Compliance and Testing Preorders Differ*

Giovanni Bernardi and Matthew Hennessy

School of Computer Science, University of Dublin, Trinity College, Ireland

Abstract

Contracts play an essential role in the Service Oriented Computing, for which they need to be equiped with a *sub-contract relation*. We compare two possible formulations, one based on *compliance* and the other on the testing theory of De Nicola and Hennessy. We show that if the language of contracts is sufficiently expressive then the resulting *sub-contract relations* are incomparable.

However if we put natural restrictions on the contract language then the *sub-contract relations* coincide, at least when applied to servers. But when formulated for clients they remain incomparable, for many reasonable contract languages. Finally we give one example of a contract language for which the client-based *sub-contract relations* coincide.

1 Introduction

Contracts play a central role in the orchestration and development of web services, [CCLP06, LP07]. Existing services are advertised for use by third parties, which may combine these existing services to construct, and in turn advertise for further use, new services. The behavioural specification of advertised services is given via *contracts*, high-level descriptions of expected behaviour, which should come equipped with a *sub-contract* relation. Intuitively $\textsc{ct}_1 \sqsubseteq_{\textsc{crt}} \textsc{ct}_2$ means that a third party requiring a service to provide contract $\textsc{ct}_1$ may use one which already provides $\textsc{ct}_2$, so in this sense $\textsc{ct}_2$ is better than $\textsc{ct}_1$. The purpose of this short technical note is to compare and contrast two different approaches to defining this *sub-contract* relation.

The first method, [LP07, CGP09, Pad10], is based on a notion of *compliance* between two contracts, where one contract notionally formalises the behaviour offered by a server p, and the other one the behaviour offered by a client r. Contracts are interpreted as abstract processes, written in process algebras similar to CCS or CSP, [Mil89, Hoa85]. However, as pointed out by [Bd10, BH12] they can also be viewed as session types [THK94, GH05]. Intuitively p and r are in compliance, written $r \dashv p$, if when viewed as abstract processes they can continuously interact, and if this interaction ever stops then the client is in a *happy state*; the formal definition is co-inductive and is given in Definition 2.4. This leads to a natural comparison between server-oriented contracts : $p_1 \sqsubseteq^{\mathsf{cpl}}_{\textsc{svr}} p_2$ if every client which complies with p_1 also complies with p_2. As suggested in [Bd10], client-oriented contracts can also be compared, but in terms of the servers with which they comply, $r_1 \sqsubseteq^{\mathsf{cpl}}_{\textsc{clt}} r_2$.

It has been pointed out by various authors [LP07, CGP09, Pad09, Pad10] that the server contract preorder, $\sqsubseteq^{\mathsf{cpl}}_{\textsc{svr}}$, bears a striking resemblance to the well-known *must-testing* preorders from [DH84]. For example, the axioms for the strong sub-contract relation in [Pad09, Table 1], are essentially the same for the testing preorder in [Hen85, Figure 3.6]; and the behavioural characterisations of the sub-contract relation use *ready sets*, which were already in the behavioural characterisation of the *must-testing* preorder [DH84]. In this approach clients are viewed as *tests* for servers and servers are compared by their ability to guarantee that tests are satisfied. This is formalised as an *inductive* relation between tests and servers. Intuitively p $\textsc{must}$ r if

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

whenever the two abstract processes p, r are executed in parallel the test r is guaranteed to reach a *happy state*. This in turn leads to a second pair of *sub-contract* relations, which we denote by $p_1 \sqsubseteq^{\mathsf{tst}}_{\text{SVR}} p_2$ and $r_1 \sqsubseteq^{\mathsf{tst}}_{\text{CLT}} r_2$ respectively.
In this paper we contrast these two different approaches to the notion of *sub-contract* by comparing the relations $\sqsubseteq^{\mathsf{cpl}}_{\star}$ and $\sqsubseteq^{\mathsf{tst}}_{\star}$, for both servers and clients. This study is of interest because the testing-based preorders have been thoroughly studied. In particular $\sqsubseteq^{\mathsf{tst}}_{\text{SVR}}$ has a behavioural characterisation, an axiomatisation (for finite terms) [DH84, Hen85], a logical characterisation [CH10], and an algorithm to decide it (on finite state LTSs) [CH93]; moreover the client preorder $\sqsubseteq^{\mathsf{tst}}_{\text{CLT}}$ has recently been investigated in [Ber13].

The outcome of the comparison depends on the expressive power of the language used to express contracts. We examine three different possibilities. The first is when there is no restriction on the contract language. We essentially allow any description of behaviour from the process calculus CCS; this includes infinite state and potentially divergent contracts. In this case the preorders are incomparable; see Section 3.1

In the second case we restrict the contract language to what we call $\mathsf{CCS}_{\mathsf{web}}$; this only allows finite-state contracts, which can never give rise to divergent behaviour; this language includes all the contract languages used in the standard literature, such as [LP07, Bd10, Pad09] and the concrete one of [CGP09]. In this setting the two server-contract preorders coincide:

$$p_1 \sqsubseteq^{\mathsf{cpl}}_{\text{SVR}} p_2 \text{ if and only if } p_1 \sqsubseteq^{\mathsf{tst}}_{\text{SVR}} p_2$$

However the client-contract preorders remain incomparable. This is discussed in Section 3.2.

