MUTUALLY TESTING PROCESSES

GIOVANNI BERNARDI AND MATTHEW HENNESSY

School of Statistics and Computer Science, Lloyd Building, Trinity College, Dublin 2, Ireland
e-mail address: bernargi@tcd.ie

School of Statistics and Computer Science, O'Reilly Institute, Trinity College, Dublin 2, Ireland
e-mail address: matthew.hennessy@scss.tcd.ie

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

1998 ACM Subject Classification: [**Theory of computation**]: Program semantics. [**Software and its engineering**]: Formal software verification. [**Computer systems organization**]: Client-server architectures.

Key words and phrases: Software equivalences, testing theory, operational semantics, behavioural reasoning, equational reasoning, process calculi, usability.

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

mutual satisfaction of both. The resulting refinement preorder is denoted by $p \mathrel{\underset{\sim}{\sqsubset}}_{\mathsf{p2p}} q$. We give a uniform behavioural characterisation of all three refinement preorders in terms of traces and *acceptances sets* [NH84, Hen88]. We also give equational characterisations for a finite process calculus for servers/clients/peers.

We use an infinitary version of 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 $p \not\mathrel{\underset{\sim}{\sqsubset}}_{\mathsf{svr}} q$. 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 $a.\,\mathsf{1} + b.\,\mathsf{0} \not\mathrel{\underset{\sim}{\sqsubset}}_{\mathsf{svr}} a.\,\mathsf{1}$ 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 $a.\,\mathsf{1} \not\mathrel{\underset{\sim}{\sqsubset}}_{\mathsf{clt}} a.\,\mathsf{0}$; $a.\,\mathsf{1}$ 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 $\mathsf{1} + b.\,\mathsf{0} \not\mathrel{\underset{\sim}{\sqsubset}}_{\mathsf{p2p}} \mathsf{1}$ because of the peer $b.\,\mathsf{1}$. In our formulation $\mathsf{1} + b.\,\mathsf{0}$ and $b.\,\mathsf{1}$ mutually satisfy each other, whereas the peers $\mathsf{1}$ and $b.\,\mathsf{1}$ 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.

It is well-known that the behaviour of processes relative to $\mathrel{\underset{\sim}{\sqsubset}}_{\mathsf{svr}}$ 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 $q = \tau.a.b.\,\mathsf{0} + \tau.a.c.\,\mathsf{0}$ has the ready set

[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.

$\{ b \}$ 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 $p = a.(b.\,\mathsf{0} + c.\,\mathsf{0}) + a.c.\,\mathsf{0}$, 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 $\sqsubseteq^{c}_{\mathrm{clt}}$, $\sqsubseteq^{c}_{\mathsf{p2p}}$ contained in the refinement preorders $\sqsubseteq_{\mathsf{clt}}$, $\sqsubseteq_{\mathsf{p2p}}$ respectively, over a finite version of CCS. The presence of the success constant $\mathsf{1}$ 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, $\sqsubseteq^{c}_{\mathrm{clt}}, \sqsubseteq^{c}_{\mathsf{svr}}$ respectively, requires extra equations to capture the behaviour of the special processes $\mathsf{1}$ and $\mathsf{0}$. For example one of the inequations required by the client preorder is $x \leq \mathsf{1}$, while those for the peer preorder include $\mu.(\mathsf{1} + x) \leq \mathsf{1} + \mu.x$.

The remainder of the paper is organised as follows. Section 2 is devoted to definitions and notation. We introduce a language for describing processes, an infinitary version of the 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 3.1) 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(Theorem 3.12). The proof of this result is delegated to a separate subsequent section, Section 4. Meanwhile Section 3 continues by showing how 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 3.14), which applies even in LTSs which are not finite-branching. Finally by combining these we get an analogous characterisation (Theorem 3.19) for the peer preorder.

Section 4, which contains the details of the behavioural characterisation theorem for clients, is divided into three sub-sections. The first is devoted to some technical results concerning the relations used in the characterisation. The *soundness* of the characterisation is the topic of the next sub-section, Section 4.2, while the converse *completeness* is covered in the final sub-section.

Section 5 is similar in structure, but deals with the behavioural characterisation of the peer preorder.

In Section 6 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 pre-congruences $\sqsubseteq^{c}_{\mathsf{svr}}$, $\sqsubseteq^{c}_{\mathsf{clt}}$ and $\sqsubseteq^{c}_{\mathsf{p2p}}$; 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 an equational characterisation of the client and peer pre-congruences; These are stated in Theorem 6.7 and Theorem 6.9 respectively. The proof of the soundness of the equations is straightforward and is left to the reader. But the completeness is considerably more complex and the details are self-contained in a separate section, Section 7. This again is divided into three sub-sections. The first is devoted to the exposition of *normal-forms* which are crucial to the completeness proofs. This is followed by two sub-sections, dealing with the client preorder first, followed by the peer preorder.

The paper ends with Section 8, where we present a summary of our results, a comparison with the existing work, and a series of open questions.

The main results in sections 3, 5, and 6 were originally reported in the extended abstract [BH13], but there the proofs were omitted. This work extends [BH13], for it contains all the technical results (even the auxiliary ones), their proofs, more examples, and detailed discussions of the material.

2. Testing processes

Let Act be a set of actions, ranged over by $a, b, c, \dots$ and let τ, $\checkmark$ be two distinct actions *not* in 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 Act_τ to denote the set $\mathsf{Act} \cup \{\, \tau \,\}$, and similarly for $\mathsf{Act}_{\tau\checkmark}$; we use μ to range over the former and λ to range over the latter. We assume 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 τs. 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 CCS, [Mil89], augmented with a *success* operator, $\mathsf{1}$. 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 CCS and where the relations $p \xrightarrow{\lambda} q$ are the least ones determined by the (standard) rules

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

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

Figure 1: Syntax of infinitary CCS.

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

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

$$\frac{p \xrightarrow{\lambda} p'}{A \xrightarrow{\lambda} p'}\ A \overset{\text{def}}{=} p;\ (\textsc{r-Const})$$

Figure 2: The operational semantics of CCS

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

$$\frac{q \xrightarrow{a} q' \quad p \xrightarrow{\overline{a}} p'}{q \mid\mid p \xrightarrow{\tau} q' \mid\mid p'}\ (\textsc{p-Synch})$$

Figure 3: The operational semantics of contract composition

in Figure 2. We use *finite branching* CCS to refer to the LTS which consists only of terms from CCS which generate finite branching structures.

To model the interactions that take place between the server and the client contracts, we introduce a binary composition of contracts, $r \mid\mid p$, whose operational semantics is in Figure (3).

A *computation* consists of series of τ actions of the form

$$p \mid\mid r = p_0 \mid\mid r_0 \xrightarrow{\tau} p_1 \mid\mid r_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} p_k \mid\mid r_k \xrightarrow{\tau} \ldots \tag{2.1}$$

It is *maximal* if it is infinite, or whenever $p_n \mid\mid r_n$ is the last state then $p_n \mid\mid r_n \overset{\tau}{\not\longrightarrow}$. 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 (2.1) 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 2.1 (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 2.2 (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 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 $s \in \mathsf{Act}^{\star}$:

$$\mathsf{Acc}(p, s) = \{\, S(q) \mid p \overset{s}{\Longrightarrow} q \overset{\tau}{\not\longrightarrow} \,\} \tag{3.1}$$

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 3.1. [DH87, Hen88] *In finite branching* 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 3.2. *One can prove that $b.a.\,\mathit{1} \sqsubseteq_{\mathsf{clt}} q$, where q denotes $b.(c.\,\mathit{0} + \mathit{1})$. However their acceptance sets are not related as required by Theorem 3.1. Calculations show that $\mathsf{Acc}(b.a.\,\mathit{1}, b) = \{\{a\}\}$. But $\{c\} \in \mathsf{Acc}(q, b)$ and so there is no set B in $\mathsf{Acc}(b.a.\,\mathit{1}, b)$ satisfying $B \subseteq \{c\}$.*

In Example 3.2 we should not require the ready set $\{c\} \in \mathsf{Acc}(q, b)$ to be matched by one in $\mathsf{Acc}(b.a.\,\mathsf{1}, b)$ 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 τs from the string t.

Definition 3.3. 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 3.1 to clients as follows:

Definition 3.4. 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 3.5. *One can show that $r \sqsubseteq_{\mathsf{clt}} c.a.\,\mathit{1}$ where r denotes the client $c.(a.\,\mathit{1} + b.\,\mathit{0})$. However they are not related by the proposed $\preccurlyeq_{\mathsf{bad}}$ in Definition 3.4. Obviously $r \Downarrow c$ and $\{a\} \in \mathsf{Acc}_{\not\checkmark}(c.a.\,\mathit{1}, c)$. 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 $a.\,\mathit{1} + b.\,\mathit{0}$.*

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 3.4 we should ignore occurrences of *unusable* actions.

Let $\mathcal{U}\mathsf{clt} = \{\, r \mid p \ \mathsf{must}\ r,$ 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

$$(r \ \mathsf{after}_{\not\checkmark}\ s) = \{\, q \mid r \overset{s}{\Longrightarrow}_{\not\checkmark} q \,\} \tag{3.2}$$

The usability of a client, then, is parametrised over traces: for every $s \in \mathsf{Act}^\star$, the client usability along 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 $r \overset{a}{\Longrightarrow}_{\not\checkmark}$ 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.

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

$$\mathsf{ua}_{\mathsf{clt}}(r,s) = \{\, a \in \mathsf{Act} \mid r \overset{sa}{\Longrightarrow}_{\not\checkmark} \text{ implies } r \;\mathsf{usbl}_{\not\checkmark}\; sa \,\} \tag{3.3}$$

Example 3.6. *We revisit Example 3.5. 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 $\mathit{0}$, which is not in $\mathcal{U}\mathsf{clt}$. Instead we have $\mathsf{ua}_{\mathsf{clt}}(r,c) = \{\, a \,\}$.*

If we amend Definition 3.4 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 $r \preccurlyeq_{\mathsf{bad}} c.a.\,\mathit{1}$; thereby correctly reflecting the fact that $r \sqsubseteq_{\mathsf{clt}} c.a.\,\mathit{1}$.

Example 3.7. *In the definition of $\mathsf{usbl}_{\not\checkmark}$ above we must consider only the unsuccessful traces rather than all the traces. Consider the client $r = b.(\tau.(\mathit{1} + a.\,\mathit{0}) + \tau.a.\tau.\,\mathit{1})$. First note that $\overline{b}.\overline{a}.\,\mathit{0} \;\mathsf{must}\; r$ while $\overline{b}.\overline{a}.\,\mathit{0} \;\not{\mathsf{m}}\mathsf{ust}\; b.\,\mathit{0}$ and therefore $r \not\sqsubseteq_{\mathsf{clt}} b.\,\mathit{0}$.*

Now consider the consequences of using $\Longrightarrow$ and after rather than $\Longrightarrow_{\not\checkmark}$ and $\mathsf{after}_{\not\checkmark}$ in the definition of the usability along traces. The amendment to the definition of $\preccurlyeq_{\mathsf{bad}}$ suggested in Example 3.6 would no longer be sound, as $r \preccurlyeq_{\mathsf{bad}} b.\,\mathit{0}$ would be true.

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

However with the correct definition of $\mathsf{usbl}_{\not\checkmark}$ this reasoning no longer works as $\mathsf{ua}_{\mathsf{clt}}(r,b) = \{\, a \,\}$.

Unfortunately the amendment to Definition 3.4 suggested in Example 3.6 is still not sufficient to obtain a complete characterisation of the client preorder.

Example 3.8. *Consider the clients $r_1 = a.(b.d.\,\mathit{0}{+}b.\,\mathit{1})$ and $r_2 = a.c.d.\,\mathit{1}$. As r_1 is not usable $r_1 \sqsubseteq_{\mathsf{clt}} r_2$, although $r_1 \not\preccurlyeq_{\mathsf{bad}} r_2$, even when $\preccurlyeq_{\mathsf{bad}}$ is amended as suggested in Example 3.6. 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 3.4 this matching is moderated by the predicate $\Downarrow s$; for example $a.(\tau^\infty + b.\,\mathsf{1}) \preccurlyeq_{\mathsf{bad}} a.c.d.\,\mathsf{1}$, where τ^∞ denotes some process which does not converge. This is because $a.(\tau^\infty + b.\,\mathsf{1}) \Downarrow a$ is false and therefore the ready set $\{\, c \,\} \in \mathsf{Acc}_{\not\checkmark}(a.c.d.\,\mathsf{1}, a)$ does not have to be matched by $a.(\tau^\infty + b.\,\mathsf{1})$. However the client preorder is largely impervious to convergence/divergence. For example $\mathsf{1} \eqsim_{\mathsf{clt}} (\mathsf{1} + \tau^\infty)$.

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

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.

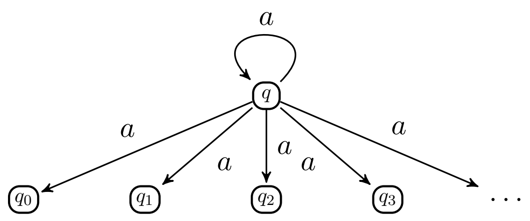

Figure 4: Infinite traces

Definition 3.9 (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 $r_1 \, \mathsf{usbl}_{\not\checkmark} \, w$, $r_2 \overset{w}{\Longrightarrow}_{\not\checkmark}$ implies $r_1 \overset{w}{\Longrightarrow}_{\not\checkmark}$.

Example 3.10. *Let us revisit the clients r_1, r_2, in Example 3.8. The client $b.d.\,\mathit{0}+b.\,\mathit{1}$ is not usable; that is $b.d.\,\mathit{0}+b.\,\mathit{1} \notin \mathcal{U}\mathsf{clt}$ 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 ε.

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 3.11. *Consider the process q from Figure 4, where q_k denotes a process which performs a sequence of k a actions followed by $\mathit{1}$. 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 ws. However condition (2) on infinite ws is not satisfied: if u denotes the infinite sequence of as 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 $A \overset{def}{=} \overline{a}.A$. 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; $A \, \not{\mathsf{m}}\mathsf{ust} \, q$.*

Theorem 3.12. *In* CCS*, $r_1 \sqsubseteq_{\mathsf{clt}} r_2$ if and only if $r_1 \precsim_{\mathsf{clt}} r_2$.*

Proof. It follows from Theorem (4.10) of Section 4. □

The server-preorder $\sqsubseteq_{\mathsf{svr}}$ can be characterised behaviourally in manner dual to that of Definition 3.9, using the set of usable servers $\mathcal{U}\mathsf{svr} = \{ p \mid p \, \mathsf{must} \, r, \text{ for some client } r \}$, the usable actions

$$\mathsf{ua}_{\mathsf{svr}}(p, s) = \{ a \in \mathsf{Act} \mid p \overset{s}{\Longrightarrow} \text{ implies } p \, \mathsf{usbl} \, sa \}$$

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 3.13 (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 $p \Downarrow_{\mathsf{svr}} w$, $q \xRightarrow{w}$ implies $p \xRightarrow{u}$.

Theorem 3.14. *In* CCS*, $p \sqsubseteq_{\mathsf{svr}} q$ if and only if $p \precsim_{\mathsf{svr}} q$.*

Proof. The standard argument of [Hen88] suffices, but for the condition on infinite traces, which we prove here.

Let $u = a_1 a_2 a_3 \ldots$, and $C_n \overset{\mathsf{def}}{=} \tau.\,\mathsf{1} + \overline{a}_n.C_{n+1}$ for every $n \in \mathbb{N}$. No C_n is successful, so the infinite computation of $C_0 \mid\mid p_2$ due to the trace u proves that $p_2 \;\not\!\mathsf{must}\; C_0$. The hypothesis $p_1 \sqsubseteq_{\mathsf{svr}} p_2$ implies that $p_1 \;\not\!\mathsf{must}\; C_0$. For a contradiction, suppose that $p_1 \overset{u}{\not\Longrightarrow}$. The assumption $p_1 \Downarrow_{\mathsf{svr}} u$ implies $p_1 \Downarrow u$, which in turn lets us prove that all the maximal computations of $C_0 \mid\mid p_1$ are client-successful. But this is not possible, as $p_1 \;\not\!\mathsf{must}\; C_0$. It follows that $p_1 \xRightarrow{u}$. □

This can be seen to be a generalisation of Theorem 3.1, 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 $\mathsf{1}$.

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

Proposition 3.15. *In* CCS*, $r_1 \sqsubseteq_{\mathsf{p2p}} r_2$ implies $r_1 \sqsubseteq_{\mathsf{clt}} r_2$.*

Proof. First note that using Theorem (3.14) one can prove that $\mathsf{1} + p \sqsubseteq_{\mathsf{svr}} p$ and that $p \sqsubseteq_{\mathsf{svr}} \mathsf{1} + p$.

Now suppose that p must r_1; it follows that $\mathsf{1} + p$ must r_1, and so $\mathsf{1} + p$ $\mathsf{must}^{\mathsf{p2p}}$ r_1 because $\mathsf{1} + p$ is trivially satisfied. The hypothesis imply that $\mathsf{1} + p$ $\mathsf{must}^{\mathsf{p2p}}$ r_2, thus $\mathsf{1} + p$ must r_2. In turn this ensures that p must r_2. □

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

Example 3.16. *It is easy to see that $a.\,0 \sqsubseteq_{\mathsf{p2p}} b.\,0$. This is true because $a.\,0$ can never be satisfied, for it offers no $\checkmark$ at all. However, $a.\,0 \not\sqsubseteq_{\mathsf{svr}} b.\,0$, as the client $\overline{a}.\,1$ is satisfied by $a.\,0$, whereas $b.\,0 \;\not\!\mathsf{must}\; \overline{a}.\,1$.*

Intuitively, the reason why $\sqsubseteq_{\mathsf{p2p}} \not\subseteq \sqsubseteq_{\mathsf{svr}}$ 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 3.17. Let $p \precsim_{u\mathsf{svr}} q$ whenever (1) for every $s \in \mathsf{Act}^\star$, if $p \Downarrow_{\mathsf{p2p}} s$ then (a) $q \Downarrow s$, 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 $q \xRightarrow{w}$, then $p \xRightarrow{w}$.

Definition 3.18 (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 3.9 and Definition 3.13. It is essential that the usable set of peers $\mathcal{U}\mathsf{p2p}$ be employed.

Theorem 3.19. *In* CCS, $p \underset{\sim}{\sqsubset}_{\mathsf{p2p}} q$ *if and only if* $p \precsim_{\mathsf{p2p}} q$.

