# Mutually testing processes[⋆]
## (Extended abstract)

Giovanni Bernardi and Matthew Hennessy

bernargi@tcd.ie, matthew.hennessy@cs.tcd.ie
School of Computer Science and Statistics,
Trinity College Dublin,
Dublin 2, Ireland

**Abstract.** In the standard testing theory of DeNicola-Hennessy one process is considered to be a refinement of another if every test guaranteed by the former is also guaranteed by the latter. In the domain of web services this has been recast, with processes viewed as *servers* and tests as *clients*. In this way the standard refinement preorder between servers is determined by their ability to satisfy clients.
But in this setting there is also a natural refinement preorder between *clients*, determined by their ability to be satisfied by *servers*. In more general settings where there is no distinction between *clients* and *servers*, but all processes are *peers*, there is a further refinement preorder based on the mutual satisfaction of *peers*.
We give a uniform account of these three preorders. In particular we give two characterisations. The first is behavioural, in terms of traces and ready sets. The second, for finite processes, is equational.

## 1 Introduction

The DeNicola-Hennessy theory of testing [NH84,DH87,Hen88] considers a process $p$ to be a refinement of process $q$ if every test passed by $p$ is also passed by $q$. Recently, in papers such as [LP07,Bd10,CGP09,Pad10], this refinement preorder has been recast with a view to providing theoretical foundations for web services. Here processes are viewed as *servers* and tests viewed as *clients*. In this terminology the standard (must) testing preorder is a refinement preorder between servers, which we denote by $p \sqsubseteq_{\mathsf{svr}} q$; this is determined by the ability of the servers $p$, $q$ to satisfy clients. However in this framework there are many other natural behavioural preorders between processes. In this paper we investigate two; the first, $p \sqsubseteq_{\mathsf{clt}} q$, is determined by the ability of the clients $p$, $q$ to be satisfied by servers. For the second we drop the distinction between clients and servers. Instead all processes are viewed as peers of each other and the purpose of interaction between two peers is the mutual satisfaction of both. The resulting refinement preorder is denoted by $p \sqsubseteq_{\mathsf{p2p}} q$. We give a uniform behavioural characterisation of all three refinement preorders in terms of traces and *acceptances*

⋆ Research supported by SFI project SFI 06 IN.1 1898.

*sets* [NH84,Hen88]. We also give equational characterisations for a finite process calculus for servers/clients/peers.

We use an infinitary version of $\mathsf{CCS}$ [Mil89] augmented by a *success* constant $\mathsf{1}$, to describe processes, be they servers, clients or peers. Thus $p = \tau.a.(b.\,\mathsf{0} + c.\,\mathsf{0}) + \tau.a.c.\,\mathsf{0}$ is a server which offers the action $a$ followed by either $b$ and $c$ depending on how choices are made, and then terminates, denoted by $\mathsf{0}$. On the other hand $r = \overline{a}.\overline{c}.\,\mathsf{1}$ is a test or a client which seeks a synchronisation on $a$ followed by one on $c$; as usual [Mil89] communication or cooperation consists of the simultaneous occurrence of an action $a$ and its complement $\overline{a}$. Thus when the server $p$ is executed in parallel with the client $r$, the latter will always be satisfied, in that it is guaranteed to reach the successful state $\mathsf{1}$ regardless of how the various choices are made. But if the client is executed with the alternative server $q = \tau.a.b.\,\mathsf{0} + \tau.a.c.\,\mathsf{0}$ there is a possibility of the client remaining unhappy; for this reason $p \mathrel{\underset{\sim}{\not\sqsubset}}_{\mathsf{svr}} q$. However it turns out that $q \mathrel{\underset{\sim}{\sqsubset}}_{\mathsf{svr}} p$ because every client satisfied by $q$ will also be satisfied by $p$.

The client preorder $p \mathrel{\underset{\sim}{\sqsubset}}_{\mathsf{clt}} q$ compares the processes as clients, and their ability to be satisfied by servers. This refinement preorder turns out to be incomparable with the server preorder. For example $a.\,\mathsf{1} + b.\,\mathsf{0} \mathrel{\underset{\sim}{\not\sqsubset}}_{\mathsf{svr}} a.\,\mathsf{1}$ because of the client $\overline{b}.\,\mathsf{1}$. But $a.\,\mathsf{1} + b.\,\mathsf{0} \mathrel{\underset{\sim}{\sqsubset}}_{\mathsf{clt}} a.\,\mathsf{1}$ because every server satisfying the former also satisfies $a.\,\mathsf{1}$; intuitively the extra component of the client $b.\,\mathsf{0}$ puts no further demands on servers, because the execution of $b$ will never lead to satisfaction. Conversely $a.\,\mathsf{1} \mathrel{\underset{\sim}{\sqsubset}}_{\mathsf{svr}} a.\,\mathsf{0}$ because $\mathsf{1}$ plays no role for processes acting a servers, while $a.\,\mathsf{1} \mathrel{\underset{\sim}{\not\sqsubset}}_{\mathsf{clt}} a.\,\mathsf{0}$; $a.\,\mathsf{1}$ as a client is satisfied by the server $\overline{a}.\,\mathsf{0}$ while $a.\,\mathsf{0}$ can never be satisfied as a client by any server. Behaviour relative to the client preorder $\mathrel{\underset{\sim}{\sqsubset}}_{\mathsf{clt}}$ is very sensitive to the presence of $\mathsf{1}$ and $\mathsf{0}$; for example $\mathsf{0}$ is a least element, that is $\mathsf{0} \mathrel{\underset{\sim}{\sqsubset}}_{\mathsf{clt}} r$ for any process $r$.[1] However in general the precise role these constants play is difficult to discern; for example, rather surprisingly we have $a.(b.\,\mathsf{0} + c.\,\mathsf{1}) + a.(b.\,\mathsf{1} + c.\,\mathsf{0}) \mathrel{\underset{\sim}{\sqsubset}}_{\mathsf{clt}} \mathsf{0}$.

If we ignore the distinction between servers and clients then every process plays an independent role as a *peer* to all other processes in its environment. This point of view leads to another behavioural preorder. Intuitively, we say that the process $p$ satisfies its peer $q$ if whenever they are executed in parallel both are guaranteed to be satisfied; in some sense both peers test their partner. Then $p_1 \mathrel{\underset{\sim}{\sqsubset}}_{\mathsf{p2p}} p_2$ means that every peer satisfied by $p_1$ is also satisfied by $p_2$.

This third refinement preorder is different from the server and client preorders. In fact we will show that $p_1 \mathrel{\underset{\sim}{\sqsubset}}_{\mathsf{p2p}} p_2$ implies $p_1 \mathrel{\underset{\sim}{\sqsubset}}_{\mathsf{clt}} p_2$; but the converse is not true in general. For example $\mathsf{1} + b.\,\mathsf{0} \mathrel{\underset{\sim}{\sqsubset}}_{\mathsf{clt}} \mathsf{1}$ but $\mathsf{1} + b.\,\mathsf{0} \mathrel{\underset{\sim}{\not\sqsubset}}_{\mathsf{p2p}} \mathsf{1}$ because of the peer $\overline{b}.\,\mathsf{1}$. In our formulation $\mathsf{1} + b.\,\mathsf{0}$ and $\overline{b}.\,\mathsf{1}$ mutually satisfy each other, whereas the peers $\mathsf{1}$ and $\overline{b}.\,\mathsf{1}$ do not.

The aim of the paper is to show that the theory of the standard (must) testing preorder [NH84,Hen88], here formulated as the server refinement preorder $\mathrel{\underset{\sim}{\sqsubset}}_{\mathsf{svr}}$, can be extended to both the client and the peer refinement preorders.

