Multiparty testing preorders

Rocco De Nicola[1] and Hernán Melgratti[2]

1 **IMT Lucca Institute for Advanced Studies, Italy**
`rocco.denicola@imtlucca.it`
2 **DC, FCEyN, Universidad de Buenos Aires - CONICET, Argentina**
`hmelgra@dc.uba.ar`

Abstract

Variants of the must testing approach have been successfully applied in Service Oriented Computing for analysing the compliance between (contracts exposed by) clients and servers or, more generally, between two peers. It has however been argued that multiparty scenarios call for more permissive notions of compliance because partners usually do not have full coordination capabilities. We propose two new testing preorders, which are obtained by restricting the set of potential observers. For the first preorder, called uncoordinated, we allow only sets of parallel observers that use different parts of the interface of a given service and have no possibility of intercommunication. For the second preorder, that we call independent, we instead rely on parallel observers that perceive as silent all the actions that are not in the interface of interest. We have that the uncoordinated preorder is coarser than the classical must testing preorder and finer than the independent one. We also provide a characterisation in terms of decorated traces for both preorders: the uncoordinated preorder is defined in terms of must-sets and Mazurkiewicz traces while the independent one is described in terms of classes of filtered traces that only contain designated visible actions and must-sets.

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

1 Introduction

A desired property of communication-centered systems is the graceful termination of the partners involved in a multiparty interaction, i.e., every possible interaction among a set of communicating partners ends successfully, in the sense that there are no messages waiting forever to be sent, or sent messages which are never received. The theories of session types [16, 8] and of contracts [5, 6, 3, 9] are commonly used to ensure such kind of properties. The key idea behind both approaches is to associate to each process a type (or *contract*) that gives an abstract description of its external, visible behavior and to use type checking to verify correctness of behaviours.

Services are often specified by sequential nondeterministic CCS processes [12] describing the communications offered by peers, built-up from invoke and accept activities, which are abstractly represented as input and output actions that take place over a set of channels or names, and internal τ actions. Basic actions can be composed sequentially (prefix operator ".") or as alternatives (non deterministic choice "+"). Moreover, the language for describing services does not provide any operator for parallel composition. It is assumed that all possible interleavings are made explicit in the description of the service and communication is used only for modelling the interaction among different peers.

Services come equipped with a notion of *compliance* that characterises all valid clients of a service, i.e., those clients that are guaranteed to terminate after any possible interaction with the service. Compliance has been characterized by using a suitable variant of the *must testing* approach [7], which allows comparing processes according to the ability of the external observers to distinguishing them. Processes that are must-equivalent are characterized by

Conference title on which this volume is based on.
Editors: Billy Editor and Bill Editors; pp. 1–13

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

the set of tests or observers that they are able to pass: any observer is defined as a unique process that runs in parallel with the tested service and, consequently, all interactions with the observed service are handled by a unique, central process, i.e. the observer. Technically, two given processes p and q are related via the must preorder ($p \sqsubseteq_{\mathsf{must}} q$) if q passes all tests that are passed by p. Consequently, p and q are considered equivalent ($p \approx_{\mathsf{must}} q$) if they pass exactly the same tests.

If one considers a multiparty setting, each service may interact with several partners and its interface is often (logically) partitioned by allowing each partner to communicate only through dedicated parts of the interface. In many cases, the peers of a specific service do not communicate with each other. In these situations, the classical testing approach to process equivalences or preorders turns out to be too demanding. Consider the following scenario involving three partners: an organisation (the broker) that sells goods produced by a different company (the producer) to a specific customer (the client). The behaviour of the broker can be described with the following process:

$$B = req.\overline{order}.\overline{inv}.$$

The broker accepts requests on channel *req* and then places an order to the producer with the message $\overline{order}$ and sends an invoice to the customer with the message $\overline{inv}$. In this scenario, the broker uses the channels *req* and *inv* to interact with the customer, while the interaction with the producer is over the channel *order*. Moreover, the customer and the producer do not know each other and are completely independent. Hence, the order in which messages $\overline{order}$ and $\overline{inv}$ are sent is completely irrelevant for them. They would be equally happy with a broker defined as follows:

$$B' = req.\overline{inv}.\overline{order}.$$

Nevertheless, these two different implementations are not considered must-equivalent.

The main goal of this paper is to introduce alternative, less discriminating, preorders that take into account the distributed nature of the peers and thus the limited coordination and interaction capabilities of the different players. A first preorder, called uncoordinated must preorder, is obtained by assuming that all clients of a given service do interact with it via disjoint sets of ports, i.e. they use different parts of its interface, have no possibility of intercommunication, and all of them terminate successfully in every possible interaction. Even though there is no intercommunication among clients, there is still some inter-dependency among them, e.g. because one client refuses to interact in a way required for another. In this way, it is possible to differentiate B from B' above when the observer refuses to synchronise over the port *order*. We introduce then a second preorder, that we call independent must preorder, which ignores possible inter-dependencies among clients actions and, thanks to the more limited discriminating power, guarantees increased acceptability of offered services.

The two preorders are as usual defined in terms of the outcomes of experiments by specific sets of observers. For defining the uncoordinated must preorder, we allow only sets of parallel observers that cannot intercommunicate and do challenge services via disjoint parts of their interface. For defining the independent must preorder, we instead rely on parallel observers that, again, cannot intercommunicate but in addition perceive as silent all the actions that are not in the interface of their interest. This is instrumental to avoid that a specific client recovers information about the other involved clients. As expected, we have that the uncoordinated preorder is coarser than the classical must testing preorder and finer than the independent one.

Just like for classical testing preorders, we provide a characterisation for both new preorders in terms of decorated traces, which avoids dealing with universal quantifications over

Conference title on which this volume is based on.
Editors: Billy Editor and Bill Editors; pp. 2–13

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

the set of observers whenever a specific relation between two processes has to be established. The alternative characterizations make it even more evident that our preorders permit action reordering. Indeed, the uncoordinated preorder is defined in terms of Mazurkiewicz traces [11] while the independent one is described in terms of classes of traces quotiented via specific sets of visible actions. We would like to remark that our two preorders are different from those defined in [4, 14, 13], which also permit action reordering by relying on buffered communication. Additional details will be provided in §6.

Synopsis The remainder of this paper is organised as follows. In §2 we recall the basics of the classical must testing approach. In §3 and §4 we present the theory of uncoordinated and independent must testing preorders and their characterisation in terms of traces. In §5 we show that the uncoordinated preorder is coarser than the must testing preorder but finer than the independent one. Finally, we discuss some related work and future developments in §6.

