

# A preliminary proposal of decidable testing relations for infinitary asynchronous CCS

(Position paper: not for the proceedings)

Giuseppe Castagna  
CNRS, Université Paris Diderot

Luca Padovani  
Università degli Studi di Urbino

We present a simple formalization of an asynchronous variant of CCS that preserves the order of outputs and devise a testing-based containment relation for its processes. We prove that the relation is decidable also in the presence on infinite processes which, in our knowledge, is the first such result for testing in asynchronous CCS. We discuss related work and the application of our framework to the theory of web-services composition.

## 1 Introduction

We define an asynchronous version of infinitary CCS that preserves the order of the output messages, and endow it with a testing-based containment relation that we prove to be decidable. Our asynchrony is effective in two different senses, inasmuch as it preserves the order of the messages (thus it effectively implements the asynchrony used on the Web) and has a decidable testing-based containment theory (static verification of service upgrade and replacement is effective).

The CCS variant we present implements asynchrony by buffering outputs in the processes that produced them. The idea of using buffers is not new (see the related works below) but we improve over existing works by proposing a tighter integration of buffers in the process syntax, yielding a simpler and, in our opinion, elegant solution. As a matter of facts in existing literature buffers appear more like an afterthought, as if they were grafted over some existing process language rather than being an integrating part of a specifically designed calculus. The idea is very simple: we use the very same syntax as synchronous CCS and, as in the synchronous case, a top-level output action is consumed only when it synchronizes with a corresponding input action. The difference is that a top-level output action does not block input actions underneath, which can thus synchronize. For instance,  $\bar{a}.b.\bar{c}.d.\mathbf{0}$  is a sequential process that performs an output on  $a$ , followed by an input on  $b$ , followed by an output on  $c$ , and followed by an input on  $d$  (the colon following output actions is just a visual clue to stress that we are dealing with asynchronous outputs, it has no other special function). This process can synchronize with an input on  $a$  and reduce to  $b.\bar{c}.d.\mathbf{0}$ . But since the output on  $a$  does not mask the next  $b$  input, then this process can also synchronize with an output on  $b$  and reduce to  $\bar{a}.\bar{c}.d.\mathbf{0}$ . This latter process offers an output on  $a$  and an input on  $d$ . The output on  $c$  is blocked by the output on  $a$  and will become visible only when this is consumed. The intuition is that  $\bar{a}.b.\bar{c}.d.\mathbf{0}$  is a process with an  $a$  in its output queue and ready to read a  $b$ , while  $\bar{a}.\bar{c}.d.\mathbf{0}$  is a process with  $a, c$  in its output queue (with  $a$  first element) and ready to read a  $d$ .

We will mostly focus on two behavioral operators  $+$  and  $\oplus$  representing external and internal choices respectively. This is consistent with many works on web-services and is also the same approach taken in some presentations of CCS [13, 15] and of the  $\pi$ -calculus [16]. For example,  $a.P + b.Q$  is the process that continues as  $P$  if it receives an input on  $a$  and as  $Q$  if it receives an input on  $b$ . Dually,  $\bar{a}.R + \bar{b}.S$  is a process that internally decides whether to output on  $a$  or  $b$  and then continues as  $R$  or  $S$ , accordingly. We

distinguish two terminal behaviors:  $\mathbf{0}$  represents the deadlocked process while  $\mathbf{1}$  represents successful termination. Finally, we allow processes to be recursively defined (precisely, our syntax includes infinite terms solutions of recursive equations). This, combined with our treatment of asynchrony, has as consequence that it is no longer possible to recognize the sequences of visible actions emitted by a process by a finite state automaton. For instance the process  $P = \bar{a}.b.P$  produces infinitely many different processes, whose buffer part contains as many outputs on  $a$  as the number of  $b$  inputs emitted by the process. In other words it reduces to all (and only) the terms of the form  $\bar{a}:\bar{a}:\dots\bar{a}.b.P$  with an arbitrary number of  $\bar{a}$ 's.

We can identify four main contributions of this work. First, the work provides the first, in our ken, decidable testing equivalence for a infinitary asynchronous CCS, thus providing a partial answer to one of the major open problems left by Boreale, De Nicola and Pugliese [20]. Second, we give a definition of the queue asynchronous CCS that preserves the syntax of synchronous CCS and whose simplicity contrasts with the solutions existing in the literature where buffers materialize by adding states in the operational semantics, by paring them with processes in some syntactic meta-levels, or by giving to them the status of full processes. Third, our results are obtained for the kind of asynchrony typical of the web, since it preserves the order of outputs and, as we will discuss later on, synchronization can be implemented without resorting to complex agreement protocols between the partners. Our theory can thus be realistically and immediately applied to web services. Last but not least, we believe that a very important contribution of this work is to promote a syntax-independent treatment of infinitary process. To account for infinite behavior we work on possibly infinite regular trees and this allows us to prove very general results by very compact proofs. In doing that we do not do anything very original since we just transpose to concurrency theory the ideas of Bruno Courcelle [11] as we already did in two previous works [7, 8]. However not only this sheds new light and, we believe, greatly clarifies the treatment of recursion in process algebras (that always appeared somewhat obscure to our eyes), but also we consider it the key ingredient that allowed us to find a proof (and a simple one) of the decidability of the containment relation.

**Overview** Section 2 defines the syntax and semantics of our asynchronous CCS, as well as the test-based refinement relation on its processes. Section 3 contains a coinductive characterization of the same relation. Since this characterization does not provide an effective way to test refinement, then in Section 4 we give an equivalent inductive characterization and prove it to be equivalent to the coinductive one and decidable. We conclude our presentation in Section 5. All proofs are relegated to the appendixes.