It turns out that the difference in the formulation of the *compliance* relation between contracts and that based on *must-testing*, one co-inductive and the other inductive, has significant implications on the client-preorders, regardless of the expressivity of the contract language. This is explained via examples in Section 3.3. In particular it is difficult to think of a reasonable contract language in which they coincide. We provide one example, also in Section 3.3, which essentially coincides with the *finite session behaviours* of [Bd10]; one can think of these as *first-order* session types [THK94]. But, as we will see, introducing recursion into this contract language will once more enable us to differentiate between the two client-preorders.

The remainder of the paper is structured as follows. In the next section, Section 2, we provide formal definitions for the concepts introduced informally above, together with a description of the abstract language CCS, which is used as a general description language for contracts. Then the three different scenarios are discussed in turn in Section 3. Finally we discuss the related literature in Section 4.

2 LTS and behavioural preorders

A labelled transition system, LTS, consists of a triple $\langle P, \longrightarrow, \mathsf{Act}_{\tau\checkmark} \rangle$ 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 let λ range over $\mathsf{Act}_{\tau\checkmark}$, and μ range over Act_{τ}. We use the infix notation $p \xrightarrow{\lambda} q$ in place of $(p, \lambda, q) \in \longrightarrow$. Let CCS be the set of terms defined by the grammar

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

where $\mu \in \mathsf{Act}_{\tau}$, I is a countable index sets, and $A, B, C, \ldots$ range over a set of definitional constants each of which has an associated definition $A \overset{\mathsf{def}}{=} p_A$. 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 $\sum_{i \in \{1,2\}} p_i$. Note that we have omitted

$$\frac{}{\mathsf{1} \xrightarrow{\checkmark} \mathsf{0}} \text{[A-OK]} \qquad \frac{}{\mu.p \xrightarrow{\mu} p} \text{[A-PRE]}$$

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

$$\frac{p \xrightarrow{\lambda} p'}{A \xrightarrow{\lambda} p'} \; A \overset{\mathsf{def}}{=} p; \text{ [R-CONST]}$$

Figure 1: The operational semantics of CCS

$$\frac{q \xrightarrow{\lambda} q'}{q \mid\mid p \xrightarrow{\lambda} q' \mid\mid p} \text{[P-LEFT]} \qquad \frac{p \xrightarrow{\lambda} p'}{q \mid\mid p \xrightarrow{\lambda} q \mid\mid p'} \text{[P-RIGHT]}$$

$$\frac{q \xrightarrow{\alpha} q' \quad p \xrightarrow{\overline{\alpha}} p'}{q \mid\mid p \xrightarrow{\tau} q' \mid\mid p'} \text{[P-SYNCH]}$$

Figure 2: The operational semantics of contract composition

the parallel operator $||$, as contracts, and their associated session types [Bd10, BH12], are normally expressed purely in terms of prefixing and choices.

The operational semantics of the language is given by the LTS generated by the relations $p \xrightarrow{\lambda} q$ determined by the rules given in Figure 1. The *happy* or successful states mentioned in the Introduction are considered to be those CCS terms satisfying $p \xrightarrow{\checkmark}$.

We use standard notation for operations in LTSs. For example $\mathsf{Act}^{\star}_{\tau\checkmark}$, ranged over by t, denotes the set of *finite* sequences of actions from the set $\mathsf{Act}_{\tau\checkmark}$, and for any $t \in \mathsf{Act}^{\star}_{\tau\checkmark}$ we let $p \xrightarrow{t} q$ be the obvious generalisation of the single transition relations to sequences. For an infinite sequence $u \in \mathsf{Act}^{\infty}_{\tau\checkmark}$ of the form $\lambda_0\lambda_1\ldots$ we write $p \xrightarrow{u}$ to mean that there is an infinite sequence of actions $p \xrightarrow{\lambda_0} p_o \xrightarrow{\lambda_1} p_1 \ldots$. These action relations are lifted to the weak case in the standard manner, giving rise to $p \overset{s}{\Longrightarrow} q$ for $s \in \mathsf{Act}^{\star}_{\checkmark}$ and $p \overset{u}{\Longrightarrow}$ for $u \in Act^{\infty}$. We write $\Longrightarrow$ in place of $\overset{\varepsilon}{\Longrightarrow}$, where ε denotes the empty string. Finally a process *diverges*, written $p \Uparrow$, if there is an infinite sequence of actions $p \xrightarrow{\tau} p_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} p_k \xrightarrow{\tau} \ldots$. Otherwise it is said to *converge*, written $p \Downarrow$.

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 (2).

Definition 2.1. [Compliance]
Let $\mathcal{F}^{\dashv} : \mathcal{P}(\mathsf{CCS}^2) \longrightarrow \mathcal{P}(\mathsf{CCS}^2)$ be the rule functional defined so that $(r, p) \in \mathcal{F}^{\dashv}(\mathcal{R})$ whenever the following conditions are true:

(a) if $p \Uparrow$ then $r \xrightarrow{\checkmark}$

(b) if $r \mid\mid p \overset{\tau}{\not\longrightarrow}$ then $r \xrightarrow{\checkmark}$

(c) if $r \mid\mid p \xrightarrow{\tau} r' \mid\mid p'$ then $r' \mathcal{R}\, p'$

If $X \subseteq \mathcal{F}^{\dashv}(X)$, then we say that X is a *co-inductive* compliance *relation*. The monotonicity of $\mathcal{F}^{\dashv}$ and the Knaster-Tarski theorem ensure that there exists the greatest solution of the equation $X = \mathcal{F}^{\dashv}(X)$; we call this solution the *compliance relation*, and we denote it $\dashv$. That is $\dashv = \nu X.\mathcal{F}^{\dashv}(X)$. If $r \dashv p$ we say that the client r *complies with* the server p. □

Thanks to its co-inductive nature, the compliance admits everlasting computations, even if the client side never reaches a happy state. This is a typical feature of the compliance relation.