Proof. See Section 5. □

4. Characterising the client behaviour

This section is devoted to the proof of behavioural characterisation of the client preorder, Theorem 3.12. For convenience it is divided into three subsections. The first gathers some preliminary technical properties of the various predicates used in the characterisation; the second is devoted to *soundness* and the final one to *completeness.*

4.1. **Preliminaries.** Here we collect some technical results on the interplay between the testing predicate $p \;\mathsf{must}\; r$ and the client and action usability predicates. The two corollaries below are the main results of the section.

Lemma 4.1. *Suppose* $p \;\mathsf{must}\; r$ *where* $p \overset{\overline{s}}{\Longrightarrow} q$. *Then* $r \overset{s}{\Longrightarrow}_{\not\checkmark} r'$ *implies* $q \;\mathsf{must}\; r'$.

Proof. Straightforward as any maximal computation from $q \mid\mid r'$ can be prefixed by an unsuccessful sequence of reduction steps to obtain a maximal computation from $p \mid\mid r$. □

Corollary 4.2. *Suppose* $p \;\mathsf{must}\; r$ *where* $p \overset{\overline{s}}{\Longrightarrow} q$. *Then* $r \;\mathsf{usbl}_{\not\checkmark}\; s$.

Proof. By induction on s. If s is the empty sequence ε then the result is immediate, as $p \;\mathsf{must}\; r$ ensures that $r \in \mathcal{U}\mathsf{clt}$. So assume s has the form $b.t$ and $r \overset{b}{\Longrightarrow}_{\not\checkmark}$. We have to show that $\bigoplus(r \;\mathsf{after}_{\not\checkmark}\; a) \;\mathsf{usbl}_{\not\checkmark}\; t$.

Let $p \overset{\overline{b}}{\Longrightarrow} p_b \overset{\overline{t}}{\Longrightarrow} q$. By the lemma we know $p_b \;\mathsf{must}\; r'$ whenever $r \overset{b}{\Longrightarrow}_{\not\checkmark} r'$. This in turn means that $p_a \;\mathsf{must}\; \bigoplus(r \;\mathsf{after}_{\not\checkmark}\; a)$. Now apply induction. □

Proposition 4.3. *Suppose* $p \;\mathsf{must}\; r$ *and* $p \overset{\overline{s.a}}{\Longrightarrow} q$. *Then* $r \overset{s}{\Longrightarrow}_{\not\checkmark} r' \overset{a}{\longrightarrow} r''$ *implies* $q \;\mathsf{must}\; r''$.

Proof. Suppose that $p \overset{\overline{s.a}}{\Longrightarrow} q$ and that $r \overset{s}{\Longrightarrow}_{\not\checkmark} \overset{a}{\longrightarrow} r''$. We prove $q \;\cancel{\mathsf{m}}\mathsf{ust}\; r''$ implies $p \;\cancel{\mathsf{m}}\mathsf{ust}\; r$.

Since $q \;\cancel{\mathsf{m}}\mathsf{ust}\; r''$ there must exist a maximal unsuccessful computation from

$$q'' \mid\mid r'' = q_0 \mid\mid r''_0 \overset{\tau}{\longrightarrow} q_1 \mid\mid r''_1 \overset{\tau}{\longrightarrow} q_1 \mid\mid r''_1 \ldots \tag{4.1}$$

such that $r''_k \overset{\checkmark}{\not\longrightarrow}$ for every $k \geq 0$. In particular $r'' \overset{\checkmark}{\not\longrightarrow}$.

The two derivations $p \overset{\overline{s.a}}{\Longrightarrow} q$ and can be zipped together $r \overset{s}{\Longrightarrow}_{\not\checkmark} r' \overset{a}{\longrightarrow} r''$ to obtain a computation

$$p \mid\mid r = p_0 \mid\mid r_0 \overset{\tau}{\longrightarrow} p_1 \mid\mid r_1 \overset{\tau}{\longrightarrow} \ldots p_n \mid\mid r_n = q \mid\mid r'' \tag{4.2}$$

Moreover here $r_i \overset{\checkmark}{\not\longrightarrow}$ for every $0 \leq i \leq n$.

Now the computation in (4.2) can be continued using the one in (4.1), leading to a maximal computation from $p \mid\mid r$ which is unsuccessful. It follows that $p \;\cancel{\mathsf{m}}\mathsf{ust}\; r$.

□

Corollary 4.4. *Suppose p must r where $p \overset{\overline{s.a}}{\Longrightarrow} q$ and $r \overset{s}{\Longrightarrow} r' \xrightarrow{a} r''$. Then $a \in \mathsf{ua}_{\mathsf{clt}}(r, s)$.*

Proof. If $r \overset{sa}{\not\Longrightarrow}_{\not\checkmark}$, then by definition (3.3) $a \in \mathsf{ua}_{\mathsf{clt}}(r, s)$. If $r \overset{sa}{\Longrightarrow}_{\not\checkmark}$, then we have to prove r $\mathsf{usbl}_{\not\checkmark}$ sa; the argument relies on the previous proposition and induction on s.

If s is empty then $\bigoplus(r \ \mathsf{after}_{\not\checkmark}\ a)$ $\mathsf{usbl}_{\not\checkmark}$ ε because of the previous proposition.

If $s = b.t$, then $p \overset{\bar{b}}{\Longrightarrow} p' \overset{\overline{t.a}}{\Longrightarrow} q$. An application of the previous proposition to $p \overset{\bar{b}}{\Longrightarrow} p'$ ensures that p' must r'' for every $r'' \in (r \ \mathsf{after}_{\not\checkmark}\ b)$, so p' must $\bigoplus(r \ \mathsf{after}_{\not\checkmark}\ b)$. As $p' \overset{\overline{ta}}{\Longrightarrow}$, induction implies that $\bigoplus(r \ \mathsf{after}_{\not\checkmark}\ b)$ $\mathsf{usbl}_{\not\checkmark}$ ta. □

Usability ensures that even when a client diverges, it can report success.

Lemma 4.5. *If $r \in \mathcal{U}\mathsf{clt}$ and $r \Uparrow$, then for every infinite reduction sequence $r = r_0 \xrightarrow{\tau} r_1 \xrightarrow{\tau} r_2 \xrightarrow{\tau} r_3 \xrightarrow{\tau} \ldots$, there exists an $n \in \mathbb{N}$ such that $r_n \xrightarrow{\checkmark}$.*

Proof. As $r \in \mathcal{U}\mathsf{clt}$ there exists a server p such that p must r. Fix a divergent computation of r, and zip it with p,

$$p \mid\mid r = p \mid\mid r_0 \xrightarrow{\tau} p \mid\mid r_1 \xrightarrow{\tau} p \mid\mid r_2 \xrightarrow{\tau} \ldots$$

The computation must be client-successful, so $r_n \xrightarrow{\checkmark}$ for some $n \in \mathbb{N}$. □

4.2. **Soundness.** Here we prove that the behavioural preorder in Definition 3.9 provides a sufficient set of conditions to capture the client-preorder. It is difficult to break the proof into a series of manageable independent results; instead we have one long monolithic proof.

Theorem 4.6 (Soundness client preorder)**.** *$r_1 \precsim_{\mathsf{clt}} r_2$ implies $r_1 \sqsubseteq_{\mathsf{clt}} r_2$.*

Proof. Fix a pair $r_1 \precsim_{\mathsf{clt}} r_2$, and let p must r_1; we have to show that all the maximal computations of the composition $r_2 \mid\mid p$ are client-successful. The argument is by contradiction, in that we show that if a maximal computation of $p \mid\mid r_2$ is not client successful, then also $p \mid\mid r_1$ performs a non client successful computation, so p $\not\mathsf{must}$ r_1.

Fix a maximal computation from $p \mid\mid r_2$,

$$p \mid\mid r_2 = p^0 \mid\mid r_2^0 \xrightarrow{\tau} p^1 \mid\mid r_2^1 \xrightarrow{\tau} p^3 \mid\mid r_2^3 \xrightarrow{\tau} p^4 \mid\mid r_2^4 \xrightarrow{\tau} \ldots \tag{4.3}$$

The computation in (4.3) is finite or infinite. We discuss the two cases separately.

Suppose that the computation is finite, and unzip it; the resulting contributions of p and r_2 are

$$r_2 \overset{s}{\Longrightarrow} r_2^k, \qquad p \overset{\bar{s}}{\Longrightarrow} p^k$$

for some $s \in \mathsf{Act}^\star$, and stable $r_2^k \mid\mid p^k$. The hypothesis p must r_1, $p \overset{\bar{s}}{\Longrightarrow}$, and Corollary 4.2 imply that r_1 $\mathsf{usbl}_{\not\checkmark}$ s. Suppose that the computation in (4.3) is not client successful, so no state in the contribution of r_2 reports success. It follows $r_2 \overset{s}{\Longrightarrow}_{\not\checkmark} r_2^k$, and as $r_2^k \overset{\tau}{\not\longrightarrow}$, $S(r_2^k) \in \mathsf{Acc}_{\not\checkmark}(r_2, s)$; so part (1b) of Definition 3.9 implies that $A \in \mathsf{Acc}_{\not\checkmark}(r_1, s)$, for some A such that $A \cap \mathsf{ua}_{\mathsf{clt}}(r_1, s) \subseteq S(r_2^k)$. Definition 3.3 implies that there exists a r_1' such that $S(r_1') = A$ and $r_1 \overset{s}{\Longrightarrow}_{\not\checkmark} r_1' \overset{\checkmark}{\not\longrightarrow}$. Zip together the contributions along s of p and r_1, the resulting computation reaches the state $r_1' \mid\mid p_k$; if this state is stable, then the computation is maximal and *not* client-successful, so p $\not\mathsf{must}$ r_1. This contradicts our assumption that p must r_1.

So it remains to show that $r'_1 \,||\, p_k$ is stable. Suppose, for a contradiction, that $p_k \xrightarrow{\overline{c}}$ and $r'_1 \xrightarrow{c}$, that is $c \in A$, for some action c. This situation matches the assumptions in Corollary 4.4 exactly, which gives that $c \in \mathsf{ua}_{\mathsf{clt}}(r_1, s)$. Since $A \cap \mathsf{ua}_{\mathsf{clt}}(r_1, s) \subseteq S(r_2^k)$ this in turn means that $r_2^k \xrightarrow{c}$, which contradicts the assumption that $p \,||\, r_2^k$ is stable.

We have discussed when the computation in (4.3) above is finite. Now let us suppose that it is infinite. As before unzip it.

Either p and r_2 perform infinite traces, or they perform finite traces and then (at least) one of them diverge.

If we are in the first case, then

$$r_2 \overset{u}{\Longrightarrow}, \qquad p \overset{\overline{u}}{\Longrightarrow}$$

The assumption p must r_1, the fact that $p \overset{\overline{u}}{\Longrightarrow}$, and Corollary 4.2 applied to every prefix of u, imply that r_1 $\mathsf{usbl}_{\not\checkmark}$ u. The proof that there is a successful term in $r_2 \overset{u}{\Longrightarrow}$ is by contradiction; for suppose that $r_2 \overset{u}{\Longrightarrow}_{\not\checkmark}$; then part (2) of Definition 3.9 implies that $r_1 \overset{u}{\Longrightarrow}_{\not\checkmark}$. By zipping $r_1 \overset{u}{\Longrightarrow}_{\not\checkmark}$ with $p \overset{\overline{u}}{\Longrightarrow}$ we obtain a maximal computation of $r_1 \,||\, p$ which is not client-successful; this implies that p $\cancel{\mathsf{must}}$ r_1, which contradicts our original assumption on p.

Suppose now that p and r_2 engage in a finite trace and then there is a divergence; by unzipping the computation in (4.3) we get the contributions

$$r_2 \overset{s}{\Longrightarrow} r_2^k, \qquad p \overset{\overline{s}}{\Longrightarrow} p^k$$

The assumption p must r_1, the fact that $p \overset{\overline{s}}{\Longrightarrow}$, and Corollary 4.2 imply that r_1 $\mathsf{usbl}_{\not\checkmark}$ s. Either p^k diverges or r_2^k diverges, or both diverge.

Suppose that p_k diverges. To prove that the computation in (4.3) is client-successful we reason by contradiction: suppose that there is no successful state among $r_2, \ldots, r_2^k$; this implies that r_2 performs the trace s unsuccessfully,

$$r_2 \overset{s}{\Longrightarrow}_{\not\checkmark}$$

Part (2) of Definition 3.9 ensures that $r_1 \overset{s}{\Longrightarrow}_{\not\checkmark} r'_1$. We zip the contribution of p with the unsuccessful transition of r_1; as p_k diverges the resulting computation is maximal,

$$p \,||\, r_1 \Longrightarrow p_k \,||\, r'_1 \Longrightarrow p_k \,||\, r'_1 \Longrightarrow \ldots \tag{4.4}$$

All the derivatives of r_1 in the maximal computation above are in $r_1 \overset{s}{\Longrightarrow}_{\not\checkmark} r'_1$, so they are not successful. It follows that the computation in (4.4) is not client-successful. However this contradicts the assumption p must r_1.

Finally suppose that r_2^k diverges. If there is a successful state in $r_2 \overset{s}{\Longrightarrow} r_2^k$ then the maximal computation we unzipped is client-successful. Therefore suppose that there is no successful state in the contribution of r_2, that is $r_2 \overset{s}{\Longrightarrow}_{\not\checkmark} r_2^k$. As r_1 $\mathsf{usbl}_{\not\checkmark}$ s, part (1a) of Definition 3.9 implies that r_2 $\mathsf{usbl}_{\not\checkmark}$ s. Now one can show that this implies that r_2^k $\mathsf{usbl}_{\not\checkmark}$ ε. So an application of Lemma 4.5 ensures that the unzipped computation is client-successful. □

4.3. **Completeness.** Here we show the converse of Theorem 4.6, which involves showing that the testing preorder $r_1 \sqsubseteq_{\mathsf{clt}} r_2$ implies the collection of properties gathered together in Definition 3.9. These in turn are quantified over all sequences $s \in \mathsf{Act}^\star$; we handle this quantification using induction over the length of s. First a technical lemma.

Lemma 4.7. *Suppose $r \in \mathcal{U}\mathsf{clt}$. Then $\mathsf{0}\ \cancel{\mathsf{m}}\mathsf{ust}\ r$ if and only if $r \overset{\varepsilon}{\Longrightarrow}_{\not\checkmark} r' \overset{\tau}{\not\longrightarrow}$ for some client r'.*

Proof. One direction is straightforward. For the converse suppose $\mathsf{0}\ \cancel{\mathsf{m}}\mathsf{ust}\ r$; we have to show that there exists some r' satisfying $r \overset{\varepsilon}{\Longrightarrow}_{\not\checkmark} r' \overset{\tau}{\not\longrightarrow}$.

Since $\mathsf{0}\ \cancel{\mathsf{m}}\mathsf{ust}\ r$ there must exist some unsuccessful maximal computation

$$\mathsf{0} \mid\mid r = \mathsf{0} \mid\mid r_o \xrightarrow{\tau} \dots \xrightarrow{\tau} \mathsf{0} \mid\mid r_k \xrightarrow{\tau} \dots \tag{4.5}$$

Suppose this is infinite. Then p must r can not be true for any server p, as $\mathsf{0}$ can be replaced in (4.5) by any p, to obtain an unsuccessful maximal computation from $p \mid\mid r$. This contracts the assumption that $r \in \mathcal{U}\mathsf{clt}$.

So (4.5) has to be finite, with terminal element $\mathsf{0} \mid\mid r_n$. The required r' is r_n. □

Proposition 4.8. *Suppose $r_1 \sqsubseteq_{\mathsf{clt}} r_2$ where $r_1 \in \mathcal{U}\mathsf{clt}$.*

(1) $r_2 \overset{a}{\Longrightarrow}_{\not\checkmark}$ implies $r_1 \overset{a}{\Longrightarrow}_{\not\checkmark}$
(2) For every $B \in \mathsf{Acc}_{\not\checkmark}(r_2, \varepsilon)$ there exists some $A \in \mathsf{Acc}_{\not\checkmark}(r_1, \varepsilon)$ such that $A \cap \mathsf{ua}_{\mathsf{clt}}(r_1, \varepsilon) \subseteq B$
(3) If $r_2 \overset{a}{\Longrightarrow}_{\not\checkmark}$ then $\bigoplus(r_1\ \mathsf{after}_{\not\checkmark}\ a) \sqsubseteq_{\mathsf{clt}} \bigoplus(r_2\ \mathsf{after}_{\not\checkmark})$

Proof. Throughout let p_1 be a server such that p_1 must r_1.

(1) Let $p = p_1 + \overline{a}.\tau^\infty$. As p diverges after the interaction on $\overline{a}$, and r_2 performs a without reaching successful states, $p\ \cancel{\mathsf{m}}\mathsf{ust}\ r_2$. The hypothesis imply $p\ \cancel{\mathsf{m}}\mathsf{ust}\ r_1$. In turn this ensures that $r_1 \overset{a}{\Longrightarrow}_{\not\checkmark}$, for otherwise the assumption on p_1 would imply that p must r_1.

(2) Let $\mathsf{Acc}_{\not\checkmark}(r_1, \varepsilon)$ be denoted by $\{\, A_i \mid i \in I \,\}$; note that the index set I may be empty, or indeed infinite. For convenience we use U to denote the set $\mathsf{ua}_{\mathsf{clt}}(r_1, \varepsilon)$. Suppose, for a contradiction, that there exists some $B \in \mathsf{Acc}_{\not\checkmark}(r_2, \varepsilon)$ such that for every $i \in I$ there is some action $a_i \in (A_i \cap U) \backslash B$. From the preceding lemma we therefore know that the index set I is non-empty. Let D_i denote the set $\{\, r' \mid r_1 \overset{\varepsilon}{\Longrightarrow}_{\not\checkmark} \xrightarrow{a_i} r' \,\}$, which we know to be non-empty because $a_i \in U$. We also know, because $a_i \in U$, that there is some server p_i satisfying p_i must r' for every $r' \in D_i$. This is true because either $r' \xrightarrow{\checkmark}$ for every $r' \in D_i$, or Definition (3.3) ensures that there exists a p_i such that p_i must $\bigoplus\{\, r' \in D_i \mid r' \overset{\checkmark}{\not\longrightarrow} \,\}$. Plainly p_i must r' for every $r' \in D_i$ such that $r' \xrightarrow{\checkmark}$, so the server p_i indeed satisfies all the clients in D_i.

Let $J = \{\, i \in I \mid r_1 \overset{a_i}{\not\Longrightarrow}_{\not\checkmark} \,\}$; this set contains the indexes of the actions that r_1 performs passing through a successful state. Now let p denote the server $\sum_{i \in I \backslash J} \overline{a_i}.p_i + \sum_{j \in J} \overline{a_j}.\,\mathsf{0}$.