**Related work** Asynchrony was first introduced for  $\pi$  independently by Gérard Boudol [4] and Honda and Tokoro [18, 17]. Asynchronous variants for CCS were introduced later, in the context of Linda-related studies [23]. Testing equivalences for asynchronous variant of CCS and  $\pi$  were subsequently studied by Castellani and Hennessy [9] and Boreale, De Nicola, and Pugliese [20]. In all these works asynchronous communications do not preserve the order of outputs and although some early works proposed buffered asynchronous communication [2, 12] this has become the standard approach to asynchrony in the community of process calculi (see [1] for a comparison). Furthermore the works on testing equivalences do not consider decidability problems for infinite processes: Castellani and Hennessy give an equational characterization only for the finite fragment, while Boreale, De Nicola, and Pugliese leave decidability as a major future work.

The mismatch between the default theoretic formalization of asynchrony and the order preserving buffered asynchrony typical of the Web was felt in the concurrency community and inspired both theoretic-oriented studies and practical-oriented ones. Among the former, a prominent place is taken

by the work by Beauxis, Palamidessi, and Valencia [1] who compare the expressive power of the two kinds of asynchrony; however they stop to expressiveness and do not tackle the study of the respective equivalence relations. Among the latter we want to signal those issued from the “web-services” theoretical community where asynchrony was studied both for the approach based on contracts [5] and for the session type approach [22, 10, 14, 19, 21]. Among the latter, several works prove the decidability of the subtyping relation for session types, a result that is less surprising than ours since, from the viewpoint of concurrency theory, session types are far more restrictive than contracts. If we add to these works the already cited early works of Jan Willem Klop and his coauthors [2, 12], we see that we are not the first to use buffered communication to implement asynchrony. However none of these works possess the simplicity of our approach, since in them buffers are grafted over the language of processes or its metatheory instead of being indissolubly embedded in the syntax as in our case. So in previous works buffers were implanted either via a stateful operational semantics [12, 22], or as some new autonomous processes [1, 19, 21, 14], or they were paired to processes by introducing a further syntactic layer [5].

## 2 Asynchronous Communicating Processes

Let  $\mathcal{N}$  be a set of *names* ranged over by  $a, b, c, \dots$  and let  $\overline{\mathcal{N}}$  be the corresponding set of *co-names*. We let  $\alpha, \beta, \dots$  range over *actions*, namely elements of  $\mathcal{N} \cup \overline{\mathcal{N}}$ ; we let  $\varphi, \varphi', \dots$  range over strings of actions; we let  $s, t, \dots$  range over strings of names; we write  $\overline{\varphi}$  for the string obtained by turning every action in  $\varphi$  into the corresponding co-action; we write  $\overline{s}$  for the string obtained by turning every name in  $s$  into the corresponding co-name; we use  $\varepsilon$  to denote the empty string.

The set of processes of our calculus are defined as follows:

**Definition 2.1** (processes). *Processes are all the possibly infinite regular terms coinductively generated by the productions*

$$P ::= \mathbf{0} \mid \mathbf{1} \mid a.P \mid \overline{a}:P \mid P+P \mid P\oplus P$$

*such that on every infinite branch of their syntax tree there occur infinitely many input actions.*

The syntax of each single constructions was already informally described in the Introduction. The reason why we consider possibly infinite regular trees that satisfy some weird condition is that we want to account for infinite behaviors in a syntax-independent way. In process algebra literature infinite behaviors are obtained by adding syntax for defining recursive computation: usually this appears in the form of recursively defined process  $\text{rec } X.P$ , or of replication  $P^*$ , or of a separate set of mutually recursive declarations. These are nothing but finite representations of the infinite (syntax) trees obtained by unfolding or expanding them. We believe that it is far simpler and enlightening to get rid of syntactic constraints and work directly on infinite trees by relying on the theory developed by Bruno Courcelle [11]. In our theory we do not consider every possible infinite tree but just those that satisfy two conditions. First, we ask trees to be regular, that is to have a finite number of distinct subtrees. This powers down the expressiveness of the language to include only and all processes that can be generated by the  $\text{rec}$ -expressions or mutually recursive declarations customary in the process algebra literature. Nevertheless all results stated in this work hold also for non regular trees, except the decidability ones of course. Second, we require that on every infinite branch of a tree there must be infinitely many input actions. This simple restriction allows us to get rid of two kinds of pathological processes, namely, infinite (possibly mixed) sums (e.g.,  $X = X + X$ ) and infinite “spammers”, that is processes that can produce an infinite sequence of consecutive outputs (e.g.,  $X = \overline{a}:X$ ): every process will eventually stop to listen for inputs or declare

$$\begin{array}{c}
\mathbf{1} \xrightarrow{\checkmark} \mathbf{1} \quad a.P \xrightarrow{a} P \quad \bar{a}.P \xrightarrow{\bar{a}} P \quad P \oplus Q \longrightarrow P \quad \frac{P \longrightarrow P'}{\bar{a}.P \longrightarrow \bar{a}.P'} \quad \frac{P \xrightarrow{b} P'}{\bar{a}.P \xrightarrow{b} \bar{a}.P'} \\
\\
\frac{P \longrightarrow P'}{P + Q \longrightarrow P' + Q} \quad \frac{P \xrightarrow{\checkmark} P'}{P + Q \xrightarrow{\checkmark} P'} \quad \frac{P \xrightarrow{a} P'}{P + Q \xrightarrow{a} P'} \quad \frac{P \xrightarrow{\bar{a}} P'}{P + Q \longrightarrow \bar{a}.P'}
\end{array}$$

Table 1: Transition relation of processes.

success. This second condition is an example of how working directly on infinite trees is somehow simpler (though not very difficult, we invite the reader to think about how to impose the same restriction with sets of mutually recursive declarations).