[1] Note in passing that this is not the case for the server preorder; $\mathsf{0}$ as a server guarantees the client $\overline{b}.\,\mathsf{0} + \tau.\,\mathsf{1}$ but the server $b.\,\mathsf{0}$ does not.

It is well-known that the behaviour of processes relative to $\sqsubseteq_{\mathsf{svr}}$ can be characterised in terms of the traces they can perform followed by *ready* or *acceptance* sets; intuitively each ready set $A$ after a trace $s$ captures a possibility for the process to deadlock when interacting with a client. For example the process $q = \tau.a.b.\,0 + \tau.a.c.\,0$ has the ready set $\{\, b \,\}$ after the (weak) sequence of actions $a$; this represents the possibility of $q$ deadlocking if servicing a client which requests an action $a$ but then is not subsequently interested in the action $b$. The process $p = a.(b.\,0 \,+\, c.\,0) \,+\, a.c.\,0$, also discussed above, has no comparable ready set and for this reason $p \not\sqsubseteq_{\mathsf{svr}} q$.

The first main result of the paper is a similar behavioural characterisation of both the client and the peer refinement preorders, in terms of certain kinds of traces and ready sets. However the details are intricate. It turns out that *unsuccessful* traces, those which can be performed without reaching a successful state, play an essential role. We also need to parametrise these concepts, relative to *usable* actions and *usable* processes; the exact meaning of *usable* will depend on the particular refinement preorder being considered.

It is also well-known that the standard testing preorders over finite processes can be characterised by a collection of (in-)equations over the process operators, [NH84,Hen88]. The second main result of the paper is a similar characterisation of the new refinement preorders. In fact there is a complication here, as these preorders are not in general preserved by the external operator $+$. A similar complication occurred in Section 7.2 of [Mil89] in the axiomatisation of *weak bisimulation equivalence*, and in the axiomatisations of the *must testing* preorder in [NH84], and we adopt the same solution. We give sound and complete (in-)equational theories for the largest pre-congruences $\sqsubseteq^{c}_{\mathsf{clt}}$, $\sqsubseteq^{c}_{\mathsf{p2p}}$ contained in the refinement preorders $\sqsubseteq_{\mathsf{clt}}$, $\sqsubseteq_{\mathsf{p2p}}$ respectively, over a finite version of $\mathsf{CCS}$. The presence of the success constant $1$ in this language complicates the axiomatisations considerably, as the behaviour of clients and peers is very dependent on their ability to immediately report success. For this reason we reformulate the axiomatisation of *must testing* preorder from [NH84], which in this paper coincides with the server preorder $\sqsubseteq^{c}_{\mathsf{svr}}$, as a two-sorted equational theory. The characterisation of the client and server preorders, $\sqsubseteq^{c}_{\mathsf{clt}}, \sqsubseteq^{c}_{\mathsf{svr}}$ respectively, requires extra equations to capture the behaviour of the special processes $1$ and $0$. For example one of the inequations required by the client preorder is $x \leq 1$, while those for the peer preorder include $\mu.(1 + x) \leq 1 + \mu.x$.

The remainder of this extended abstract is organised as follows. Section 2 is devoted to definitions and notation. We introduce a language for describing processes, an infinitary version of the $\mathsf{CCS}$ used in [Mil89], and give the standard intensional interpretation of it as a labelled transition system, LTS. For the remainder of the paper, processes will then be considered to be states in the resulting LTS. We also formally define the three different refinement preorders discussed informally in the Introduction, by generalising the standard notion from [NH84] of applying tests to processes. We begin Section 3 by recalling the well-known characterisation of the must preorder (Theorem 1) for finite branching LTSs from [NH84] in terms of traces and ready sets. To adapt this for the client preorder we need some extra technical notation. This is motivated by a series of examples, until we finally obtain a statement of the characterisation (Theorem 2).

The notation used in this characterisation of the client preorder can be modified in a uniform manner to give an analogous characterisation of the server preorder, (Theorem 3), which applies even in LTSs which are not finite-branching. Finally by combining these we get an analogous characterisation (Theorem 5) for the peer preorder.

In Section 4 we restrict our attention to a finite sub-language $\mathsf{CCS}^{\mathrm{f}}$ and address the question of equational characterisations. We first show why the client and peer refinement preorders are not preserved by the external choice operator $+$, and give a simple behavioural characterisation of the associated precongruences $\sqsubseteq^{\mathrm{c}}_{\mathsf{svr}}$, $\sqsubseteq^{\mathrm{c}}_{\mathrm{clt}}$ and $\sqsubseteq^{\mathrm{c}}_{\mathsf{p2p}}$; this simply involves taking into account the initial behaviour of processes. We then explain the equations which need to be added to the standard set in order to obtain completeness (Theorem 9 and Theorem 8). The paper ends with Section 5, where we present a summary of our results, a comparison with the existing work, and a series of open questions.

In this extended abstract all proofs are omitted. The proofs of the various behavioural characterisations from Section 3 will appear in [Ber13].

## 2 Testing processes

Let $\mathsf{Act}$ be a set of actions, ranged over by $a, b, c, \ldots$ and let $\tau$, $\checkmark$ be two distinct actions *not* in $\mathsf{Act}$; the first will denote internal unobservable activity while the second will be used to report the success of an experiment. To emphasise their distinctness we use $\mathsf{Act}_\tau$ to denote the set $\mathsf{Act} \cup \{\, \tau \,\}$, and similarly for $\mathsf{Act}_{\tau\checkmark}$; we use $\mu$ to range over the former and $\lambda$ to range over the latter. We assume $\mathsf{Act}$ has an idempotent complementation function, with $\overline{a}$ being the complement to $a$. A labelled transition system, LTS, consists of a triple $\langle\, P, \mathsf{Act}_{\tau\checkmark}, \longrightarrow \rangle$, where $P$ is a set of processes and $\longrightarrow \subseteq P \times \mathsf{Act}_{\tau\checkmark} \times P$ is a transition relation between processes decorated with labels drawn from the set $\mathsf{Act}_{\tau\checkmark}$. We use the infix notation $p \xrightarrow{\lambda} q$ in place of $(p, \lambda, q) \in \longrightarrow$. An LTS is finite-branching if for all $p \in P$ and for all $\lambda \in \mathsf{Act}_{\tau\checkmark}$, the set $\{\, q \mid p \xrightarrow{\lambda} q \,\}$ is finite. Single transitions $p \xrightarrow{\lambda} q$ are extended to sequences of transitions $p \xrightarrow{t} q$, where $t \in (\mathsf{Act}_{\tau\checkmark})^\star$, in the standard manner. 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 $\tau$s. Somewhat nonstandard is the use of infinite weak transitions, $p \overset{u}{\Longrightarrow}$, for $u \in (\mathsf{Act})^\infty$.

It will be convenient to have a notation for describing LTSs; we use an infinitary version of $\mathsf{CCS}$, [Mil89], augmented with a *success* operator, 1. The syntax of the language is depicted in Figure 1. We use $\mathsf{0}$ to denote the empty external sum $\sum_{i \in \emptyset} p_i$ and $p_1 + p_2$ for the binary sum $\sum_{i \in \{\, 1,2 \,\}} p_i$. If $I$ is a non-empty set, we use $\bigoplus_{i \in I} p_i$ to denote the sum $\sum_{i \in I} \tau.p_i$. For the remainder of the paper we use the LTS whose states are the terms in $\mathsf{CCS}$ and where the relations $p \xrightarrow{\lambda} q$ are the least ones determined by the (standard) rules in Figure 2. We use *finite*

$$p, q, r ::= 1 \mid A \mid \mu.p \mid \sum_{i\in I} p_i \mid p \,||\, q$$

where $I$ is a countable index set, and $A$ ranges over a set of definitional constants each of which has an associated definition $A \overset{\mathsf{def}}{=} p_A$.