2 Processes and testing preorders

Let $\mathcal{N}$ be a countable set of action names, ranged over by $a, b, \ldots$. As usual, we write co-names in $\overline{\mathcal{N}}$ as $\overline{a}, \overline{b}, \ldots$ and assume $\overline{\overline{a}} = a$. We will use α, β to range over $\mathsf{Act} = (\mathcal{N} \cup \overline{\mathcal{N}})$. Moreover, we consider a distinguished internal action τ not in Act and use μ to range over $\mathsf{Act} \cup \{\tau\}$. We fix the language of defining services as the sequential fragment of CCS extended with a *success* operator, as specified by the following grammar.

$$p, q ::= \mathbf{0} \mid \mathbf{1} \mid \mu.p \mid p + q \mid X \mid \mathbf{rec}_X.p$$

The process $\mathbf{0}$ stands for the terminated process, $\mathbf{1}$ for the process that reports success and then terminates, and $\mu.p$ for a service that executes μ and then continues as p. Alternative behaviours are specified by terms of the form $p + q$, while recursive ones are introduced by terms like $\mathbf{rec}_X.p$. We sometimes omit trailing $\mathbf{0}$ and write, e.g., $a.b + c$ instead of $a.b.\mathbf{0} + c.\mathbf{0}$. We write $\mathtt{n}(p)$ for the set of names $a \in \mathcal{N}$ such that either a or $\overline{a}$ occur in p.

The operational semantics of processes is given in terms of a labelled transition system (LTS) $p \xrightarrow{\lambda} q$ with $\lambda \in \mathsf{Act} \cup \{\tau, \checkmark\}$, where $\checkmark$ signals the successful termination of an execution.

▶ **Definition 1** (Transition relation)**.** The transition relation on processes, noted $\xrightarrow{\lambda}$, is the least relation satisfying the following rules

$$\mathbf{1} \xrightarrow{\checkmark} \mathbf{0} \qquad \mu.p \xrightarrow{\mu} p \qquad \frac{p \xrightarrow{\lambda} p'}{p + q \xrightarrow{\lambda} p'} \qquad \frac{q \xrightarrow{\lambda} q'}{p + q \xrightarrow{\lambda} q'} \qquad \frac{p[\mathbf{rec}_X.p/X] \xrightarrow{\lambda} p'}{\mathbf{rec}_X.p \xrightarrow{\lambda} p'}$$

◀

Multiparty applications, named *configurations*, are built by composing processes concurrently. Formally, configurations are given by the following grammar.

$$c, d ::= p \mid c \| d$$

We sometimes write $\Pi_{i \in 0..n} p_i$ for the parallel composition $p_0 \parallel \ldots \parallel p_n$. The operational semantics of configurations, which accounts for the communication between peers, is obtained by extending the LTS in Definition 1 with the following rules:

$$\frac{c \xrightarrow{\mu} c'}{c \parallel d \xrightarrow{\mu} c' \parallel d} \qquad \frac{d \xrightarrow{\mu} d'}{c \parallel d \xrightarrow{\mu} c \parallel d'} \qquad \frac{c \xrightarrow{\alpha} c' \quad d \xrightarrow{\overline{\alpha}} d'}{c \parallel d \xrightarrow{\tau} c' \parallel d'} \qquad \frac{c \xrightarrow{\checkmark} c' \quad d \xrightarrow{\checkmark} d'}{c \parallel d \xrightarrow{\checkmark} c' \parallel d'}$$

Conference title on which this volume is based on.
Editors: Billy Editor and Bill Editors; pp. 3–13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

All rules are standard apart for the last one that is not present in [7]. This rule states that the concurrent composition of processes can report success only when all processes in the composition do so.

We write $c \xrightarrow{\lambda}$ when there exists c' s.t. $c \xrightarrow{\lambda} c'$; $\Rightarrow$ for the reflexive and transitive closure of $\xrightarrow{\tau}$; $c \stackrel{\lambda}{\Longrightarrow} c'$ for $\lambda \in \mathsf{Act} \cup \{\checkmark\}$ and $c \Longrightarrow \xrightarrow{\lambda} \Longrightarrow$; $c \stackrel{\lambda_0 \ldots \lambda_n}{\Longrightarrow} c'$ for $c \stackrel{\lambda_0}{\Longrightarrow} \ldots \stackrel{\lambda_n}{\Longrightarrow} c'$, and $c \stackrel{s}{\Longrightarrow}$ with $s \in (\mathsf{Act} \cup \{\checkmark\})^*$ if there exists c' s.t. $c \stackrel{s}{\Longrightarrow} c'$. We write $\mathsf{str}(c)$ and $\mathsf{init}(c)$ to denote the sets of strings and enabled actions of c, defined as follows

$$\mathsf{str}(c) = \{s \in (\mathsf{Act} \cup \{\checkmark\})^* \mid c \stackrel{s}{\Longrightarrow}\} \qquad \mathsf{init}(c) = \{\lambda \in \mathsf{Act} \cup \{\checkmark\} \mid c \stackrel{\lambda}{\Longrightarrow}\}$$

As behavioural semantics, we will consider the must-testing preorder [7]. We take all possible configurations as the set $\mathcal{O}$ of *observers*, ranged over by $o, o_0, \ldots, o', \ldots$. If we do only allow observers to report success and use only sequential observers we recover the standard framework of [7].

▶ **Definition 2** (must). A sequence of transitions $p_0 \parallel o_0 \xrightarrow{\tau} \ldots \xrightarrow{\tau} p_k \parallel o_k \xrightarrow{\tau} \ldots$ is a maximal computation if either it is infinite or the last term $p_n \parallel o_n$ is such that $p_n \parallel o_n \not\xrightarrow{\tau}$. Let p must o iff for each maximal computation $p \parallel o = p_0 \parallel o_0 \xrightarrow{\tau} \ldots \xrightarrow{\tau} p_k \parallel o_k \xrightarrow{\tau} \ldots$ there exists $n \geq 0$ such that $o_n \xrightarrow{\checkmark}$. ◀

We say that a computation $c_o \xrightarrow{\mu_0} \ldots c_i \xrightarrow{\mu_i} \ldots \xrightarrow{\mu_n} c_{n+1}$ is unsuccessful when $c_j \not\xrightarrow{\checkmark}$ for all $0 \leq j \leq n+1$, we say it successful otherwise.

The notion of passing a test represents the fact that a configuration, i.e., a set of partners, is able to successfully terminate every possible interaction with the process under test. Then, it is natural to compare processes accordingly to their capacity to satisfy partners.

