is defined.*

Proof. Fix an $r \in \mathsf{CCS}$, and consider the set $R = \{ r \} \cup \mathsf{sumsRdx}(r)$.

We prove that for every $\texttt{acm}$ such that

$$\texttt{acm} \subseteq R \qquad \textbf{(INV)}$$

the computation of $\mathsf{isUsable}(r, \texttt{acm})$ terminates. We show this by establishing first that

1. the property (**INV**) above is an invariant: in $\mathsf{isUsable}(r, \texttt{acm})$ the set `acm'` of every recursive call $\mathsf{isUsable}(\_, \texttt{acm}')$ satisfies (**INV**), and
2. in $\mathsf{isUsable}(r, \texttt{acm})$ the set `acm'` of every recursive call $\mathsf{isUsable}(\_, \texttt{acm}')$ is strictly bigger than $\texttt{acm}$: $\texttt{acm} \subsetneq \texttt{acm}'$.

We prove (1) and (2) together. Suppose that $\mathsf{isUsable}(r, \texttt{acm})$ performs a recursive call on line 24, $\mathsf{isUsable}(r', \texttt{acm}')$. This means that the first if clause (line 2) is false, thus $r \notin \texttt{acm}$. By definition of the recursive call (line 24) $\texttt{acm}' = \texttt{acm} \cup \{ r \}$, thus $\texttt{acm} \subseteq \texttt{acm}'$, which proves (2). Now observe that in every recursive call $r' \in \mathsf{sumsRdx}(r)$. The last statement is true because an induction on $a_1 \ldots a_n$ ensures that

$$\bigoplus(r \ \mathsf{after}_{\not\checkmark} \ a_1 \ldots a_n) = \bigoplus((\bigoplus(\ldots(\bigoplus \ \mathsf{after}_{\not\checkmark} \ a_1)\ldots) \ \mathsf{after}_{\not\checkmark} \ a_{n-1}) \ \mathsf{after}_{\not\checkmark} \ a_n)$$

Since by assumption $\texttt{acm} \subseteq R$, it follows that also $\texttt{acm}' \subseteq R$, which satisfies 1.

Now we prove that if the invariant is true $\mathsf{isUsable}(r, \texttt{acm})$ terminates, with an argument on induction on the number $m = |R \setminus \texttt{acm}|$. If $m = 0$ then it must be the case that $r \in \texttt{acm}$, meaning that the first if clause on line 2 returns false, thus $\mathsf{isUsable}(r, \texttt{acm})$ terminates. In the inductive case, if the algorithm performs a recursive call $\mathsf{isUsable}(r', \texttt{acm}')$, then $\texttt{acm} \subsetneq \texttt{acm}'$. point (2) ensures that $|R \setminus \texttt{acm}'| < m$, and point (1) ensures that $\texttt{acm}' \subseteq R$, thus we can apply the inductive hypothesis, which ensures that $\mathsf{isUsable}(r', \texttt{acm}')$ terminates. As the argument applies to all the recursive calls, the original call $\mathsf{isUsable}(r, \texttt{acm})$ terminates.