**Fig. 1.** Syntax of infinitary $\mathsf{CCS}$.

$$\frac{}{\mathsf{1} \xrightarrow{\checkmark} \mathsf{0}}\ (\textsc{a-Ok}) \qquad \frac{}{\mu.p \xrightarrow{\mu} p}\ (\textsc{a-Pre})$$

$$\frac{p \xrightarrow{\lambda} p'}{p + q \xrightarrow{\lambda} p'}\ (\textsc{r-Ext-l}) \qquad \frac{q \xrightarrow{\lambda} q'}{p + q \xrightarrow{\lambda} q'}\ (\textsc{r-Ext-r})$$

$$\frac{q \xrightarrow{\lambda} q'}{q \,||\, p \xrightarrow{\lambda} q' \,||\, p}\ (\textsc{p-Left}) \qquad \frac{p \xrightarrow{\lambda} p'}{q \,||\, p \xrightarrow{\lambda} q \,||\, p'}\ (\textsc{p-Right})$$

$$\frac{q \xrightarrow{a} q' \quad p \xrightarrow{\overline{a}} p'}{q \,||\, p \xrightarrow{\tau} q' \,||\, p'}\ (\textsc{p-Synch}) \qquad \frac{p \xrightarrow{\lambda} p'}{A \xrightarrow{\lambda} p'}\ A \overset{\mathsf{def}}{=} p;\ (\textsc{r-Const})$$

**Fig. 2.** The operational semantics of $\mathsf{CCS}$

*branching* $\mathsf{CCS}$ to refer to the LTS which consists only of terms from $\mathsf{CCS}$ which generate finite branching structures.

A *computation* consists of series of $\tau$ actions of the form

$$p \,||\, r = p_0 \,||\, r_0 \xrightarrow{\tau} p_1 \,||\, r_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} p_k \,||\, r_k \xrightarrow{\tau} \ldots \tag{1}$$

It is *maximal* if it is infinite, or whenever $p_n \,||\, r_n$ is the last state then $p_n \,||\, r_n \not\xrightarrow{\tau}$. A computation may be viewed as two processes $p$, $r$, one a server and the other a client, co-operating to achieve individual goals, which may or may not be independent. We say (1) is *client-successful* if there exists some $k \geq 0$ such that $r_k \xrightarrow{\checkmark}$. It is *successful* if it is *client-successful* and there exists an $l \geq 0$ such that $p_l \xrightarrow{\checkmark}$. In a *client-successful* computation the client can report *success* while in a *successful* one both the client and the server can report success; note however that they are not required to do so at the same time.

**Definition 1 ( Passing tests ).** *We write $p$ $\mathsf{must}$ $r$ if every maximal computation from $p \,||\, r$ is* client-successful*. We write $p$ $\mathsf{must}^{\mathsf{p2p}}$ $r$ if every such computation is* successful. □

Intuitively, $p$ $\mathsf{must}$ $r$ means that the client $r$ is satisfied by the server $p$, as $r$ always reaches a state where it can report success. On the other hand, $p$ $\mathsf{must}^{\mathsf{p2p}}$ $r$

means that $p$ passes $r$ *and* $r$ also passes $p$; so $p$ and $r$ *have to collaborate* in order to pass each other. Thus, when using the testing relation $\mathsf{must}^{\mathsf{p2p}}$ we think of $p$ and $r$ as two peers rather than a server and a client.

**Definition 2 ( Testing preorders ).** *In an arbitrary LTS we write*

*(1)* $p_1 \sqsubseteq_{\mathsf{svr}} p_2$ *if for every* $r$*,* $p_1$ $\mathsf{must}$ $r$ *implies* $p_2$ $\mathsf{must}$ $r$
*(2)* $r_1 \sqsubseteq_{\mathsf{clt}} r_2$ *if for every* $p$*,* $p$ $\mathsf{must}$ $r_1$ *implies* $p$ $\mathsf{must}$ $r_2$
*(3)* $p \sqsubseteq_{\mathsf{p2p}} q$ *if for every* $r$*,* $p$ $\mathsf{must}^{\mathsf{p2p}}$ $r$ *implies* $q$ $\mathsf{must}^{\mathsf{p2p}}$ $r$*.*

*We use the obvious notation for the kernel of these preorders; for instance* $p_1 \eqsim_{\mathsf{p2p}} p_2$ *means that* $p_1 \sqsubseteq_{\mathsf{p2p}} p_2$ *and* $p_2 \sqsubseteq_{\mathsf{p2p}} p_1$*.* □

The preorder $\sqsubseteq_{\mathsf{svr}}$ is meant to compare servers, as $p_1 \sqsubseteq_{\mathsf{svr}} p_2$ ensures that all the clients passed (wrt $\mathsf{must}$) by $p_1$ are passed also by $p_2$. The preorder $\sqsubseteq_{\mathsf{clt}}$ relates processes seen as clients, because $r_1 \sqsubseteq_{\mathsf{clt}} r_2$ means that all the servers that satisfy $r_1$ satisfy also $r_2$. The third preorder, $\sqsubseteq_{\mathsf{p2p}}$, relates processes seen as *peers*; this follows from the fact that $p$ $\mathsf{must}^{\mathsf{p2p}}$ $r$ is true only if $p$ and $r$ mutually satisfy each other.

## 3 Semantic characterisations

The standard (must) testing preorder from [NH84,Hen88] has been characterised for finite-branching LTSs using two behavioural predicates. The first, $p \Downarrow s$, says that $p$ can never come across a divergent residual while executing the sequence of actions $s \in \mathsf{Act}^\star$. We use the notation $p \Downarrow$, $p$ *converges*, to mean that there is no infinite sequence $p \xrightarrow{\tau} p_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} p_k \xrightarrow{\tau} \ldots$. Then the general convergence predicate is defined inductively as follows:

(a) $p \Downarrow \varepsilon$ whenever $p \Downarrow$
(b) $p \Downarrow a.s$ whenever $p \Downarrow$ and $p \overset{a}{\Longrightarrow}$ implies $\bigoplus(p\ \mathsf{after}\ a) \Downarrow s$

where $(p\ \mathsf{after}\ s)$ denotes the set $\{\, p' \mid p \overset{s}{\Longrightarrow} p' \,\}$. Note that $p \overset{a}{\Longrightarrow}$ ensures that $(p\ \mathsf{after}\ a)$ is non-empty; thus $\bigoplus(p\ \mathsf{after}\ a)$ represents a (well-formed) process consisting of the choice between the elements of the non-empty set $(p\ \mathsf{after}\ a)$, which may in general be infinite. The second predicate codifies the possible deadlocks which may occur when a process $p$ attempts to execute the trace of actions $s \in \mathsf{Act}^\star$:

$$\mathsf{Acc}(p,s) = \{\, S(q) \mid p \overset{s}{\Longrightarrow} q \overset{\tau}{\not\longrightarrow} \,\} \tag{2}$$

where $S(q) = \{\, a \in \mathsf{Act} \mid q \xrightarrow{a} \,\}$. The sets $S(q)$ are called *ready sets*, while we say that $\mathsf{Acc}(p,s)$ is the *acceptance set* of $p$ *after* a trace $s$. Ready sets are essentially the complements of the *refusal sets* used in [Hoa85]. The sets in $\mathsf{Acc}(p,s)$ describe the interactions that can lead $p$ out of a possible deadlock, reached by executing the trace $s$ of external actions.