▶ **Definition 3** (must preorder). $p \sqsubseteq_{\mathsf{must}} q$ iff $\forall o \in \mathcal{O} : p$ must o implies q must o. We write $p \approx_{\mathsf{must}} q$ when both $p \sqsubseteq_{\mathsf{must}} q$ and $q \sqsubseteq_{\mathsf{must}} p$. ◀

2.1 Semantic characterisation

The must testing preorder has been characterised in [7] in terms of the sequences of actions that a process may perform and the possible sets of actions that it may perform after executing a particular sequence of actions. This characterisation relies on a few auxiliary predicates and functions that are presented below. A process p *diverges*, written $p \Uparrow$, when it exhibits an infinite, internal computation $p \xrightarrow{\tau} p_0 \xrightarrow{\tau} p_1 \xrightarrow{\tau} \ldots$. We say p *converges*, written $p \Downarrow$, otherwise. For $s \in \mathsf{Act}^*$, the convergence predicate is inductively defined by the following rules:

- $p \Downarrow \epsilon$ if $p \Downarrow$.
- $p \Downarrow \alpha.s$ if $p \Downarrow$ and $p \stackrel{\alpha}{\Longrightarrow} p'$ implies $p' \Downarrow s$.

The *residual* of a process p (or a set of processes P) after the execution of $s \in \mathsf{Act}^*$ is given by the following equations

- $(p \text{ after } s) = \{p' \mid p \stackrel{s}{\Longrightarrow} p'\}$.
- $(P \text{ after } s) = \bigcup_{p \in P} (p \text{ after } s)$.

▶ **Definition 4** (Must-set). A *must-set* of process p (or set of processes P) is $L \subseteq \mathsf{Act}$, L finite s.t.

- p MUST L iff $\forall p'$ s.t. $p \Longrightarrow p'$, $\exists \alpha \in L$ such that $p' \stackrel{\alpha}{\Longrightarrow}$.
- P MUST L iff $\forall p \in P.p$ MUST L. ◀

Conference title on which this volume is based on.
Editors: Billy Editor and Bill Editors; pp. 4–13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Then, the must testing preorder can be characterised in terms of strings and must-sets as follows.

▶ **Definition 5.** $p \preceq_{\mathsf{must}} q$ if for every $s \in \mathsf{Act}^*$, for all finite $L \subseteq \mathsf{Act}$, if $p \Downarrow s$ then

- $q \Downarrow s$.
- (p after s) MUST L implies (q after s) MUST L. ◀

▶ **Theorem 6** ([7]). $\sqsubseteq_{\mathsf{must}} = \preceq_{\mathsf{must}}$.

3 A testing preorder with uncoordinated observers

The must testing preorder is defined in terms of the tests that each process is able to pass. Remarkably, the classical setting can be formulated by considering only sequential tests (see the characterisation of minimal tests in [7]). Each sequential test is a unique, centralised process that handles all the interaction with the service under test and, therefore, has a complete view of the externally observable behaviour of the service. For this reason, we refer to the classical must testing preorder as a *centralised preorder*. Multiparty interactions are generally structured in such a way that pairs of partners communicate through dedicated channels, for example, partner links in service oriented models or buffers in communicating machines [1]. Conceptually, the interface (i.e., the set of channels) of a service is partitioned and the service interacts with each partner by using only specific sets of channels in its interface. In addition, there are scenarios in which partners frequently do not know each other and cannot communicate directly. As a direct consequence, the partners of a process cannot establish causal dependencies among actions that take place over different parts of the interface. These constraints reduce the discriminating power of partners and call for coarser equivalences that equate processes that cannot be distinguished by independent sets of sequential processes only interested in specific interactions.

▶ **Example 7.** Consider the classical scenario for planning a trip. A user U interacts with a broker B, which is responsible for booking flights provided by a service F and hotel rooms available at service H. The expected interaction can be described as follows: U makes a booking request by sending a message req to B (we will just describe the interaction and abstract away from data details such as trip destination, departure dates and duration). Depending on the request, B may contact service F (for booking just a flight ticket), H (for booking rooms) or both. Service B uses channels $reqF$ and $reqH$ to respectively contact F and H (for the sake of simplicity, we assume that any request to F and H will be granted). Then, the expected behaviour of B can be described with the following process:

$$B_0 \stackrel{def}{=} req.(\tau.\overline{reqF} + \tau.\overline{reqH} + \tau.\overline{reqH}.\overline{reqF})$$

In this process, the third branch represents B's choice to contact first H and then F. Nevertheless, the other partners (U, F and H) are not affected in any way by this choice, thus they would be equally happy with alternative definitions such as:

$$B_1 \stackrel{def}{=} req.(\tau.\overline{reqF} + \tau.\overline{reqH} + \tau.\overline{reqF}.\overline{reqH})$$
$$B_2 \stackrel{def}{=} req.(\tau.\overline{reqF} + \tau.\overline{reqH} + \tau.\overline{reqH}.\overline{reqF} + \tau.\overline{reqF}.\overline{reqH})$$

Unfortunately, B_0, B_1 and B_2 are distinguished by the must testing equivalence. It suffices to consider $o_0 = \overline{req}.(\tau.\mathbf{1} + reqF.(\tau.\mathbf{1} + reqH.\mathbf{0}))$ for showing that $B_0 \not\sqsubseteq_{\mathsf{must}} B_1$ and that $B_0 \not\sqsubseteq_{\mathsf{must}} B_2$, and use $o_1 = \overline{req}.(\tau.\mathbf{1} + reqH.(\tau.\mathbf{1} + reqF.\mathbf{0}))$ for proving that $B_1 \not\sqsubseteq_{\mathsf{must}} B_2$. ◀

Conference title on which this volume is based on.
Editors: Billy Editor and Bill Editors; pp. 5–13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

This section is devoted to the definition and characterization of a preorder that is coarser than the classical must preorder and relates processes that cannot be distinguished by distributed contexts. We start by introducing the notion of uncoordinated observers.

▶ **Definition 8** (Uncoordinated observer)**.** A process $\Pi_{i\in 0..n} o_i = o_0 \parallel \ldots \parallel o_n$ is an *uncoordinated observer* if $\mathtt{n}(o_i) \cap \mathtt{n}(o_j) = \emptyset$ for all $i \neq j$. ◀

Obviously, the condition $\mathtt{n}(o_i) \cap \mathtt{n}(o_j) = \emptyset$ forbids the direct communication between the sequential components of an uncoordinated observer. As a consequence, a distributed observer cannot impose a total order between actions that are controlled by different components of the observer. Indeed, the executions of a distributed observer are the interleavings of the executions of all sequential components $\{o_i\}_{i\in 0..n}$ (this property is formally stated in Section 3.1, Lemma 11). We remark that a configuration does report success (i.e., perform action $\checkmark$) only when all sequential processes in the composition do report success; an uncoordinated observer reports success when all its components report success simultaneously.

The uncoordinated must testing preorder is obtained by restricting the set of observers to consider just uncoordinated observers over a suitable partition of the interface of a process. We will say $\mathbb{I} = \{I_i\}_{i\in 0\ldots n}$ is an interface whenever $\mathbb{I}$ is a partition of Act and $\forall \alpha \in \mathsf{Act}$, $\alpha \in I_i$ implies $\overline{\alpha} \in I_i$. In the remaining of this paper we usually write only the relevant part of an interface. For instance, we will write $\{\{a\},\{b\}\}$ for any interface $\{I_0, I_1\}$ such that $a \in I_0$ and $b \in I_1$.