The semantics of processes is defined by the transition system of Table 1 (plus the symmetric rules for the sums). Labels of the transition system are ranged over by  $\mu$  and are elements of  $\mathcal{N} \cup \overline{\mathcal{N}} \cup \{\checkmark\}$ . We write  $\bar{\mu}$  for the co-label of  $\mu$ , where  $\overline{\checkmark} = \checkmark$ . Technically the transition system is composed of two relations: a labeled one  $\xrightarrow{\mu}$  describing the external visible actions, and an unlabeled one  $\longrightarrow$  describing internal invisible actions. The process  $\mathbf{1}$  emits  $\checkmark$  to signal successful termination and reduces to itself, while no reduction stems from  $\mathbf{0}$ . Top-level actions are immediately emitted and internal choices resolved by an internal transition. Asynchrony is implemented by the last two rules in the first row of Table 1: they state that a top-level output blocks neither the internal transitions nor the input actions produced by its continuation. Notice that any output emitted by the continuation is blocked instead since, intuitively, it is buffered in the queue after the top-level output. Transitions for external choices are standard except for outputs, as illustrated by the last rule of Table 1. This rule (already introduced in [9]) states that whenever a branch of an external choice has decided to fire an output, then it does so independently from the presence of a processes willing to accept it.

Observe that the condition on infinitely many input actions in Definition 2.1 ensures that there is no infinite sequence of internal moves: every process eventually stops to wait for an input or to signal success. Of course this holds true only for internal transitions as infinite trees may produce infinite labeled reductions. For instance consider again the process defined in the introduction, that is the infinite tree solution of  $P = \bar{a}.b.P$ . First notice that this *is* a process since it is regular (it has two distinct subtrees:  $b.P$  and  $P$ ) and satisfies the condition of Definition 2.1 (this single-branched tree has infinitely many inputs on  $b$ ). An example of transition issued from this term is

$$\bar{a}.b.P \xrightarrow{\bar{a}} b.P \xrightarrow{b} P \xrightarrow{b} \bar{a}.P \xrightarrow{b} \bar{a}.\bar{a}.P \xrightarrow{b} \bar{a}.\bar{a}.\bar{a}.P \xrightarrow{b} \dots$$

Notice that, apart from the first two terms, all generated processes are not subtrees of the original process, whence the fact that the processes of our calculus does not correspond to finite state automata.

As customary we write  $\Longrightarrow$  for the reflexive and transitive closure of  $\longrightarrow$ ; we write  $\xRightarrow{\mu}$  for  $\Longrightarrow \xrightarrow{\mu} \Longrightarrow$ ; we write  $P \xrightarrow{\mu}$  if there exists  $P'$  such that  $P \xrightarrow{\mu} P'$ , and similarly for  $\xRightarrow{\mu}$ ; we write  $P \not\rightarrow$  if there exists no  $P'$  such that  $P \longrightarrow P'$ , and similarly for  $\not\xrightarrow{\mu}$ .

Process synchronization takes place at the higher level of *systems*

**Definition 2.2** (system). *Let  $P$  and  $Q$  be processes. A system is a pair  $P\|Q$  indicating that  $P$  and  $Q$  have established a connection and can interact with each other.*

and is defined as follows:

**Definition 2.3** (system transitions). *The transition relation for systems is defined by the rules*

$$\frac{P \longrightarrow P'}{P \parallel Q \longrightarrow P' \parallel Q} \quad \frac{P \xrightarrow{\alpha} P'}{P \parallel Q \xrightarrow{\alpha} P' \parallel Q} \quad \frac{P \xrightarrow{\bar{\alpha}} P' \quad Q \xrightarrow{\alpha} Q'}{P \parallel Q \longrightarrow P' \parallel Q'}$$

plus the symmetric of the first two rules.

Next we define our test-based containment relation for processes. Applying a test is materialized by the notion of *compliance* (this terminology is inherited from the web-services domain, where a client is compliant with a service if every interaction stops only with client's satisfaction).

**Definition 2.4** (compliance).  *$P$  is compliant with  $Q$ , written  $P \triangleleft Q$ , if  $P \parallel Q \Longrightarrow P' \parallel Q' \dashv\vdash$  implies  $P' \xrightarrow{\checkmark}$ .*

As in the testing approach, we use compliance for defining the refinement for processes:

**Definition 2.5** (refinement). *We say that  $Q$  is a refinement of  $P$ , notation  $P \preceq Q$ , if  $R \triangleleft P$  implies  $R \triangleleft Q$  for every  $R$ . We write  $\simeq$  for the equivalence relation induced by  $\preceq$ , namely  $\simeq = \preceq \cap \succeq$ .*

The intuition is that a process can be safely used in the place of a less refined one without disrupting interactions with the clients of the latter.

### 3 Coinductive Characterizations of Compliance and Refinement

Definition 2.5 emphasizes the notion of safe replacement associated to refinement, but gives little insight on the properties that this relations enjoys, in particular decidability. To obviate it in this section we give equivalent coinductive characterizations of compliance and of refinement.

Before we move on to the alternative characterization we need two auxiliary notions, that of process continuation and that of observables. For what concerns continuation, observe that the relation for contracts we have given in Section 2 is subjective in the sense that it describes the evolution of a process from the subjective point of the process itself. For example, we have  $a.P + a.Q \xrightarrow{a} P$ , namely the process has received an output on  $a$  and chosen the left branch of the external choice thus reducing to  $P$ . A process that sends an output on  $a$  to this process, however, has no clue as to which branch the receiving process has taken. Hence, the objective evolution of the above contract after the action  $a$  is  $P \oplus Q$ . The objective view of a computation is embodied by the notion of *continuation*:

**Definition 3.1** (continuation). *Let  $P \xrightarrow{\alpha}$ . The continuation of  $P$  with respect to  $\alpha$  is defined as  $P(\alpha) \stackrel{\text{def}}{=} \bigoplus_{P \xrightarrow{\alpha, P'}} P'$ . We generalize this to finite sequences of actions, so that  $P(\varepsilon) = P$  and  $P(\alpha\varphi) = P(\alpha)(\varphi)$ .*

Next we define the *observables*, in the sense of [3], of our calculus.