**Theorem 1** [DH87,Hen88] In finite branching $\mathsf{CCS}$, $p \sqsubseteq_{\mathsf{svr}} q$ if and only if, for every $s \in \mathsf{Act}^\star$, if $p \Downarrow s$ then (i) $q \Downarrow s$, (ii) for every $B \in \mathsf{Acc}(q, s)$ there exists some $A \in \mathsf{Acc}(p, s)$ such that $A \subseteq B$, and (iii) if $q \overset{s}{\Longrightarrow}$ then $p \overset{s}{\Longrightarrow}$. □

As might be expected, this behavioural characterisation does not work for $\sqsubseteq_{\mathsf{clt}}$:

*Example 1.* One can prove that $b.a.\,1 \sqsubseteq_{\mathsf{clt}} q$, where $q$ denotes $b.(c.\,0 + 1)$. However their acceptance sets are not related as required by Theorem 1. Calculations show that $\mathsf{Acc}(b.a.\,1, b) = \{\,\{\,a\,\}\,\}$. But $\{\,c\,\} \in \mathsf{Acc}(q, b)$ and so there is no set $B$ in $\mathsf{Acc}(b.a.\,1, b)$ satisfying $B \subseteq \{\,c\,\}$. □

In Example 1 we should not require the ready set $\{\,c\,\} \in \mathsf{Acc}(q, b)$ to be matched by one in $\mathsf{Acc}(b.a.\,1, b)$ because $q$ can report success immediately after performing $b$. We formalise this intuition. For every $s \in \mathsf{Act}^\star$ let $p \overset{s}{\Longrightarrow}_{\not\checkmark} q$ be the least relation satisfying

(1) $p \overset{\checkmark}{\not\longrightarrow}$ implies $p \overset{\varepsilon}{\Longrightarrow}_{\not\checkmark} p$

(2) if $p' \overset{s}{\Longrightarrow}_{\not\checkmark} q$ and $p \overset{\checkmark}{\not\longrightarrow}$ then
   – $p \xrightarrow{a} p'$ implies $p \overset{as}{\Longrightarrow}_{\not\checkmark} q$
   – $p \xrightarrow{\tau} p'$ implies $p \overset{s}{\Longrightarrow}_{\not\checkmark} q$

Intuitively, $p \overset{s}{\Longrightarrow}_{\not\checkmark} q$ means that $p$ can perform the sequence of external actions $s$ ending up in state $q$ without passing through any state which can report success; in particular neither $p$ nor $q$ can report success. This notation is extended to infinite traces, $u \in \mathsf{Act}^\infty$, by letting $p \overset{u}{\Longrightarrow}_{\not\checkmark}$ whenever there exists a $t \in (\mathsf{Act}_\tau)^\infty$ such that $t = \mu_1\mu_2\ldots$, (a) $p = p_0 \xrightarrow{\mu_1} p_1 \xrightarrow{\mu_2} p_2 \xrightarrow{\mu_3} \ldots$ implies that $p_i \overset{\checkmark}{\not\longrightarrow}$ for every $p_i$, and (b) for every $n \in \mathbb{N}$ and some $k \in \mathbb{N}$, $u_n = \langle t_k \rangle_{\backslash\tau}$; where $\langle t \rangle_{\backslash\tau}$ removes the $\tau$s from the string $t$.

**Definition 3.** *For every process $p$ and trace $s \in \mathsf{Act}^\star$, let*

$$\mathsf{Acc}_{\not\checkmark}(p, s) = \{\, S(q) \mid p \overset{s}{\Longrightarrow}_{\not\checkmark} q \overset{\tau}{\not\longrightarrow} \,\}$$

*We call the set $\mathsf{Acc}_{\not\checkmark}(p, s)$ the* unsuccessful *acceptance set of $p$ after $s$.* □

We can now try to adapt the characterisation for servers in Theorem 1 to clients as follows:

**Definition 4.** *Let $r_1 \preccurlyeq_{\mathsf{bad}} r_2$ if for every $s \in \mathsf{Act}^\star$, if $r_1 \Downarrow s$ then (i) $r_2 \Downarrow s$, and (ii) for every $B \in \mathsf{Acc}_{\not\checkmark}(r_2, s)$, there exists some $A \in \mathsf{Acc}_{\not\checkmark}(r_1, s)$ such that $A \subseteq B$.* □

*Example 2.* One can show that $r \sqsubseteq_{\mathsf{clt}} c.a.\,1$ where $r$ denotes the client $c.(a.\,1 + b.\,0)$. However they are not related by the proposed $\preccurlyeq_{\mathsf{bad}}$ in Definition 4. Obviously $r \Downarrow c$ and $\{\,a\,\} \in \mathsf{Acc}_{\not\checkmark}(c.a.\,1, c)$. But there is no $B \in \mathsf{Acc}_{\not\checkmark}(r, c)$ such that $B \subseteq \{\,a\,\}$; this is because $\mathsf{Acc}_{\not\checkmark}(r, c) = \{\,\{\,a, b\,\}\,\}$. The problem is the presence of $b$ in the ready set of $a.\,1 + b.\,0$. □

Intuitively, the action $b$ is *unusable* for $r$ after having performed the unsuccessful trace $c$; this is because performing $b$ leads to a client, $\mathsf{0}$, which is *unusable*, in the sense that it can never be satisfied by any server. When comparing ready sets after unsuccessful traces in Definition 4 we should ignore occurrences of *unusable* actions.

Let $\mathcal{U}\mathsf{clt} = \{ r \mid p \;\mathsf{must}\; r, \text{ for some server } p \}$. The set $\mathcal{U}\mathsf{clt}$ contains the usable clients, those satisfied by at least one server. We also need to consider the residuals of a client $r$ only after unsuccessful traces: for any process $r$ and $s \in \mathsf{Act}^\star$ let

$$(r \;\mathsf{after}_{\not\checkmark}\; s) = \{ q \mid r \xRightarrow{s}_{\not\checkmark} q \}$$

Now the set of usable actions for a client after $s$ can be defined as

$$\mathsf{ua}_{\mathsf{clt}}(r, s) = \{ a \in \mathsf{Act} \mid \bigoplus (r \;\mathsf{after}_{\not\checkmark}\; sa) \in \mathcal{U}\mathsf{clt} \} \qquad (3)$$

Thus if $a \in \mathsf{ua}_{\mathsf{clt}}(r, s)$ we know that the set of clients $(r \;\mathsf{after}_{\not\checkmark}\; sa)$ is non-empty, and the client given by the choice among those clients is usable; that is, there is some server which satisfies it.

*Example 3.* We revisit Example 2. Although $r$ can perform the sequence $cb$, $b$ is not in $\mathsf{ua}_{\mathsf{clt}}(r, c)$ because $(r \;\mathsf{after}_{\not\checkmark}\; cb)$ is the singleton set containing $\mathsf{0}$, which is not in $\mathcal{U}\mathsf{clt}$. Instead we have $\mathsf{ua}_{\mathsf{clt}}(r, c) = \{ a \}$.

If we amend Definition 4 by replacing the set inclusion $A \subseteq B$ with the more relaxed condition $A \cap \mathsf{ua}_{\mathsf{clt}}(r_1, s) \subseteq B$, it follows that $r \preccurlyeq_{\mathsf{bad}} c.a.\mathsf{1}$; thereby correctly reflecting the fact that $r \mathrel{\underset{\sim}{\sqsubset}}_{\mathsf{clt}} c.a.\mathsf{1}$. □

*Example 4.* In (3) above we must consider only the unsuccessful traces rather than all the traces. Consider the client $r = b.(\tau.(1 + a.\mathsf{0}) + \tau.a.\tau.\mathsf{1})$. First note that $\overline{b}.\overline{a}.\mathsf{0} \;\mathsf{must}\; r$ while $\overline{b}.\overline{a}.\mathsf{0} \;\not\mathsf{must}\; b.\mathsf{0}$ and therefore $r \not\mathrel{\underset{\sim}{\sqsubset}}_{\mathsf{clt}} b.\mathsf{0}$.