▶ **Definition 9** (Uncoordinated must preorder $\sqsubseteq^{\mathbb{I}}_{\mathsf{unc}}$)**.** Let $\mathbb{I} = \{I_i\}_{i\in 0\ldots n}$ be an interface. We say $p \sqsubseteq^{\mathbb{I}}_{\mathsf{unc}} q$ iff for all $\Pi_{i\in 0..n} o_i$ such that $\mathtt{n}(o_i) \subseteq I_i$, p must $\Pi_{i\in 0..n} o_i$ implies q must $\Pi_{i\in 0..n} o_i$. We write $p \approx^{\mathbb{I}}_{\mathsf{unc}} q$ when both $p \sqsubseteq^{\mathbb{I}}_{\mathsf{unc}} q$ and $q \sqsubseteq^{\mathbb{I}}_{\mathsf{unc}} p$. ◀

▶ **Example 10.** Consider the scenario presented in Example 7 and the following interface $\mathbb{I} = \{\{req\},\{reqF\},\{reqH\}\}$ for the process B that thus interacts with each of the other partners by using a dedicated part of its interface. It can be shown that the three definitions for B in Example 7 are equivalent when considering the uncoordinated must testing preorder, i.e., $B_0 \approx^{\mathbb{I}}_{\mathsf{unc}} B_1 \approx^{\mathbb{I}}_{\mathsf{unc}} B_2$. The actual proof, which uses the (trace-based) alternative characterization of the preorder, is deferred to Example 15. ◀

3.1 Semantic characterisation

We now address the problem of characterising the uncoordinated must testing preorder in terms of traces and must-sets. In order to do that, we shift from strings to Mazurkiewicz traces [10]. A *Mazurkiewicz trace* is a set of strings, obtained by permuting independent symbols. Traces represent concurrent computations, in which commuting letters stand for actions that execute independently of one another and non-commuting symbols are causally dependent actions. We start by summarizing the basics of the theory of traces in [10].

Let $D \subseteq \mathsf{Act} \times \mathsf{Act}$ be a finite, equivalence relation, called the *dependency relation*, that relates the actions that cannot be commuted. Thus if $(\alpha,\beta) \in D$, the two actions have to be considered causally dependent. Symmetrically, $I_D = (\mathsf{Act} \times \mathsf{Act}) \setminus D$ stands for the *independency* relation with $(\alpha,\beta) \in I_D$ meaning that α and β are concurrent.

The trace equivalence induced by the dependency relation D is the least congruence $\equiv_D$ in Act such that for all $\alpha,\beta \in \mathsf{Act} : (\alpha,\beta) \in I_D \implies \alpha\beta \equiv_D \beta\alpha$.

The equivalence classes of $\equiv_D$, denoted by $[s]_D$, are the (Mazurkiewicz) *traces*, namely the strings quotiented via $\equiv_D$. The trace monoid, denoted as $\mathbb{M}(D)$, is the quotient monoid $\mathbb{M}(D) = \mathsf{Act}^*/_{\equiv_D}$ whose elements are the traces induced by D. We remark that no action can commute with $\checkmark$ because I_D is defined over $\mathsf{Act} \times \mathsf{Act}$.

Conference title on which this volume is based on.
Editors: Billy Editor and Bill Editors; pp. 6–13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Let $\mathbb{I}$ be an interface, the *dependency relation induced by* $\mathbb{I}$ is $D = \bigcup_{I \in \mathbb{I}} I \times I$. The alternative characterisation of the uncoordinated preorder is defined in terms of equivalence classes of traces. Hence, we extend the transition relation and the notions of convergence and residuals to equivalence classes of strings:

- $q \stackrel{[s]_D}{\Longrightarrow} q'$ if and only if $\exists s' \in [s]_D$ such that $q \stackrel{s'}{\Longrightarrow} q'$
- $p \Downarrow [s]_D$ if $\forall s' \in [s]_D . p \Downarrow s'$
- $(p \text{ after } [s]_D) = \{p' \mid p \stackrel{[s]_D}{\Longrightarrow} p'\}$

Now we are able to characterise the behaviour of an uncoordinated observer. We formally state that an uncoordinated observer reaches the same processes after executing any of the sequences of actions in an equivalence class. This result is instrumental to the proof of the alternative characterisation of the uncoordinated preorder.

▶ **Lemma 11.** *Let $o = \Pi_{i \in 0..n} o_i$ be an observer for the interface $\mathbb{I} = \{I_i\}_{i \in 0..n}$ and D the dependency relation induced by $\mathbb{I}$. Then, for all $s \in \mathsf{Act}^*$ and $s' \in [s]_D$ we have $o \stackrel{s}{\Longrightarrow} o'$ iff $o \stackrel{s'}{\Longrightarrow} o'$.*

▶ **Corollary 12.** *Let $o = \Pi_{i \in 0..n} o_i$ be an observer for the interface $\mathbb{I} = \{I_i\}_{i \in 0..n}$ and D the dependency relation induced by $\mathbb{I}$. Then, $\forall s \in \mathsf{Act}^*, s' \in [s]_D$,*

1. *$s \in \mathsf{str}(o)$ implies $s' \in \mathsf{str}(o)$.*
2. *$o \Downarrow s$ implies $o \Downarrow s'$.*
3. *$(o \text{ after } s)$ MUST L implies $(o \text{ after } s')$ MUST L.*
4. *If there exists an unsuccessful computation $o \stackrel{s}{\Longrightarrow}$, then there exists an unsuccessful computation $o \stackrel{s'}{\Longrightarrow}$.*

The alternative characterisation for the uncoordinated preorder mimics the definition of the classical one, but relies on traces. In the definition below, the condition $L \subseteq I$ with $I \in \mathbb{I}$, captures the idea that each observation is relative to a specific part of the interface.

▶ **Definition 13.** Let $\mathbb{I}$ be an interface and D the dependency relation induced by $\mathbb{I}$. Then, $p \preceq^{\mathbb{I}}_{\mathsf{unc}} q$ if for every $s \in \mathsf{Act}^*$, for any part $I \in \mathbb{I}$, for all finite $L \subseteq I$, if $p \Downarrow [s]_D$ then

1. $q \Downarrow [s]_D$
2. $(p \text{ after } [s]_D)$ MUST L implies $(q \text{ after } [s]_D)$ MUST L ◀

▶ **Theorem 14.** $\sqsubseteq^{\mathbb{I}}_{\mathsf{unc}} = \preceq^{\mathbb{I}}_{\mathsf{unc}}$.

In the following we will write $L^I_{p,[s]_D}$ for the smallest set such that if $(p \text{ after } [s]_D)$ MUST L and $L \subseteq I$ then $L^I_{p,[s]_D} \subseteq L$.