**Definition 3.2.** *Let  $P$  be a process and  $\text{init}(P) \stackrel{\text{def}}{=} \{\mu \mid P \xrightarrow{\mu}\}$ .*

1. *We say that  $P$  may  $\bar{a}$ , notation  $P \downarrow \bar{a}$ , if  $P \xrightarrow{\bar{a}}$ .*
2. *We say that  $P$  guarantees  $a$ , notation  $P \Downarrow a$ , if  $P \Longrightarrow P'$  implies  $P' \xrightarrow{a}$ .*
3. *We say that  $P$  has ready set  $R$ , notation  $P \downarrow R$ , if there exists  $P'$  such that  $P \Longrightarrow P'$  and  $R = \text{init}(P') \subseteq \mathcal{N}$ .*
4. *We say that  $P$  is testable, notation  $P \Downarrow$ , if  $P \Longrightarrow P'$  implies that there exists  $a$  such that  $P' \downarrow \bar{a}$ .*

The first two observables signal, respectively, that a process may emit an output and that it guarantees it will accept a given output. A ready set constitutes a set of input actions that the process may offer. Finally, in an asynchronous calculus it is possible to test only the presence of an output: inputs and success cannot be tested. So we say that a process is *testable* if it emits an output, whatever it is. We now possess all the technical tools to give a coinductive definition of compliance:

**Definition 3.3** (coinductive compliance).  $\mathcal{C}$  is a coinductive compliance relation if  $(P, Q) \in \mathcal{C}$  implies:

1.  $P \downarrow \bar{a}$  implies  $Q \Downarrow a$ ;
2.  $P \downarrow R$  implies  $Q \downarrow$  and  $Q \downarrow \bar{a}$  implies  $a \in R$ ;
3.  $P \xrightarrow{\bar{\alpha}}$  and  $Q \xrightarrow{\alpha}$  implies  $(P(\bar{\alpha}), Q(\alpha)) \in \mathcal{C}$ .

The definition states that a client  $P$  is compliant with a process  $Q$  if and only if: (1) if  $P$  may emit an output, then  $Q$  guarantees that it will eventually accept it, (2) if  $P$  may offer  $Q$  to choose among a set of inputs  $R$ , then  $Q$  will eventually make a choice and emit an output that is present in all sets of inputs that  $P$  may offer, and (3) if  $P$  and  $Q$  synchronize, then their continuation are compliant.

**Theorem 3.4.**  $\triangleleft$  is the largest coinductive compliance.

Finally, the coinductive definition of refinement is:

**Definition 3.5** (coinductive refinement).  $\mathcal{S}$  is a coinductive refinement relation if  $(P, Q) \in \mathcal{S}$  implies:

1.  $P \downarrow$  implies  $Q \downarrow$ , and
2.  $Q \downarrow \bar{a}$  implies  $P \downarrow \bar{a}$  and  $(P(\bar{a}), Q(\bar{a})) \in \mathcal{S}$ , and
3.  $P \Downarrow a$  implies  $Q \Downarrow a$  and  $(P(a), Q(a)) \in \mathcal{S}$ .

The intuition is that we want that all clients of (i.e., compliant with)  $P$  must also be compliant with  $Q$ , thus: (1) if  $P$  is testable, then there might be clients of  $P$  that will stop and listen for an input, so  $Q$  must also be testable; (2) all outputs that  $Q$  may emit must be handled by the clients of  $P$  and therefore  $P$  must emit the same output and, finally, (3) if  $P$  guarantees an input its clients may send the corresponding output and therefore  $Q$  must guarantee the same input as well.

**Theorem 3.6.**  $\preceq$  is the largest coinductive refinement.

None of the two coinductive characterizations provides an effective way to test the corresponding relation since, in general, it is not possible to find a *finite* coinductive compliance or refinement that contains two related process. This hinders the direct application of standard memoization techniques. For instance consider again our paradigmatic example  $P = \bar{a}.b.P$  together with the process  $Q = \bar{a}:(b.Q + b.\bar{c}.1)$ . It is easy to verify that  $Q \preceq P$ : if we define  $P_0 = P$ ,  $P_{i+1} = \bar{a}.P_i$ , and similarly for  $Q_i$ , then

$$\{(Q, P)\} \cup \{(b.Q + b.\bar{c}.1, b.P)\} \cup \{(Q \oplus \bar{c}.1, P)\} \cup \{(Q_i \oplus \bar{a}.\bar{c}.1, P_i)\}_{i>0}$$

is a coinductive refinement containing  $(Q, P)$ . Note however that the rightmost set in the union above contains infinitely many distinct processes (none of which is a subtree of the original processes); and since this is the smallest coinductive refinement containing  $(Q, P)$ , then it is not possible to directly apply standard memoization techniques to prove  $Q \preceq P$ . In order to decide the compliance and refinement relations we need some extra effort.

## 4 Decidable Characterizations of Compliance and Refinement

One of the central results of this work is that both compliance and refinement are decidable relations. This is far from being obvious—as witnessed by the time this problem has been open for—since even if we have a coinductive characterization of both relations, they both demand to check the continuations of two processes. We saw that, because of asynchrony, there may be infinitely many distinct continuations for a process and, in particular, that buffers can grow indefinitely (the reader must not be misled by our condition on infinite inputs: this condition ensures only that all computations will eventually stop, but does not limit the length of buffers which can grow indefinitely). The key observation that makes it possible to prove the result is that for deciding compliance and refinement one needs to look only at the head of the buffers and at the residuals after the outputs (i.e., the processes immediately following the end of a buffer): since the possible combinations of these are finitely many, we can deduce the decidability property. In order to show it we need to give further characterizations of the compliance and refinement that do not use continuations.