(a) $p\ \cancel{\mathsf{m}}\mathsf{ust}\ r_2$. A finite unsuccessful maximal computation is ensured by the existence of B in $\mathsf{Acc}_{\not\checkmark}(r_2, \varepsilon)$.

(b) But p must r_1, which contradicts (a). To prove that p must r_1 is a little delicate. But consider a maximal computation

$$p \mid\mid r_1 = p^0 \mid\mid r^0 \xrightarrow{\tau} \dots p^k \mid\mid r^k \dots \tag{4.6}$$

If the server p remains untouched then the same sequence of clients can be used to construct a maximal computation from $p_1 \mid\mid r_1$; so some r^k must report success. On the other hand suppose p is touched. For example p^k is p while p^{k+1} is $\mathsf{0}$ or p_i for some $i \in I$. If no r^j, for $j \leq k$, reports success then $r^{k+1} \in D_i$, from which p_i must r^{k+1} follows, and p^{k+1} indeed equals p_i. This means again that (4.6) above is successful.

(3) For convenience let $\hat{r}_i$ denote $\bigoplus(r_i \;\mathsf{after}_{\not\checkmark}\; a)$, for $i = 1, 2$. Suppose p must $\hat{r}_1$. Then one can argue that $p_1 + \overline{a}.p$ must r_1. Since $r_1 \sqsubseteq_{\mathsf{clt}} r_2$ this ensures that $p_1 + \overline{a}.p$ must r_2. From this it is easy to see that p must r' for every r' in the non-empty set $r_2 \;\mathsf{after}_{\not\checkmark}\; a$. Now the required result, p must $\hat{r}_2$, follows.

□

The first part in the proposition above depends on the possibility of servers diverging. If divergence is not admitted, then one can prove that $a.\,\mathsf{1} \sqsubseteq_{\mathsf{clt}} a.\tau.\,\mathsf{1}$ (see Example 4.2.26 in [Ber13]). In absence of divergence, this would provide a counter example to part (1) above, for $a.\tau.\,\mathsf{1} \overset{a}{\Longrightarrow}_{\not\checkmark}$, whereas $a.\,\mathsf{1} \overset{a}{\not\Longrightarrow}_{\not\checkmark}$.

Unfortunately one property in Definition 3.9, that involving infinite sequences, requires special treatment.

Lemma 4.9. *Suppose $r_1 \sqsubseteq_{\mathsf{clt}} r_2$. If r_1 $\mathsf{usbl}_{\not\checkmark}$ u and $r_2 \overset{u}{\Longrightarrow}_{\not\checkmark}$, where $u \in \mathsf{Act}^\infty$, then $r_1 \overset{u}{\Longrightarrow}_{\not\checkmark}$.*

Proof. Let $u = a_1 a_2 a_3 \ldots$. To show that $r_1 \overset{u}{\Longrightarrow}_{\not\checkmark}$ we have to exhibit a $t \in \mathsf{Act}_\tau^\infty$ such that $t = \mu_1\mu_2\mu_3\ldots$ and

- $r_1 = r_1^0 \xrightarrow{\mu_1} r_1^1 \xrightarrow{\mu_2} r_1^2 \xrightarrow{\mu_3} \ldots$
- for every $n \in \mathbb{N}$, $u_n = \langle t_k \rangle_{\setminus\tau}$ for some $k \in \mathbb{N}$
- for every $n \in \mathbb{N}$, $r_1^n \overset{\checkmark}{\not\longrightarrow}$

The hypothesis r_1 $\mathsf{usbl}_{\not\checkmark}$ u ensures that for every u_k there is a p_k such that p_k must $\bigoplus(r_1 \;\mathsf{after}_{\not\checkmark}\; u_k)$. For every $k \in \mathbb{N}$, let $A_k \overset{\mathsf{def}}{=} p_k + \overline{a}_{k+1}.A_{k+1}$.

By zipping $r_2 \overset{u}{\Longrightarrow}_{\not\checkmark}$ with $A_0 \overset{\overline{u}}{\Longrightarrow}$ one sees that A_0 $\not\mathsf{must}$ r_2, for the client r_2 does not report success. In turn A_0 $\not\mathsf{must}$ r_1, so there exists a maximal computation of $r_1 \mid\mid A_0$ which is not client-successful. Given the construction of the A's and the P_k's, this is possible only if the computation is due to the infinite trace u. So $r_1 \overset{u}{\Longrightarrow}$, which ensures the first two properties above. As the computation is unsuccessful, $r_1^i \overset{\checkmark}{\not\longrightarrow}$ for every $i \in \mathbb{N}$. □

We have now gathered sufficient material to give the proof of completeness.

Theorem 4.10 (Completeness)**.** *$r_1 \sqsubseteq_{\mathsf{clt}} r_2$ implies $r_1 \precsim_{\mathsf{clt}} r_2$.*

Proof. We have to infer all the properties used in Definition 3.9. The property (2) for $w \in \mathsf{Act}^\infty$ follows directly from the preceding lemma. All other properties are parametrised on $s \in \mathsf{Act}^\star$; they can be inferred using induction on the length of s, and Proposition 4.8. Here we give one example, and the remaining ones can be established in a similar manner.

We show that r_1 $\mathsf{usbl}_{\not\checkmark}$ s implies r_2 $\mathsf{usbl}_{\not\checkmark}$ s. If s is the empty string this follows immediately. So suppose it has the form bt and r_1 $\mathsf{usbl}_{\not\checkmark}$ $b.t$; we have to prove that r_2 $\mathsf{usbl}_{\not\checkmark}$ $b.t$ follows. This requires establishing (a) $r_2 \in \mathcal{U}\mathsf{clt}$, which is a consequence of r_1 being in $\mathcal{U}\mathsf{clt}$ and (b) if $r_2 \overset{b}{\Longrightarrow}_{\not\checkmark}$ then $\bigoplus(r_2 \;\mathsf{after}_{\not\checkmark}\; b)$ $\mathsf{usbl}_{\not\checkmark}$ t. So suppose $r_2 \overset{b}{\Longrightarrow}_{\not\checkmark}$. But by part (3) of Proposition 4.8 we know that $\bigoplus(r_1 \ \mathsf{after}_{\not\checkmark}\ b) \mathrel{\eqsim_{\mathsf{clt}}} \bigoplus(r_2 \ \mathsf{after}_{\not\checkmark}\ b)$. Moreover unravelling the assumption $r_1 \ \mathsf{usbl}_{\not\checkmark}\ s$ gives that $\bigoplus(r_1 \ \mathsf{after}_{\not\checkmark}\ b) \ \mathsf{usbl}_{\not\checkmark}\ t$. The required result, $\bigoplus(r_2 \ \mathsf{after}_{\not\checkmark}\ b) \ \mathsf{usbl}_{\not\checkmark}\ t$, now follows by induction. □

5. Characterising the peer behaviour

In this section we are concerned with the behavioural characterisation of the peer preorder, Theorem (3.19). The material is organised in three subsections, where we respectively gather ancillary results, we prove the *soundness* of the characterisation, and then prove its *completeness*.

5.1. Preliminaries.

Corollary 5.1. *For every process p, $p \eqsim_{\mathsf{svr}} p + \sum_{i \in I} 1$.*

Proof. This follows from Theorem 3.14 and the fact that adding 1 to the terms has no impact on their traces and acceptance sets. □

Lemma 5.2. *If $r \in \mathcal{U}\mathsf{clt}$, then there exists a p such that $p \ \mathsf{must}\ r$, $p \overset{\checkmark}{\not\longrightarrow}$ and $p \overset{\tau}{\not\longrightarrow}$.*

Proof. We prove that if $r \in \mathcal{U}\mathsf{clt}$ then there exists a process p such that $p \overset{\checkmark}{\not\longrightarrow}$. The assumption that $r \in \mathcal{U}\mathsf{clt}$ implies that there exists a p such that $p \ \mathsf{must}\ r$. If $p \overset{\checkmark}{\not\longrightarrow}$ there is nothing more to prove. If $p \xrightarrow{\checkmark}$, then $p = p' + \sum_{i \in I} 1$ for some non empty set I and p' such that $p' \overset{\checkmark}{\not\longrightarrow}$. Corollary 5.1 implies that $p' \eqsim_{\mathsf{svr}} p$, so $p' \ \mathsf{must}\ r$.

Now we prove that there exists a process p such that $p \ \mathsf{must}\ r$ and $p \overset{\tau}{\not\longrightarrow}$. The assumption that $r \in \mathcal{U}\mathsf{clt}$ implies that there exists a p such that $p \ \mathsf{must}\ r$. If $p \overset{\tau}{\not\longrightarrow}$, then there is nothing more to prove. If $p \xrightarrow{\tau}$, then either p diverges or p converges. If p diverges, then $p \ \mathsf{must}\ r$ implies that $r \xrightarrow{\checkmark}$. It follows that $0 \ \mathsf{must}\ r$; as $0 \overset{\tau}{\not\longrightarrow}$, the process 0 suits our aims.

If p converges then there exists a stable p' such that all the maximal computation of $r \mid\mid p'$ are extensions of the computation $r \mid\mid p \Longrightarrow r \mid\mid p'$. Since $p \ \mathsf{must}\ r$, it follows that all the maximal computations of $r \mid\mid p'$ are client-successful, and so $p' \ \mathsf{must}\ r$. □

The next lemma tells what it means for a process r to be usable along an unsuccessful trace s.

Lemma 5.3. *For every process r and trace s, if $r \ \mathsf{usbl}\ s$ then for every s' prefix of s if $r \overset{s'}{\Longrightarrow}_{\not\checkmark}$ there exists a server p such that $p \ \mathsf{must} \bigoplus(r \ \mathsf{after}_{\not\checkmark}\ s')$.*

Proof. As $r \ \mathsf{usbl}\ s$, $p \ \mathsf{must}\ r$ for some p.

We reason by induction on s. In the base case ($s = \varepsilon$) observe that $p \ \mathsf{must} \bigoplus(r \ \mathsf{after}\ \varepsilon)$.

In the inductive case $s = a.s'$. Fix a prefix s' of s such that $r \overset{s'}{\Longrightarrow}_{\not\checkmark}$; we have to show a server p which must pass $\bigoplus(r \ \mathsf{after}_{\not\checkmark}\ s')$. If s' is empty we reason as in the base case. Suppose $s' = a.s''$. As $r \overset{s'}{\Longrightarrow}_{\not\checkmark}$, the term $\hat{r} = \bigoplus(r \ \mathsf{after}_{\not\checkmark}\ a)$ is well-defined, and the hypothesis $r \ \mathsf{usbl}\ s$ ensures that $\hat{r} \ \mathsf{usbl}\ s''$. The inductive hypothesis ensures that for every $\hat{s}$ prefix of s'', if $\hat{r} \overset{\hat{s}}{\Longrightarrow}_{\not\checkmark}$ then there exists a server p such that $p \ \mathsf{must} \bigoplus(\hat{r} \ \mathsf{after}\ \hat{s})$.

The equality $\bigoplus(\hat{r} \text{ after } s'') = \bigoplus(r \text{ after}_{\not\checkmark} s')$ implies that there exists a p such that $p \text{ must } \bigoplus(r \text{ after}_{\not\checkmark} s')$. □

The next result gives a proof method for the predicate $\Downarrow$. This proof method is based on the convergence of the residuals of processes after traces.

Lemma 5.4. *If for every s' prefix of s, $p \overset{s'}{\Longrightarrow} p'$ implies $p' \Downarrow$, then $p \Downarrow s$.*

Proof. Let us assume that for every s' prefix of s, $p \overset{s'}{\Longrightarrow} p'$ implies $p' \Downarrow$. The string ε is a prefix of every string s, so the assumption and $p \overset{\varepsilon}{\Longrightarrow} p$ imply that $p \Downarrow$.

We proceed by induction. As $p \Downarrow$ the base case is true. In the inductive case $s = a.s'$ and either $p \overset{a}{\not\Longrightarrow}$ or $p \overset{a}{\Longrightarrow}$. In the first case $p \Downarrow a.s'$ follows, while in the second case $\bigoplus(p \text{ after } a) \overset{s'}{\Longrightarrow}$. Induction on s' implies that $\bigoplus(p \text{ after } a) \Downarrow s'$, and so $p \Downarrow a.s'$.

□

5.2. **Soundness.** Our aim in this section is to prove that the peer preorder contains the behavioural preorder of Definition 3.18. Roughly speaking, the proof is a combination of the standard arguments that show the soundness of the server preorder [Hen88], with the arguments on usability that we used to prove the soundness of the client preorder, Theorem 4.6. Much in the same style of Section 4.2, the proof is monolithic.

Theorem 5.5 (Soundness peer)**.** *$p \precsim_{\mathsf{p2p}} q$ implies $p \sqsubseteq_{\mathsf{p2p}} q$.*

Proof. Fix two processes p and q such that $p \precsim_{\mathsf{svr}} q$. We are required to show that $p \sqsubseteq_{\mathsf{p2p}} q$, that is $p \text{ must}^{\mathsf{p2p}} r$ implies $q \text{ must}^{\mathsf{p2p}} r$ for every process r. Fix a process r such that $p \text{ must}^{\mathsf{p2p}} r$; we explain why all the maximal computations of $q \mid\mid r$ are *successful*.

The definition of $\precsim_{\mathsf{p2p}}$ ensures that $p \precsim_{\mathsf{clt}} q$, so Theorem 4.6 implies that $p \sqsubseteq_{\mathsf{clt}} q$. The assumption $p \text{ must}^{\mathsf{p2p}} r$ ensures that $r \text{ must } p$, thus $r \text{ must } q$. It follows that all the maximal computations of $q \mid\mid r$ are client-successful, that is q reach a successful state.

What is left to prove is that the maximal computations of $q \mid\mid r$ contain a state $q' \mid\mid r'$ wherein $r' \xrightarrow{\checkmark}$.

Fix a maximal computation of $q \mid\mid r$,

$$q \mid\mid r = q_0 \mid\mid r_0 \xrightarrow{\tau} q_1 \mid\mid r_1 \xrightarrow{\tau} q_2 \mid\mid r_2 \xrightarrow{\tau} \dots \tag{5.1}$$

Unzip the computation above. We obtain the contributions

$$q \overset{w}{\Longrightarrow}, \quad r \overset{\overline{w}}{\Longrightarrow}$$

for some possibly infinite w.

The argument now depends on p. Either $p \not\Downarrow w$ or $p \Downarrow w$.

In the first case p performs a prefix of w, say s, and reaches a state p' that diverges: $p \overset{s}{\Longrightarrow} p' \xrightarrow{\tau} p' \xrightarrow{\tau} \dots$. Zip this diverging trace of p with a prefix of the trace $r \overset{\overline{s}}{\Longrightarrow}$, and let p' diverge. The result is an infinite (i.e. maximal) computation of $p \mid\mid r$ that contains a successful derivative of r, because $p \text{ must}^{\mathsf{p2p}} r$. The successful derivative of r appears also in (5.1) above.

Suppose now that $p \Downarrow w$.

The computation in (5.1) above is either finite or infinite. Suppose it is finite. Then the contributions are

$$q \overset{s}{\Longrightarrow} q_k, \quad r \overset{\overline{s}}{\Longrightarrow} r_k$$

where $q_k \parallel r_k \overset{\tau}{\not\longrightarrow}$ and $s = w$. The last fact ensures that $p \Downarrow s$. Since $r \overset{\overline{s}}{\Longrightarrow} r_k$, and $p \ \mathsf{must}^{\mathsf{p2p}}\ r$ implies $r \ \mathsf{must}\ p$, Corollary 4.2 guarantees that $p \ \mathsf{usbl}_{\not\checkmark}\ s$.

Note that $q \overset{s}{\Longrightarrow} q_k \overset{\checkmark}{\not\longrightarrow}$ ensures $S(q_k) \in \mathsf{Acc}(q, s)$. As $p \ \mathsf{usbl}_{\not\checkmark}\ s$ and $p \Downarrow s$, we know $p \Downarrow_{\mathsf{p2p}} s$, so part (1b) of Definition 3.17 implies that there exists a set $A \in \mathsf{Acc}(p, s)$ such that $A \cap \mathsf{ua}_{\mathsf{clt}}(p, s) \subseteq S(q_k)$. In turn this means that there exists a stable p' such that $S(p') = A$ and $p \overset{s}{\Longrightarrow} p'$. Consider the computation $p \parallel r \Longrightarrow p' \parallel r_k$. If the state $p' \parallel r_k$ is stable, then the computation is maximal, thus $p \ \mathsf{must}^{\mathsf{p2p}}\ r$ ensures that one of the derivatives of r is successful. This derivative appears also in (5.1) above.

We have to prove that $p' \parallel r_k \overset{\tau}{\not\longrightarrow}$. The reasoning here is analogous to the one used in Theorem 4.6 to show that the state $r'_1 \parallel p_k$ is stable, and relies on Corollary 4.2.

Thus far we have proven that if (5.1) above is finite, then r reaches a successful state. This is the case also if the computation is infinite. Let us see why.

Either q and r engage in infinite traces, or (at least) one of them diverge.

Suppose that the contributions obtained by by unzipping the computation in (5.1) are infinite

$$q \overset{u}{\Longrightarrow}, \quad r \overset{\overline{u}}{\Longrightarrow} \tag{5.2}$$

with $u = w$. We have to show that one of the derivatives of r is successful.

As $r \overset{\overline{u}}{\Longrightarrow}$ and $p \ \mathsf{must}^{\mathsf{p2p}}\ r$, Corollary 4.2 applied to every finite prefix of u implies that $p \ \mathsf{usbl}_{\not\checkmark}\ u$. As $p \Downarrow u$ it follows that $p \Downarrow_{\mathsf{p2p}} u$. Since $q \overset{u}{\Longrightarrow}$, part (2) of Definition 3.17 implies that $p \overset{u}{\Longrightarrow}$. Zip this infinite trace of p with $r \overset{\overline{u}}{\Longrightarrow}$. The resulting computation of $p \parallel r$ is infinite as well, so the assumption $p \ \mathsf{must}^{\mathsf{p2p}}\ r$ ensures that r reaches a successful state. This state appears in (5.1) above.