Now consider the consequences of using $\mathsf{after}$ rather than $\mathsf{after}_{\not\checkmark}$ in the definition (3) above. The amendment to the definition of $\preccurlyeq_{\mathsf{bad}}$ suggested in Example 3 would no longer be sound, as $r \preccurlyeq_{\mathsf{bad}} b.\mathsf{0}$ would be true.

This is because $(r \;\mathsf{after}\; ba)$ is the set $\{ \mathsf{0}, \mathsf{1} \}$ and so $\bigoplus (r \;\mathsf{after}\; ba)$ is the client $\tau.\mathsf{0} + \tau.\mathsf{1}$ , which is not in $\mathcal{U}\mathsf{clt}$. This leads to $\mathsf{ua}_{\mathsf{clt}}(r, b)$ being $\emptyset$, from which $r \preccurlyeq_{\mathsf{bad}} b.\mathsf{0}$ would follow. The incorrect reasoning involves the unsuccessful acceptance set after the trace $b$. $\mathsf{Acc}_{\not\checkmark}(b.\mathsf{0}, b) = \{ \emptyset \}$ and the unique ready set it contains, $\emptyset$, can be matched by $A \cap \emptyset$ for some $A \in \mathsf{Acc}_{\not\checkmark}(r, b)$, namely $A = \{ a \}$.

However with the correct definition (3) this reasoning no longer works as $\mathsf{ua}_{\mathsf{clt}}(r, b) = \{ a \}$. □

Unfortunately the amendment to Definition 4 suggested in Example 3 is still not sufficient to obtain a complete characterisation of the client preorder.

*Example 5.* Consider the clients $r_1 = a.(b.d.\mathsf{0} + b.\mathsf{1})$ and $r_2 = a.c.d.\mathsf{1}$. As $r_1$ is not usable $r_1 \mathrel{\underset{\sim}{\sqsubset}}_{\mathsf{clt}} r_2$, although $r_1 \not\preccurlyeq_{\mathsf{bad}} r_2$, even when $\preccurlyeq_{\mathsf{bad}}$ is amended as suggested in Example 3. To see this first note $\{ d \} \in \mathsf{Acc}_{\not\checkmark}(r_2, ac)$, and $r_1 \Downarrow ac$, although $r_1$ can not actually perform the sequence of actions $ac$; $r_1 \Downarrow ac$ merely says that if $r_1$ can perform any prefix of the sequence $ac$ to reach $r'$ then $r'$ must converge. Consequently $\mathsf{Acc}_{\not\checkmark}(r_1, ac)$ is empty and thus no ready set $B$ can be found to match the ready set $\{ d \}$. □

To fix this problem we need to reconsider when ready sets are to be matched. In Definition 4 this matching is moderated by the predicate $\Downarrow s$; for example $a.(\tau^\infty + b.\,1) \preccurlyeq_{\mathsf{bad}} a.c.d.\,1$, where $\tau^\infty$ denotes some process which does not converge. This is because $a.(\tau^\infty + b.\,1) \Downarrow a$ is false and therefore the ready set $\{\,c\,\} \in \mathsf{Acc}_{\not\checkmark}(a.c.d.\,1, a)$ does not have to be matched by $a.(\tau^\infty + b.\,1)$. However the client preorder is largely impervious to convergence/divergence. For example $1 \eqsim_{\mathsf{clt}} (1 + \tau^\infty)$.

It turns out that we have to moderate the matching of ready sets, not via the convergence predicate, but instead via *usability*.

For every $s \in \mathsf{Act}^\star$, the *client* usability after an unsuccessful trace $s$, denoted $\mathsf{usbl}_{\not\checkmark}\ s$, is defined by induction on $s$:

- $r\ \mathsf{usbl}_{\not\checkmark}\ \varepsilon$ if $r \in \mathcal{U}\mathsf{clt}$
- $r\ \mathsf{usbl}_{\not\checkmark}\ a.s$ if $r \in \mathcal{U}\mathsf{clt}$, and if $r \xRightarrow{a}_{\not\checkmark}$ then $\bigoplus(r\ \mathsf{after}_{\not\checkmark}\ a)\ \mathsf{usbl}_{\not\checkmark}\ s$

It is extended to infinite traces $u \in \mathsf{Act}^\infty$ in the obvious manner. Intuitively $r\ \mathsf{usbl}_{\not\checkmark}\ s$ means that any state reachable from $r$ by performing any subsequence of $s$ is usable. Note that only unsuccessful traces have to be taken into the account.

One can show that if $r_1 \sqsubseteq_{\mathsf{clt}} r_2$ and $r_1\ \mathsf{usbl}_{\not\checkmark}\ s$ then $r_2\ \mathsf{usbl}_{\not\checkmark}\ s$. In fact this predicate describes precisely when we expect ready sets of clients to be compared.

**Definition 5 ( Semantic client-preorder ).** *In any LTS, let $r_1 \precsim_{\mathsf{clt}} r_2$ if (1) for every $s \in \mathsf{Act}^\star$ such that $r_1\ \mathsf{usbl}_{\not\checkmark}\ s$, (a) $r_2\ \mathsf{usbl}_{\not\checkmark}\ s$, and (b) for every $B \in \mathsf{Acc}_{\not\checkmark}(r_2, s)$ there exists some $A \in \mathsf{Acc}_{\not\checkmark}(r_1, s)$ such that*

$$A \cap \mathsf{ua}_{\mathsf{clt}}(r_1, s) \subseteq B$$

*(2) for $w \in \mathsf{Act}^\star \cup \mathsf{Act}^\infty$ such that $r_1\ \mathsf{usbl}_{\not\checkmark}\ w$, $r_2 \xRightarrow{w}_{\not\checkmark}$ implies $r_1 \xRightarrow{w}_{\not\checkmark}$.* □

*Example 6.* Let us revisit the clients $r_1$, $r_2$, in Example 5. The client $b.d.\,0 + b.\,1$ is not usable; that is $b.d.\,0 + b.\,1 \notin \mathcal{U}\mathsf{clt}$ because it cannot be satisfied by any server. Consequently $r_1\ \mathsf{usbl}_{\not\checkmark}\ ac$ does not hold, and therefore when checking whether $r_1 \precsim_{\mathsf{clt}} r_2$ holds the ready set $\{\,d\,\} \in \mathsf{Acc}_{\not\checkmark}(r_2, ac)$ does not have to be matched by $r_1$.

Indeed it is now straightforward to check that $r_1 \precsim_{\mathsf{clt}} r_2$; the only $s \in \mathsf{Act}^\star$ for which $\mathsf{Acc}_{\not\checkmark}(r_2, s)$ is non-empty and $r_1\ \mathsf{usbl}_{\not\checkmark}\ s$ is the empty sequence $\varepsilon$. □

In general, and in particular in LTSs which are not finite branching, the condition on the existence of infinite computations in (2) does not follow from the condition on finite computations.