**Theorem 4.1** (decidable compliance).  $P \triangleleft Q$  if and only if for every  $\varphi$  such that  $P \xrightarrow{\bar{\varphi}}$  and  $Q \xrightarrow{\varphi}$  we have: (1)  $P(\bar{\varphi}) \downarrow \bar{a}$  implies  $Q(\varphi) \downarrow a$ ; (2)  $P(\bar{\varphi}) \downarrow R$  implies  $Q(\varphi) \downarrow$  and  $Q(\varphi) \downarrow \bar{a}$  implies  $a \in R$ .

The adjective “decidable” in Theorem 4.1 might seem inappropriate, as the Theorem still quantifies over an infinite number of strings  $\varphi$ . However, all that matters is the observables of the continuations of  $P$  and  $Q$  after  $\varphi$ , which are finitely many, as we will show at the end of the section.

A similar result can be stated for checking refinement. In this case one has to check all the paths of the smaller process that are formed by the outputs that the process may emit and the inputs that the process guarantees. Formally, they are defined as follows:

**Definition 4.2.** The relation  $P \Downarrow \varphi$  is the least relation defined by the following rules:

$$P \Downarrow \varepsilon \quad \frac{P \Downarrow a \quad P(a) \Downarrow \varphi}{P \Downarrow a\varphi} \quad \frac{P \downarrow \bar{a} \quad P(\bar{a}) \Downarrow \varphi}{P \Downarrow \bar{a}\varphi}$$

Refinement can then be verified by checking that all these paths yield to states where (1) if the smaller process emits some output, so must the larger one do since some clients of the smaller may be waiting for that output, (2) if the larger process emits a given output then this must be expected by the clients of the smaller process, and this is guaranteed only if the smaller process emits the same output, and (3) if the smaller process guarantees that it will accept a given input, then so must the larger process do in order to handle a possible output of the smaller process’s clients.

**Theorem 4.3** (decidable refinement).  $P \preceq Q$  if and only if for every  $\varphi$  such that  $P \Downarrow \varphi$  and  $Q \xrightarrow{\varphi}$  we have: (1)  $P(\varphi) \downarrow$  implies  $Q(\varphi) \downarrow$ ; (2)  $Q(\varphi) \downarrow \bar{a}$  implies  $P(\varphi) \downarrow \bar{a}$ ; (3)  $P(\varphi) \downarrow a$  implies  $Q(\varphi) \downarrow a$ .

The last step to prove the decidability of the two relations is to show that all observables of  $P(\varphi)$  can be checked on a set  $D(P)$  that is finite for every  $P$ . To this aim we define  $\mathcal{T}(P)$  as the set of syntactic subtrees of  $P$ . Because of regularity,  $\mathcal{T}(P)$  is finite. Then, we define  $\mathcal{T}^*(P) \stackrel{\text{def}}{=} \{P'' \mid \exists P' \in \mathcal{T}(P), P' \Longrightarrow P''\}$ . Namely,  $\mathcal{T}^*(P)$  is the set of residuals of syntactic subtrees of  $P$  that are reachable by means of internal transitions only. This set is also finite, because of the restriction that on every infinite branch of  $P$  there are infinitely many occurrences of the input prefix (technically,  $P$  is everywhere convergent, none of its residuals ever engage in an infinite sequence of internal transitions).

**Definition 4.4** (observable residuals). The set of observable residuals of  $P$  after  $\varphi$  is defined as  $D(P, \varphi) \stackrel{\text{def}}{=} \{(\bar{a}, Q) \mid \exists s : P \xrightarrow{\varphi} \bar{a}s : Q, Q \in \mathcal{T}^*(P)\} \cup \{(\varepsilon, Q) \mid P \xrightarrow{\varphi} Q, Q \in \mathcal{T}^*(P)\}$ . The set of observable residuals of  $P$  is defined as let  $D(P) \stackrel{\text{def}}{=} \bigcup_{P \xrightarrow{\varphi}} D(P, \varphi)$ .

A direct consequence of the conditions on regularity and infinite inputs for processes we have:

**Proposition 4.5.**  $D(P)$  is finite for every  $P$ .

Note that for every process  $P$  it is possible to construct  $D(P)$  inductively (by induction on the length of  $\varphi$ ): the proposition above ensures that the construction will eventually stop on a fixpoint.

Finally it remains to prove that for all  $P$  and  $\varphi$  such that  $P \xRightarrow{\varphi}$ , all observables of  $P(\varphi)$  can be checked on  $D(P)$ , that is on a finite set.

**Proposition 4.6.** Let  $P \xRightarrow{\varphi}$ . The following properties hold:

1.  $P(\varphi) \downarrow$  if and only if  $(\bar{s}, Q) \in D(P, \varphi)$  implies  $\bar{s}:Q \downarrow$ ;
2.  $P(\varphi) \downarrow \bar{a}$  if and only if  $\bar{s}:Q \downarrow \bar{a}$  for some  $(\bar{s}, Q) \in D(P, \varphi)$ ;
3.  $P(\varphi) \downarrow R$  if and only if  $\bar{s}:Q \downarrow R$  for some  $(\bar{s}, Q) \in D(P, \varphi)$ ;
4.  $P(\varphi) \downarrow a$  if and only if  $Q \downarrow a$  for every  $(\bar{s}, Q) \in D(P, \varphi)$ .

This proposition, together with Theorems 4.1 and 4.3, immediately yields the following corollary.

**Corollary 4.7.** The relations  $\triangleleft$  and  $\preceq$  are decidable.

Coming back to our example with  $P = \bar{a}:b.P$  and  $Q = \bar{a}:(b.Q + b.\bar{c}:\mathbf{1})$ , it suffices to check the conditions of Theorem 4.3 on the following finite sets:  $D(P) = \{(\varepsilon, b.P), (\bar{a}, b.P)\}$  and  $D(Q) = \{(\varepsilon, b.Q + b.\bar{c}:\mathbf{1}), (\bar{a}, b.Q + b.\bar{c}:\mathbf{1}), (\bar{a}, \mathbf{1})\}$  (in order to apply Proposition 4.6 we must also consider  $D(b.P)$ ,  $D(b.Q + b.\bar{c}:\mathbf{1})$ , and  $D(\mathbf{1})$ ).