Example 2.2. Let $C \overset{\mathsf{def}}{=} \tau.\alpha.C$ and $S \overset{\mathsf{def}}{=} \overline{\alpha}.S$. Even if C can not reach a happy state, it complies with S, for $\{ (C, S), (\alpha.C, S) \}$ is a co-inductive compliance. This set enjoys the properties required by Definition 2.1: Point (a) is trivially true for S converges, point (b) is true because $C \mid\mid S \xrightarrow{\tau}$ and $\alpha.C \mid\mid S \xrightarrow{\tau}$. A routine check shows that also point (c) is true. □

Another property of $\dashv$ is that it is preserved by the interactions of contracts.

Lemma 2.3. If $r \dashv p$ and $r \mid\mid p \xrightarrow{\tau}^{*} r' \mid\mid p'$ then $r' \dashv p'$.

Proof. It follows from induction on the number of reduction steps in $\xrightarrow{\tau}^{*}$, and point (c) of Definition 2.1. □

Definition 2.4. [Compliance preorders]
In an arbitrary LTS we write

(1) $p_1 \sqsubseteq^{\mathsf{cpl}}_{\text{SVR}} p_2$ if for every r, $r \dashv p_1$ implies $r \dashv p_2$

(2) $r_1 \sqsubseteq^{\mathsf{cpl}}_{\text{CLT}} r_2$ if for every p, $r_1 \dashv p$ implies $r_2 \dashv p$ □

Note that our compliance relation is slightly different than that of [LP07]; we require that a client that complies with a divergent server report success immediately, whereas in [LP07] the client may report success in the future, and cannot engage in any interaction. This does not affect the resulting *sub-contract* relations on the language of contracts discussed in [CGP09, Pad10].

We also briefly recall the notion of *must-testing* from [DH84] A *computation* consists of series of τ actions of the form

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

It is *maximal* if it is infinite, or whenever $r_n \mid\mid p_n$ is the last state then $r_n \mid\mid p_n \overset{\tau}{\not\longrightarrow}$. A computation may be viewed as two processes p, r, one a server and the other a client, co-operating to achieve individual goals. We say that (1) is *client-successful* if there exists some $k \geq 0$ such that $r_k \xrightarrow{\checkmark}$.

Definition 2.5. [Testing preorders]
In an arbitrary LTS we write p MUST r if every maximal computation of $r \mid\mid p$ is *client-successful.* Then

(1) $p_1 \sqsubseteq^{\mathsf{tst}}_{\text{SVR}} p_2$ if for every r, p_1 MUST r implies p_2 MUST r

(2) $r_1 \sqsubseteq^{\mathsf{tst}}_{\text{CLT}} r_2$ if for every p, p MUST r_1 implies p MUST r_2 □

Before comparing the testing and the compliance preorders, we highlight the differences between $\dashv$ and $\textsc{must}$. We use standard examples [LP07, Ber13]. The discussion on the preorders will mirror the differences shown in these examples.

Example 2.6. [Meaning of livelocks]
In this example we prove that $r \dashv p$ does not imply $p \ \textsc{must}\ r$. Recall the contracts C and S from Example 2.2. In that example we have seen that since $\{(C,\, S),\ (\alpha.C,\, S)\}$ is a co-inductive compliance, $C \dashv S$.

The fact that $S \ \not\!\textsc{must}\ C$ is true because C does not perform $\checkmark$, and so no computation of $C \mid\mid S$ is client-successful. $\square$

The previous example shows that while the compliance admits livelocks where clients do not report success, the must testing does not. The testing relation requires clients to reach a successful state in every (maximal) computation.

Example 2.7. [Meaning $\checkmark$]
In this example we prove that $p \ \textsc{must}\ r$ does not imply $r \dashv p$. Let $r = \mathsf{1} + \tau.\mathsf{0}$. For every p, $p \ \textsc{must}\ r$ because $r \xrightarrow{\checkmark}$, so all the computations of $r \mid\mid p$ are client-successful. For every p, the proof that $r \not\dashv p$ relies on the following computation,

$$r \mid\mid p \xrightarrow{\tau} \mathsf{0} \mid\mid p \xrightarrow{\tau} \ldots$$

Since $\mathsf{0} \not\dashv p$, Lemma 2.3 implies that $r \not\dashv p$. $\square$

In the must testing, the behaviour of a client that has reported success is completely disregarded; that is $p \ \textsc{must}\ r$ and $r \mid\mid p \xrightarrow{\tau} r' \mid\mid p'$ does not imply $p' \ \textsc{must}\ r'$. For the compliance it is the contrary, as we have seen in Lemma 2.3.

3 Examples

We have three sub-sections, each examining one of the scenarios for contracts alluded to in the Introduction.