*Example 7.* Consider the process $q$ from Figure 3, where $q_k$ denotes a process which performs a sequence of $k$ $a$ actions followed by 1. Let $p$ be a similar process, but without the self loop. Then $p\ \mathsf{usbl}_{\not\checkmark}\ s$ and $q\ \mathsf{usbl}_{\not\checkmark}\ s$ for every $s$, and the pair $(p, q)$ satisfies condition (1) of $\precsim_{\mathsf{clt}}$, and condition (2) on finite $w$s. However

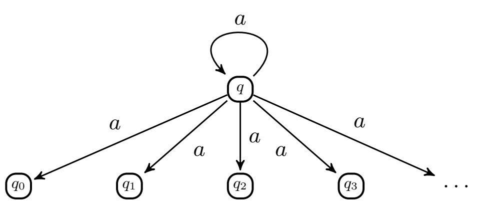

**Fig. 3.** Infinite traces

condition (2) on infinite $w$s is not satisfied: if $u$ denotes the infinite sequence of $a$s then $q \overset{u}{\Longrightarrow}_{\not\checkmark}$ but $p \overset{u}{\not\Longrightarrow}_{\not\checkmark}$.

In fact $p \not\sqsubseteq_{\mathsf{clt}} q$. For consider the process $A \overset{\mathsf{def}}{=} \overline{a}.A$. When $p$ is run as a test on $A$, or as a *client* using the *server* $A$, every computation is finite and successful; $A$ $\mathsf{must}$ $p$. However when $q$ is run as a test, there is the possibility of an infinite computation, the indefinite synchronisation on $a$, which is not successful; $A$ $\not\mathsf{must}$ $q$. □

**Theorem 2** In $\mathsf{CCS}$, $r_1 \sqsubseteq_{\mathsf{clt}} r_2$ if and only if $r_1 \precsim_{\mathsf{clt}} r_2$. □

The server-preorder $\sqsubseteq_{\mathsf{svr}}$ can be characterised behaviourally in manner dual to that of Definition 5, using the set of usable servers $\mathcal{U}\mathsf{svr} = \{ p \mid p \ \mathsf{must}\ r$, for some client $r \}$, the usable actions $\mathsf{ua}_{\mathsf{svr}}(p, s) = \{ a \in \mathsf{Act} \mid \bigoplus(p \ \mathsf{after}\ sa) \in \mathcal{U}\mathsf{svr} \}$, and the *server* convergence predicate $p \Downarrow_{\mathsf{svr}} s$, defined as the conjunction of $p \Downarrow s$ and a server usability predicate $p \ \mathsf{usbl}\ s$. This latter predicate is defined inductively in a manner similar to $\mathsf{usbl}_{\not\checkmark}\ s$, but over all traces $s$, rather than simply the unsuccessful ones.

**Definition 6 ( Semantic server-preorder ).** *In any LTS, let* $p \precsim_{\mathsf{svr}} q$ *if (1) for every* $s \in \mathsf{Act}^\star$ *such that* $p \Downarrow_{\mathsf{svr}} s$*, (a)* $q \Downarrow_{\mathsf{svr}} s$*, and (b) for every* $B \in \mathsf{Acc}(q, s)$ *there exists some* $A \in \mathsf{Acc}(p, s)$ *such that*

$$A \cap \mathsf{ua}_{\mathsf{svr}}(p, s) \subseteq B$$

*(2) for every* $w \in \mathsf{Act}^\star \cup \mathsf{Act}^\infty$ *such that* $p \Downarrow_{\mathsf{svr}} w$*,* $q \overset{w}{\Longrightarrow}$ *implies* $p \overset{u}{\Longrightarrow}$. □

**Theorem 3** In $\mathsf{CCS}$, $p \sqsubseteq_{\mathsf{svr}} q$ if and only if $p \precsim_{\mathsf{svr}} q$. □

This can be seen to be a generalisation of Theorem 1, as the server usability predicate $\mathcal{U}\mathsf{svr}$ is degenerate; it holds for every process, since any process used as a server trivially satisfies the degenerate client 1.

Let us now consider the peer preorder. The following result is hopeful:

**Proposition 4** In $\mathsf{CCS}$, $p \sqsubseteq_{\mathsf{p2p}} q$ implies $p \sqsubseteq_{\mathsf{clt}} q$. □

| | | | |
|---|---|---|---|
| **(S1a)** | $\mu.x_{\not\checkmark} + \mu.y = \mu.(\tau.x_{\not\checkmark} + \tau.y)$ | **(S3)** | $\mu.x + \tau.(\mu.y + z) = \tau.(\mu.x + \mu.y + z)$ |
| **(S1b)** | $\tau.x \leq \tau.\tau.x$ | **(S4)** | $\tau.x + \tau.y \leq x$ |
| **(S2)** | $x_{\not\checkmark} + \tau.y = \tau.(x_{\not\checkmark} + y) + \tau.y$ | **(S5)** | $\Omega \leq x$ |

**Fig. 4.** Standard inequations

Unfortunately, the peer preorder is *not* contained in the server preorder:

*Example 8.* It is easy to see that $a.\mathsf{0} \mathrel{\underset{\sim}{\sqsubset}}_{\mathsf{p2p}} b.\mathsf{0}$. This is true because $a.\mathsf{0}$ can never be satisfied, for it offers no $\checkmark$ at all. However, $a.\mathsf{0} \mathrel{\underset{\sim}{\not\sqsubset}}_{\mathsf{svr}} b.\mathsf{0}$, as the client $\overline{a}.\mathsf{1}$ is satisfied by $a.\mathsf{0}$, whereas $b.\mathsf{0} \;\not{\mathsf{m}}\mathsf{ust}\; \overline{a}.\mathsf{1}$. $\square$

Intuitively, the reason why $\mathrel{\underset{\sim}{\sqsubset}}_{\mathsf{p2p}} \not\subseteq \mathrel{\underset{\sim}{\sqsubset}}_{\mathsf{svr}}$ is that the server preorder does not take into account the requirement that servers should now act as peers; they should also be satisfied by their interactions with clients. To take this into account we introduce the usability of peers and amend the definition of $\precsim_{\mathsf{svr}}$ accordingly. In principle we should introduce the set of usable peers, $\mathcal{U}\mathsf{p2p} = \{\, p \mid p \; \mathsf{must}^{\mathsf{p2p}} \; r$ for some peer $r \,\}$. However, since $\mathcal{U}\mathsf{p2p}$ turns out to coincide with $\mathcal{U}\mathsf{clt}$, instead we define the peer convergence predicate by using the usability predicate of *clients*. For every $w \in \mathsf{Act}^{\star} \cup \mathsf{Act}^{\infty}$, let $p \Downarrow_{\mathsf{p2p}} w$ whenever $p \Downarrow w$ and $p \; \mathsf{usbl}_{\not\checkmark} \; w$.

**Definition 7.** *Let $p \precsim_{u\mathsf{svr}} q$ whenever (1) for every $s \in \mathsf{Act}^{\star}$, if $p \Downarrow_{\mathsf{p2p}} s$ then (a) $q \Downarrow_{\mathsf{p2p}} s$, and (b) for every $B \in \mathsf{Acc}(q, s)$ there exists some $A \in \mathsf{Acc}(p, s)$ such that*

$$A \cap \mathsf{ua}_{\mathsf{clt}}(p, s) \subseteq B$$

*(2) for every $w \in \mathsf{Act}^{\star} \cup \mathsf{Act}^{\infty}$, if $p \Downarrow_{\mathsf{p2p}} w$, and $q \overset{w}{\Longrightarrow}$, then $p \overset{w}{\Longrightarrow}$.* $\square$

**Definition 8 ( Semantic peer-preorder ).** *Let $p \precsim_{\mathsf{p2p}} q$ if $p \precsim_{\mathsf{clt}} q$ and $p \precsim_{u\mathsf{svr}} q$.*

Note that the definition of $p \precsim_{\mathsf{p2p}} q$ is **not** simply the conjunction of the client and server preorders from Definition 5 and Definition 6. It is essential that the usable set of peers $\mathcal{U}\mathsf{p2p}$ be employed.