## 5 Conclusion

We have shown how to formalize in CCS the kind output order preserving asynchrony typical of the web and studied testing equivalences of processes since, as argued here and in several other works, this kind of equivalences is the right one to characterize the behavior of web services. We gave the first proof, in our knowledge, of decidability for a testing equivalence for an asynchronous CCS.

Certainly the refinement relation we defined is still too gross in the perspective of applying it to choreographies of services. Currently, compliance is an asymmetric relation that is defined only on the basis of client satisfaction, independently from what the service will do. This is an approach we already took in previous works [6, 7], which nicely fits a client-service asymmetric scenario but can hardly be extended to choreographic assemblages of services. In a choreography each participant acts as client and service at the same time. By taking the symmetric closure of compliance, which is normally called *duality relation*, both the tester and the process being tested must reach a successful test. In this setting the refinement relation depends on a *viability predicate* that characterizes those processes that do have a dual. Refinement seems to be decidable, but only up to decidability of viability, which we have not been able to prove.

As a final note it should also be mentioned that, despite the decidability results, we have not produced an actual algorithm for deciding compliance and refinement. This involves relating the sequences  $\varphi$  of actions performed by processes with their syntax trees.

## References

- [1] R. Beauxis, C. Palamidessi, and F. Valencia. On the asynchronous nature of the asynchronous pi-calculus. In *Concurrency, Graphs and Models*, LNCS 5065, pages 473–492, 2008.

- [2] J. A. Bergstra, J. W. Klop, and J. V. Tucker. Process algebra with asynchronous communication mechanisms. In *Proceedings of the Seminar on Concurrency*, number 197 in LNCS. Springer, 1985.
- [3] M. Boreale, R. D. Nicola, and R. Pugliese. Basic observables for processes. In *ICALP*, volume 1256 of LNCS, pages 482–492. Springer, 1997.
- [4] G. Boudol. Asynchrony and the pi-calculus. Technical Report RR-1702, INRIA, 1992.
- [5] M. Bravetti and G. Zavattaro. Contract compliance and choreography conformance in the presence of message queues. In *WS-FM'08*, number 5387 in LNCS. Springer, 2008.
- [6] S. Carpineti, G. Castagna, C. Laneve, and L. Padovani. A formal account of contracts for Web Services. In *WS-FM '06, 3rd Int. Workshop on Web Services and Formal Methods*, number 4184 in LNCS, pages 148–162. Springer, 2006.
- [7] G. Castagna, N. Gesbert, and L. Padovani. A theory of contracts for Web Services. *ACM TOPLAS*, 31(5), 2009.
- [8] G. Castagna and L. Padovani. Contracts for mobile processes. In *CONCUR 2009, 20th. International Conference on Concurrency Theory*, LNCS. Springer, 2009.
- [9] I. Castellani and M. Hennessy. Testing theories for asynchronous languages. In *Proceedings FST-TCS'98*, volume 1530 of *Lecture Notes in Computer Science*. Springer Verlag, 1998.
- [10] M. Coppo, M. Dezani-Ciancaglini, and N. Yoshida. Asynchronous Session Types and Progress for Object-Oriented Languages. In *FMOODS'07*, LNCS 4468. Springer, 2007.
- [11] B. Courcelle. Fundamental properties of infinite trees. *Theoretical Computer Science*, 25:95–169, 1983.
- [12] F. de Boer, J. Klop, and C. Palamidessi. Asynchronous communication in process algebra. In *LICS '92, 7th IEEE Symposium on Logic in Computer Science*, pages 137–147, 1992.
- [13] R. De Nicola and M. Hennessy. CCS without  $\tau$ 's. In *TAPSOFT/CAAP'87*, LNCS 249, pages 138–152. Springer, 1987.
- [14] S. Gay and V. Vasconcelos. Linear type theory for asynchronous session types. Unpublished manuscript. Subsumes Technical Report 2007–251, University of Glasgow, 2008.
- [15] M. Hennessy. *Algebraic Theory of Processes*. Foundation of Computing, MIT Press, 1988.
- [16] M. Hennessy. A fully abstract denotational semantics for the pi-calculus. *Theoretical Computer Science*, 278:53–89(37), 2002.
- [17] K. Honda and M. Tokoro. An object calculus for asynchronous communication. In *ECOOP '91, European Conference on Object-Oriented Programming*, LNCS, pages 133–147. Springer, 1991.
- [18] K. Honda and M. Tokoro. On asynchronous communication semantics. In *Object-Based Concurrent Computing*, volume 612 of LNCS, pages 21–51. Springer, 1991.
- [19] K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. In *POPL '08: the 35th annual ACM Symp. on Principles of Programming Languages*, 2008.
- [20] R. P. M. Boreale, R. De Nicola. Trace and testing equivalence on asynchronous processes. *Information and Computation*, 172(2):139–164, Jan. 2002.
- [21] D. Mostrous, N. Yoshida, and K. Honda. Global principal typing in partially commutative asynchronous sessions. In *Proc. of ESOP '09*, LNCS 5502, pages 316–332, 2009.
- [22] M. Neubauer and P. Thiemann. Session types for asynchronous communication. Unpublished manuscript, Feb. 2004.
- [23] R. Pugliese. A process calculus with asynchronous communications. In *Proc. of the 5th Italian Conference on Theoretical Computer Science*. World Scientific, 1995.