3.1 General contracts

Here we assume that contracts may be any term in the language CCS defined above. First we show that the server-contract preorders are incomparable.

Example 3.1. [Infinite traces and servers]
Here we prove that $p \sqsubseteq^{\mathsf{cpl}}_{\text{SVR}} q$ but $p \not\sqsubseteq^{\mathsf{tst}}_{\text{SVR}} q$ where these terms are depicted in Figure 3.

The symbol p_k denotes a process which performs a sequence of k α actions and then becomes $\mathsf{0}$; so the process p performs every finite sequence of αs. In contrast, the process q performs also an infinite sequence of αs.

To prove that $p \sqsubseteq^{\mathsf{cpl}}_{\text{SVR}} q$, we have to show that $r \dashv p$ then $r \dashv q$. It suffices to prove that the following relation is a co-inductive compliance,

$$\mathcal{R} = \{\, (r',\, q) \ \mid\ r \dashv p,\, r \overset{\overline{\alpha}^k}{\Longrightarrow} r',\ \text{for some } k \in \mathbb{N} \text{ and } r \in \mathsf{CCS}\,\}$$

We have to show that if $r' \ \mathcal{R}\ q$ then the pair $(r',\, q)$ satisfies the conditions given in Definition 2.1.

Pick a pair $(r',\, q)$ in the relation $\mathcal{R}$. By construction of $\mathcal{R}$ and of q, we know that $r \overset{\overline{\alpha}^k}{\Longrightarrow} r'$ for some $k \in \mathbb{N}$ and some r such that $r \dashv p$.

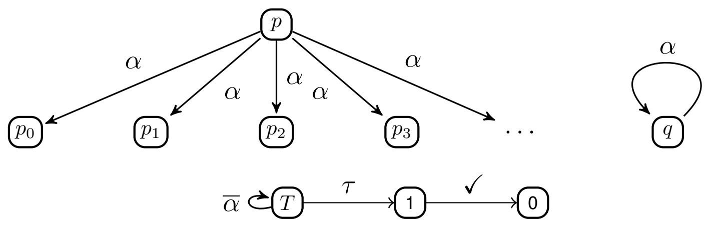

Figure 3: While $p \sqsubseteq^{\mathsf{cpl}}_{\mathrm{SVR}} q$, the test T witnesses that $p \not\sqsubset\!\!\sim^{\mathsf{tst}}_{\mathrm{SVR}} q$ (see Example 3.1)

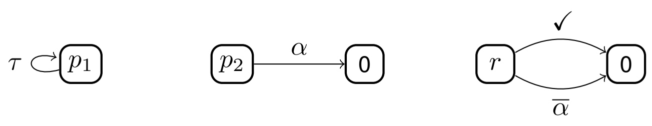

Figure 4: While $p_1 \sqsubset\!\!\sim^{\mathsf{tst}}_{\mathrm{SVR}} p_2$, the client r lets us prove that $p_1 \not\sqsubseteq^{\mathsf{cpl}}_{\mathrm{SVR}} p_2$ (see Example 3.2)

Condition (a) is trivially true, because q converges. We discuss condition (b) and (c). Suppose that $r' \parallel q \overset{\tau}{\not\longrightarrow}$; this implies that $r' \overset{\tau}{\not\longrightarrow}$. By construction $p \overset{\alpha^k}{\Longrightarrow} \mathsf{0}$, so we infer $r \parallel p \Longrightarrow r' \parallel \mathsf{0} \overset{\tau}{\not\longrightarrow}$. Now $r \dashv p$ and Lemma 2.3 imply that $r' \dashv \mathsf{0}$; Definition 2.1 ensures that $r' \overset{\checkmark}{\longrightarrow}$.

Suppose that $r' \parallel q \overset{\tau}{\longrightarrow} r'' \parallel q'$; we prove that $r'' \mathcal{R} q'$. The argument is a case analysis on the rule used to infer the reduction. In every case $q = q'$. If rule [P-LEFT] was applied then $r' \overset{\tau}{\longrightarrow} r''$; as $r \overset{\overline{\alpha}^k}{\Longrightarrow} r''$ the definition of $\mathcal{R}$ implies that $r'' \mathcal{R} q'$. Rule [P-RIGHT] cannot have been applied, for $q \overset{\tau}{\not\longrightarrow}$. If rule [P-SYNCH] was applied, then the reduction is due to an interaction. As q engages only in α, it follows $r \overset{\overline{\alpha}^{k+1}}{\Longrightarrow} r''$. The definition of $\mathcal{R}$ implies that $r'' \mathcal{R} q'$.

We have proven that the relation $\mathcal{R}$ is a co-inductive compliance, so $p \sqsubseteq^{\mathsf{cpl}}_{\mathrm{SVR}} q$.

Now we prove that $p \not\sqsubset\!\!\sim^{\mathsf{tst}}_{\mathrm{SVR}} q$; we define a test that is passed by p and not by q. Let $T \overset{\mathsf{def}}{=} \tau.\mathsf{1} + \overline{\alpha}.T$. The LTS of T is depicted in Figure (3). Every computation of $T \parallel p$ is finite and successful, so p MUST T. However when q is run as a server interacting with T, there is the possibility of an indefinite synchronisation on α, which is not a successful computation; q ~~M~~UST T. □

Example 3.2. [Convergence of servers]
In this example we prove that $p_1 \sqsubset\!\!\sim^{\mathsf{tst}}_{\mathrm{SVR}} p_2$ but $p_1 \not\sqsubseteq^{\mathsf{cpl}}_{\mathrm{SVR}} p_2$, where $p_1 = \tau^\infty$ and $p_2 = \alpha.\mathsf{0}$. The LTS of these processes is in Figure (4).