**Theorem 5** In $\mathsf{CCS}$, $p \mathrel{\underset{\sim}{\sqsubset}}_{\mathsf{p2p}} q$ if and only if $p \precsim_{\mathsf{p2p}} q$. $\square$

## 4 Equational characterisation

We use $\mathsf{CCS}^{\mathsf{f}}$ to denote the finite sub-language of $\mathsf{CCS}$; this consists of all finite words constructed from the operators $\mathsf{0}$, $\mathsf{1}, +, \mu.\_$ for each $\mu \in \mathsf{Act}_{\tau}$, together with the special operator $\Omega$; this last denotes the term $\tau^{\infty}$ from $\mathsf{CCS}$ and its inclusion enables us to consider the algebraic properties of divergent processes. Our intention is to use equations, or more generally inequations, to characterise the three behavioural preorders $p \mathrel{\underset{\sim}{\sqsubset}}_{\star} q$ over this finite algebra, where $\star$ ranges

| | | | |
|---|---|---|---|
| (**Za**) | $\tau.0 \leq \Omega$ | (**Zb**) | $\mu.0 \leq 0$ |
| (**CLT1a**) | $x \leq 1$ | (**P2P1**) | $0 \leq 1$ |
| (**CLT1b**) | $1 \leq x + 1$ | (**P2P2**) | $\mu.(1+x) \leq 1 + \mu.x$ |
| (**CLT1c**) | $0 \leq \mu.1$ | (**P2P3**) | $\mu.(1+x) + \mu.(1+y) \leq \mu.(1+\tau.x + \tau.y)$ |

**Fig. 5.** Client and peer inequations

over svr, clt and p2p. Minor variations on standard equations, [Mil89], can be used for the other operators, such as parallel and hiding. For a given set of inequations $E$ we will use $p \sqsubseteq_E q$ to denote the fact that the inequation $p \leq q$ can be derived from $E$ using standard equational reasoning, while $p =_E q$ means that both $p \sqsubseteq_E q$ and $q \sqsubseteq_E p$ can be derived.

There are two immediate obstacles. The first is that these preorders are not pre-congruences for the language $\mathsf{CCS}^{\mathsf{f}}$; specifically they are not preserved by the choice operator $+$.

*Example 9.* Using the behavioural characterisation in Definition 8 it is easy to check that $0 \underset{\sim}{\sqsubset}_{\mathsf{p2p}} b.0$; in fact this is trivial because $0 \notin \mathcal{U}\mathsf{p2p}$. However $a.1 + 0 \not\underset{\sim}{\sqsubset}_{\mathsf{p2p}} a.1 + b.1$ because $\overline{a}.1 + \overline{b}.0 \;\mathsf{must}^{\mathsf{p2p}}\; a.1 + 0$ while $\overline{a}.1 + \overline{b}.0 \;\not\mathsf{must}^{\mathsf{p2p}}\; a.1 + b.0$; the latter follows because of the possible communication on $b$.

The same counter-example also shows that the other preorders are also not preserved. □

So in order to discuss equational reasoning we focus on the largest $\mathsf{CCS}^{\mathsf{f}}$ pre-congruence contained in $\underset{\sim}{\sqsubset}_{\star}$ which we denote by $\underset{\sim}{\sqsubset}^{\mathsf{c}}_{\star}$; by definition this is preserved by all the operators. But it is convenient to have alternative more amenable characterisations. To this end we let $p \underset{\sim}{\sqsubset}^{+}_{\star} q$ to mean that $a.1 + p \underset{\sim}{\sqsubset}_{\star} a.1 + q$ for some fresh action name $a$.

**Proposition 6** In $\mathsf{CCS}$, $p \underset{\sim}{\sqsubset}^{\mathsf{c}}_{\star} q$ if and only if $p \underset{\sim}{\sqsubset}^{+}_{\star} q$. □

Note that this is similar to the characterisation of observation-congruence in Section 7.2 of [Mil89]; the same technique is also used in [NH84].

The second obstacle is that the behavioural preorders are very sensitive to the ability of processes to immediately report success, with the result that many of the expected equations are not in general valid. For example the innocuous

$$a.\tau.x = a.x,$$

valid in the theories of [Mil89,NH84], is not in general satisfied by two of our behavioural theories. For example $a.1 \not\underset{\sim}{\sqsubset}^{+}_{\mathsf{p2p}} a.\tau.1$ because of the peer $\overline{a}.(1 + \Omega)$.

In order to have a more elegant presentation of the axioms we will use two sorts of variables, the standard $x, y, \ldots$ which may be instantiated with any process from $\mathsf{CCS}^{\mathsf{f}}$, and $x_{\not\checkmark}, y_{\not\checkmark}, \ldots$ which may only be instantiated by a process $p$

satisfying $p \not\xrightarrow{\checkmark}$; in $\mathsf{CCS}^{\mathrm{f}}$ such processes $p$ in fact have a simple syntactic characterisation. With this convention in mind consider the five *standard* inequations given in Figure 4, which are satisfied by all three behavioural preorders $\underset{\sim}{\sqsubset}^{+}_{\star}$. We also assume the standard equations for $(\mathsf{CCS}^{\mathrm{f}}, +, \mathsf{0})$ being a commutative monoid. Let **SVR** denote the set of inequations obtained by adding

$$(\textbf{SVR1}) \qquad \mathsf{1} = \mathsf{0}$$

Intuitively $\mathsf{1}$ has no significance for server behaviour; this extra equation captures this intuition and is sufficient to characterise the server preorder:

**Theorem 7** [Soundness and completeness for server-testing] In $\mathsf{CCS}^{\mathrm{f}}$, $p \underset{\sim}{\sqsubset}^{\mathrm{c}}_{\mathsf{svr}} q$ if and only $p \sqsubseteq_{\mathbf{SVR}} q$. $\Box$

In order to characterise the client and peer preorders we need to replace the equation **SVR1** with inequations which capture the significance of the operators $\mathsf{1}$ and $\mathsf{0}$ for clients and peers respectively. A sufficient set of inequations for clients is also given in Figure 4. Thus the client preorder has both a least element $\Omega$ from (**S5**), which by (**Za**) is also equivalent to $\tau.\mathsf{0}$, and a greatest element $\mathsf{1}$ from (**CLT1a**). Let **CLT** denote the resulting set of inequations.

**Theorem 8** [Soundness and Completeness for client-testing] In $\mathsf{CCS}^{\mathrm{f}}$, $p \underset{\sim}{\sqsubset}^{\mathrm{c}}_{\mathrm{clt}} q$ if and only $p \sqsubseteq_{\mathbf{CLT}} q$. $\Box$

Both the inequations (**Za**) and (**Zb**) remain valid for the peer preorder, but none of the unit inequations (**CLT1a**) - (**CLT1c**) are. They need to be replaced by unit inequations appropriate to peers. Let **P2P** denote the set obtained by replacing them with (**P2P1**) - (**P2P3**).

**Theorem 9** [Soundness and Completeness for peer-testing] In $\mathsf{CCS}^{\mathrm{f}}_{\mathrm{w}\tau}$, $p \underset{\sim}{\sqsubset}^{\mathrm{c}}_{\mathsf{p2p}} q$ if and only $p \sqsubseteq_{\mathbf{P2P}} q$. $\Box$

## 5 Conclusions

Much of the recent work on behavioural preorders for processes has been carried out using formalisms for contracts for web-services, proposed first in [CCLP06]. Spurred on by the recasting of the standard must preorder from [NH84] as a server-preorder between contracts, these ideas have been developed further in [LP07,CGP09,Bd10,Pad10].