▶ **Example 15.** We take advantage of the alternative characterisation of the uncoordinated preorder to show that the three processes for the broker in Example 7 are equivalent when considering $\mathbb{I} = \{\{req\}, \{reqF\}, \{reqH\}\}$. Actually, we will only consider $B_0 \approx^{\mathbb{I}}_{\mathsf{unc}} B_1$, being that the proofs for $B_0 \approx^{\mathbb{I}}_{\mathsf{unc}} B_2$ and $B_1 \approx^{\mathbb{I}}_{\mathsf{unc}} B_2$ are analogous.

Firstly, we have to consider that $B_0 \Downarrow s$ and $B_1 \Downarrow s$ for any s because B_0 and B_1 do not have infinite computations. The relation between must-sets are described in the two tables below. The first table shows the sets $(B_0 \text{ after } [s]_D)$ and $L^I_{B_0,[s]_D}$. Note that $[s]_D$ in the first column will be represented by any string $s' \in [s]_D$. Moreover, we write "$-$" in the tree last columns whenever $L^I_{B_0,[s]_D}$ does not exist. The second table does the same for B_1. In the tables, we let B'_0 stand for $\tau.\overline{reqF} + \tau.\overline{reqH} + \tau.\overline{reqH}.\overline{reqF}$ and B'_1 stand for $\tau.\overline{reqF} + \tau.\overline{reqH} + \tau.\overline{reqF}.\overline{reqH}$.

Conference title on which this volume is based on.
Editors: Billy Editor and Bill Editors; pp. 7–13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

$[s]_D$	B_0 after $[s]_D$	$L^{\{req\}}_{B_0,[s]_D}$	$L^{\{reqH\}}_{B_0,[s]_D}$	$L^{\{reqF\}}_{B_0,[s]_D}$
ϵ	B_0	$\{req\}$	–	–
req	$\{B_0', \overline{reqF}, \overline{reqH}, \overline{reqH}.\overline{reqF}\}$	–	–	–
$req.\overline{reqF}$	$\{0\}$	–	–	–
$req.\overline{reqH}$	$\{0, \overline{reqF}\}$	–	–	–
$req.\overline{reqF}.\overline{reqH}$	$\{0\}$	–	–	–
other	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

$[s]_D$	B_1 after $[s]_D$	$L^{\{req\}}_{B_0,[s]_D}$	$L^{\{reqH\}}_{B_0,[s]_D}$	$L^{\{reqF\}}_{B_0,[s]_D}$
ϵ	B_1	$\{req\}$	–	–
req	$\{B_1', \overline{reqF}, \overline{reqH}, \overline{reqF}.\overline{reqH}\}$	–	–	–
$req.\overline{reqF}$	$\{0, \overline{reqH}\}$	–	–	–
$req.\overline{reqH}$	$\{0\}$	–	–	–
$req.\overline{reqF}.\overline{reqH}$	$\{0\}$	–	–	–
other	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

By inspecting the tables, we can check that for any possible trace $[s]_D$ and $I \in \mathbb{I}$, it holds that $L^I_{B_0,[s]_D} = L^I_{B_1,[s]_D}$. Consequently, ($B_0$ after $[s]_D$) MUST L iff (B_1 after $[s]_D$) MUST L and thus we have $B_0 \approx^{\mathbb{I}}_{\mathsf{unc}} B_1$.

We now present two additional examples that help us in understanding the discriminating capability of the uncoordinated preorder, its differences with the classical must preorder and its adequacy for modelling process conformance.

The first of these examples shows that a process that does not communicate its internal choices to all of its clients is useless in a distributed context.

▶ **Example 16.** Consider the process $p = \tau.a + \tau.b$ that is intended to be used by two partners with the following interface: $\mathbb{I} = \{\{a\}, \{b\}\}$. We show that this process is less useful than 0 in an uncoordinated context, i.e., $\tau.a + \tau.b \sqsubseteq^{\mathbb{I}}_{\mathsf{unc}} 0$. It is immediate to see that p and 0 strongly converge for any $s \in \mathsf{Act}^*$, then the minimal sets $L^{\{a\}}_{p,[s]_D}$, $L^{\{b\}}_{p,[s]_D}$, $L^{\{a\}}_{0,[s]_D}$ and $L^{\{b\}}_{0,[s]_D}$ presented in the tables below are sufficient for proving our claim.

$[s]_D$	p after $[s]_D$	$L^{\{a\}}_{p,[s]_D}$	$L^{\{b\}}_{p,[s]_D}$
ϵ	p, a, b	–	–
a	$\{0\}$	–	–
b	$\{0\}$	–	–
other	$\emptyset$	$\emptyset$	$\emptyset$

$[s]_D$	0 after $[s]_D$	$L^{\{a\}}_{0,[s]_D}$	$L^{\{b\}}_{0,[s]_D}$
ϵ	0	–	–
a	$\emptyset$	$\emptyset$	$\emptyset$
b	$\emptyset$	$\emptyset$	$\emptyset$
other	$\emptyset$	$\emptyset$	$\emptyset$

Note that differently from the classical must preorder, the uncoordinated preorder does not consider the must-set $\{a, b\}$ to distinguish p from 0 because this set involves channels in different parts of the interface. The key point here is that each internal reduction of p is observed just by one part of the interface: the choice of branch a is only observed by one client and the choice of b is observed by the other one. Since uncoordinated observers do not intercommunicate, they can only report success simultaneously if they can do it independently from the interactions with the tested process, but such observers are exactly the ones that 0 can pass.

Like in the classical must preorder, we have that $0 \not\sqsubseteq^{\mathbb{I}}_{\mathsf{unc}} \tau.a + \tau.b$. This is witnessed by the observer $o = \overline{a}.\mathbf{0} + \tau.\mathbf{1} \parallel \mathbf{1}$ that is passed by 0 but not by $\tau.a + \tau.b$. ◀

The second example shows that the uncoordinated preorder falls somehow short with respect to the target we set in the introduction of allowing servers to swap actions that are targeted to different clients.