We prove that $p_1 \sqsubset\!\!\sim^{\mathsf{tst}}_{\mathrm{SVR}} p_2$. First note that $p_1 \Uparrow$, so if p_1 MUST r, then $r \overset{\checkmark}{\longrightarrow}$; this is because of the infinite computation due only to the divergence of p_1. It follows that if p_1 MUST r then p_2 MUST r.

Now we define a client that lets us prove $p_1 \not\sqsubseteq^{\mathsf{cpl}}_{\mathrm{SVR}} p_2$. Let $r = \mathsf{1} + \overline{\alpha}.\mathsf{0}$. To prove that $r \dashv p_1$

Definition 2.1 requires us to show a co-inductive compliance that contains the pair $(r,\, p_1)$. The following relation $\{(r,\, p_1)\}$ will do, because the only state ever reached by $r \mid\mid p_1$ is itself. We have to prove that $r \not\dashv p_2$. Consider the computation $r \mid\mid p_2 \Longrightarrow \mathsf{0} \mid\mid \mathsf{0} \overset{\tau}{\not\longrightarrow}$. Since $\mathsf{0} \overset{\checkmark}{\not\longrightarrow}$, Definition 2.1 ensures that $\mathsf{0} \not\dashv \mathsf{0}$. An application of Lemma 2.3 leads to $r \not\dashv p_2$. □

Let us now consider the client preorders in this setting of general contracts. The fact that $\mathrel{\underset{\sim}{\sqsubset}}^{\mathsf{tst}}_{\mathrm{CLT}} \not\subseteq \sqsubseteq^{\mathsf{cpl}}_{\mathrm{CLT}}$ will follow from Example 3.5. One final example is needed to show the converse.

Example 3.3. [Infinite traces and clients]
Here we prove that $\sqsubseteq^{\mathsf{cpl}}_{\mathrm{CLT}} \not\subseteq \mathrel{\underset{\sim}{\sqsubset}}^{\mathsf{tst}}_{\mathrm{CLT}}$. Let us define r as the the process p of Example 3.1, but with a $\checkmark$ transition after each finite sequence of αs. Recall also the process T of Example 3.1. To see why $r \sqsubseteq^{\mathsf{cpl}}_{\mathrm{CLT}} T$, it is enough to check that the relation

$$\begin{aligned}\mathcal{R} = \quad & \{\, (T,\, p') \mid r \dashv p,\, p \overset{\overline{\alpha}^k}{\Longrightarrow} p' \text{ for some } k \in \mathbb{N} \text{ and } p \in \mathsf{CCS} \,\} \\ & \cup \{\, (\mathsf{1},\, p) \mid p \in \mathsf{CCS} \,\}\end{aligned}$$

is a co-inductive compliance. To prove this, an argument similar to the one of Example 3.1 will do.

Now we show that $r \mathrel{\not\underset{\sim}{\sqsubset}}^{\mathsf{tst}}_{\mathrm{CLT}} T$; to see why, consider the server $S \overset{\mathsf{def}}{=} \overline{\alpha}.S$. All the maximal computations of $r \mid\mid S$ are client-successful, so S MUST r; while $T \mid\mid S$ performs an infinite computation with no client-successful states. □

The two essential differences in how servers are treated by the compliance relation and the testing relation are crystallised Example 3.1 and Example 3.2. In the former we see that a server may fail a test because of the presence of an infinite sequence of actions, although this does not impede the test, or client, from complying with the server. In the latter we see that divergent computations affect the preorders differently. The relation $\mathrel{\underset{\sim}{\sqsubset}}^{\mathsf{tst}}_{\mathrm{SVR}}$ is sensitive to the divergence of servers: any server that diverges is a least element of $\mathrel{\underset{\sim}{\sqsubset}}^{\mathsf{tst}}_{\mathrm{SVR}}$. So if $p_1 \mathrel{\underset{\sim}{\sqsubset}}^{\mathsf{tst}}_{\mathrm{SVR}} p_2$ and p_1 diverges, the traces that p_2 performs need not be matched by the traces of p_1. This is not the case if $p_1 \sqsubseteq^{\mathsf{cpl}}_{\mathrm{SVR}} p_2$; the traces of p_2 have to be matched suitably by the traces of p_1, regardless of the divergence of p_1.

3.2 Contracts for web-services

There are natural constraints on the contract language which avoid the phenomena described above. We say that a process p *converges strongly* if for every $s \in Act^\star$, $p \overset{s}{\Longrightarrow} p'$ implies $p' \Downarrow$. Then let $\mathsf{CCS}_{\mathsf{web}}$ denote the subset of processes in CCS which both strongly converge and are finite-state. Note that Konigs Lemma ensures that for every $p \in \mathsf{CCS}_{\mathsf{web}}$, p can perform an infinite sequence of actions u whenever it can perform all finite subsequences of u. Thus neither Example 3.1 nor Example 3.2 can be formulated in $\mathsf{CCS}_{\mathsf{web}}$. Nevertheless it is still a very expressive contract language. It encompasses (via an interpretation) first-order session types [Bd10, BH12], and, up to syntactic differences, the LTSs of contracts for web-services used in [LP07, CGP09, Pad10] are contained in the LTS $\langle\, \mathsf{CCS}_{\mathsf{web}},\, \longrightarrow,\, \mathsf{Act}_{\tau\checkmark} \,\rangle$.