In these publications the standard refinements are referred to as *subcontracts* or *sub-server* relations and [LP07,CGP09,Pad10,Bd10] contain a range of alternative characterisations. For example in [LP07,CGP09] the characterisations are coinductive and essentially rely on traces and ready sets; in [Bd10] the characterisation is coinductive and syntax-oriented.

To the best of our knowledge, the first paper to use a preorder for clients is [Bd10]. But their setting is much more restricted; they use so-called *session* *behaviours* which correspond to a much smaller class of processes than our language $\mathsf{CCS}$. As there are fewer contexts, their sub-server preorder differs from our server preorder: $a_1.\,1 \preceq_s a_1.\,1 + a_2.\,1$, whereas $a_1.\,1 \not\underset{\sim}{\sqsubset}_{\mathsf{svr}} a_1.\,1 + a_2.\,1$.

The refinements in the papers mentioned above depend on a *compliance* relation, rather than must testing; this is also why in [Bd10] the peer preorder $\preceq$: coincides with the intersection of the client and the server preorders; this is not the case for the must preorders (Example 8 can be tailored to the setting of session behaviours). Moreover, in a general infinite branching and non-deterministic LTS the refinements in the above papers differ from the preorder $\underset{\sim}{\sqsubset}_{\mathsf{svr}}$. The subcontract relation of [LP07] turns out to be not comparable with $\underset{\sim}{\sqsubset}_{\mathsf{svr}}$, whereas the strong subcontract $\sqsubseteq$ of [Pad10] is strictly contained in $\underset{\sim}{\sqsubset}_{\mathsf{svr}}$, as the LTS there is convergent and finite branching. The comparison of $\underset{\sim}{\sqsubset}_{\mathsf{svr}}$ with the refinement preorder of [CGP09] is complicated by their use of a non-standard LTS.

In [BMPR09] a symmetric refinement due to the compliance, $\sqsubseteq^{\mathsf{ds}}$, is studied; it differs from our peer preorder ($\underset{\sim}{\sqsubset}_{\mathsf{p2p}} \not\subseteq \sqsubseteq^{\mathsf{ds}}$), and its characterisation does not mention usability. This is because of the restrictions of the LTS in [BMPR09]. In more general settings the usability of contracts/services is crucial; [Pad11] talks of *viability*, while [MSV10] talks of *controllability*.

Also subcontracts/subtyping for peers inspired by the should/fair-testing of [RV07] have been proposed in [BZ09,BMPR09,Pad11]. In [BZ09] the fair-testing preorder is used as proof method for relating contracts, but no characterisation of their refinement preorder is given. A sound but incomplete characterisation is given in [BMPR09]. The focus of [Pad11] is on multi-party *session types* which, roughly speaking, cannot express all the behaviours of our language $\mathsf{CCS}$. In view of the restricted form of session types, they can give a syntax-oriented characterisation of their subtyping relation, $\leqslant$; this is in general incomparable with our $\underset{\sim}{\sqsubset}_{\mathsf{p2p}}$.

*Future work:* The most obvious open question about our two new refinement preorders $\underset{\sim}{\sqsubset}_{\mathsf{clt}}$ and $\underset{\sim}{\sqsubset}_{\mathsf{p2p}}$ is the development of algorithms for finite-state systems. The ability to check efficiently whether a process is *usable* will play an important role.

Another interesting question would be to characterise in some equational manner the refinement preorders $\underset{\sim}{\sqsubset}_{\mathsf{clt}}$, $\underset{\sim}{\sqsubset}_{\mathsf{p2p}}$ themselves rather than their associated pre-congruences $\underset{\sim}{\sqsubset}^{+}_{\mathsf{clt}}$ and $\underset{\sim}{\sqsubset}^{+}_{\mathsf{p2p}}$. In the resulting equational theory we would have to restrict in some way the form of reasoning allowed under the external choice operator $- + -$, but the extra inequations needed in such a proof system might be simpler.

We have confined our attention to refinement preorders based on must testing. But one can also define client and peer preorders based on the standard *may testing* of [NH84]. We believe that these refinement preorders can be completely characterised using a modified notion of *trace*, which takes into account the *usability* of residuals. Other variations on client and peer preorders are worth investigating: a "synchronous" formulation of $\underset{\sim}{\sqsubset}_{\mathsf{p2p}}$ where a computation is successful only if the peers report success *at the same time*; the client preorders for fair settings [Pad11,BZ09], or the ones based on the compliance [Pad10].

*Acknowledgements* The first author would like to acknowledge Vasileios Koutavas, for his help in unravelling the client preorder.

## References

Bd10. Franco Barbanera and Ugo de'Liguoro. Two notions of sub-behaviour for session-based client/server systems. In Temur Kutsia, Wolfgang Schreiner, and Maribel Fernández, editors, *PPDP*, pages 155–164. ACM, 2010.

Ber13. Giovanni Bernardi. *Behavioural Equivalences for Web Services*. PhD thesis, Trinity College Dublin, 2013. Available from https://www.scss.tcd.ie/∼bernargi.

BMPR09. Michele Bugliesi, Damiano Macedonio, Luca Pino, and Sabina Rossi. Compliance preorders for web services. In Cosimo Laneve and Jianwen Su, editors, *WS-FM*, volume 6194 of *Lecture Notes in Computer Science*, pages 76–91. Springer, 2009.

BZ09. Mario Bravetti and Gianluigi Zavattaro. Contract-based discovery and composition of web services. In Marco Bernardo, Luca Padovani, and Gianluigi Zavattaro, editors, *SFM*, volume 5569 of *Lecture Notes in Computer Science*, pages 261–295. Springer, 2009.

CCLP06. Samuele Carpineti, Giuseppe Castagna, Cosimo Laneve, and Luca Padovani. A formal account of contracts for web services. In Mario Bravetti, Manuel Núñez, and Gianluigi Zavattaro, editors, *WS-FM*, volume 4184 of *Lecture Notes in Computer Science*, pages 148–162. Springer, 2006.

CGP09. Giuseppe Castagna, Nils Gesbert, and Luca Padovani. A theory of contracts for web services. *ACM Trans. Program. Lang. Syst.*, 31(5):1–61, 2009. Supersedes the article in POPL '08.

DH87. Rocco De Nicola and Matthew Hennessy. Ccs without tau's. In Hartmut Ehrig, Robert A. Kowalski, Giorgio Levi, and Ugo Montanari, editors, *TAPSOFT, Vol.1*, volume 249 of *Lecture Notes in Computer Science*, pages 138–152. Springer, 1987.

Hen88. Matthew Hennessy. *Algebraic Theory of Processes*. The MIT Press, Cambridge, Mass., 1988.

Hoa85. C. A. R. Hoare. *Communicating sequential processes*. Prentice-Hall, 1985.

LP07. Cosimo Laneve and Luca Padovani. The must preorder revisited. In *Proceedings of the 18th international conference on Concurrency Theory*, pages 212–225, Berlin, Heidelberg, 2007. Springer-Verlag.

Mil89. R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

MSV10. Arjan J. Mooij, Christian Stahl, and Marc Voorhoeve. Relating fair testing and accordance for service replaceability. *J. Log. Algebr. Program.*, 79(3-5):233–244, 2010.

NH84. R. De Nicola and M. C. B. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34(1–2):83–133, November 1984.

Pad10. Luca Padovani. Contract-based discovery of web services modulo simple orchestrators. *Theor. Comput. Sci.*, 411(37):3328–3347, 2010.

Pad11. Luca Padovani. Fair Subtyping for Multi-Party Session Types. In *Proceedings of the 13th Conference on Coordination Models and Languages*, volume LNCS 6721, pages 127–141. Springer, 2011.

RV07. A. Rensink and W. Vogler. Fair testing. *Information and Computation*, 205(2):125–198, 2007.