Conference title on which this volume is based on.
Editors: Billy Editor and Bill Editors; pp. 8–13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

▶ **Example 17.** Consider the interface $\mathbb{I} = \{\{a\}, \{b\}\}$ and the two pairs of processes

- $a.b + a + b$ and $b.a + a + b$
- $a.b$ and $b.a$

By inspecting traces and must-sets in the two tables below, where we use p and q to denote $a.b + a + b$ and $b.a + a + b$

$[s]_D$	p after $[s]_D$	$L^{\{a\}}_{p,[s]_D}$	$L^{\{b\}}_{p,[s]_D}$
ϵ	$\{p\}$	$\{a\}$	$\{b\}$
a	$\{b, 0\}$	–	–
b	$\{0\}$	–	–
ab	$\{0\}$	–	–
other	$\emptyset$	$\emptyset$	$\emptyset$

$[s]_D$	q after $[s]_D$	$L^{\{a\}}_{q,[s]_D}$	$L^{\{b\}}_{q,[s]_D}$
ϵ	$\{p\}$	$\{a\}$	$\{b\}$
a	$\{0\}$	–	–
b	$\{a, 0\}$	–	–
ab	$\{0\}$	–	–
other	$\emptyset$	$\emptyset$	$\emptyset$

It is easy to see that

$$a.b + a + b \approx^{\mathbb{I}}_{\mathsf{unc}} b.a + a + b$$

However, by using $o = \overline{a}.\mathbf{1} \parallel \mathbf{1}$ and $o' = \mathbf{1} \parallel \overline{b}.\mathbf{1}$ as observers, it can be shown that

$$a.b \not\approx^{\mathbb{I}}_{\mathsf{unc}} b.a \qquad \text{and} \qquad b.a \not\approx^{\mathbb{I}}_{\mathsf{unc}} a.b$$

Note that $o = \overline{a}.\mathbf{1} \parallel \mathbf{1}$ actually interacts with the process under test by using just one part of the interface and relies on the fact that the remaining part of the interface stays idle. Thanks to this ability, uncoordinated observers have still a limited power to track some dependencies among actions on different parts of the interface.

The preorder presented in the next section limits further the discriminating power of observers and allows us to equate processes $a.b$ and $b.a$. ◀

4 A testing preorder with independent observers

In this section we explore a notion of equivalence equating processes that can freely permute actions over different parts of their interface. As for the uncoordinated observers, the targeted scenario is that of a service with a partitioned interface interacting with two or more independent partners by using separate sets of ports. In addition, each component of an observer cannot exploit any knowledge about the design choices made by the other components, i.e., each of them has a local view of the behaviour of the process that ignores all actions controlled by the remaining components. Local views are characterised in terms of a projection operator defined as follows.

▶ **Definition 18** (Projection)**.** Let $V \subseteq \mathcal{N}$ be a set of observable ports. We write $p \upharpoonright V$ for the process obtained by hiding all actions of p over channels that are not in V. Formally,

$$\frac{p \xrightarrow{\alpha} p' \quad \alpha \in V \cup \overline{V}}{p \upharpoonright V \xrightarrow{\alpha} p' \upharpoonright V} \qquad \frac{p \xrightarrow{\alpha} p' \quad \alpha \notin V \cup \overline{V}}{p \upharpoonright V \xrightarrow{\tau} p' \upharpoonright V}$$

◀

▶ **Definition 19** (Independent (must) preorder $\sqsubseteq^{\mathbb{I}}_{\mathsf{ind}}$)**.** Let $\mathbb{I} = \{I_i\}_{i \in 0..n}$ be an interface. We say $p \sqsubseteq^{\mathbb{I}}_{\mathsf{ind}} q$ iff for all $\Pi_{i \in 0..n} o_i$ such that $\mathfrak{n}(o_i) \subseteq I_i$, $p \upharpoonright I_i$ must o_i implies $q \upharpoonright I_i$ must o_i. ◀

Note that $a.b$ and $b.a$ cannot be distinguished anymore by the observer $o = \overline{a}.\mathbf{1} \parallel \mathbf{1}$ used in the previous section to prove $a.b \not\approx^{\{\{a\},\{b\}\}}_{\mathsf{unc}} b.a$ (Example 17), because $a.b \upharpoonright \{a\}$ must $\overline{a}.\mathbf{1}$, $b.a \upharpoonright \{a\}$ must $\overline{a}.\mathbf{1}$, $a.b \upharpoonright \{b\}$ must $\mathbf{1}$ and $b.a \upharpoonright \{b\}$ must $\mathbf{1}$. Indeed, later (Example 24) we will see that:

$$a.b \approx^{\{\{a\},\{b\}\}}_{\mathsf{ind}} b.a$$

Conference title on which this volume is based on.
Editors: Billy Editor and Bill Editors; pp. 9–13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

4.1 Semantic characterisation

In this section we address the characterisation of the independent preorder in terms of traces. We start by introducing an equivalence notion of traces that ignores hidden actions.

▶ **Definition 20** (Filtered traces). Let $I \subseteq \mathsf{Act}$. Two strings $s_1, s_2 \in \mathsf{Act}^*$ are *equivalent up-to* I, written $s_1 \stackrel{\bullet}{\equiv}_I s_2$, if there exist $s_1', s_2' \in (\mathsf{Act} \setminus I)^*$ s.t. $s_1 s_1' \equiv_D s_2 s_2'$ where D is the dependency relation induced by $\{I, \mathsf{Act} \setminus I\}$. We write $[[s]]_I$ for the equivalence class of s. ◀

Basically, two traces are equivalent up-to I when they coincide after the removal of hidden actions. Note that the set $s' \in [[s]]_I \cap I^*$ has a unique element, which is the string obtained by removing from s all actions that are not in I. We write $s \restriction I$ to denote that element. As for the distributed preorder, we extend the notions of reduction, convergence and residuals to equivalence classes of strings.

- $q \stackrel{[[s]]_I}{\Longrightarrow} q'$ if and only if $\exists t \in [[s]]_I$ such that $q \stackrel{t}{\Longrightarrow} q'$
- $p \Downarrow [[s]]_I$ if and only if $\forall t \in [[s]]_I . p \Downarrow t$
- $(p \text{ after } [[s]]_I) = \{p' \mid p \stackrel{[[s]]_I}{\Longrightarrow} p'\}$

The following auxiliary result establishes properties relating reductions, hiding and filtered traces, which will be useful in the proof of the correspondence theorem.

▶ **Lemma 21.**
1. *$p \stackrel{s}{\Longrightarrow} p'$ implies $p \restriction I \stackrel{s\restriction I}{\Longrightarrow} p' \restriction I$.*
2. *$p \restriction I \stackrel{s}{\Longrightarrow} p' \restriction I$ implies $\exists t \in [[s]]_I$ and $p \stackrel{t}{\Longrightarrow} p'$.*
3. *$p \Uparrow [[s]]_I$ implies $p \restriction I \Uparrow s \restriction I$.*
4. *$(p \text{ after } [[s]]_I)$ MUST L iff $(p \restriction I \text{ after } s \restriction I)$ MUST $L \cap I$.*

The alternative characterisation for the independent preorder is given in terms of filtered traces.