Theorem 3.4. In $\mathsf{CCS}_{\mathsf{web}}$, $p_1 \mathrel{\underset{\sim}{\sqsubset}}^{\mathsf{tst}}_{\mathrm{SVR}} p_2$ if and only if $p_1 \sqsubseteq^{\mathsf{cpl}}_{\mathrm{SVR}} p_2$.

Proof. See Proposition 5.1.21 of [Ber13]. The proof relies on the behavioural characterisation of the two preorders, which is the same relation $\precsim_{\mathrm{SVR}}$. Roughly speaking, $p_1 \precsim_{\mathrm{SVR}} p_2$ if and only if for every trace $s \in Act^\star$, the potential deadlocks of p_2 after s are matched the potential

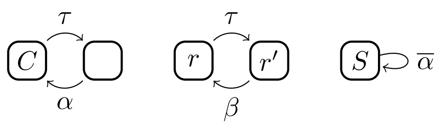

Figure 5: In any LTS that contains C and r, and where $\overline{\alpha} \neq \overline{\beta}$, S witnesses that $C \not\sqsubseteq^{\mathsf{cpl}}_{\text{CLT}} r_2$. However $C \sqsubseteq\!\!\!\sim^{\mathsf{tst}}_{\text{CLT}} r$ (see Example 3.5)

deadlocks[1] of p_1 after s. These properties characterise both $\sqsubseteq\!\!\!\sim^{\mathsf{tst}}_{\text{SVR}}$ and $\sqsubseteq^{\mathsf{cpl}}_{\text{SVR}}$, that is $\sqsubseteq\!\!\!\sim^{\mathsf{tst}}_{\text{SVR}} = \precsim_{\text{SVR}}$ and $\sqsubseteq^{\mathsf{cpl}}_{\text{SVR}} = \precsim_{\text{SVR}}$. The theorem follows from these equalities. $\square$

However even in $\mathsf{CCS}_{\mathsf{web}}$ the client sub-contract preorders remain different. In Example 3.5 and Example 3.6 below we prove that the client preorders are not comparable; Theorem 3.4 is false for the client preorders. and Also the converse (negative) inequality is true; we prove it in Example 3.6 below.

Example 3.5. [Client preorders and livelocks]
In this example we prove that in $\mathsf{CCS}_{\mathsf{web}}$, $\sqsubseteq\!\!\!\sim^{\mathsf{tst}}_{\text{CLT}} \not\subseteq \sqsubseteq^{\mathsf{cpl}}_{\text{CLT}}$. Suppose that for two actions α, β we have $\overline{\alpha} \neq \overline{\beta}$, recall the processes C, S of Example 2.6. Their LTS are depicted in Figure 5 along with the LTS of a process r.

We prove that $C \sqsubseteq\!\!\!\sim^{\mathsf{tst}}_{\text{CLT}} r$ and that $C \not\sqsubseteq^{\mathsf{cpl}}_{\text{CLT}} r$. The inequality $C \sqsubseteq\!\!\!\sim^{\mathsf{tst}}_{\text{CLT}} r$ is trivially true, because C does not perform $\checkmark$, so $p \not{\text{M}}\text{UST } C$ for every C.

To show that $C \not\sqsubseteq^{\mathsf{cpl}}_{\text{CLT}} r$ we have to exhibit a server with which C complies, while r does not. This server is S. In Example 2.2 we have already proven that $C \dashv S$. On the other hand, since $\overline{\alpha}$ cannot interact with β, we have $r' \parallel S \overset{\tau}{\not\longrightarrow}$. As $r' \overset{\checkmark}{\not\longrightarrow}$, Definition 2.1 and Lemma 2.3 ensure that $r \not\dashv S$. $\square$

3.3 Finite session behaviours

Underlying Example 3.5 is the treatment of *livelocks*. These are catastrophic for the testing based preorder, but can be accommodated by the compliance based one. However, there is another completely independent reason for which the two client preorders are different. Both are sensitive to the presence of the $\checkmark$ action, but in different ways.

In the examples below, Example 3.6 and Example 3.7, we prove that because of this difference, even for finite clients, with no recursion, the client preorders are incomparable. These examples show that any test which immediately performs $\checkmark$ is a top element in the testing based preorder, even if it subsequently evolves to a state in which $\checkmark$ is no longer possible. On the other hand for the compliance relation the action $\checkmark$ matters only in the stuck states of the client; its presence in all other states is immaterial.

Example 3.6. [$\mathsf{1}$ and internal moves]
Here we prove that $\sqsubseteq^{\mathsf{cpl}}_{\text{CLT}} \not\subseteq \sqsubseteq\!\!\!\sim^{\mathsf{tst}}_{\text{CLT}}$ even for finite clients (without recursion). Recall the client r of Example 2.7; its LTS is depicted in column (a) of Figure 6.

[1]More precisely, the acceptance sets.