Now we discuss the case of (5.1) being due to finite traces and divergence of q or r. To unzip (5.1) gives the following contributions,

$$q \overset{s}{\Longrightarrow} q_k, \quad r \overset{\overline{s}}{\Longrightarrow} r_k$$

with $w = s$. Note that $p \Downarrow s$. The fact that $r \overset{\overline{s}}{\Longrightarrow} r_k$ implies $p \ \mathsf{usbl}_{\not\checkmark}\ s$, so $p \Downarrow_{\mathsf{p2p}} s$. Part (1a) of Definition 3.17 implies that $q \Downarrow s$, so the divergent process must be r_k.

Part (2) of Definition 3.17, $q \overset{s}{\Longrightarrow}$, and $p \Downarrow_{\mathsf{p2p}} s$ imply that $p \overset{s}{\Longrightarrow}$. Zipping this trace of p with the trace $r \overset{\overline{s}}{\Longrightarrow} r_k$, and let r_k diverge. The resulting computation of $p \parallel r$ is infinite, so one of the derivatives of r in it is successful; this is true because of the assumption $p \ \mathsf{must}^{\mathsf{p2p}}\ r$. This successful derivative of r appears also in (5.1) above. $\square$

5.3. **Completeness.** This section contains the proof that the behavioural characterisation given in Definition 3.18 is complete with respect to the peer preorder. This result is the converse inclusion of Theorem 5.5.

In view of Proposition 3.15, the bulk of the work is to prove that the peer preorder is contained in the behavioural preorder $\precsim_{u\mathsf{svr}}$, Proposition 5.11.

In Section 4 we have proven a similar result for the client preorder and its characterisation, Theorem 3.12. Our reasoning there is inductive, and relies on the property proven in part (3) of Proposition 4.8. That property is not true for the peer preorder and traces, so here we will reason using techniques analogous to the standard ones of [Hen88].

Example 5.6. *It is not true that if $p \precsim_{\mathsf{p2p}} q$ and $q \overset{a}{\Longrightarrow}$ for some action a, then $\bigoplus(p \text{ after } a) \precsim_{\mathsf{p2p}} \bigoplus(q \text{ after } a)$. An example are the peers $p = a.\,1$ and $q = 1 + a.\,0$. First, the inequality $p \precsim_{\mathsf{p2p}} q$ is true, because the peers p and q engage exactly in the same interactions, and the latter is trivially satisfied (i.e. $q \xrightarrow{\checkmark}$). Second, $\bigoplus(p \text{ after } a) = \tau.\,1$ and $\bigoplus(q \text{ after } a) = \tau.\,0$. A peer that witnesses $\tau.\,1 \not\precsim_{\mathsf{p2p}} \tau.\,0$ is 1, for $\tau.\,1 \ \mathsf{must}^{\mathsf{p2p}}\ 1$, while $\tau.\,0 \ \not{\mathsf{m}}\mathsf{ust}^{\mathsf{p2p}}\ 1$.*

The remaining part of the section essentially shows what properties typical of server behaviours are enjoyed by peers. During *unsuccessful* execution of traces, peers behave at the same time as clients and servers, whereas after reporting success they behave only as servers. The tests that we will use in the oncoming proofs witness this intuition, for they are a combination of the tests used to reason on the client and on the server behaviours.

Lemma 5.7. *if $p \precsim_{\mathsf{p2p}} q$, $p \Downarrow_{\mathsf{p2p}} s$, and $q \overset{s}{\Longrightarrow} q'$, then $q' \Downarrow$.*

Proof. It is enough to show a peer C such that $q \ \mathsf{must}^{\mathsf{p2p}}\ C$ and $C \overset{s}{\Longrightarrow}_{\not\checkmark} \hat{C}$, for some $\hat{C}$. These facts imply that if $q \overset{s}{\Longrightarrow} q'$ then $q' \Downarrow$. This is true for otherwise there exists a maximal computation of $q \mid\mid C$ which is not successful, namely

$$q \mid\mid C \Longrightarrow q' \mid\mid \hat{C} \xrightarrow{\tau} q^1 \mid\mid \hat{C} \xrightarrow{\tau} q^2 \mid\mid \hat{C} \xrightarrow{\tau} \dots$$

As $q \ \mathsf{must}^{\mathsf{p2p}}\ C$ follows from $p \ \mathsf{must}^{\mathsf{p2p}}\ C$, we define C and prove the latter fact. Let $s = a_1 a_2 \dots a_n$ and let s' be the longest prefix of s such that $p \overset{s'}{\Longrightarrow}_{\not\checkmark}$. The precise definition of C depends on the existence of s', so we treat the two cases separately.

s' does not exist. In this case $p \xrightarrow{\checkmark}$. For every $0 \leq k \leq n$, let

$$C_k \overset{\mathsf{def}}{=} \begin{cases} (\,\tau.\,1\,) + \overline{a}_{k+1}.C_{k+1} & \text{if } 0 \leq k < n \\ \tau.\,1 & \text{if } k = n+1 \end{cases}$$

The reason why $p \ \mathsf{must}\ C_0$, is that $p \Downarrow s$. This follows from $p \Downarrow_{\mathsf{p2p}} s$, and ensures that all the maximal computations of $C_0 \mid\mid p$ contain a stable state $C' \mid\mid p'$. As $C_0 \overset{\overline{s'}}{\Longrightarrow} C'$ for some s' is a prefix of s, the definition of the C_i's ensures that $C' \xrightarrow{\checkmark}$.

Since $p \xrightarrow{\checkmark}$ we also know that $C \ \mathsf{must}\ p$, and so $p \ \mathsf{must}^{\mathsf{p2p}}\ C$. The hypothesis $p \precsim_{\mathsf{p2p}} q$ implies that $q \ \mathsf{must}^{\mathsf{p2p}}\ C$.

s' exists. In this case $p \overset{s'}{\Longrightarrow}_{\not\checkmark}$ for some s'; let $s' = a_1 a_2 \dots a_m$, with $m \leq n$. For every $0 \leq j \leq m$ the assumption $p \overset{s'}{\Longrightarrow}_{\not\checkmark}$ ensures that $p \overset{s_j}{\Longrightarrow}_{\not\checkmark}$. Lemma 5.3 ensures that for every $0 \leq j \leq m$ there exists a $\hat{r}_j$ such that

$$\hat{r}_j \ \mathsf{must} \bigoplus(p \ \mathsf{after}_{\not\checkmark}\ s_j) \tag{5.3}$$

For every $0 \leq k \leq n+1$ let

$$C_k \stackrel{\mathtt{def}}{=} \begin{cases} (\,\tau.(\hat{r}_k + \mathsf{1})\,) + \overline{a}_{k+1}.C_{k+1} & \text{if } 0 \leq k \leq m \\ (\,\tau.\,\mathsf{1}\,) + \overline{a}_{k+1}.C_{k+1} & \text{if } m < k \leq n \\ \tau.(\hat{r}_n + \mathsf{1}) & \text{if } k = n+1,\, m = n \\ \tau.\,\mathsf{1} & \text{if } k = n+1,\, m < n \end{cases}$$

We prove that $p \;\mathsf{must}^{\mathsf{p2p}}\; C_0$. Fix a maximal computation of $p \mid\mid C_0$,

$$p \mid\mid C_0 = p^0 \mid\mid C_0^0 \xrightarrow{\tau} p^1 \mid\mid C_0^1 \xrightarrow{\tau} p^2 \mid\mid C_0^2 \xrightarrow{\tau} \ldots \tag{5.4}$$

Intuitively, if one of the $\hat{r}_k + \mathsf{1}$'s appears in the computation, then there is a state $p^k \mid\mid C_0^k$ with $C_0^k = \hat{r}_i + \mathsf{1}$. The peer C_0 reaches a successful state (namely C_0^k itself). As for p, either $p^j \xrightarrow{\checkmark}$ for some $j \leq k$, or (5.3) above ensures that in the computation of $p^k \mid\mid C_0^k$ the peer p^k reaches a successful state.

If no $\hat{r}_k + \mathsf{1}$ appears in the computation then the convergence of p, $p \Downarrow s$, ensures that C_0 reaches $\mathsf{1}$. Moreover the construction of the C_k's and the $\hat{r}_k$'s imply that p reaches a successful state in the computation.

$\square$

Corollary 5.8. *if $p \sqsubseteq_{\mathsf{p2p}} q$ and $p \Downarrow_{\mathsf{p2p}} s$, then $q \Downarrow s$.*

Proof. For every s' prefix of s, the hypothesis imply that $p \Downarrow_{\mathsf{p2p}} s'$ and that $q \stackrel{s'}{\Longrightarrow} q'$, so Lemma 5.7 ensures that $q' \Downarrow$. Lemma 5.4 implies that $q \Downarrow s$. $\square$

Lemma 5.9. *Let $p \sqsubseteq_{\mathsf{p2p}} q$. For every $s \in \mathsf{Act}^\star$, if $p \Downarrow_{\mathsf{p2p}} s$ and $q \stackrel{s}{\Longrightarrow}$, then $p \stackrel{s}{\Longrightarrow}$.*

Proof. It suffices to define a peer C such that $p \;\not\mathsf{must}\; C$, $C \stackrel{\overline{s}}{\Longrightarrow}_{\not\checkmark} \hat{C} \stackrel{\tau}{\not\longrightarrow}$, and for every s' proper prefix of s, $C \stackrel{\overline{s'}}{\Longrightarrow}_{\not\checkmark} C'$ implies $C' \xrightarrow{\tau}\xrightarrow{\checkmark}$. These three conditions and $p \Downarrow s$ ensure that $p \stackrel{s}{\Longrightarrow}$, for otherwise all the maximal computations of $p \mid\mid C$ would be client-successful, thereby contradicting $p \;\not\mathsf{must}\; C$. To prove that $p \;\not\mathsf{must}\; C$, it suffices to show that $q \;\not\mathsf{must}^{\mathsf{p2p}}\; C$ and C must p. We show the first fact. The hypothesis imply that there exists a q' such that $q \stackrel{s}{\Longrightarrow} q'$. If q' diverges we infer the maximal computation

$$C_0 \mid\mid q \Longrightarrow \hat{C} \mid\mid q' \xrightarrow{\tau} \hat{C} \mid\mid q^1 \xrightarrow{\tau} \hat{C} \mid\mid q^2 \xrightarrow{\tau} \ldots$$

If q' does not diverge, then there exists a q'' such that $q' \stackrel{\varepsilon}{\Longrightarrow} q'' \stackrel{\tau}{\not\longrightarrow}$, and we infer the maximal computation $C_0 \mid\mid q \Longrightarrow \hat{C} \mid\mid q'' \stackrel{\tau}{\not\longrightarrow}$. In both cases we have shown non client-successful computation of $C_0 \mid\mid q$, so we have proven that $q \;\not\mathsf{must}\; C_0$. This ensures $q \;\not\mathsf{must}^{\mathsf{p2p}}$ C_0.

Now we define a suitable C, and prove that C must p. Let n be the length of s, $s = a_1 a_2 \ldots a_n$, and let s' be the longest prefix of s such that $p \stackrel{s'}{\Longrightarrow}_{\not\checkmark}$. The construction of C depends on the existence of s', and so rest of the proof is divided in two parts.

s' does not exist. In this case $p \xrightarrow{\checkmark}$. Let for every $0 \leq i \leq n+1$,

$$C_i \overset{\mathsf{def}}{=} \begin{cases} (\tau.\,\mathsf{1}\,) + \overline{a}_{i+1}.C_{i+1} & \text{if } 0 \leq i < n \\ \mathsf{0} & \text{if } i = n+1 \end{cases}$$

We prove that $q \not\mathsf{must}\ C_0$. The hypothesis imply that there exists a q' such that $q \overset{s}{\Longrightarrow} q'$. If q' diverges we infer the maximal computation

$$C_0 \mid\mid q \Longrightarrow \mathsf{0} \mid\mid q' \xrightarrow{\tau} \mathsf{0} \mid\mid q^1 \xrightarrow{\tau} \mathsf{0} \mid\mid q^2 \xrightarrow{\tau} \ldots$$

If q' does not diverge, then there exists a q'' such that $q' \overset{\varepsilon}{\Longrightarrow} q'' \overset{\tau}{\not\longrightarrow}$, and we infer the maximal computation $C_0 \mid\mid q \Longrightarrow \mathsf{0} \mid\mid q'' \overset{\tau}{\not\longrightarrow}$. In both cases we have shown non client-successful computation of $C_0 \mid\mid q$, so we have proven that $q \not\mathsf{must}\ C_0$. This ensures $q \not\mathsf{must}^{\mathsf{p2p}}$ C_0. Since $p \xrightarrow{\checkmark}$, it is clear that C must p.

s' exists. Let $s' = a_0 a_1 \ldots a_m$, with $m \leq n$. For every $0 \leq k \leq m$, the assumption $p \overset{s}{\Longrightarrow}_{\not\checkmark}$ ensures that $p \overset{s_k}{\Longrightarrow}_{\not\checkmark}$, and so Lemma 5.3 and Lemma 5.2 implies that there exists a $\hat{r}_k$ such that $\hat{r}_k$ must $\bigoplus(p \ \mathsf{after}_{\not\checkmark}\ s_k)$, $\hat{r}_k \overset{\tau}{\not\longrightarrow}$ and $\hat{r}_k \overset{\checkmark}{\not\longrightarrow}$. For every $0 \leq i \leq n+1$, let

$$C_i \overset{\mathsf{def}}{=} \begin{cases} (\tau.(\hat{r}_i + \mathsf{1})) + \overline{a}_i.C_{i+1} & \text{if } 0 \leq i \leq m \\ (\tau.\,\mathsf{1}) + \overline{a}_{i+1}.C_{i+1} & \text{if } m < i < n \\ \hat{r}_n & \text{if } i = n+1,\ m = n \\ \mathsf{0} & \text{if } i = n+1,\ m < n \end{cases}$$

We prove that C_0 must p. Fix a maximal computation

$$C_0 \mid\mid p \xrightarrow{\tau} C_0^1 \mid\mid p^1 \xrightarrow{\tau} C_0^2 \mid\mid p^2 \xrightarrow{\tau} C_0^3 \mid\mid p^3 \xrightarrow{\tau} \ldots$$

Either one of the $\hat{r}_i + \mathsf{1}$ (or $\hat{r}_n$) appears in the computation, or none does. Suppose $C_0^k = \hat{r}_i + \mathsf{1}$ or $C_0^k = \hat{r}_n$ for some state $C_0^k \mid\mid p^k$. If some p^j with $j \leq k$ is successful the computation is client-successful. If no p^j is successful, then the part of the computation starting at $C_0^k \mid\mid p^k$ is client-successful, for $\hat{r}_i$ must p^k. If neither an $\hat{r}_i + \mathsf{1}$ nor $\hat{r}_n$ appear in the computation then there must a state $C_0^k \mid\mid p^k$ with $C_0^k = C_{n+1}$, and $C_0^k \in \{\,\mathsf{0}, \mathsf{1}\,\}$, because neither $\hat{r}_n$ not $\hat{r}_i + \mathsf{1}$ appear. This and the construction of C ensure that p reaches a successful state in the computation above, for otherwise $C_0^k = \hat{r}_n$ or $C_0^k = \hat{r}_i + \mathsf{1}$ for some i.

□

Lemma 5.10. *If $p \sqsubseteq_{\mathsf{p2p}} q$ and $p \Downarrow_{\mathsf{p2p}} s$, then for every $B \in \mathsf{Acc}(q, s)$ there exists a set A such that $A \in \mathsf{Acc}(p, s)$ and $A \cap \mathsf{ua}_{\mathsf{clt}}(p, s) \subseteq B$.*

Proof. Let $s = b_1 b_2 \ldots b_n$. We reason by contradiction. Suppose that there exists a $A \in \mathsf{Acc}(p, s)$ such that $A \cap \mathsf{ua}_{\mathsf{clt}}(p, s) \not\subseteq B$. We use this assumption to define a peer C such that p $\mathsf{must}^{\mathsf{p2p}}$ C and that $q \not\mathsf{must}^{\mathsf{p2p}}$ C, thereby proving that $p \not\sqsubseteq_{\mathsf{p2p}} q$.

In particular, we build a C such that $C \overset{s}{\Longrightarrow}_{\not\checkmark} C'$, where C' is similar to the external sum we used to prove part (2) of Proposition 4.8. This allows us to prove that $q \not\mathsf{must}^{\mathsf{p2p}}$ C.

Let I be the index set of $\mathsf{Acc}(p, s)$, let J be the subset of I which ranges over the ready sets of p after successful executions of s, and let $\overline{a_i}$ an action in $A_i \cap \mathsf{ua}_{\mathsf{clt}}(p, s)$.

The construction of C depends on the longest s' prefix of s that is performed unsuccessfully by p.

s' does not exist. In this case for every $0 \leq k \leq n+1$ we let

$$C_k \stackrel{\texttt{def}}{=} \begin{cases} (\,\tau.\,\mathsf{1}\,) + \overline{b}_{k+1}.C_{k+1} & \text{if } 0 \leq k \leq n \\ \sum_{j \in J} \overline{a_j}.\,\mathsf{1} & \text{if } k = n+1 \end{cases}$$

All the maximal computations of $p \mid\mid C_0$ are successful. This is true because the assumption that s' does not exist implies that $p \xrightarrow{\checkmark}$, and because the hypothesis $p \Downarrow_{\mathsf{p2p}} s$ ensures that $p \Downarrow s$. In turn this let us prove that C_0 reaches a successful state in the maximal computations of $p \mid\mid C_0$. We have proven that $p \;\mathsf{must}^{\mathsf{p2p}}\; C_0$

To obtain an unsuccessful maximal computation of $q \mid\mid C_0$ zip together $C_0 \overset{\overline{s}}{\Longrightarrow} C_n$ with the execution of s that leads q to the state q' with ready set B. The state $q' \mid\mid C'$ is stable, so in the computation C_0 does not report success. This shows that $q \;\not{\mathsf{m}}\mathsf{ust}^{\mathsf{p2p}}\; C_0$, in turn leading to the mentioned contradiction, $p \not\sqsubseteq_{\mathsf{p2p}} q$.

s' exists. Let $s' = a_0 a_1 \ldots a_m$, with $m \leq n$. The construction of C in this case is more involved then the previous case. For every $0 \leq k \leq m$, $p \overset{s_k}{\Longrightarrow}_{\not\checkmark}$ so the hypothesis $p \Downarrow_{\mathsf{p2p}} s$ implies that there exists a $\hat{r}_k$ such that $\hat{r}_k \;\mathsf{must}\; \bigoplus(p \;\mathsf{after}_{\not\checkmark}\; s_k)$. For every $0 \leq k \leq n+1$ we let

$$C_k \stackrel{\texttt{def}}{=} \begin{cases} (\,\tau.(\hat{r}_k + \mathsf{1})\,) + \overline{b}_k.C_{k+1} & \text{if } 0 \leq i \leq m \\ (\,\tau.\,\mathsf{1}\,) + \overline{b}_{k+1}.C_{k+1} & \text{if } m < k < n \\ \sum_{i \in I \setminus J} \overline{a_i}.p_i + \sum_{j \in J} \overline{a_j}.\,\mathsf{0} & \text{if } k = n+1,\, m = n \\ \sum_{j \in J} \overline{a_j}.\,\mathsf{0} & \text{if } k = n+1,\, m < n \end{cases}$$

To prove that $p \;\mathsf{must}^{\mathsf{p2p}}\; C_0$ we show that $C_0 \;\mathsf{must}\; p$ and that $p \;\mathsf{must}\; C_0$. The reason why $C_0 \;\mathsf{must}\; p$ is same we used in Lemma 5.9. The symmetric statement, $p \;\mathsf{must}\; C_0$, follows from the convergence of p, which is ensures by the hypothesis $p \Downarrow_{\mathsf{p2p}} s$, and the fact that the stable states reached by C_0 are successful, except C_n. This state, though, can interact with the derivative of p at hand, and reduce to a successful state.