## A Proofs of Section 3

*Proof of Theorem 3.4.* We begin by showing that  $\triangleleft$  is a coinductive compliance relation. Suppose  $P \triangleleft Q$ . Suppose by contradiction  $P \downarrow \bar{a}$  and  $Q \not\Downarrow a$ . Then there exist  $P'$  and  $Q'$  such that  $P \Longrightarrow P' \not\rightarrow$  and  $Q \Longrightarrow Q' \rightarrow$  and  $P' \xrightarrow{\bar{a}}$  and  $Q' \not\xrightarrow{a}$ . Then  $P \parallel Q \Longrightarrow P' \parallel Q' \not\rightarrow$  which is absurd because, since  $P' \xrightarrow{\bar{a}}$ , then this implies that  $P'$  is of the form  $\bar{a}:P''$  and therefore  $P' \not\xrightarrow{a}$ . Suppose  $P \downarrow R$  and suppose, by contradiction, that  $Q \not\Downarrow$ . Then there exist  $P'$  and  $Q'$  such that  $P \Longrightarrow P' \not\rightarrow$  and  $Q \Longrightarrow Q' \rightarrow$  and  $\text{init}(P') \subseteq R \subseteq \mathcal{N}$  and  $\text{init}(Q') \subseteq \mathcal{N} \cup \{\checkmark\}$ . We deduce  $P \parallel Q \Longrightarrow P' \parallel Q' \not\rightarrow$  which is absurd since  $P' \not\xrightarrow{a}$ . The case for  $Q \downarrow \bar{a}$  with  $a \notin R$  is similar. Suppose  $P \xrightarrow{\bar{\alpha}}$  and  $Q \xrightarrow{\alpha}$  and suppose by contradiction that  $P(\bar{\alpha}) \not\triangleleft Q(\alpha)$ . Then there exists a derivation  $P(\bar{\alpha}) \parallel Q(\alpha) \Longrightarrow P' \parallel Q' \rightarrow$  such that  $P' \not\xrightarrow{a}$ . This is absurd since  $P \Longrightarrow P'$  and  $Q \Longrightarrow Q'$ .

Now we prove that  $\triangleleft$  is indeed the largest coinductive compliance. Suppose  $(P, Q) \in \mathcal{C}$  for some coinductive compliance  $\mathcal{C}$  and consider a derivation  $P \parallel Q \Longrightarrow P' \parallel Q' \rightarrow$ . By unzipping this derivation we obtain that there exists a sequence  $\alpha_1, \dots, \alpha_n$  of actions such that  $P \xrightarrow{\bar{\alpha}_1 \dots \bar{\alpha}_n} P'$  and  $Q \xrightarrow{\alpha_1 \dots \alpha_n} Q'$ . We prove that  $P' \not\xrightarrow{a}$  by induction on  $n$ . Suppose  $n = 0$ . If  $P' \xrightarrow{\bar{a}}$ , then  $P \downarrow \bar{a}$ . From condition (1) of Definition 3.3 we deduce  $Q \not\Downarrow a$ , hence  $Q' \not\Downarrow a$ , which is absurd since  $P' \parallel Q' \rightarrow$ . Hence  $P'$  cannot emit any output. Suppose  $\text{init}(P') \subseteq \mathcal{N}$ . Then  $P \downarrow \text{init}(P')$ . From condition (2) of Definition 3.3 we deduce  $Q \downarrow$  and  $Q \downarrow \bar{a}$  implies  $P' \xrightarrow{a}$ . Then there exists  $a$  such that  $Q' \xrightarrow{a}$  and  $P \xrightarrow{a}$ , which is absurd again from the hypothesis  $P' \parallel Q' \rightarrow$ . We deduce  $P' \not\xrightarrow{a}$ . Suppose  $n > 0$ . Then  $P \xrightarrow{\bar{\alpha}_1}$  and  $Q \xrightarrow{\alpha_1}$ . From condition (3) of Definition 3.3 we deduce  $(P(\bar{\alpha}_1), Q(\alpha_1)) \in \mathcal{C}$ . By induction hypothesis we conclude  $P' \not\xrightarrow{a}$ .  $\square$

*Proof of Theorem 3.6.* First we prove that  $\preceq$  is a coinductive refinement. Suppose  $P \preceq Q$ . As regards condition (1) of Definition 3.5, suppose  $P \downarrow$  and suppose, by contradiction, that  $Q \not\Downarrow$ . Let  $R \stackrel{\text{def}}{=} \sum_{P \downarrow \bar{a}} a.1$ . We have  $R \triangleleft P$  and  $R \not\triangleleft Q$ , which is absurd. Hence  $Q \downarrow$ . As regards condition (2) of Definition 3.5, suppose  $Q \downarrow \bar{a}$  and suppose, by contradiction, that  $P \not\Downarrow \bar{a}$ . Then  $\mathbf{1} + a \triangleleft P$  and  $\mathbf{1} + a \not\triangleleft Q$ , which is absurd. Hence  $P \downarrow \bar{a}$ . Consider an arbitrary  $R'$  such that  $R' \triangleleft P(\bar{a})$  and let  $R \stackrel{\text{def}}{=} \mathbf{1} + a.R'$ . We have  $R \triangleleft P$  from which we deduce  $R \triangleleft Q$ . We conclude  $P(\bar{a}) \preceq Q(\bar{a})$  since  $R'$  is arbitrary. As regards condition (3) of Definition 3.5, suppose  $P \not\Downarrow a$  and suppose, by contradiction, that  $Q \not\Downarrow a$ . Let  $R \stackrel{\text{def}}{=} \bar{a}.1$ . We have  $R \triangleleft P$  and  $R \not\triangleleft Q$ , which is absurd. Hence  $Q \downarrow a$ . Let  $R'$  be such that  $R' \triangleleft P(a)$  and let  $R \stackrel{\text{def}}{=} \bar{a}.R'$ . We have  $R \triangleleft P$  hence  $R \triangleleft Q$ . We conclude  $P(a) \preceq Q(a)$  since  $R'$  is arbitrary.