▶ **Definition 22.** Let $p \preceq^{\mathbb{I}}_{\mathsf{ind}} q$ if for every $I \in \mathbb{I}$, for every $s \in I^*$, and for all finite $L \subseteq I$, if $p \Downarrow [[s]]_I$ then
1. $q \Downarrow [[s]]_I$
2. $(p \text{ after } [[s]]_I)$ MUST $L \cup (\mathsf{Act} \setminus I)$ implies $(q \text{ after } [[s]]_I)$ MUST $L \cup (\mathsf{Act} \setminus I)$

We would like to draw attention to condition 2 above; it only considers must-sets that always include all the actions in $(\mathsf{Act} \setminus I)$ to avoid the possibility of distinguishing reachable states because of actions that are not in I. Consider that this condition could be formulated as follows: for all finite $L \subseteq \mathsf{Act}$,

$$(p \text{ after } [[s]]_I) \text{ MUST } L \text{ implies } \exists L' \text{ s.t } (q \text{ after } [[s]]_I) \text{ MUST } L' \text{ and } L \cap I = L' \cap I$$

that makes evident that only the actions from observable part of the interface are relevant.

▶ **Theorem 23.** $\sqsubseteq^{\mathbb{I}}_{\mathsf{ind}} = \preceq^{\mathbb{I}}_{\mathsf{ind}}$.

▶ **Example 24.** Consider the processes $p = a.b$ and $q = b.a$ and the interface $\mathbb{I} = \{\{a\}, \{b\}\}$. The table below shows the analysis for the part of the interface $\{a\}$.

$[[s]]_{\{a\}}$	p after $[[s]]_{\{a\}}$	$L^{\{a\}}_{p,[[s]]_I}$	q after $[[s]]_{\{a\}}$	$L^{\{a\}}_{q,[[s]]_I}$
ϵ	$\{p\}$	$\{a\}$	$\{q, a\}$	$\{a\}$
a	$\{0, b\}$	$-$	$\{0\}$	$-$
other	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Conference title on which this volume is based on.
Editors: Billy Editor and Bill Editors; pp. 10–13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

When analysing the sets $(p \text{ after } [[\epsilon]]_{\{a\}}) = \{p\}$ and $(q \text{ after } [[\epsilon]]_{\{a\}}) = \{q, a\}$, we ignore the fact that q starts with a hidden action b; the only relevant residuals are those performing a. With a similar analysis we conclude that the condition on must-sets also holds for set $\{b\}$. Hence, $a.b \approx^{\mathbb{I}}_{\mathsf{ind}} b.a$ holds. ◀

The following example illustrates also the fact that independent observers are unable to track causal dependencies between choices made in different parts of the interface.

▶ **Example 25.** Let $p_1 = a.c + b.d$ and $p_2 = a.d + b.c$ be two alternative implementations for a service with interface $\mathbb{I} = \{\{a, b\}, \{c, d\}\}$. These two implementations are distinguished by the uncoordinated preorder ($p_1 \not\approx^{\{\{a,b\},\{c,d\}\}}_{\mathsf{unc}} p_2$) because of the observers $o_1 = \overline{a}.\mathbf{1} \parallel \overline{c}.\mathbf{1}$ ($p_1 \not\sqsubseteq^{\{\{a,b\},\{c,d\}\}}_{\mathsf{unc}} p_2$) and $o_2 = \overline{b}.\mathbf{1} \parallel \overline{c}.\mathbf{1}$ ($p_2 \not\sqsubseteq^{\{\{a,b\},\{c,d\}\}}_{\mathsf{unc}} p_1$).

They are instead equated by the independent preorder, $p_1 \approx^{\mathbb{I}}_{\mathsf{ind}} p_2$, Indeed, if only the part of the interface $\{a, b\}$ is of interest, we have that p_1 and p_2 are equivalent because they exhibit the same interactions over channels a and b. Similarly, without any a priori knowledge of the choices made for $\{a, b\}$, the behaviour observed over $\{c, d\}$ can be described by the non-deterministic choice $\tau.c + \tau.d$, and hence, p_1 and p_2 are indistinguishable also over $\{c, d\}$.

We use the alternative characterisation to prove our claim. As usual, $p_1 \Downarrow s$ and $p_2 \Downarrow s$ for any s. The tables below show coincidence of the must-sets. We would only like to remark that $ac \in [[a]]_{\{a,b\}}$ and, consequently, p_1 after $[[a]]_{\{a,b\}}$ contains also process 0.

$[[s]]_{\{a,b\}}$	p_1 after $[[s]]_{\{a,b\}}$	$L^{\{a,b\}}_{p_1,[[s]]_I}$	p_2 after $[[s]]_{\{a,b\}}$	$L^{\{a,b\}}_{p_2,[[s]]_I}$
ϵ	p_1	$\{a, b\}$	p_2	$\{a, b\}$
a	$\{c, 0\}$	–	$\{d, 0\}$	–
b	$\{d, 0\}$	–	$\{c, 0\}$	–
other	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

$[[s]]_{\{c,d\}}$	p_1 after $[[s]]_{\{a,b\}}$	$L^{\{c,d\}}_{p_1,[[s]]_I}$	p_2 after $[[s]]_{\{a,b\}}$	$L^{\{c,d\}}_{p_2,[[s]]_I}$
ϵ	p_1	$\{c, d\}$	p_2	$\{c, d\}$
c	$\{0\}$	–	$\{0\}$	–
d	$\{0\}$	–	$\{0\}$	–
other	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

◀

5 Relation between must, uncoordinated and independent preorders

In this section, we formally study the relationships between the classical must preorder and the two preorders we have introduced. We start by showing that a refinement of an interface induces a coarser preorder, e.g., splitting the observation among more uncoordinated observers decreases the discriminating power of the tests. We say that an interface $\mathbb{I}'$ is a *refinement* of another interface $\mathbb{I}$ when the partition $\mathbb{I}'$ is finer than the partition $\mathbb{I}$.

▶ **Lemma 26.** *Let $\mathbb{I}$ be an interface and $\mathbb{I}'$ a refinement of $\mathbb{I}$. Then, $p \sqsubseteq^{\mathbb{I}}_{\mathsf{unc}} q$ implies $p \sqsubseteq^{\mathbb{I}'}_{\mathsf{unc}} q$.*

This result allows us to conclude that the uncoordinated preorder is coarser than the classical must testing preorder. It suffices to note that the preorder associated to the maximal element of the partition lattice, i.e., the trivial partition $\mathbb{I} = \{\mathsf{Act}\}$, corresponds to $\sqsubseteq_{\mathsf{must}}$.

▶ **Proposition 27.** *Let $\mathbb{I}$ be an interface. Then, $p \sqsubseteq_{\mathsf{must}} q$ implies $p \sqsubseteq^{\mathbb{I}}_{\mathsf{unc}} q$.*

Conference title on which this volume is based on.
Editors: Billy Editor and Bill Editors; pp. 11–13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The converse of Lemma 26 and Proposition 27 do not hold. Consider the processes $p = a.b + a + b$ and $q = b.a + a + b$. It has been shown, in Example 17, that we have $p \sqsubseteq_{\mathsf{unc}}^{\{\{a\},\{b\}\}} q$. Nonetheless, it is easy to check that $p \not\sqsubseteq_{\mathsf{must}} q$ (i.e., $p \sqsubseteq_{\mathsf{unc}}^{\{\mathsf{Act}\}} q$) by using $o = \overline{b}.(\tau.\mathbf{1} + \overline{a}.\mathbf{0})$ as observer.