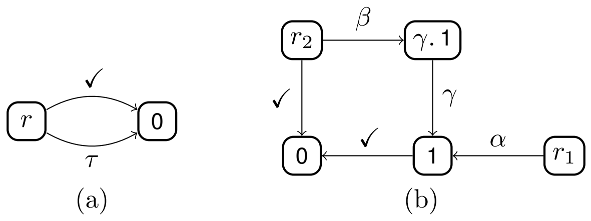

Figure 6: Clients that let us prove that the client preorders are not comparable even in the finite fragment of $\mathsf{CCS}_{\mathsf{web}}$ (see Example 3.6 and Example 3.7)

On the one hand, $r \sqsubseteq^{\mathsf{cpl}}_{\mathrm{CLT}} \mathsf{0}$. This is true because for every p, $r \mid\mid p \xrightarrow{\tau} \mathsf{0} \mid\mid p$; thus $r \dashv p$ and Lemma 2.3 imply that $\mathsf{0} \dashv p$. On the other hand $r \not\sqsubset\!\!\!\sim^{\mathsf{tst}}_{\mathrm{CLT}} \mathsf{0}$, because $\mathsf{0}$ MUST r (as $r \xrightarrow{\checkmark}$). However $\mathsf{0}$ $\not$MUST $\mathsf{0}$. □

Example 3.7. [$\mathsf{1}$ and interactions]
Here we show that $\sqsubset\!\!\!\sim^{\mathsf{tst}}_{\mathrm{CLT}} \not\subseteq \sqsubseteq^{\mathsf{cpl}}_{\mathrm{CLT}}$. Let $r_1 = \alpha.\,\mathsf{1}$ and $r_2 = \mathsf{1} + \beta.\gamma.\,\mathsf{1}$; their LTS is in column (b) of Figure 6.

Regardless of the server p we have p MUST r_2 because $r_2 \xrightarrow{\checkmark}$. It follows trivially that $r_1 \sqsubset\!\!\!\sim^{\mathsf{tst}}_{\mathrm{CLT}} r_2$. However, $r_1 \not\sqsubseteq^{\mathsf{cpl}}_{\mathrm{CLT}} r_2$; a typical server which distinguishes the two clients is $p = \overline{\alpha}.\,\mathsf{0} + \overline{\beta}.\,\mathsf{0}$. The proof that $r_1 \dashv p$ amounts to checking that the relation $\{\,(r_1,\,p),\,(\mathsf{1},\,\mathsf{0})\,\}$ is a co-inductive compliance. The fact that $r_2 \not\dashv p$ is due to Lemma 2.3 and the computation $r_2 \mid\mid p \xrightarrow{\tau} \gamma.\,\mathsf{1} \mid\mid \mathsf{0} \overset{\tau}{\not\longrightarrow}$. □

A further restriction of $\mathsf{CCS}_{\mathsf{web}}$ provides a language in which this difference in the treatment of ✓ does not materialise. Let SB^{f} be the language given by the following grammar,

$$p, q, r \quad ::= \quad \mathsf{1} \mid \textstyle\sum_{i \in I} \alpha_i.p_i \mid \sum_{i \in I} \tau.\overline{\alpha_i}.p_i$$

where $\alpha \in \mathit{Act}$, I is a finite non-empty set, and the actions α_is are pairwise distinct. This language gives rise to the LTS $\langle\ \mathsf{SB}^{\mathsf{f}},\ \longrightarrow,\ \mathsf{Act}_{\tau\checkmark}\ \rangle$ in the usual manner. The language SB^{f} is essentially the finite part of the session behaviours of [Bd10], which we can think of as the *first-order* part of the session types used in [GH05].

Here the language is finite so as to avoid duplicating Example 3.5. But the finiteness of the language implies also that the computations are finite. This is sufficient to show that a client and a server in compliance are related by the MUST testing as well.

Lemma 3.8. If SB^{f} if $r \dashv p$ then p MUST r.

Proof. Suppose that $r \dashv p$. We have to show that all the maximal computations of $r \mid\mid p$ are succesful. Fix such a computation. Since r and p are finite, the computation must have a terminal state, say $r' \mid\mid p' \overset{\tau}{\not\longrightarrow}$. The hypothesis $r \dashv p$ and the reductions $r \mid\mid p \Longrightarrow r' \mid\mid p'$ imply that $r' \dashv p'$. As this state is stable, the definition of compliance ensures that $r' \xrightarrow{\checkmark}$; thus the maximal computation we picked is succesful. □

Thanks to the restrictive syntax of SB^{f}, the converse of Lemma 3.8 is also true. This is due to two reasons. One is a general property of MUST borne out by Lemma 3.10. The other reason is the next lemma, which explains why the different treatment of ✓ does not materialse in SB^{f}.

Lemma 3.9. In SB^{f}, if $p \xrightarrow{\checkmark}$ then $p = \mathsf{1}$.

Proof. In principle $p \xrightarrow{\checkmark}$ may be proven by using one of the rules [A-OK], [R-EXT-L], and [R-EXT-R]. But the last two rules can be used only on external sums, and in SB^{f} these sums do not enagage in $\checkmark$. It follows that $p \xrightarrow{\checkmark}$ must be due to rule [A-OK], hence $p = \mathsf{1}$. □

The previous lemma is not true for CCS. For instance the client r of Example (2.7) performs the action $\checkmark$, but $r \neq \mathsf{1}$.