To prove that $q \;\not{\mathsf{m}}\mathsf{ust}^{\mathsf{p2p}}\; C_0$ we proceed as we did in the case that s' does not exist. □

Proposition 5.11. *$p \sqsubseteq_{\mathsf{p2p}} q$ implies $p \precsim_{u\mathsf{svr}} q$.*

Proof. It is a consequence of Corollary 5.8, Lemma 5.10, Lemma 5.9 and of a fourth property of $\sqsubseteq_{\mathsf{p2p}}$ that we prove here.

We have to show that $p \sqsubseteq_{\mathsf{p2p}} q$, $p \Downarrow_{\mathsf{p2p}} u$ and $q \overset{u}{\Longrightarrow}$, then $p \overset{u}{\Longrightarrow}$. Fix a pair $p \sqsubseteq_{\mathsf{p2p}} q$ that satisfies the first three conditions. $p \Downarrow u$ is true because $p \Downarrow_{\mathsf{p2p}} u$. If $p \xrightarrow{\checkmark}$, then the argument is the same we used in Theorem 3.14. If $p \overset{\checkmark}{\not\longrightarrow}$, then either $p \overset{u}{\Longrightarrow}_{\not\checkmark}$ or $p \overset{u}{\not\Longrightarrow}_{\not\checkmark}$. In the first case $p \overset{u}{\Longrightarrow}$ follows immediately. In the second case, thanks to $p \overset{\checkmark}{\not\longrightarrow}$ there exists the greatest $m \in \mathbb{N}$ such that $p \overset{u_m}{\Longrightarrow}_{\not\checkmark}$. The hypothesis $p \Downarrow_{\mathsf{p2p}} u$ ensures that $p \;\mathsf{usbl}_{\not\checkmark}\; u$, so for every $0 \leq i \leq m$ there exists a process r_i such that $r_i \;\mathsf{must}\; \bigoplus(p \;\mathsf{after}\; u_i)$. For every $n \in N$ let

$$C_k \stackrel{\texttt{def}}{=} \begin{cases} \tau.(r_k + \mathsf{1}) + \overline{a}_k.C_{k+1} & \text{if } i \leq m \\ \tau.\,\mathsf{1} + \overline{a}_k.C_{k+1} & \text{otherwise} \end{cases}$$

$$\begin{array}{llcl llcl}
\textbf{(S1a)} & \mu.x_{\not\checkmark} + \mu.y & = & \mu.(\tau.x_{\not\checkmark} + \tau.y) & \textbf{(S3)} & \mu.x + \tau.(\mu.y + z) & = & \tau.(\mu.x + \mu.y + z) \\
\textbf{(S1b)} & \tau.x & \leq & \tau.\tau.x & \textbf{(S4)} & \tau.x + \tau.y & \leq & x \\
\textbf{(S2)} & x_{\not\checkmark} + \tau.y & = & \tau.(x_{\not\checkmark} + y) + \tau.y & \textbf{(S5)} & \Omega & \leq & x
\end{array}$$

Figure 5: Standard inequations

The remaining part of the proof is analogous to the one of Theorem 3.14, and relies on the fact that $p \Downarrow u$ and that $r_k + \mathsf{1}\ \mathsf{must}\ (p\ \mathsf{after}_{\not\checkmark}\ u_k)$. $\square$

Now the proof of completeness is straightforward.

Theorem 5.12 (Completeness peer)**.** *$p \underset{\sim}{\sqsubset}_{\mathsf{p2p}} q$ implies $p \precsim_{\mathsf{p2p}} q$.*

Proof. Fix a pair $p \underset{\sim}{\sqsubset}_{\mathsf{p2p}} q$. We have to show that $p \precsim_{\mathsf{clt}} q$ and $p \precsim_{u\mathsf{svr}} q$. The first fact follows from Proposition 3.15 and Theorem 3.12. The second fact is Proposition 5.11. $\square$

6. Equational characterisation

We use $\mathsf{CCS}^{\mathsf{f}}$ to denote the finite sub-language of 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 Ω; this last denotes the term τ^∞ from 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 $p \underset{\sim}{\sqsubset}_\star q$ over this finite algebra, where $\star$ ranges over svr, clt and $\mathsf{p2p}$. For a given set of inequations E we will use $p \sqsubseteq_E q$ denote the fact that the inequation $p \leq q$ can be derived from E using standard equational reasoning, while $t =_E u$ means that both $t \sqsubseteq_E u$ and $u \sqsubseteq_E t$ can be derived.

There are two immediate obstacles. The first is that none of these preorders are pre-congruences for the language $\mathsf{CCS}^{\mathsf{f}}$; specifically they are not preserved by the choice operator $+$.

Example 6.1. *Using the behavioural characterisation it is easy to check that $\mathit{0} \underset{\sim}{\sqsubset}_{\mathsf{p2p}} b.\,\mathit{0}$; in fact this is trivial because $\mathit{0} \notin \mathcal{U}\mathsf{p2p}$. However $a.\,\mathit{1} + \mathit{0} \not\underset{\sim}{\sqsubset}_{\mathsf{p2p}} a.\,\mathit{1} + b.\,\mathit{0}$ because $\overline{a}.\,\mathit{1} + \overline{b}.\,\mathit{0}\ \mathsf{must}^{\mathsf{p2p}}\ a.\,\mathit{1} + \mathit{0}$ while $\overline{a}.\,\mathit{1} + \overline{b}.\,\mathit{0}\ \not\mathsf{must}^{\mathsf{p2p}}\ a.\,\mathit{1} + b.\,\mathit{0}$; the latter follows because of the possible communication on b.*

A similar example shows that the client preorder is not preserved by parallel composition. The same counter-example also shows 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 an alternative more amenable characterisation. To this end we let $p \underset{\sim}{\sqsubset}^{+}_\star q$ to mean that $\mathsf{f}.\,\mathsf{1} + p \underset{\sim}{\sqsubset}_\star \mathsf{f}.\,\mathsf{1} + q$ for some fresh action f.

Proposition 6.2. *In an arbitrary LTS, $p \underset{\sim}{\sqsubset}^{\mathsf{c}}_\star q$ if and only if $p \underset{\sim}{\sqsubset}^{+}_\star q$.*

Proof. One direction is immediate, namely $p \underset{\sim}{\sqsubset}^{\mathsf{c}}_\star q$ implies $p \underset{\sim}{\sqsubset}^{+}_\star q$. To prove the converse it is sufficient to prove that each preorder $\underset{\sim}{\sqsubset}^{+}_\star$ is preserved by the two operators $- + -$ and $\mu.-$. The details are straightforward, and left to the reader. $\square$

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].

Lemma 6.2 gives a convenient characterisation of the behavioural precongruences $p \sqsubseteq^{\mathsf{c}}_{\star} q$ which we will use in the sequel. One useful property of this characterisation is the following:

Lemma 6.3. *Suppose $p \xrightarrow{\tau}$. Then $p \sqsubseteq_{\star} q$ implies $p \sqsubseteq^{+}_{\star} q$.*

Proof. Suppose $p \xrightarrow{\tau}$. Then for any client r, $\mathsf{f}.\,\mathsf{1} + p$ must r if and only if p must r. This is sufficient to prove that $p \sqsubseteq_{\mathsf{svr}} q$ implies $\mathsf{f}.\,\mathsf{1} + p \sqsubseteq_{\mathsf{svr}} \mathsf{f}.\,\mathsf{1} + q$.

Minor variations on this argument will show that the result also holds for the client and peer preorders. □

The second obstacle to the equational characterisation of the behavioural preorders is that they 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

$$a.\tau.x = a.x, \tag{6.1}$$

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

Accordingly in order to have a more elegant presentation of the inequational theory we will use two sorts of variables, the standard $x, y, \ldots$ which may be instantiated with any process from $\mathsf{CCS}^{\mathrm{f}}$, 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 5,which are satisfied by all three behavioural orders $\sqsubseteq^{+}_{\star}$. 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

$$\mathsf{1} \;=\; \mathsf{0} \tag{\textbf{SVR1}}$$

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

Theorem 6.4 (Soundness and completeness for server-testing)**.** *In CCS^{f}, $p \sqsubseteq^{c}_{\mathsf{svr}} q$ if and only $p \sqsubseteq_{\mathbf{SVR}} q$.*

Outline. The equation (**SVR**) means that every term can be reduced to one which does not contain any occurrence of the unit $\mathsf{1}$. This means that all terms can now match the special variables $x_{\not\checkmark}, y_{\not\checkmark}, \ldots$ and therefore the equations (**S1**) - (**S5**) can be rewritten with them replaced with the standard variables $x, y, \ldots$. The resulting inequational theory coincides with that from [NH84] which characterises the *must* testing preorder over finite terms;[2] this we know coincides with our server preorder $\sqsubseteq^{\mathsf{c}}_{\mathsf{svr}}$. □

[2]This is referred to as $\sqsubseteq_2$ in [NH84].

All the standard inequations in Figure 5 are also valid for the client and peer pre-congruences. Indeed the reason for introducing the two sorts of variables was to ensure that they remain valid for these new pre-congruences.

Example 6.5 (The need for two sorted inequations)**.** *We have already seen why* (**S1a**) *would no longer hold, for the client and peer pre-congruences, if the meta-variable $x_{\not\checkmark}$ were replaced by the standard variable x. The innocuous equation (6.1) above would be a derived equation from this altered version of* (**S1a**)*, because of the idempotency of $+$.*

Let r_1, r_2 denote the clients $1 + \tau.a.\,1$ and $\tau.(1 + a.\,1) + \tau.a.\,1$ respectively. Then $(1 + \tau^\infty)$ must r_1 whereas $(1 + \tau^\infty)$ $\not\mathsf{must}$ r_2, and thus $r_1 \not\sqsubseteq_{\mathsf{clt}} r_2$. This shows the need for the meta-variable $x_{\not\checkmark}$ in (**S2**)*, for otherwise $r_1 = r_2$ would be an instantiation.*

The same example can be used to show that this restriction is also necessary for the peer pre-congruence.

Despite the use of two sorts of variables, much of the standard equational reasoning, for example from [NH84] remains valid. Here is a typical example, where we use **ST** to denote the inequational theory generated by the standard inequations.

Lemma 6.6. *In the equational theory* **ST***, $\tau.x + \tau.y =_{\mathbf{ST}} \tau.(\tau.x + \tau.y)$ is a derived equation.*

Proof. Note that this is a generalisation of the standard equation (**S1a**) with μ set to τ, which requires that at least one of the meta-variables be $\not\checkmark$. But let us give a more general proof.

Using (**S1b**) and (**S4**), together with the idempotency of $+$, we have the derived equation $\tau.\tau.x = \tau.x$. Then applying this twice we obtain:

$$\begin{aligned} \tau.x + \tau.y =_{\mathbf{ST}} \tau.\tau.x + \tau.y \quad &=_{\mathbf{ST}} \quad \tau.(\tau.\tau.x + \tau.y) && (\mathbf{S1a}) \\ &=_{\mathbf{ST}} \quad \tau.(\tau.x + \tau.y) \end{aligned}$$

□

The equation (**SVR**) is obviously not satisfied by either the client or the peer pre-congruence. In order to characterise them we to replace it with inequations which capture the significance of the operators $\mathsf{1}$ and $\mathsf{0}$ for client and peers respectively. First we consider the client case. It is easy to see that $\mathsf{1}$ is a maximal element for the preorder $\sqsubseteq_{\mathsf{clt}}$, and also for the contextual preorder $\sqsubseteq^{c}_{\mathsf{clt}}$: p must $\mathsf{f}.\,\mathsf{1} + \mathsf{1}$ for every server p, from which $r \sqsubseteq^{c}_{\mathsf{clt}} \mathsf{1}$ follows for every client r. We also have that $r + \mathsf{1} \eqsim_{\mathsf{clt}} \mathsf{1}$ for every client r; intuitively once a client can report success immediately then it does not matter what other behaviour it has. This client behaviour of $\mathsf{1}$ is adequately captured by the two inequations (**CLT1a**) and (**CLT1b**) in Figure 7.

Another property of $\mathsf{1}$ stems from the fact that for every client r,

$$r \sqsubseteq_{\mathsf{clt}} r + \mu.\,\mathsf{1}$$

for every $\mu \in \mathsf{Act}_\tau$; adding the capability $\mu.\,\mathsf{1}$ to a client does not decrease its ability to satisfy servers. This property is captured by the inequation (**CLT1c**). In a dual manner, adding the capability $\mu.\,\mathsf{0}$ to a client does not increase its ability to satisfy; for every client r

$$r + \mu.\,\mathsf{0} \sqsubseteq_{\mathsf{clt}} r$$

This is captured by (**Zb**).

(**Za**)	$\tau.\mathsf{0}$	$\leq$	Ω	(**Zb**)	$\mu.\mathsf{0}$	$\leq$	$\mathsf{0}$
(**CLT1a**)	x	$\leq$	$\mathsf{1}$	(**P2P1**)	$\mathsf{0}$	$\leq$	$\mathsf{1}$
(**CLT1b**)	$\mathsf{1}$	$\leq$	$x+\mathsf{1}$	(**P2P2**)	$\mu.(\mathsf{1}+x)$	$\leq$	$\mathsf{1}+\mu.x$
(**CLT1c**)	$\mathsf{0}$	$\leq$	$\mu.\mathsf{1}$	(**P2P3**)	$\mu.(\mathsf{1}+x)+\mu.(\mathsf{1}+y)$	$\leq$	$\mu.(\mathsf{1}+\tau.x+\tau.y)$

Figure 6: Client and peer inequations

Let us now look in more detail at the zero $\mathsf{0}$. Since p must $\mathsf{0}$ for no server p it follows that $\mathsf{0} \sqsubseteq_{\mathsf{clt}} r$ for every client r. But $\mathsf{0} \sqsubseteq^{\mathsf{c}}_{\mathsf{clt}} r$ does not not in general hold. For example $\bar{\mathsf{f}}.\mathsf{0} + \bar{b}.\mathsf{0}$ must $\mathsf{f}.\mathsf{1} + \mathsf{0}$ but $\bar{\mathsf{f}}.\mathsf{0} + \bar{b}.\mathsf{0}$ $\not\mathsf{must}$ $\mathsf{f}.\mathsf{1} + b.\mathsf{0}$. In the latter, the synchronisation on b leads to the possibility of the client not being satisfied. It follows that $\mathsf{0} \not\sqsubseteq^{\mathsf{c}}_{\mathsf{clt}} b.\mathsf{0}$.

However p must $\mathsf{f}.\mathsf{1} + \tau.\mathsf{0}$ for no server p, with the result that $\tau.\mathsf{0} \sqsubseteq^{\mathsf{c}}_{\mathsf{clt}} r$ for every client r; it follows that $\tau.\mathsf{0}$ is a minimal element in the client theory. Recall that Ω is also a minimal element, and therefore to capture this property of $\mathsf{0}$ it is sufficient to add the inequation (**Za**).

Note that an application of (**S1a**), together with the idempotency of $+$, gives the derived equation $\mu.x_{\not\checkmark} = \mu.\tau.x_{\not\checkmark}$; this combined with (**Za**), (**S5**) gives the useful derived inequation

$$\mu.\mathsf{0} \;\leq\; \mu.x \qquad (\mathbf{DZ1})$$

which we will use extensively in the sequel; intuitively this means that $\mathsf{0}$ acts like a zero underneath a prefix.

Let **CLT** denote the set of inequations obtained by adding to the standard one, the client inequations we have just discussed, (**Za**), (**Zb**) and (**CLT1a**) - (**CLT1c**).

Theorem 6.7 (Soundness and Completeness for client-testing)**.** *In* CCS^f, *$p \sqsubseteq^{c}_{clt} q$ if and only $p \sqsubseteq_{\mathbf{CLT}} q$.*

Proof. To prove soundness, again it is sufficient to show that $\sqsubseteq^{+}_{\mathsf{clt}}$ satisfies of the the inequations concerned. Completeness requires the development of *normal forms* for clients. This is the topic of Section 7.2, and the result is actually proved in Theorem 7.12. □

Both the inequations (**Za**) and (**Zb**) remain valid for the peer preorder, but none of the unit inequations (**CLT1a**) - (**CLT1c**).

Example 6.8. *First consider* (**CLT1a**)*. It is easy to see that $\bar{a}.\mathsf{1}$ must $\mathsf{f}.\mathsf{1} + a.\mathsf{1}$ as both peers always evolve to success states. However $\bar{a}.\mathsf{1}$ $\not\mathsf{must}^{\mathsf{p2p}}$ $\mathsf{f}.\mathsf{1} + \mathsf{1}$, because the peer $\mathsf{1}$ can not help the partner $\bar{a}.\mathsf{1}$ achieve success. It follows that $a.\mathsf{1} \not\sqsubseteq^{+}_{\mathsf{p2p}} \mathsf{1}$.*

Moving to (**CLT1b**)*, $\mathsf{1}$ $\not\mathsf{must}^{\mathsf{p2p}}$ $\tau.\mathsf{0} + \mathsf{1}$ because the activation of the internal action can preempt one of the peers achieving success. However trivially $\mathsf{1}$ must $\mathsf{1}$, with the result that $\mathsf{1} \not\sqsubseteq^{+}_{\mathsf{p2p}} \tau.\mathsf{0} + \mathsf{1}$; this is a counterexample to* (**CLT1b**) *for the peer pre-congruence.*

For the final counter-example note that $\bar{a}.\mathsf{0} + \bar{\mathsf{f}}.\mathsf{1}$ must $\mathsf{f}.\mathsf{1} + \mathsf{0}$ because the co-action $\bar{a}$ is never activated. However $\bar{a}.\mathsf{0} + \bar{\mathsf{f}}.\mathsf{1}$ $\not\mathsf{must}^{\mathsf{p2p}}$ $\mathsf{f}.\mathsf{1} + a.\mathsf{1}$ because the here it is activated, and the activation prevents one of the peers from achieving success. Thus (**CLT1c**) *does not hold for the pre-congruence $\sqsubseteq^{+}_{\mathsf{p2p}}$ for any external μ; a minor variation demonstrates that it also does not hold when μ is τ.*

They need to be replaced by unit inequations appropriate to peer. There are various possibilities we could add; we justify our particular[3] choice by considering properties we would like of the inequational theory; in total we add three new inequations.