Now we prove that  $\preceq$  is the largest coinductive refinement, by showing that every coinductive refinement is included in it. Let  $\mathcal{S}$  be a coinductive refinement such that  $(P, Q) \in \mathcal{S}$  and let  $R \triangleleft P$ . Consider a derivation  $R \parallel Q \Longrightarrow R' \parallel Q' \rightarrow$ . By unzipping this derivation we obtain that there exist  $\mu_1, \dots, \mu_n$  such that  $R \xrightarrow{\bar{\mu}_1 \dots \bar{\mu}_n} R'$  and  $Q \xrightarrow{\mu_1 \dots \mu_n} Q'$ . We reason by induction on  $n$  to show that  $R' \not\xrightarrow{a}$ . Suppose  $n = 0$ . We distinguish three subcases. If  $R' \downarrow \bar{a}$ , then from  $R \triangleleft P$  we deduce  $P \downarrow a$  and  $R(\bar{a}) \triangleleft P(a)$ . From  $(P, Q) \in \mathcal{S}$  and condition (3) of Definition 3.5 we conclude  $Q \downarrow a$ , which contradicts  $R' \parallel Q' \rightarrow$ . Hence  $R'$  cannot emit any output. Suppose  $R' \not\xrightarrow{a}$ . From  $R \triangleleft P$  we deduce  $P \downarrow$ . From  $(P, Q) \in \mathcal{S}$  and condition (1) of Definition 3.5 we deduce  $Q \downarrow$ . Hence there exists  $a$  such that  $Q' \downarrow \bar{a}$ . From condition (2) of Definition 3.5 we deduce  $P \downarrow \bar{a}$ . From  $R' \not\Downarrow$  we deduce  $R' \not\Downarrow a$ , which contradicts  $R' \parallel Q' \rightarrow$ . Hence we conclude  $R' \not\xrightarrow{a}$ . Suppose  $n > 0$ . We distinguish two subcases. If  $\mu_1 = a$ , then from the hypothesis  $R \triangleleft P$  we deduce  $P \downarrow a$  and  $R(\bar{a}) \triangleleft P(a)$ . From condition (3) of Definition 3.5 we deduce  $Q \downarrow a$  and  $(P(a), Q(a)) \in \mathcal{S}$ . If  $\mu_1 = \bar{a}$ , then from the hypothesis  $(P, Q) \in \mathcal{S}$  we deduce  $P \downarrow \bar{a}$  and  $(P(\bar{a}), Q(\bar{a})) \in \mathcal{S}$ . From  $R \triangleleft P$  we

derive  $R(a) \triangleleft P(\bar{a})$ . In both subcases we have  $R(\bar{\mu}_1) \triangleleft P(\mu_1)$  and  $(P(\mu_1), Q(\mu_1)) \in \mathcal{S}$ . Furthermore, we have  $R(\bar{\mu}_1) \xrightarrow{\mu_2 \cdots \mu_n} R'$  and  $Q(\mu_1) \xrightarrow{\mu_2 \cdots \mu_n} Q'$ . By induction hypothesis we conclude  $R' \checkmark$ .  $\square$

## B Proofs of Section 4

*Proof of Theorem 4.1.* (“if” part) It is sufficient to show that

$$\mathcal{C} \stackrel{\text{def}}{=} \{(P(\bar{\varphi}), Q(\varphi)) \mid P \xrightarrow{\bar{\varphi}}, Q \xrightarrow{\varphi}\}$$

is a coinductive compliance. Let  $(P', Q') \in \mathcal{C}$ . Then there exists  $\varphi$  such that  $P \xrightarrow{\bar{\varphi}}$  and  $Q \xrightarrow{\varphi}$  and  $P' = P(\bar{\varphi})$  and  $Q' = Q(\varphi)$ . Conditions (1) and (2) of Definition 3.3 immediately follow since they exactly correspond to items (1) and (2) above. As regards condition (3) of Definition 3.3, suppose  $P' \xrightarrow{\bar{\alpha}}$  and  $Q' \xrightarrow{\alpha}$ . Then  $P \xrightarrow{\bar{\varphi}\bar{\alpha}}$  and  $Q \xrightarrow{\varphi\alpha}$  and  $(P(\bar{\varphi}\bar{\alpha}), Q(\varphi\alpha)) \in \mathcal{C}$  by definition of  $\mathcal{C}$ . We conclude by observing that  $P'(\bar{\alpha}) = P(\bar{\varphi}\bar{\alpha})$  and  $Q'(\alpha) = Q(\varphi\alpha)$ .

(“only if” part) Suppose  $P \triangleleft Q$ . Then there exists a coinductive compliance  $\mathcal{C}$  such that  $(P, Q) \in \mathcal{C}$ . Suppose  $P \xrightarrow{\bar{\varphi}}$  and  $Q \xrightarrow{\varphi}$ . We reason by induction on  $\varphi$  to prove items (1) and (2) above. If  $\varphi = \varepsilon$ , then items (1) and (2) follow immediately from the corresponding conditions (1) and (2) of Definition 3.3. If  $\varphi = \alpha\varphi'$  for some  $\alpha$  and  $\varphi'$ , then from condition (3) of Definition 3.3 we deduce  $(P(\bar{\alpha}), Q(\alpha)) \in \mathcal{C}$ . By induction hypothesis we conclude that items (1) and (2) hold for  $P(\bar{\alpha})(\bar{\varphi}') = P(\varphi)$  and  $Q(\alpha)(\varphi') = Q(\varphi)$ .  $\square$

*Proof of Theorem 4.3.* (“if” part) It is sufficient to show that the relation

$$\mathcal{S} \stackrel{\text{def}}{=} \{(P(\varphi), Q(\varphi)) \mid P \Downarrow \varphi, Q \xrightarrow{\varphi}\}$$

is a coinductive refinement. Let  $(P', Q') \in \mathcal{S}$ . Then there exists  $\varphi$  such that  $P \Downarrow \varphi$  and  $Q \xrightarrow{\varphi}$  and  $P' = P(\varphi)$  and  $Q' = Q(\varphi)$ . Condition (1) of Definition 3.5 immediately follows from item (1). As regards condition (2) of Definition 3.5, suppose

(“only if” part)  $\square$