Lemma 3.10. In CCS if p MUST r, $r \mid\mid p \xrightarrow{\tau} r' \mid\mid p'$ and $r' \not\xrightarrow{\checkmark}$ then p' MUST p'.

Proof. (Outline) To prove the result it suffices to add to every maximal computation of $r' \mid\mid p'$ the suffix $r \mid\mid p \xrightarrow{\tau} r' \mid\mid p'$, and then use the hypothesis p MUST r and $r \not\xrightarrow{\checkmark}$. □

Lemma 3.11. In SB^{f}, if p MUST r then $r \dashv p$.

Proof. We show that the next relation is a co-inductive compliance,

$$\mathcal{R} = \{\, (r, p) \;\mid\; p \text{ MUST } r \,\}$$

Pick a pair $r \mathrel{\mathcal{R}} p$. Suppose $r \mid\mid p \not\xrightarrow{\tau}$. Then the definition of MUST ensures that $r \xrightarrow{\checkmark}$. Suppose that $r \mid\mid p \xrightarrow{\tau} r' \mid\mid p'$. If $r \xrightarrow{\checkmark}$ then Lemma 3.9 lets us prove that $r' \xrightarrow{\checkmark}$. In turn this ensures that p' MUST r'. If $r \not\xrightarrow{\checkmark}$, then Lemma 3.10 implies that p' MUST r'. □

Theorem 3.12. In SB^{f},

(1) $p_1 \mathrel{\sqsubseteq^{\mathsf{tst}}_{\text{SVR}}} p_2$ if and only if $p_1 \mathrel{\sqsubseteq^{\mathsf{cpl}}_{\text{SVR}}} p_2$

(2) $r_1 \mathrel{\sqsubseteq^{\mathsf{tst}}_{\text{CLT}}} r_2$ if and only if $r_1 \mathrel{\sqsubseteq^{\mathsf{cpl}}_{\text{CLT}}} r_2$

Proof. This is a direct consequence of Lemma 3.11 and Lemma 3.8. □

4 Conclusion

In this paper we have shown the differences between the sub-contract preorders [CGP09, Pad10, Ber13] and the testing preorders [DH84, BH13]. Another study of sub-contract relations is [BMPR10]. There different compliances are used; two similar to $\dashv$, and a fair one.

The sub-contract relation was first proposed in [CCLP06], and further developed in [LP07, CGP09, Pad10]. For instance, the latter papers show how to adapt the behaviour of contracts by applying filters, or orchestrators; thereby defining weak sub-contracts, whose elements can be forced (by filtering) into the sub-contract. [CGP09] also shows an encoding of WS-BPEL activities into the language of contracts. A result similar to Theorem 3.4 was already established in [LP07], and it has been referenced by [Pad09],[Pad10, Proposition 2.7], and [CGP09, pag. 13]. The sub-contract for clients was proposed first in [Bd10], and it is instrumental in modelling the subtyping for first-order session types [GH05]. The preorder that models the subtyping coincides with a combination of the client sub-contract and a server one. This model was proven sound in [Bd10], fully-abstract in [BH12], and extended to higher-order session types in [Ber13].

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, June 2013. Available at https://www.scss.tcd.ie/∼bernargi.

[BH12] Giovanni Bernardi and Matthew Hennessy. Modelling session types using contracts. In Sascha Ossowski and Paola Lecca, editors, *SAC*, pages 1941–1946. ACM, 2012.

[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.

[BMPR10] Michele Bugliesi, Damiano Macedonio, Luca Pino, and Sabina Rossi. Compliance preorders for web services. In Cosimo Laneve and Jianwen Su, editors, *Web Services and Formal Methods*, volume 6194 of *Lecture Notes in Computer Science*, pages 76–91. Springer Berlin / Heidelberg, 2010.

[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.

[CH93] Rance Cleaveland and Matthew Hennessy. Testing equivalence as a bisimulation equivalence. *Formal Asp. Comput.*, 5(1):1–20, 1993.

[CH10] Andrea Cerone and Matthew Hennessy. Process behaviour: Formulae vs. tests (extended abstract). In Sibylle B. Fröschle and Frank D. Valencia, editors, *EXPRESS'10*, volume 41 of *EPTCS*, pages 31–45, 2010.

[DH84] Rocco De Nicola and Matthew Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.

[GH05] Simon J. Gay and Malcolm Hole. Subtyping for session types in the pi calculus. *Acta Inf.*, 42(2-3):191–225, 2005.

[Hen85] Matthew Hennessy. *Algebraic theory of processes.* MIT Press, Cambridge, MA, USA, 1985.

[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] Robin Milner. *Communication and concurrency.* PHI Series in computer science. Prentice Hall, 1989.

[Pad09] Luca Padovani. Contract-based discovery and adaptation of web services. In Marco Bernardo, Luca Padovani, and Gianluigi Zavattaro, editors, *SFM*, volume 5569 of *Lecture Notes in Computer Science*, pages 213–260. Springer, 2009.

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

[THK94] Kaku Takeuchi, Kohei Honda, and Makoto Kubo. An interaction-based language and its typing system. In Constantine Halatsis, Dimitris G. Maritsas, George Philokyprou, and Sergios Theodoridis, editors, *PARLE*, volume 817 of *Lecture Notes in Computer Science*, pages 398–413. Springer, 1994.