In both the server and the client theory we know that for every action μ and processes p, q there is another process r satisfying

$$\mu.p + \mu.q = \mu.r \tag{6.2}$$

Indeed this is one of the most important laws which delineates behavioural theories based on testing, rather than say *bisimulation equivalence* [Mil89]. It is derivable in the theory of servers, where r can be taken to be $\tau.p + \tau.q$. it is also derivable in the theory of clients, although the form r takes depends on whether both p, q can immediately report success. If at least one of p, q can not report success immediately this is an instance of (**S1a**) in Figure 5. If this is not the case then (**S1a**) can not be employed. But it turns out that in the algebraic theory of clients (6.2) still remains derivable, taking the required r to be $\mathsf{1}$.

We also require (6.2) to be derivable in the algebraic theory of peers. Again if either p or q can not immediately report success then this will be an instance of (**S1a**). One can also check that

$$\mu.(\mathsf{1} + p) + \mu.(\mathsf{1} + q) \eqsim^{+}_{\mathsf{p2p}} \mu.(\mathsf{1} + \tau.p + \tau.p)$$

for all p, q. In order to make these derivable in the algebraic theory it is sufficient to add the inequation (**P2P3**), given in Figure 6. From (**P2P3**) and (**S4**) one then obtains the derived equation

$$\mu.(\mathsf{1} + x) + \mu.(\mathsf{1} + y) \;=\; \mu.(\mathsf{1} + \tau.x + \tau.y) \tag{6.3}$$

Another intrinsic property of extensional behavioural theories is the ability to abstract from internal activity. One equation capturing this is have already been discussed in (6.1) above. This is valid in the server theory, and enables us to forget about the intermediate internal action τ. We have also seen that it does not hold in the client theory; nor does it hold in the peer theory. However we are still able to abstract from intermediate internal actions in certain circumstances. For example

$$\mu.\tau.x_{\not\checkmark} \quad = \quad \mu.x_{\not\checkmark}$$

is easily derivable from (**S1a**). Other circumstances, the presence of $\mathsf{1}$, are summed up by

$$\mu.(\mathsf{1} + \tau.x) \eqsim^{+}_{\mathsf{p2p}} \qquad \mu.(\mathsf{1} + x) \tag{6.4}$$

This is easily seen to be derivable from (6.3) above.

Our other two additions are motivated by the requirement for both peers to always report success. So adding this possibility to a peer does not affect its overall behaviour. This is summed up be the identity

$$p \eqsim^{+}_{\mathsf{p2p}} p + \mathsf{1}$$

for every peer p. This is captured as a derived equation if we add the inequation (**P2P1**) in Figure 7 to the theory.

Success does not have to be reported simultaneously by interacting pairs of peers; in particular the ability of a peer is not damaged by bringing forward the reporting of success.

[3]current

$$
\begin{array}{lrcl}
(\mathbf{D1}) & \sum_{1\leq i\leq n}\tau.x_i & = & \tau.(\sum_{1\leq i\leq n}\tau.x_i) \\
(\mathbf{D2}) & x_{\not\checkmark} + \tau.(x_{\not\checkmark} + y) & = & \tau.(x_{\not\checkmark} + y) \\
(\mathbf{D3}) & \mu.x + \Omega & = & \Omega \\
 & & & \\
(\mathbf{DP1}) & \mathsf{1} + \mu.x & = & \mathsf{1} + \mu.(x + \mathsf{1}) \\
(\mathbf{DP2}) & \tau.(\mathsf{1} + \sum_{1\leq i\leq n}\tau.x_i) & = & \sum_{1\leq i\leq n}\tau.(\mathsf{1} + x_i) \\
(\mathbf{DP3}) & \mu.x & \leq & \mu.(\mathsf{1} + \tau.x) \\
 & & & \\
(\mathbf{D4a}) & \tau.x_{\not\checkmark} + \tau.y & = & \tau.x_{\not\checkmark} + \tau.y + \tau.(x_{\not\checkmark} + y) \\
(\mathbf{D4b}) & \tau.(x_{\not\checkmark} + \mathsf{1}) + \tau.(y_{\not\checkmark} + \mathsf{1}) & = & \tau.(x_{\not\checkmark} + \mathsf{1}) + \tau.(y_{\not\checkmark} + \mathsf{1}) + \tau.(x_{\not\checkmark} + y_{\not\checkmark} + \mathsf{1}) \\
 & & & \\
(\mathbf{D5a}) & \tau.x + \tau.(x + y_{\not\checkmark} + z) & = & \tau.x + \tau.(x + y_{\not\checkmark}) + \tau.(x + y_{\not\checkmark} + z) \\
(\mathbf{D5b}) & \tau.x + \tau.(x + (y_{\not\checkmark} + \mathsf{1}) + z) & = & \tau.x + \tau.(x + (y_{\not\checkmark} + \mathsf{1})) + \tau.(x + (y_{\not\checkmark} + \mathsf{1}) + z)
\end{array}
$$

Figure 7: Some derived equations

This motivates the use of (**P2P2**) in Figure 7. As we will see an interesting consequence is the derived equation:

$$\mathsf{1} + \mu.x \quad = \quad \mathsf{1} + \mu.(x + \mathsf{1}) \qquad (\mathbf{DP1})$$

As we will see this is a derived equation in our inequational theory for peers. This is taken to consist of the standard inequations from Figure 5, together with (**Za**), (**Za**) and (**P2P1**) - (**P2P3**).

Theorem 6.9 (Soundness and Completeness for peer-testing)**.** *In* $\mathsf{CCS}^f_{w\tau}$*,* $p \sqsubseteq^c_{\mathsf{p2p}} q$ *if and only* $p \sqsubseteq_{\mathbf{P2P}} q$*.*

Proof. Again to prove soundness it is sufficient to show that all of the inequations are valid for the preorder $\sqsubseteq^+_{\mathsf{p2p}}$. Completeness is proved in Theorem 7.16. □

7. Completeness proofs

In this section we use a number of derived inequations, gathered in Figure 7. These are justified in Appendix A.

7.1. Normal forms.

It will be notationally convenient to consider $\mathsf{1}$ as a prefix term, say $\checkmark.\mathsf{0}$, thus including $\checkmark$ as a possible prefix action. We also use $p\checkmark$ to denote that p can perform the success action, $p \xrightarrow{\checkmark}$, and $p\not\checkmark$ for the converse.

The normal forms we use are an extension of those in [NH84]; considerable complications arise because of the presence of the unit operator $\mathsf{1}$. The central idea is that of *saturated* collections of sets. Let $\mathcal{A}$ be a collection of finite subsets of $\mathsf{Act}_\checkmark$. It is said to be saturated if whenever $X, Y \in \mathcal{A}$,

(i) $X \cup Y \in \mathcal{A}$
(ii) $Z \in \mathcal{A}$ whenever $X \subseteq Z \subseteq Y$

Lemma 7.1. *For every collection $\mathcal{A}$ of finite subsets of* $\mathsf{Act}_{\checkmark}$ *there exists a least collection* $\mathsf{cl}(\mathcal{A})$ *containing $\mathcal{A}$ which is saturated.*

Proof. Straightforward. The existence of $\mathsf{cl}(\mathcal{A})$ can be shown from general principles, but we can also give a constructive definition. Let

$$\mathcal{B} = \{ Z \mid X \subseteq Z \subseteq \cup\mathcal{A}, \text{ for some } X \in \mathcal{A} \}$$

By definition $\mathcal{A} \subseteq \mathcal{B}$ and one can check that $\mathcal{B}$ is saturated, that is it satisfies (i), (ii) above.

Now let $\mathcal{C}$ be any other saturated set containing $\mathcal{B}$. Since it is closed under set theoretic union it must contain the set $\cup\mathcal{A}$. Therefore, since it satisfies (ii) above it must also contain all sets in $\mathcal{B}$; that is $\mathcal{B} \subseteq \mathcal{C}$. So we can set $\mathsf{cl}(\mathcal{A})$ to be $\mathcal{B}$. □

Definition 7.2 (Peer-normal forms, pnfs)**.** (1) $\Omega\ \{+\ \mathsf{1}\}$ is a pnf.
(2) $n = (\sum_{a \in A} a.n_a)\ \{+\ \mathsf{1}\}$, for $A \subseteq \mathsf{Act}$, is a pnf, provided $n\checkmark$ implies $n_a\checkmark$
(3) Let $\mathcal{A}$ be a non-empty saturated set of non-empty finite subsets of $\mathsf{Act}_{\checkmark}$. Suppose that for each $\lambda \in \cup\mathcal{A}$, n_λ is a pnf. Then $\sum_{A \in \mathcal{A}} \tau.n_A\ \{+1\}$ is a pnf, where n_A denotes the term $\sum_{\lambda \in A} n_\lambda$, provided $n\checkmark$ implies $n_a\checkmark$.

Here we use the notation $p\ \{+\ \mathsf{1}\}$ to indicate that the presence of $+\ \mathsf{1}$ is optional. Thus by (1), both Ω and $\Omega + \mathsf{1}$ are pnfs; by (2) $\mathsf{0}$ is a pnf, as are $a.\mathsf{0} + \mathsf{1}$ and $a.\mathsf{0}$.[4]

Before showing that all finite terms can be transformed into normal forms we need to develop some syntactic machinery for manipulating terms. We continue to use the notation introduced in Definition 7.2, using n_A, where $A \subseteq \mathsf{Act}_{\checkmark}$ to denote the term $\sum_{\lambda \in A} n_\lambda$, for some (assumed) collection of terms n_λ and $n_{\mathcal{A}}$ for the term $\sum_{A \in \mathcal{A}} n_A$.

Proposition 7.3 (Saturation)**.** *Let $\mathcal{B} = \mathsf{cl}(\mathcal{A})$. Then $n_{\mathcal{A}} =_{\mathbf{P2P}} n_{\mathcal{B}}$.*

Proof. This relies on two auxiliary results. Suppose A, B, $C \subseteq \mathsf{Act}_{\checkmark}$, where $A \subseteq C \subseteq B$. Then

$$\tau.n_A + \tau.n_B =_{\mathbf{P2P}} \tau.n_A + \tau.n_B + \tau.n_{A\cup B} \qquad \text{(Union)}$$
$$\tau.n_A + \tau.n_B =_{\mathbf{P2P}} \tau.n_A + \tau.n_C + \tau.n_B \qquad \text{(Sub)}$$

By systematically employing both equalities, from left to right, we can transform $n_{\mathcal{A}}$ into $n_{\mathcal{B}}$. So we concentrate on proving these properties.

First we consider (Union). Suppose $\checkmark \notin A$. Then the equality follows from an application of the derived equation (**D4a**) in Figure 7. This can also be applied if $\checkmark \notin B$. Finally if $\checkmark \in A \cap B$ then we can use (**D4b**).

The proof of (Sub) above is similar, depending on whether $\checkmark \in C$. If it is (**D5b**) can be used, and otherwise (**D5a**) above. □

The next property has already been alluded to in (6.2).

Proposition 7.4 (Uniqueness of derivatives)**.** *For all p, $q \in \mathsf{CCS}^f$ and all actions $\mu \in \mathsf{Act}_\tau$, there exists some term r such that $\mu.p + \mu.q =_{\mathbf{P2P}} \mu.r$.*

Proof. If $p\not\checkmark$, or $q\not\checkmark$ then we can apply (**S1a**) directly, obtaining $r = \tau.p + \tau.q$. Otherwise we have both $p\checkmark$ and $q\checkmark$ and the required r is $\mathsf{1} + \tau.p + \tau.q$. In one direction this is an application of (**P2P3**). The reverse follows from two applications of $\tau.x \leq x$, which is derivable from (**S4**). □

[4]unfortunately so is $\mathsf{0} + \mathsf{1}$; this will be treated as $\mathsf{1}$.

We now show how to transform all terms into pnfs. The main work is done in the following two lemmas.

Lemma 7.5. *If n_1, n_2 are pnfs then there exists a pnf m such that $\tau.n_1 + \tau.n_2 =_{\mathbf{P2P}} m$.*

Proof. By induction on the combined size of n_1, n_2 and an analysis of their structure. There are many cases to consider. We omit the cases when either take the form Ω $\{+ \mathsf{1}\}$ as the result follows from the derived rule (**D3**).

(a) Suppose $n_1 = \sum_{A\in\mathcal{A}} \tau.n_A$ and $n_2 = \sum_{B\in\mathcal{B}} \tau.m_B$. An application of the derived rule (**D1**) from Figure 7 gives

$$\tau.n_1 + \tau.n_2 \quad =_{\mathbf{P2P}} \quad \sum_{A\in\mathcal{A}} \tau.n_A + \sum_{B\in\mathcal{B}} \tau.m_B$$

Now suppose there exists some external action $c \in A \cap B$, where $A \in \mathcal{A}$ and $B \in \mathcal{B}$ such that $n_c \neq m_c$. We isolate the subterm $\tau.n_A + \tau.m_B$ so as to unify the c-derivatives. This subterm has the form $\tau.(p + c.n_c) + \tau.(q + c.m_c)$ which can be rewritten to

$$\begin{aligned} &=_{\mathbf{P2P}} \quad c.n_c + \tau.(p + c.n_c) + c.m_c + \tau.(q + c.m_c) && \text{by } (\mathbf{D2}) \\ &=_{\mathbf{P2P}} \quad \tau.(p + c.n_c + c.m_c) + \tau.(q + c.n_c + c.m_c) && \text{by } (\mathbf{S3}) \end{aligned}$$

Now suppose at least one of n_c, m_c satisfies $\not\checkmark$. Then we can use (**S1a**) to proceed thus:

$$=_{\mathbf{P2P}} \quad \tau.(p + c.(\tau.n_c + \tau.m_c) + \tau.(q + c.(\tau.n_c + \tau.m_c)$$

By induction there exists a normal form $o_c =_{\mathbf{P2P}} \tau.n_c + \tau.m_c$ and so we may transform the subterm to

$$=_{\mathbf{P2P}} \quad \tau.(p + c.o_c) + \tau.(q + c.o_c)$$

On the other hand if both $n_c \checkmark$ and $m_c \checkmark$, we can imitate the above sequence of steps, this time using (**P2P3**), or rather its derived version (6.3) above, to obtain

$$=_{\mathbf{P2P}} \quad \tau.(\mathsf{1} + p + c.(\mathsf{1} + o_c)) + \tau.(\mathsf{1} + p + c.(\mathsf{1} + o_c))$$

By systematically applying the *derivative unification transformation* we can now assume that $\tau.n_1 + \tau.n_2$ has the form $\sum_{C\in\mathcal{C}} \tau.s_C$, where each s_c is a pnf. Moreover by Proposition 7.3 this can be transformed into $\sum_{D\in\mathcal{D}} \tau.s_D$ where $\mathcal{D}$ is saturated; this is the required pnf.

(b) Suppose $n_1 = \mathsf{1} + \sum_{A\in\mathcal{A}} \tau.n_A$ and $n_2 = \sum_{B\in\mathcal{B}} \tau.m_B$.

Applying the derived equation (**DP2**) we can rewrite $\tau.n_1$ to the form $\sum_{A\in\mathcal{A}} \tau.(\mathsf{1} + n_A)$ which by (**D1**) can be transformed into $\tau.\sum_{A\in\mathcal{A}} \tau.(\mathsf{1} + n_A)$. We can now proceed as in the previous case. The same holds if m has $\mathsf{1}$ as a summand.

(c) Suppose $n_1 = \sum_{a\in A} a.n_a$ $\{+ \mathsf{1}\}$ and n_2 is as in the previous case. Then $\tau.n_1 =_{\mathbf{P2P}} \tau.(\tau.(\sum_{A\in\{A\}} n_A \; \{+ \mathsf{1}\})$ by (**D1**) and we can proceed as in case (a).

Finally if n_2 contains $\mathsf{1}$ as a summand we can proceed in much the same way, but using case (b).

□

Lemma 7.6. *If n_1, n_2 are* pnfs *then there exists a* pnf *m such that $n_1 + n_2 =_{\mathbf{P2P}} m$.*

Proof. Again the proof proceeds by induction on the combined sizes of n_1, n_2 and a case analysis of their form.

(a) Suppose $n_1 = \sum_{a \in A} a.n_a$ and $n_2 = \sum_{B \in \mathcal{B}} \tau.m_B$; this is the central case.
We know that $\mathcal{B}$ is not empty. So using (**D1**),(**S2**) we have $n_1 + n_2 =_{\mathbf{P2P}} \tau.(n_1 + m_{B1}) + \tau.n_2$, for some $B1 \in \mathcal{B}$. By induction $n_1 + m_{B1}$ has a pnf. The required result now follows from the previous lemma.

(b) Suppose $n_1 = 1 + n_1'$, where $n_1' = \sum_{a \in A} a.n_a$ and n_2 is as in the previous case. Then we can construct, as in case (a), a pnf for $n_1' + n_2$, which takes the form $\sum_{D \in \mathcal{D}} \tau.o_D$. Then the required pnf is

$$1 + \sum_{D \in \mathcal{D}} \tau.(1 + \sum_{d \in D} d.(1 + o_d)$$

This requires the repeated application of the derived rule (**DP1**).
The case where n_2 has 1 as an additional summand is handled in a similar manner.

(c) Suppose $n_1 = \sum_{A \in \mathcal{A}} a.n_A$ and $n_2 = \sum_{B \in \mathcal{B}} \tau.m_B$. Using (**D1**) we have $n_1 + n_2 = \tau.n_1 + \tau.n_2$ and the result now follows by the previous lemma.
If either n_1 or n_2, or both, have 1 as an additional summand we can proceed in the same manner. We may then have to apply (**DP1**) to ensure that the resulting pnf $1 + \sum_{D \in \mathcal{D}} o_d$ is such that $o_d \checkmark$ for every $d \in \cup\mathcal{D}$.

(d) Suppose $n_1 = \sum_{a \in A} a.n_a \;\{+\,1\,\}$ and $n_2 = \sum_{b \in B} b.m_b \;\{+\,1\,\}$. Then using Proposition 7.4 and induction we can construct a pnf of the form $\sum_{d \in A \cup B} d.o_d \;\{+\,1\,\}$.

(e) The final possibility, when either n_1 or n_2 is Ω is straightforward, using the derived equation (**D3**).

□

Theorem 7.7 (Peer normal forms)**.** *For every $p \in$ CCS^f there exists a* pnf *n such that $p =_{\mathbf{P2P}} n$.*

Proof. By structural induction on p. The main case is covered by Lemma 7.6. □

One consequence of the completeness theorem will be that $p \sqsubseteq_{\mathbf{P2P}} 0$, whenever $p \notin \mathcal{U}\mathsf{p2p}$, because for such p we know $p \lesssim^{+}_{\mathsf{p2p}} 0$.[5] However it is useful to already have this result when proving completeness. A direct proof of this fact is not obvious. For example consider $p = a.(b.0 + c.1) + a.(b.1 + c.0)$ which we know not to be in $\mathcal{U}\mathsf{p2p}$. The derivation of $p \sqsubseteq_{\mathbf{P2P}} 0$ is not straightforward. But it becomes so if we first convert p to a pnf. This turns out to be

$$n_p = a.\sum_{A \in \mathcal{A}} n_A \qquad \text{where } \mathcal{A} = \{\,\{\,b\,\},\,\{\,c\,\},\,\{\,b,c\,\}\,\} \text{ and } n_b = n_c = \tau.1 + \tau.0$$

Now (**S4**) gives $n_b = n_c \sqsubseteq_{\mathbf{P2P}} 0$ and $n_p \sqsubseteq 0$ then follows by applications of the rule (**Zb**). This technique is quite general, and powerful.

Lemma 7.8. *Suppose $p \notin \mathcal{U}\mathsf{p2p}$. Then*
(1) $p \lesssim^{+}_{\mathsf{p2p}} q$ implies $p \sqsubseteq_{\mathbf{P2P}} q$
(2) $p \sqsubseteq_{\mathbf{P2P}} 0$.

[5] In general $0 \lesssim^{+}_{\mathsf{p2p}} p$ is not true, even if $p \notin \mathcal{U}\mathsf{p2p}$.

Proof. Part (2) is an immediate consequence of part (1) and the observation that $p \notin \mathcal{U}\mathsf{p2p}$ implies $p \sqsubseteq^{+}_{\approx\mathsf{p2p}} \mathsf{0}$. So we concentrate on the part (1), and we may assume that p is a pnf. We now proceed by induction on it's size and a case analysis of its form. However we know that $p \not\checkmark$ because otherwise we would have $\mathsf{1}$ must p; this eliminates many of possible forms. Also if p is Ω then the result is immediate, since Ω is a least element. So in effect we are left with two possibilities.

(a) Suppose p has the form $\sum_{a \in A} a.n_a$ for some $A \subseteq \mathsf{Act}$.

If $n_a \in \mathcal{U}\mathsf{p2p}$ for some $a \in A$ then it would follow that $p \in \mathcal{U}\mathsf{p2p}$ for p must $\overline{a}.q_a$ for any q_a satisfying n_a must q_a. So by induction we have $n_a \sqsubseteq_{\mathbf{P2P}} \mathsf{0}$ for every $a \in A$.

Because $p \sqsubseteq^{+}_{\approx\mathsf{p2p}} q$ we also know that q has essentially only one possible form, namely m_B $\{+\ \mathsf{1}\}$ for some $B \subseteq \mathsf{Act}$. Moreover since $\mathsf{f}.\,\mathsf{1} + p$ must $\overline{\mathsf{f}}.\,\mathsf{1} + \overline{c}.\,\mathsf{0}$ for any $c \in \mathsf{Act}\backslash A$ we have that $B \subseteq A$. We can now reason as follows:

$$\begin{aligned}
\sum_{a\in A} a.n_a \quad &\sqsubseteq_{\mathbf{P2P}} \quad \sum_{a\in A} a.\,\mathsf{0} && \text{Induction}\\
&\sqsubseteq_{\mathbf{P2P}} \quad \sum_{b\in B} b.\,\mathsf{0} && (\mathbf{Zb})\\
&\sqsubseteq_{\mathbf{P2P}} \quad \sum_{b\in B} b.m_b && (\mathbf{DZ1})
\end{aligned}$$

Finally suppose q has the summand $\mathsf{1}$. If A is empty we can use (**P2P1**). Otherwise the extra summand $\mathsf{1}$ can be added by an initial application of the derived equation (**DP3**).

(b) Suppose p has the form $\sum_{A \in \mathcal{A}} \tau.n_A$ for some saturated set $\mathcal{A}$.

Now suppose the empty set is in $\mathcal{A}$, that is $\tau.\,\mathsf{0}$ is a summand. Then using (**S4**) we obtain $p \sqsubseteq_{\mathbf{P2P}} \tau.\,\mathsf{0}$ and the required result now follows since $\tau.\,\mathsf{0}$ is a least element.

So we can assume $\emptyset \notin \mathcal{A}$. As a preliminary argument suppose that for all $A \in \mathcal{A}$, either $\mathsf{1} \in A$ or there exists some $a_A \in A$ such that $n_{a_A} \in \mathcal{U}\mathsf{p2p}$. Then let p' denote the peer

$$\mathsf{1} + \sum_{A \in \mathcal{A}, \mathsf{1} \notin A} \overline{a}.p'_{a_A}$$

where p'_{a_A} is chosen so that p'_{a_A} must n_{a_A}. Then one can check that p' must p, contradicting the fact that $p \notin \mathcal{U}\mathsf{p2p}$.

So we can assume that there is some $A_0 \in \mathcal{A}$ such that $\mathsf{1} \notin A_0$ and $n_a \notin \mathcal{U}\mathsf{p2p}$ for every $a \in A_0$. By induction $n_a \sqsubseteq_{\mathbf{P2P}} \mathsf{0}$ and by (**Zb**), $a.n_a \sqsubseteq_{\mathbf{P2P}} \mathsf{0}$ for every $a \in A_0$. As a result $n_{A_0} \sqsubseteq_{\mathbf{P2P}} \mathsf{0}$. Now an application of (**S4**) allows us to conclude $p \sqsubseteq_{\mathbf{P2P}} \tau.\,\mathsf{0}$, from which again the required result follows.

□

Client normal forms are simplifications of their peer counterparts. Note that the three extra peer inequations (**P2P1**) - (**P2P3**) are easily derivable from the two client inequations (**CLT1a**) and (**CLT1b**). So any inequation derived in the peer theory is also available in the client theory.

Definition 7.9 (Client normal forms, cnfs)**.** (1) Both Ω, $\mathsf{1}$ and $\tau.\,\mathsf{1}$ are cnfs.
(2) For any $A \subseteq \mathsf{Act}$ the sum $\sum_{a \in A} a.n_a$ is a cnf, provided each n_a is a cnf.

(3) Let $\mathcal{A}$ be a non-empty saturated set of non-empty subsets of Act. Suppose that for each $a \in \cup\mathcal{A}$, n_a is a cnf. Then $n = (\sum_{A\in\mathcal{A}} \tau.n_A)\ \{+\tau.1\}$ is a cnf.

Theorem 7.10 (Client normal forms)**.** *For every $p \in \mathsf{CCS}^f$ there exists a* cnf *n such that $p =_{\mathbf{CLT}} n$.*

Proof. First recall that the usability sets $\mathcal{U}\mathsf{clt}$ and $\mathcal{U}\mathsf{p2p}$ are identical. Using Theorem 7.7, we can assume that p can be transformed in to a pnf m, that is $p =_{\mathbf{CLT}} m$. Then by induction, and a systematic application of the derived unit equation $x + \mathsf{1} = \mathsf{1}$, m can then be transformed into a cnf. For example if m has the form $\mathsf{1} + m'$ then the resulting cnf is $\mathsf{1}$. Suppose it has the form $(\sum_{A\in\mathcal{A}} \tau.m_A)$, where $\mathcal{A}$ is a saturated set of non-empty subsets of Act_τ. Let $\mathcal{B} = \{\, A \in \mathcal{A} \mid \mathsf{1} \notin A \,\}$; $\mathcal{B}$ is still saturated. If it is empty the required cnf is $\tau.\mathsf{1}$. Otherwise it is

$$\sum_{B\in\mathcal{B}} \tau.m'_B\ \{+\tau.\mathsf{1}\ \text{ if } \mathsf{1} \in \cup\mathcal{A}\}$$

where for each $b \in \cup\mathcal{B}$, m'_b is the cnf obtained from m_b by induction. Here again the derived equation $x + \mathsf{1} = \mathsf{1}$ is used to transform $\tau.m_A$ into $\tau.\mathsf{1}$ for any A containing $\checkmark$. □

7.2. **Completeness for clients.** We first tackle the more straightforward case, the client preorder. For convenience we isolate a particularly significant case in the following lemma.

Lemma 7.11 (Stable state)**.** *Suppose $n = (\sum_{A\in\mathcal{A}} \tau.n_A)\ \{+\tau.1\}$ and m_B are both* cnfs *such that $n \sqsubseteq_{\mathsf{clt}} m_B$, and $n \in \mathcal{U}\mathsf{clt}$. Let $N = \{\, a \in \cup\mathcal{A} \mid n_a \in \mathcal{U}\mathsf{clt} \,\}$ and $B_0 = \{\, b \in B \mid m_b \text{ is different from } \mathsf{1} \,\}$. Then*

(1) there exists some $A \in \mathcal{A}$ such that $B_0 \subseteq A$ and $A \cap N \subseteq B$
(2) $\tau.n_b \sqsubseteq_{\mathsf{clt}} m_b$, for every $b \in B\ \cap\ \cup\mathcal{A}$.
(3) $n_b\checkmark$ implies $m_b\checkmark$ for every $b \in B\ \cap\ \cup\mathcal{A}$.

Proof. (1) Since $n \in \mathcal{U}\mathsf{clt}$ we know that there exists some server p_n such that p_n must n. Now suppose there is some $b \in B_0 \backslash (\cup\mathcal{A})$. Then $p_n + \overline{b}.\tau^\infty$ must n, from which $p_n + \overline{b}.\tau^\infty$ must m follows. But m_b is a cnf which is different from $\mathsf{1}$. By examining the other possibilities for m_b we see that τ^∞ must m_b is not possible, which contradicts $p_n + \overline{b}.\tau^\infty$ must m. So we can conclude that $B_0 \subseteq \cup\mathcal{A}$.

Now suppose, for another contradiction, that for every $A \in \mathcal{A}$ there exists some $a_A \in (A \cap N)\backslash B$. Let p denote the server $\sum_{A\in\mathcal{A}} a_A.p_a$, where the servers p_a are chosen so that p_a must n_a. Then because $\mathcal{A}$ is not empty p must n. This would imply p must m_B, which is clearly not possible. What this means is that there is some $A_1 \in \mathcal{A}$ such that $(A_1 \cap N) \subseteq B$. Let $A = A_1 \cup B_0$. Then since $B_0 \subseteq \cup\mathcal{A}$ and $\mathcal{A}$ is saturated we know $A \in \mathcal{A}$, and by construction it has the required properties.

(2) Suppose p must $\tau.n_b$, where $b \in B$; we have to show that p must m_b follows. Let p_n be the server used in part (1); it satisfies p_n must n. Then one can show that $p_n + \overline{b}.p$ must n, from which $p_n + \overline{b}.p$ must m follows. But this is only possible if p must m_b.

(3) Let $b \in B \cap \cup\mathcal{A}$ be such that $n_b\checkmark$. Then $p_n + b.\tau^\infty$ must n from which $p_n + b.\tau^\infty$ must m_B follows. But this will only be possible if $m_b\checkmark$.

□

Theorem 7.12 (Completeness: clients)**.** *In* CCS^f*,* $p \precsim^+_{clt} q$ *implies* $p \sqsubseteq_{\mathbf{C}} q$*.*

Proof. Let n, m be the cnfs for p, q respectively; we know that $n \precsim^+_{\mathsf{clt}} m$. The proof proceeds by induction on the combined size of n, m, and an exhaustive analysis of their possible structure, dictated by Definition 7.9. Note that because of Lemma 7.8 we can assume $n \in \mathcal{U}\mathsf{clt}$.

(a) If n is Ω the result is obvious, since Ω is a least element in the client equational theory. If it is $\mathsf{1}$, the argument is is also straightforward. We have $\mathsf{1} + \tau^\infty$ must m, since this server is guaranteed by $n = \mathsf{1}$. But this is only possible if $m\checkmark$; looking at the possible forms of cnfs in Definition 7.9 we see that m also has to be $\mathsf{1}$.

A similar argument, using the server $\mathsf{0}$, gives the result when n is $\tau.\mathsf{0}$.

(b) Now suppose n has the form n_A. Let us first look at the possible forms for the cnf m. Because $\mathsf{f}.\mathsf{1} + n_A \precsim_{\mathsf{clt}} \mathsf{f}.\mathsf{1} + m$ where f is fresh, m can not perform a τ action. So the only remaining possibility is that $m = m_B$ for some set of actions B.

Now suppose that there exists some $b \in B \backslash A$. Then since $\overline{\mathsf{f}}.\mathsf{0} + \overline{b}.\tau^\infty$ must $\mathsf{f}.\mathsf{1} + n_A$ we must have that τ^∞ must m_b, for this is the only way to ensure that $\overline{\mathsf{f}}.\mathsf{0} + \overline{b}.\tau^\infty$ must $\mathsf{f}.\mathsf{1} + m_B$. But m_b is a cnf and so it must be precisely $\mathsf{1}$.

If A is the empty set then the result now is immediate, since then $n = \mathsf{0}$, and we can apply (**CLT1c**) repeatedly to obtain $\mathsf{0} \sqsubseteq_{\mathbf{C}} m_B$.

At this stage we can use information available from Lemma 7.11 because $\sum_{A \in \{A\}} \tau.n_A \sqsubseteq_{\mathbf{C}} n_A \precsim^+_{\mathsf{clt}} m_B$. Part (1) gives that $B_0 \subseteq A$, and $N \subseteq B$ where B_0 and N are as defined in the statement of the lemma. So from part (2) we have that $\tau.n_a \precsim_{\mathsf{clt}} m_a$ for every $a \in A \cap B$. From Lemma 6.3 this gives $\tau.n_a \precsim^+_{\mathsf{clt}} m_a$; now using induction, which recall is on the combined size of the terms, we can assume $\tau.n_a \sqsubseteq_{\mathbf{C}} m_a$, and therefore $a.\tau.n_a \sqsubseteq_{\mathbf{C}} a.m_a$. If $n_a \not\checkmark$ an application of the standard equation (**S1a**), and the idempotence of $+$ we obtain $a.n_a \sqsubseteq_{\mathbf{C}} a.m_a$. On the other hand if $n_a\checkmark$ then from part (3) of Lemma 7.11 we also have $m_a\checkmark$. But both are cnfs and therefore both must coincide with $\mathsf{1}$. So for every $a \in A \cap B$ we have established $a.n_a \sqsubseteq_{\mathbf{C}} a.m_a$.

The argument is now completed as follows:

$$
\begin{array}{rlr}
n_A & = \sum_{a \in N} a.n_a + \sum_{a \in A \backslash N} a.n_a & \\
& \sqsubseteq_{\mathbf{C}} \sum_{a \in N} a.m_a + \sum_{a \in A \backslash N} a.n_a & \text{as argued above} \\
& \sqsubseteq_{\mathbf{C}} \sum_{a \in N} a.m_a + \sum_{a \in (A \backslash N) \cap B} a.n_a + \sum_{a \in (A \backslash N) \backslash B)} a.n_a & \\
& \sqsubseteq_{\mathbf{C}} \sum_{a \in N} a.m_a + \sum_{a \in (A \backslash N) \cap B} a.n_a & \text{Lemma 7.8, } (\mathbf{Zb}) \\
& \sqsubseteq_{\mathbf{C}} \sum_{a \in N} a.m_a + \sum_{a \in (A \backslash N) \cap B} a.m_a & \text{Lemma 7.8, } (\mathbf{DZ1}) \\
& \sqsubseteq_{\mathbf{C}} \sum_{a \in N} a.m_a + \sum_{a \in w} a.m_a + \sum_{b \in B \backslash A} b.\mathsf{1} & (\mathbf{CLT1c}) \\
& = \sum_{b \in B} b.m_b &
\end{array}
$$

The last line follows because

- B can be decomposed into the three disjoint sets N, $(A\backslash N) \cap B$ and $B\backslash A$
- if $b \in B\backslash A$ then m_b is $\mathsf{1}$; this follows because $B_0 \subseteq A$.

(c) There is one remaining case for the structure of n, namely $(\sum_{A\in\mathcal{A}} \tau.n_A)\ \{+\tau.1\}$. Here again we have to look at the possible structure of m. There are only two interesting cases.

The first is when m has the form m_B for some set $B \subseteq \mathsf{Act}$. This case fits the statement of Lemma 7.11 precisely. There must be some $A \in \mathcal{A}$ such that $B_0 \subseteq A$ and $A \cap N \subseteq B$, where again B_0 and N are as defined in the lemma.

Now using the fact that

$$A \;=\; (A\cap N) \;\uplus\; (A\backslash N)\cap B \;\uplus\; (A\backslash N)\backslash B$$

we can proceed as in case (b) to show

$$n_A \sqsubseteq_{\mathbf{C}} \sum_{a\in A\cap N} a.m_a \;+\; \sum_{a\in(A\backslash N)\cap B} a.m_a \tag{7.1}$$

The set B can also be decomposed as

$$B \;=\; (A\cap N) \;\uplus\; (A\backslash N)\cap B \;\uplus\; B\cap(N\backslash A)$$

Moreover since $B_0 \subseteq A$ for every $b \in B\cap(N\backslash A)$ the residual m_b must be $\mathsf{1}$. Therefore using applications of (**CLT1c**) to (7.1) we can obtain $n_A \sqsubseteq_{\mathbf{C}} n_B$. The required result, $n \sqsubseteq_{\mathbf{C}} m_B$ now follows by (**S4**).

The other interesting case is when m has the form $(\sum_{B\in\mathcal{B}} \tau.m_B)\ \{+\tau.1\}$. Here $m \mathrel{\underset{\approx}{\sqsubset}}^{+}_{\text{clt}} m_B$ for every $B \in \mathcal{B}$, from which $n \mathrel{\underset{\approx}{\sqsubset}}^{+}_{\text{clt}} m_B$ follows. Again we can proceed as in (b) to show $n \sqsubseteq_{\mathbf{C}} m_B$.

To complete we use the fact that $n =_{\mathbf{CLT}} \sum_{B\in\mathcal{B}} \tau.n$. This follows from the derived law (**D1**) and the idempotency of $+$.

□

7.3. **Completeness for peers.** The completeness result for peers follows the same structure as that for clients. But it is complicated by the more intricate form of pnfs; in particular pnfs of the form $\mathsf{1} + n$, where n is non-trivial. We need a generalisation of Lemma 7.11 for peers, which in turn requires a preliminary result.

Lemma 7.13. *Suppose $p \in \mathcal{U}\mathsf{p2p}$ and $p\not\checkmark$. Then there exists some q such that $q\not\checkmark$ and q must p.*

Proof. We may assume that p is a pnf. If it has the form $\sum_{a\in A} a.n_a$ there must exist some $a \in A$ such that $n_a \in \mathcal{U}\mathsf{p2p}$. From this we get some q_a satisfying q_a must n_a, and the required q is $\overline{a}.q_a$.

Otherwise p must have the form $\sum_{A\in\mathcal{A}} \tau.n_A$. From the analysis carried out in the proof of Lemma 7.8 we know that for every $A \in \mathcal{A}$ either $\mathsf{1} \in A$ or there exists some $a_A \in A$ such that $n_a \in \mathcal{U}\mathsf{p2p}$. The required q is then $\tau.\mathsf{1} + \sum_{A\in\mathcal{A},\mathsf{1}\notin A} \overline{a}.q_{a_A}$, where the peer q_{a_A} is chosen so that n_a must q_{a_A}.

□

Lemma 7.14 (Stable state)**.** *Suppose* $n = (\sum_{A\in\mathcal{A}} \tau.n_A)\ \{+1\}$ *and* m_B*, where* $B \subseteq \mathsf{Act}_{\checkmark}$*, are both* pnfs *such that* $n \sqsubseteq_{\mathsf{p2p}} m_B$ *and* $n \in \mathcal{U}\mathsf{clt}$*. Let* $N = \{\, a \in \cup\mathcal{A} \mid n_a \in \mathcal{U}\mathsf{p2p} \,\}$ *and* $B_0 = B\backslash \checkmark$*. Then*

(1) there exists some $A \in \mathcal{A}$ *such that* $B_0 \subseteq A$ *and* $A \cap N \subseteq B_0$
(2) $\tau.n_b \sqsubseteq_{\mathsf{p2p}} m_b$*, for all* $b \in B\ \cap\ \cup\mathcal{A}$
(3) $n_b \checkmark$ *implies* $m_b \checkmark$*, for all* $b \in B\ \cap\ \cup\mathcal{A}$*.*

Proof. Let p_n be any peer satisfying p_n must n. We know at least one exists and because of the previous lemma we may assume that $p_n \not\checkmark$.

(1) This is similar to the proof of part (1) of Lemma 7.11 although here we are dealing with peers rather than clients. Suppose there is some $b \in B$ such that $b \notin \cup\mathcal{A}$. Then $p_n + \overline{b}.\mathsf{0}$ must n. This contradicts the fact that $n \sqsubseteq_{\mathsf{p2p}} m_B$ since m_B can not guarantee the success of the peer $p_n + \overline{b}.\mathsf{0}$. So we have established $B_0 \subseteq \cup\mathcal{A}$.
 We can continue as in part (1) of Lemma 7.11 to show that there exists some $A_1 \in \mathcal{A}$ such that $A_1 \cap N \subseteq B_0$. The required A can now be taken to be $A_1 \cup B_0$.
(2) Suppose p must $\tau.n_b$. This means that $p_n + \overline{b}.p$ must n, from which $p_n + \overline{b}.p$ must m_B follows. By construction $p_n \not\checkmark$ and so if $\checkmark \notin B$ this implies that p must m_b. On the other hand if $\checkmark \in B$ we can only deduce that m_b must p. But by the construction of pnfs, if $\checkmark \in B$ then we also know that $m_b \checkmark$. The required p must m_b now follows.
(3) For an arbitrary $b \in B\ \cap\ \cup\mathcal{A}$ suppose $n_b \checkmark$. Then $p_n + \overline{b}.(\mathsf{1} + \tau^\infty)$ must n, and so this must also be true of m_B. But, since $p_n \not\checkmark$, this is only possible if $m_b \checkmark$.

□

Before embarking on the main proof of completeness it is convenient to isolate one particular case.

Lemma 7.15. $n \sqsubseteq^+_{\mathsf{p2p}} \mathit{1}$ *implies* $n \sqsubseteq_{\mathbf{P2P}} \mathit{1}$*.*

Proof. We may assume that n is a pnf, and we use a case analysis on its structure. When it has the form $\Omega\ \{+\ \mathsf{1}\}$ the result is obvious.

(1) So consider the case when it has the form $\sum_{a\in A} a.n_a\ \{+\ \mathsf{1}\}$ for some $A \subseteq \mathsf{Act}$. Now suppose there is some $a \in A$ such that $n_a \in \mathcal{U}\mathsf{p2p}$; so there is a peer p_a such that p_a must n_a. This means that $\overline{a}.p_a$ must n. But this would imply that $\overline{a}.p_a$ must $\mathsf{1}$, which is impossible since $\mathsf{1}$ $\not\text{m}$ust $\overline{a}.p_a$. So what we have shown is that $n_a \notin \mathcal{U}\mathsf{p2p}$ for every a in Act and therefore $\sum_{a\in A} n_a \notin \mathcal{U}\mathsf{p2p}$. The result now follows from Lemma 7.8.
(2) The only other possibility is that it has the form $\sum_{A\in\mathcal{A}} \tau.n_A\ \{+\ \mathsf{1}\}$. For a contradiction suppose that for all $A \in \mathcal{A}$ there exists some $a_A \in A$ such that p_A must n_{a_A}. This means that n must p, where p is the peer $\sum_{A\in\mathcal{A}} \overline{a_A}.p_a$. But this contradicts the fact that $n \sqsubseteq_{\mathsf{p2p}} \mathsf{1}$ since $\mathsf{1}$ $\not\text{m}$ust p; the peer $\mathsf{1}$ cannot induce p into a successful state.
 So we have established that there is some $A \in \mathcal{A}$ such that $n_a \notin \mathcal{U}\mathsf{p2p}$ for every $a \in A$. Using Lemma 7.8 and (**DZ1**) we can derive $n_A \sqsubseteq_{\mathbf{P2P}} \mathsf{0}\ \{+\ \mathsf{1}\}$, from which the result follows, since $n \sqsubseteq_{\mathbf{P2P}} \tau.n_A$.

□

Theorem 7.16 (Completeness: peers)**.** *In* CCS^f*,* $p \sqsubseteq^+_{\mathsf{p2p}} q$ *implies* $p \sqsubseteq_{\mathbf{P2P}} q$*.*

Proof. The proof follows the same structure as that of Theorem 7.12, but there are more details to be considered. Here let n, m be the pnfs for p, q respectively; the proof proceeds by induction on the combined size of n, m, and an analysis of their possible structure, as given in Definition 7.2. Because of Lemma 7.8 we may also assume that $n \in \mathcal{U}\mathsf{p2p}$. We also leave the uninteresting case when it has the form $\Omega\ \{+\ \mathsf{1}\}$ to the reader.

(a) $n = (\sum_{A \in \mathcal{A}} \tau.n_A)\ \{+\mathsf{1}\}$ and $m = m_B$ for some $B \subseteq \mathsf{Act}_\checkmark$. This is precisely the case to which Lemma 7.14 applies. Let $A \in \mathcal{A}, N$ and B_0 be as given in the statement of the lemma; because of Lemma 7.15 we can assume that B_0 is not empty. Let A_0 be $A\backslash\{\ \checkmark\ \}$. Our aim is to show

$$n_{A_o} \sqsubseteq_{\mathbf{P2P}} m_{B_0} \tag{7.2}$$

from which the required result will follow. This is a consequence of the following:

- if $\mathsf{1}$ is a summand of n then it must also be a summand of m_B; to see this consider the peer $\mathsf{1} + \tau^\infty$.
- $m_{B_0} \sqsubseteq_{\mathbf{P2P}} m_B$; for if $\checkmark \in B$ then by condition (2)(ii) of Definition 7.2 m_b must be of the form $\mathsf{1} + m'_b$ for every $b \in B_0$, and because B_0 is not empty we can apply an instance of (**P2P2**) to one summand $b.m_b$ of m_{B_0}, to obtain $m_{B_0} \sqsubseteq_{\mathbf{P2P}} m_B$.

So let us concentrate on establishing (7.2). This relies on the following set decompositions:

$$\begin{aligned} A_0 &= (A_0 \cap N) \ \uplus\ (A_0\backslash N) \cap B_0 \ \uplus\ (A_0\backslash N)\backslash B_0 \\ B_0 &= (A_0 \cap N) \ \uplus\ (A_0\backslash N) \cap B_0 \end{aligned}$$

The argument now proceeds in much the same way as in the corresponding case, (b), of Theorem 7.12:

$$\begin{aligned} n_{A_0} &= \sum_{a \in (A_0 \cap N)} a.n_a + \sum_{a \in (A_0\backslash N)\cap B_0} a.n_a + \sum_{a \in A_0\backslash N\backslash B_0} a.n_a \\ &\sqsubseteq_{\mathbf{P2P}} \sum_{a \in (A_0 \cap N)} a.m_a + \sum_{a \in (A_0\backslash N)\cap B_0} a.n_a + \sum_{a \in A_0\backslash N\backslash B_0} a.n_a && (\star) \\ &\sqsubseteq_{\mathbf{P2P}} \sum_{a \in (A_0 \cap N)} a.m_a + \sum_{a \in (A_0\backslash N)\cap B_0} a.n_a && \text{Lemma 7.8, } (\mathbf{Zb}) \\ &\sqsubseteq_{\mathbf{P2P}} \sum_{a \in (A_0 \cap N)} a.m_a + \sum_{a \in (A_0\backslash N)\cap B_0} a.m_a && \text{Lemma 7.8, } (\mathbf{DZ1}) \\ &= m_{B_0} \end{aligned}$$

The step $(\star)$ is uses induction. From part (2) of Lemma 7.14 we know $\tau.n_a \sqsubsim_{\mathsf{p2p}} m_a$ for every $a \in A_0 \cap N$. Lemma 6.3 and induction gives $\tau.n_a \sqsubseteq_{\mathbf{P2P}} m_a$. There are now two cases. If $n_a \not\checkmark$ then an application of (**S1a**) gives $a.n_a \sqsubseteq_{\mathbf{P2P}} a.m_a$. However if $n_a \checkmark$ this equation can not be used. However we can achieve the same conclusion as follows:

$$\begin{aligned} a.n_a &\sqsubseteq_{\mathbf{P2P}} a.(\mathsf{1} + \tau.n_a) && (\mathbf{DP3}) \\ &\sqsubseteq_{\mathbf{P2P}} a.(\mathsf{1} + m_a) && \text{Induction} \\ &= a.m_a \end{aligned}$$

The last line follows from part (3) of Lemma 7.14.

(b) Suppose n is as in the previous case but that m is $(\sum_{B \in \mathcal{B}} \tau.m_B)\ \{+\mathsf{1}\}$. Here we proceed as in case (c) of Theorem 7.12. Regardless of the presence or absence of the optional units, one can show that $n \sqsubsim_{\mathsf{p2p}} m_B$ for every $B \in \mathcal{B}$. Therefore by part (a) we have $n \sqsubseteq_{\mathbf{P2P}} m_B$.

Now suppose $n \not\checkmark$, that is n does not contain $\mathsf{1}$ as a summand. Then using the derived **D1** we have

$$n = \tau.n \;=\sum_{B\in\mathcal{B}} \tau.n \qquad \sqsubseteq_{\mathbf{P2P}} \sum_{B\in\mathcal{B}} \tau.m_B \tag{7.3}$$

If m also does not contain the summand $\mathsf{1}$ we are finished. But if it does, we know that $\mathcal{B}$ is non-empty and that each $B \in \mathcal{B}$ contains $\checkmark$. Pick one such B_0, and applying (**P2P**) we obtain $\tau.m_{B_0} \sqsubseteq_{\mathbf{P2P}} \tau.m_{B_0} + \mathsf{1}$. Using this in (7.3) above we obtain the required $n \sqsubseteq_{\mathbf{P2P}} \sum_{B\in\mathcal{B}} m_B + \mathsf{1}$.

Finally if both n and m have $\mathsf{1}$ as a summand a simple variation on the argument (7.3) above suffices.

(c) Now suppose that n has the form n_A for some $A \subseteq \mathsf{Act}_{\checkmark}$. Reasoning as in the corresponding case of Theorem 7.12 we see that the only possible form for m is m_B for some $B \subseteq \mathsf{Act}_{\checkmark}$. Now we use the fact that $\sum_{A\in\{A\}} \tau.n_A \sqsubseteq^{+}_{\approx \mathsf{p2p}} n_A \sqsubseteq_{\approx \mathsf{p2p}} m_B$ to apply Lemma 7.14. This gives that $B_0 \subseteq A$, $A \cap N \subseteq B_0$, where B_0, N are as described in that lemma. Now we can repeat the argument used in case (b) to show that $n_{A_0} \sqsubseteq_{\mathbf{P2P}} m_{B_0}$ where again A_0 denotes $A\backslash\{\checkmark\}$. Again a simple case analysis on whether $\checkmark$ is in either of A, B, as used also in case (b), will allow us to conclude that $n_A \sqsubseteq_{\mathbf{P2P}} m_B$.

$\square$

8. 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 *subserver* 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 CCS. As there are fewer contexts, their sub-server preorder differs from our server preorder: $a_1.\mathsf{1} \preceq_s a_1.\mathsf{1} + a_2.\mathsf{1}$, whereas $a_1.\mathsf{1} \not\sqsubseteq_{\approx \mathsf{svr}} a_1.\mathsf{1} + a_2.\mathsf{1}$.

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 3.16 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 $\sqsubseteq_{\approx \mathsf{svr}}$. The subcontract relation of [LP07] turns out to be not comparable with $\sqsubseteq_{\approx \mathsf{svr}}$, whereas the strong subcontract $\sqsubseteq$ of [Pad10] is strictly contained in $\sqsubseteq_{\approx \mathsf{svr}}$, as the LTS there is convergent and finite branching. The comparison of $\sqsubseteq_{\approx \mathsf{svr}}$ 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 ($\sqsubseteq_{\approx \mathsf{p2p}} \not\subseteq \sqsubseteq^{\mathsf{ds}}$), 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*, both similar to our notion of *usability*.

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 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 $\sqsubseteq_{\mathsf{p2p}}$.

Future work: The most obvious open question about our two new refinement preorders $\sqsubseteq_{\mathsf{clt}}$ and $\sqsubseteq_{\mathsf{p2p}}$ 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 $\sqsubseteq_{\mathsf{clt}}$, $\sqsubseteq_{\mathsf{p2p}}$ themselves rather than their associated pre-congruences $\sqsubseteq^{+}_{\mathsf{clt}}$ and $\sqsubseteq^{+}_{\mathsf{p2p}}$. 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.

A further interesting question is the possible use of the parallel operator between clients and peers, either by allowing multi-party interactions as in [BZ09, BMPR09], or by deciding on how a parallel combination of clients should report success.

We have also 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 $\sqsubseteq_{\mathsf{p2p}}$ 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 of [Pad10].

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

Appendix A. Justifying the derived equations

(**D1**): The proof is by induction on n. For $i = 1$, the result follows by (**S1b**) and (**S4**). Assume it is true for k; that is $\tau.z = z$, where z abbreviates $\sum_{1\leq i\leq k} \tau.x_i$. Then

$$\begin{aligned} \tau.z + \tau.x_{k+1} &= \tau.(\tau.z + \tau.x_{k+1}) && (\mathbf{S1a}) \\ &= \tau.(z + \tau.x_{k+1}) && \text{Induction} \end{aligned}$$

(**D2**):

$$\begin{aligned} x_{\not\checkmark} + \tau.(x_{\not\checkmark} + y) &= \tau.(x_{\not\checkmark} + x_{\not\checkmark} + y) + \tau.(x_{\not\checkmark} + y) && (\mathbf{S2}) \\ &= \tau.(x_{\not\checkmark} + y) && \text{Idempotency} \end{aligned}$$

(**D3**): One direction is immediate from (**S5**). Here is the converse:

$$\begin{array}{rcll} \mu.x + \Omega & \leq & \mu.x + \tau.\Omega & (\mathbf{S5}) \\ & \leq & \tau.(\mu.x + \Omega) + \tau.\Omega & (\mathbf{S2}) \\ & \leq & \Omega & (\mathbf{S4}) \end{array}$$

(**DP1**): One direction is straightforward from (**P2P2**). Conversely:

$$\begin{array}{rcll} x & \leq & x + 1 & (\mathbf{P2P1}) \\ 1 + \mu.x & \leq & 1 + \mu.(x + 1) & \text{Pre-congruence} \\ & \leq & 1 + \mu.(\tau.x + 1) & (\mathbf{P2P3}\text{, Idempotency} \\ & \leq & 1 + \mu.(\tau.x + 1) & (\mathbf{S4})\text{, Idempotency} \end{array}$$

(**DP2**): This is a generalisation of (6.3) above. It is proved by induction on n. The case when $n = 1$ has already been discussed in (6.4) above. For the inductive case let r denote $\sum_{1 \leq i \leq k} \tau.x_i$. By (**D1**) $r = \tau.\tau.r$ can be derived. Then

$$\begin{array}{rcll} \tau.(1 + \tau.x_{k+1} + r) & = & \tau.(1 + \tau.x_{k+1} + \tau.(\tau.r)) & \\ & = & \tau(1 + x_{(k+1)}) + \tau(1 + \tau.z) & (6.3) \text{ above} \\ & = & \tau(1 + x_{(k+1)}) + \sum_{1 \leq k} \tau.(1 + x_i) & \text{Induction} \end{array}$$

(**DP3**):

$$\begin{array}{rcll} \mu.x & \leq & \mu.(1 + x) & \mathbf{P2P1}\text{, Pre-congruence} \\ & \leq & \mu.(1 + \tau.x) & \mathbf{P2P3}\text{, Idempotency} \end{array}$$

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.

[BH13] Giovanni Bernardi and Matthew Hennessy. Mutually testing processes - (extended abstract). In Pedro R. D'Argenio and Hernán C. Melgratti, editors, *CONCUR*, volume 8052 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 2013.

[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.