We also have that the independent preorder is coarser than the uncoordinated one.

▶ **Proposition 28.** *Let $\mathbb{I}$ be an interface. Then, $p \sqsubseteq_{\mathsf{unc}}^{\mathbb{I}} q$ implies $p \sqsubseteq_{\mathsf{ind}}^{\mathbb{I}} q$.*

The converse does not hold, i.e., $p \sqsubseteq_{\mathsf{ind}}^{\mathbb{I}} q$ does not imply $q \sqsubseteq_{\mathsf{unc}}^{\mathbb{I}} p$. Indeed, we have that $a.b \sqsubseteq_{\mathsf{ind}}^{\{\{a\},\{b\}\}} b.a$ (Example 24) but $a.b \not\sqsubseteq_{\mathsf{unc}}^{\{\{a\},\{b\}\}} b.a$ (Example 17).

6 Conclusions and related works

In this paper we have explored different relaxations of the must testing preorder tailored to define new behavioural relations that, in the framework of Service Oriented Computing, are better suited to study compliance between contracts exposed by clients and servers interacting via synchronous binary communication primitives.

In particular, we have considered two different scenarios in which contexts of a service are represented by processes with distributed control. The first variant, that we called uncoordinated preorder, corresponds to multiparty contexts without runtime communication between peers but with the possibility of one peer blocking another because it does not perform the expected action. Indeed, the observations at the basis of our experiments are designed with the assumption that users of a service interact only via dedicated ports but might be influenced by the fact that other partners do not perform expected actions. The second preorder we introduced is called independent preorder. It accounts for partners that are completely independent from the behaviour of the other ones. Indeed, from a viewpoint of a client, actions by other clients are considered as unobservable actions.

We have shown that the discriminating power of the induced equivalences decreases as observers become weaker; and thus that the independent preorder is coarser than the uncoordinated preorder which in turn is coarser than the classical testing preorder. As future work we plan to consider different "real life" scenarios and to assess the impact of the different assumptions at the basis of the two new preorders and the identifications/orderings they induce. We plan also to perform further studies to get a fuller account, possibly via axiomatizations, of their discriminating power. In the near future, we will also consider the impact of our testing framework on calculi based on asynchronous interactions.

Several variants of the must testing preorder, contract compliance and sub-contract relation have been developed in the literature to deal with different aspects of services compositions, such as buffered asynchronous communication [4, 14, 13], fairness [15], peer-interaction [2], among others. We remark that these approaches deal with aspects that are orthogonal to the discriminating power of the distributed tests analysed in this work. Our preorders have some similarities with those relying on buffered communications in that both aim at guaranteeing that actions performed by independent peers can be reordered. Nevertheless, our work considers a model with synchronous communication and, hence, message reordering is not obtained by swapping buffered messages. As mentioned above, we have left the study of distributed tests under asynchronous communication as a future work. However, we would like to remark that, even the uncoordinated and the independent preorders are different from those in [4, 14, 13] that permit explicit action reordering. The paradigmatic example is the equivalence $a.c + b.d \approx_{\mathsf{ind}}^{\{a,b\},\{c,d\}} a.d + b.c$, which does not hold for any of the preorders with buffered communication. The main reason is that, even in presence of buffered communication, the causality, e.g., between a and c is always observed.

Conference title on which this volume is based on.
Editors: Billy Editor and Bill Editors; pp. 12–13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Acknowledgments. We thank the anonymous reviewers of CONCUR and TGC 2015 for their careful reading of our manuscript and their many insightful comments and suggestions.

References

1 Samik Basu, Tevfik Bultan, and Meriem Ouederni. Deciding choreography realizability. In *ACM SIGPLAN Notices*, volume 47, pages 191–202. ACM, 2012.

2 Giovanni Bernardi and Matthew Hennessy. Mutually testing processes. In *CONCUR 2013–Concurrency Theory*, pages 61–75. Springer, 2013.

3 Mario Bravetti and Gianluigi Zavattaro. Towards a unifying theory for choreography conformance and contract compliance. In *SC*, volume 4829 of *Lect. Notes in Comput. Sci.*, pages 34–50. Springer Verlag, 2007.

4 Mario Bravetti and Gianluigi Zavattaro. A foundational theory of contracts for multi-party service composition. *Fundamenta Informaticae*, 89(4):451–478, 2008.

5 Giuseppe Castagna, Nils Gesbert, and Luca Padovani. A theory of contracts for web services. In *POPL*, pages 261–272, 2008.

6 Giuseppe Castagna, Nils Gesbert, and Luca Padovani. A theory of contracts for web services. *ACM Trans. Program. Lang. Syst.*, 31(5), 2009.

7 Rocco De Nicola and Matthew Hennessy. Testing equivalences for processes. *Theor. Comput. Sci.*, 34:83–133, 1984.

8 Kohei Honda, Vasco Thudichum Vasconcelos, and Makoto Kubo. Language primitives and type discipline for structured communication-based programming. In *ESOP*, volume 1381 of *Lect. Notes in Comput. Sci.*, pages 122–138. Springer Verlag, 1998.

9 Cosimo Laneve and Luca Padovani. The Must preorder revisited. In *CONCUR*, volume 4703 of *Lect. Notes in Comput. Sci.*, pages 212–225, 2007.

10 Antoni W. Mazurkiewicz. Trace theory. In Wilfried Brauer, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Advances in Petri Nets*, volume 255 of *Lecture Notes in Computer Science*, pages 279–324. Springer, 1986.

11 Antoni W. Mazurkiewicz. Introduction to trace theory. *The Book of Traces*, pages 3–41, 1995.

12 R. Milner. *Communication and Concurrency*. Prentice Hall International, 1989.

13 Dimitris Mostrous, Nobuko Yoshida, and Kohei Honda. Global principal typing in partially commutative asynchronous sessions. In *Programming Languages and Systems*, pages 316–332. Springer, 2009.

14 Luca Padovani. Contract-based discovery of web services modulo simple orchestrators. *Theoretical Computer Science*, 411(37):3328–3347, 2010.

15 Luca Padovani. Fair subtyping for multi-party session types. In *Coordination Models and Languages*, pages 127–141. Springer, 2011.

16 Kaku Takeuchi, Kohei Honda, and Makoto Kubo. An interaction-based language and its typing system. In *PARLE*, volume 817 of *Lect. Notes in Comput. Sci.*, pages 398–413. Springer Verlag, 1994.

Conference title on which this volume is based on.
Editors: Billy Editor and Bill Editors; pp. 13–13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany