The Journal of Logic and Algebraic Programming 79 (2010) 233–244

Contents lists available at ScienceDirect

The Journal of Logic and Algebraic Programming

journal homepage: www.elsevier.com/locate/jlap

Relating fair testing and accordance for service replaceability

Arjan J. Mooij [a,*], Christian Stahl [a,b], Marc Voorhoeve [a]

[a] Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, P.O. Box 513, 5600 MB Eindhoven, The Netherlands
[b] Institut für Informatik, Humboldt-Universität zu Berlin, Unter den Linden 6, 10099 Berlin, Germany

ARTICLE INFO

Article history:
Received 29 June 2009
Accepted 14 December 2009
Available online 14 January 2010

Keywords:
Formal methods
Behavioral equivalences
Refinement
Service-oriented computing

ABSTRACT

The accordance pre-order describes whether a service can safely be replaced by another service. That is, all partners for the original service should be partners for the new service. Partners for a service interact with the service in such a way that always a certain common goal can be reached.

We relate the accordance pre-order to the pre-orders known from the linear–branching time spectrum, notably fair testing. The differences between accordance and fair testing include the modeling of termination and success, and the parts of the services that cannot be used reliably by any partner. Apart from the theoretical results, we address the practical relevance of the introduced concepts.

1. Introduction

Service-oriented computing (SOC) aims at building systems by using widely available and independently-developed building blocks called *services*. Services sometimes need to be replaced, e.g., when new features have been implemented, or bugs have been fixed. Service replaceability is the problem of determining which services can replace each other without endangering any current use.

The accordance pre-order [18,19] indicates whether a service y can be replaced by a service x without affecting any services that use y. Service x is said to *accord with* y if every partner for y is a partner for x. A *partner* for a service x is a service z such that the composition of x and z is correct. Several correctness notions can be considered, but we focus on weak termination. Weak termination denotes that a final state is always reachable, and hence it excludes deadlocks and non-terminating loops; in the temporal logic CTL (Computation Tree Logic, [8]) it can be expressed using the "AG EF" pattern.

In this paper we study the relation between accordance and the pre-orders from the linear-branching time spectrum [20,21]. In Fig. 1, some of these pre-orders and the relations between them are depicted, featuring B (strong bisimilarity), RS (ready simulation), PF (possible futures), FT (fair testing [4,15,17]), MT (must testing [10]), CT (completed trace), and T (trace) pre-orders. An arrow denotes the inclusion relation, so, e.g., B implies (is finer than) PF; if an arrow is absent (in the transitive closure), then the inclusion does not hold.

As noticed by [2,13], fair testing implies accordance (called conflict pre-order in [13], and sub-contract in [2]), but accordance does not imply fair testing. In [2] a counter-example is given, whereas in [13] it is also shown that accordance implies a new variant of failures pre-order. This last result, however, gives no conditions under which accordance coincides with a single traditional pre-order.

In this paper we show that fair testing is the coarsest traditional pre-order that implies accordance. Moreover, we take the extra step to analyze under which conditions accordance would imply fair testing. In this way, we identify the real differences between accordance and fair testing.

* Corresponding author. Tel.: +31 40 2472733; fax: +31 40 2463992.
E-mail addresses: A.J.Mooij@tue.nl (A.J. Mooij), C.Stahl@tue.nl (C. Stahl), M.Voorhoeve@tue.nl (M. Voorhoeve).

1567-8326/$ - see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.jlap.2009.12.001

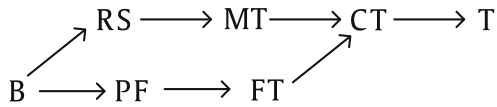


Fig. 1. Some traditional pre-orders.

A first difference is the modeling of *success* and *termination*. In case of accordance, the composed services must terminate together, which we model using a synchronized action ✓. In case of fair testing, only the test needs to perform a "success" action, say ⊛. In general, a test that synchronizes on the termination action ✓ is not guaranteed to (eventually) perform the success action ⊛, nor vice versa.

Another major difference is that accordance gives no information about the *uncoverable* states, which are states that cannot be reached with any partner. For example, a service that receives a request and then deadlocks is accordance equivalent to a service that deadlocks right away; in both cases there is no partner that can reliably use (this part of) the service.

This aspect is of interest within the context of software engineering. A service that contains bugs can be used as long as these unreliable parts are avoided; this phenomenon is witnessed all too often. A new version where some of the bugs are fixed should accord with the old version, but the two versions may be unrelated in the trace pre-order, let alone any finer pre-orders.

In this paper we define a novel variant of fair testing that turns out to be equivalent to accordance. Moreover, we show how restrictions on the tests in this variant can be transformed into restrictions on the services being tested. Finally we discuss the practicality of these restrictions, in particular within the realm of services that communicate asynchronously. Nevertheless, the main theory is developed in terms of synchronous communication.

Related work. In the literature several pre-orders are described that are related to accordance. The IR-equivalence from [23] is used for the replaceability of Petri nets, but it is an asymmetric notion, i.e, it focuses only on the tests and not on the service being tested. In [23] it has been proved that IR-equivalence coincides with fair testing.

The sub-contract pre-order from [7,12] is also asymmetric, but it only requires that the test never gets stuck. In [12] it has been proved that this sub-contract relation coincides with must-testing. Must testing is known to be incomparable to fair testing, as depicted in Fig. 1.

Stuck-free conformance from [11] only excludes deadlocks. In [11] it has been proved that CSP stable-failure refinement (which, for finitely branching processes without divergences, is equivalent to must-testing; see [9]) does not imply stuck-free conformance, and stuck-free conformance is strictly larger than the refusal pre-order [16].

Overview. In Section 2 we provide some basic definitions. In Section 3 we compare accordance and fair testing without any restrictions. In Section 4 we introduce some restrictions on the tests used for fair testing, and prove that this kind of fair testing is equivalent to accordance. In Section 5 we replace a restriction on the tests by some restrictions on the services being tested. Finally, in Section 6 we discuss some practical aspects, and in Section 7 we summarize the obtained results.

2. Preliminaries

In this section we define the most important concepts that we will use.

2.1. Processes

Let A be a set of *actions* containing three special actions: the silent action τ, the termination action ✓, and the success action ⊛. Processes are represented as non-empty (but possibly infinite) *tree-shaped* labeled transition systems; for every reachable state, there is exactly one path from the initial state. A *branch* of a tree is an action in the tree followed by the remainder of the tree.

For any two states p and p', we write $p \xrightarrow{a} p'$ if an a-labeled edge exists from p to p'. We write $p \xrightarrow{a}$ if there exists a p' such that $p \xrightarrow{a} p'$. If B is a set, then B^* consists of the lists of elements from B; ϵ denotes the empty list, and lists are concatenated by juxtaposition. For $w \in A^*$, $\stackrel{w}{\Longrightarrow}$ is the least relation satisfying:

- $p \stackrel{\epsilon}{\Longrightarrow} p$;
- $p \stackrel{w}{\Longrightarrow} p' \wedge p' \xrightarrow{a} q \Rightarrow p \stackrel{w\,a}{\Longrightarrow} q$, for any action $a \in A\backslash\{\tau\}$;
- $p \stackrel{w}{\Longrightarrow} p' \wedge p' \xrightarrow{\tau} q \Rightarrow p \stackrel{w}{\Longrightarrow} q$.

We write $p \stackrel{w}{\Longrightarrow}$ if there exists a p' such that $p \stackrel{w}{\Longrightarrow} p'$, and we write $p \Longrightarrow p'$ if there exists a w such that $p \stackrel{w}{\Longrightarrow} p'$.

We build example processes in ACP-style [1] with the action prefix operators "$a \cdot$", for each action $a \in A$, and infix choice operator "+" (which is commutative and associative). The "deadlock" process δ represents a state without outgoing edges,

and for each action $a \in A$, we abbreviate the process $a \cdot \delta$ by a. The merge operator $\|_H$ on two processes denotes composition by synchronizing the actions from the set H and interleaving the other actions; the set H should contain the action $\surd$, and should not contain the actions $\circledast$ and τ. We also use the renaming operator ρ_f, where f is a function on A such that $f(\tau) = \tau$. The operational rules are as follows:

$$\frac{p \xrightarrow{b} p',\ b \in A\backslash H}{p \|_H q \xrightarrow{b} p' \|_H q}, \qquad \frac{q \xrightarrow{b} q',\ b \in A\backslash H}{p \|_H q \xrightarrow{b} p \|_H q'}, \qquad \frac{p \xrightarrow{a} p',\ q \xrightarrow{a} q',\ a \in H}{p \|_H q \xrightarrow{a} p' \|_H q'}, \qquad \frac{p \xrightarrow{a} p'}{\rho_f(p) \xrightarrow{f(a)} \rho_f(p')}.$$

For conciseness, in the merge operator $\|_H$ we often leave the set H of synchronized actions implicit. Given the operational rules, we can use $p\|q \stackrel{w}{\Longrightarrow} p'\|q'$ to denote that there exist u and v such that $p \stackrel{u}{\Rightarrow} p'$, $q \stackrel{v}{\Rightarrow} q'$, and w can be obtained from u and v by synchronizing the actions from H and interleaving the other actions.

Definition 1 (*Visited state*)**.** Let x and y be processes. Process x can *visit* a state r of process y iff there exists a state q of process x such that $x\|y \Rightarrow q\|r$.

We introduce the abbreviation $p \rightsquigarrow a$, for any process p and any action a, to specify a class of properties related to the testable properties from [22].

Definition 2 (*Leads to*)**.** Let p be a process, and let a be an action. We use $p \rightsquigarrow a$ to denote that in every execution of process p, as long as action a has not occurred, action a can occur in the future, i.e.,

$$(\forall w, q : w \in (A\backslash\{a\})^* : p \stackrel{w}{\Longrightarrow} q \Rightarrow (\exists v : v \in A^* : q \stackrel{v\,a}{\Longrightarrow})).$$

2.2. Accordance

The accordance pre-order indicates whether some process y can be replaced by some other process x, without endangering the interaction with any partner for y. To formalize this, we first define a partner for a process y as a process z such that their composition $y\|z$ has a certain behavioral property.

In this paper we consider the weak termination property, i.e., a final state is always reachable. We use the action $\surd$ to mark the final states; it is a synchronized action in order to deal with the final states in a composed process.

Definition 3 (*Partner*)**.** A process z is a *partner* for a process x iff the composition of x and z can always reach a final state, i.e., $x\|z \rightsquigarrow \surd$.

Definition 4 (*Accordance*)**.** Two processes x and y are related in the *accordance pre-order*, i.e., $x \sqsubseteq_{acc} y$, iff each partner for y is a partner for x. Two processes x and y are accordance equivalent, iff $x \sqsubseteq_{acc} y$ and $y \sqsubseteq_{acc} x$.

Note that the roles of x and y follow the convention from [14]. Though we could have swapped them, the current convention seems to be more natural when generalizing accordance to sets of processes; see [14].

This kind of accordance differs from the one in [18], which considers deadlock-freedom instead of weak termination. The accordance notion from [19] addresses weak termination, but it is restricted to finite and acyclic processes (and hence deadlock freedom coincides with weak termination). Equivalent pre-orders are the conflict pre-order from [13], and the sub-contract pre-order from [2].

As an example, suppose a and b are synchronized actions. A process becomes smaller in the accordance pre-order by reducing some of its internal choices, e.g., $\tau \cdot a \cdot \surd \ \sqsubseteq_{acc}\ \tau \cdot a \cdot \surd + \tau \cdot b \cdot \surd$. In contrast to what was expected in [7], and to some cases in Section 6.1, a process does not become smaller by increasing some of its external choices, e.g., $a \cdot \surd + b \cdot \surd$ and $a \cdot \surd$ are unrelated (consider the potential partners $b \cdot \surd$ and $a \cdot \surd + b$). Nevertheless, both processes are smaller than $a \cdot \surd + b$, where the branch b (or $b \cdot \delta$) denotes that no partner may synchronize on b in this state.

So far there is no algorithm to decide accordance, but there is some ongoing work in this direction for finite-state processes. The main issue is to compare two possibly-infinite sets of partners. For a weaker notion of partner, accordance can be decided using a finite representation of the sets of partners [18]. In [25], we developed a related representation for the current notion of partner, whereas an algorithm is still work in progress.

For some general properties of accordance-like pre-orders we refer to [14]. In particular, accordance is a pre-congruence with respect to the associative operator $\|_H$. However, accordance is not a congruence with respect to operator $+$; e.g., $a \ \sqsubseteq_{acc}\ \delta$ and $b \cdot \surd \ \sqsubseteq_{acc}\ b \cdot \surd$, but $a + b \cdot \surd \ \not\sqsubseteq_{acc}\ \delta + b \cdot \surd$ (consider the potential partner $a + b \cdot \surd$).

For some processes there exist no partners; such processes are called *uncontrollable* [24]. The simplest example of an uncontrollable process is δ.

Definition 5 (*Controllable process*)**.** A process x is *controllable* iff there exists a partner for x.

2.3. Fair testing

To formalize the fair testing pre-order [4,15,17], we first define the notion of a successful test. We use the action $\circledast$ to model success of a test; it is an unsynchronized action as it should not occur in the process being tested.

Definition 6 (*Successful test*)**.** Let x be a process that does not contain the action $\circledast$. A process t is a *successful test* on x iff the composition of x and t can always reach a state where $\circledast$ can be performed, i.e., $x \| t \rightsquigarrow \circledast$.

Definition 7 (*Fair testing*)**.** Let x and y be two processes that do not contain the action $\circledast$. Processes x and y are related in the *fair testing pre-order*, i.e., $x \sqsubseteq_{ft} y$, iff each successful test on y is a successful test on x.

Fair testing is known to imply the (completed) trace pre-order. The other way around, processes $x = a \cdot \checkmark$ and $y = a + a \cdot \checkmark$ are trace equivalent, but in terms of fair testing we only have $x \sqsubseteq_{ft} y$. This is also depicted in Fig. 1. An algorithm to decide fair testing for finite-state processes has been presented in [17].

Being a partner is a symmetric notion (z is a partner for x iff x is a partner for z), but being a successful test is not: given the restrictions on $\circledast$, if t is a successful test on x, then x is not a successful test on t. Although for some processes there exist no partners (in terms of accordance), for every process there exists a successful test. Moreover, some tests are successful on every process. The simplest example of such a trivial test is just $\circledast$.

Definition 8 (*Trivial test*)**.** A *trivial test* is a successful test on every process that does not contain the action $\circledast$.

3. Accordance versus fair testing

This section describes our first attempt to relate the accordance pre-order to some traditional pre-order, and in particular to fair testing. We prove that fair testing implies accordance, and show that, in general, accordance does not imply fair testing.

3.1. Fair testing implies accordance

To position accordance in the spectrum from Fig. 1, consider the processes $x = a^* a \cdot \checkmark$ and $y = a^* a \cdot \checkmark + a^* \delta$; the binary Kleene star $a^* p$, for any action a and process p, is the least fix-point X of the equation $X = a \cdot X + p$. Accordance distinguishes between x and y, as x has a partner, e.g., $a^* \checkmark$, whereas y is uncontrollable. On the other hand, ready simulation does not distinguish between x and y; see [22]. Thus we conclude that accordance is not implied by any of the pre-orders depicted at the top of Fig. 1.

In [13,2] it is shown that fair testing implies accordance. For completeness reasons, and for later use, we also provide a proof for it.

Theorem 1 (Fair testing implies accordance)**.** *For any two processes x and y that do not contain the action $\circledast$ holds*:

$$x \sqsubseteq_{ft} y \;\Rightarrow\; x \sqsubseteq_{acc} y.$$

Proof. Let x and y be processes that do not contain the action $\circledast$. Let $x \sqsubseteq_{ft} y$, i.e., each successful test on y is a successful test on x. To prove $x \sqsubseteq_{acc} y$, let z be a partner for y. What remains to be proved is that z is a partner for x.

As $\circledast$ is an unsynchronized action, we can reduce the case that partner z contains $\circledast$ to a partner that does not contain $\circledast$ (by renaming $\circledast$ to τ). In what follows, we consider the case that z does not contain the action $\circledast$.

We can complete the proof if there exists a test t such that for every process p (like x and y) that does not contain $\circledast$: $p \| z \rightsquigarrow \checkmark \;\Leftrightarrow\; p \| t \rightsquigarrow \circledast$. Such a process t can be constructed from the given partner z (see also [13,2], which contain the same observation) by replacing every occurrence of the action $\checkmark$ by the term $\checkmark \cdot \circledast$. The proof of $\Rightarrow$ uses that $\circledast$ is an unsynchronized action, and the proof of $\Leftarrow$ uses that p and z do not contain the action $\circledast$. □

As an example, the processes $x = a \cdot \checkmark + b \cdot \checkmark$ and $y = \tau \cdot a \cdot \checkmark + b \cdot \checkmark$ are related by $x \sqsubseteq_{ft} y$ and $x \sqsubseteq_{acc} y$. In the proof, the example partner $z = a \cdot \checkmark$ is replaced by the test $t = a \cdot \checkmark \cdot \circledast$.

3.2. Restricted testing

In the proof of Theorem 1 we have only used a limited number of successful $\sqsubseteq_{ft}$-tests, viz., tests where each $\checkmark$ is immediately followed by a trivial test. To make this explicit, we first define a restricted kind of fair testing.

Definition 9 (*Restricted test*)**.** A *restricted test* is a test in which each occurrence of action $\checkmark$ is immediately followed by a trivial test.

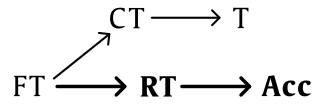


Fig. 2. Pre-orders related to accordance.

Definition 10 (*Restricted testing*)**.** Let x and y be two processes that do not contain the action $\circledast$. Processes x and y are related in the *restricted testing pre-order*, i.e., $x \sqsubseteq_{rt} y$, iff each restricted test that is a successful test on y is a successful test on x.

From the definition of restricted testing, we can immediately conclude its relation to fair testing.

Corollary 1 (Fair testing implies restricted testing)**.** *For any two processes x and y that do not contain the action $\circledast$ holds*: $x \sqsubseteq_{ft} y \Rightarrow x \sqsubseteq_{rt} y$.

Using these definitions, we can strengthen Theorem 1 based on its previously given proof.

Theorem 2 (Restricted testing implies accordance)**.** *For any two processes x and y that do not contain the action $\circledast$ holds*:

$$x \sqsubseteq_{rt} y \Rightarrow x \sqsubseteq_{acc} y.$$

3.3. Counter-examples

Accordance does not imply restricted testing, as the processes $x = a$ and $y = \delta$ indicate. As x and y are uncontrollable, $x \sqsubseteq_{acc} y$ holds. However, $t = a + \circledast$ is a restricted test that is a successful test on y, but it is not a successful test on x. A similar observation in terms of fair testing has been made in [2,13]. This example shows that accordance gives no information on uncontrollable processes, whereas fair testing does.

Restricted testing is unrelated to the (completed) trace pre-order. The processes $x = a \cdot \checkmark$ and $y = a \cdot \checkmark \cdot b$ are restricted testing equivalent, but not (completed) trace equivalent. Note that we treat the two termination actions $\checkmark$ and $\circledast$ as ordinary actions, and hence they are not necessarily the last actions of a process.

Similarly, the processes $x = a$ and $y = a + a \cdot \checkmark$ are restricted testing equivalent (in both cases, after an a, every successful test is always able to reach $\circledast$ as long as no synchronization occurs; moreover, after any synchronization on $\checkmark$ the test must be trivial), but not (completed) trace equivalent. This shows that restricted testing (and hence accordance) does not imply the (completed) trace pre-order, nor any finer pre-order like fair testing. Using our earlier observation that ready simulation does not imply accordance, we can conclude that the (completed) trace pre-order does not imply restricted testing.

These results indicate that fair testing is the coarsest traditional pre-order that implies accordance. In Fig. 2, our pre-orders RT (restricted testing) and Acc (accordance) are shown in relation to those in Fig. 1. In what follows, we study the differences between restricted testing and accordance.

4. A covering kind of fair testing

In this section we combine the observations from the previous section and define a kind of fair testing that is equivalent to accordance. As accordance gives no information on uncoverable states, the idea is to restrict the tests such that they can only visit states that can be visited by partners.

4.1. Covering tests

For every process, the set of states can be partitioned into the states that can be visited using any partner and the states that cannot. The states that can be visited using a partner are called coverable.

Definition 11 (*Coverable state*)**.** A state q of a process x is *coverable* iff there exists a partner z for x that can visit state q.

Definition 12 (*Covering process*)**.** A process y is a *covering* process for a process x iff process y can only visit coverable states of process x.

As an example, the process $x = a \cdot (b + b \cdot \checkmark + c \cdot \checkmark)$ has three uncoverable states: after the first b, after the second b, and after the first $\checkmark$. In particular the state after the second b is uncoverable, as no partner can use this b due to the first b. As illustrated by the state after the first $\checkmark$, whenever a process is in an uncoverable state, it cannot enter a coverable state anymore.

To compute the coverable states, we can use the notion of a most-permissive partner [24], which is a partner that can visit all the states that can be visited using any partner. So, the coverable states of a process are the states that can be visited by a most-permissive partner. An algorithm for computing a most-permissive partner of a finite-state process can be found in [24].

Every partner for a process x is a covering process for x. For every uncontrollable process, there exist no covering processes, because uncoverable processes have no coverable states, and every process has at least one state. In the accordance pre-order, the process and its partner should always be able to reach a final state before reaching an uncoverable state. A similar phenomenon occurs in the safe-must pre-order [6], where a process and its observer must reach a success state before reaching a catastrophic (i.e., diverging) one.

From a software-engineering perspective, it is also useful to distinguish between coverable and uncoverable states. For example, testing should concentrate on the coverable states, as the other states cannot be used reliably anyhow. Using these definitions, we can define a notion of covering restricted testing, such that the tests may only visit coverable states of the process being tested.

Definition 13 (*Covering restricted testing*)**.** Let x and y be two processes that do not contain the action $\circledast$. Processes x and y are related in the *covering restricted testing pre-order*, i.e., $x \sqsubseteq_{crt} y$, iff each restricted test that is a successful covering test on y is a successful covering test on x.

The covering restricted testing pre-order is a variant of fair testing, whereas others try to relate accordance to different traditional pre-orders, e.g., in [13] to a variant of failures pre-order. Although we restrict the considered set of tests, in [13] similar sets are extended. Moreover, in [13] it is not proved that accordance coincides with their variant of failures pre-order, whereas in Corollary 2 we will prove that accordance coincides with covering restricted testing.

4.2. Covering restricted testing implies accordance

As mentioned before, the successful $\sqsubseteq_{ft}$-tests that we have used in the proof of Theorem 1 are restricted tests. Moreover, the used tests turn out to be covering tests, as every partner is a covering process. To make this explicit, we again strengthen Theorem 1 based on its previously given proof.

Theorem 3 (Covering restricted testing implies accordance)**.** *For any two processes x and y that do not contain the action $\circledast$ holds*:

$$x \sqsubseteq_{crt} y \;\Rightarrow\; x \sqsubseteq_{acc} y.$$

4.3. Expansion

Before proving that accordance implies covering restricted testing, we first develop some theory to relate partners and covering tests. We introduce the notion of an expansion of a test, which corresponds to adding certain branches after occurrences of action $\circledast$.

Definition 14 (*Expansion*)**.** An *expansion* of a test t is a process that can be obtained from t by adding branches to states that occur after any action $\circledast$.

A covering test on process x cannot visit any uncoverable state of x. So, a successful covering test on x must be expandable into a partner for x. We prove the following two useful lemmata relating partners and covering tests.

Lemma 1. *Let x be a process that does not contain the action $\circledast$. Every successful covering test on x is expandable into a partner for x.*

Proof. Let x be a process that does not contain the action $\circledast$. Let t be a successful covering test on x. What remains to be proved is that there exists a partner z for x that is an expansion of t.

As t is a successful covering test on x, in the composition $x \| t$ always some state $q \| r$ is reachable, in which q is a covering state of x and r is a state of t, such that $\circledast$ has already occurred at least once. As q is a covering state of x, there exists a partner z' for x such that in the composition $x \| z'$ a state $q \| r'$ is reachable, in which r' denotes a state of z'. If before reaching any such state q the action $\checkmark$ has not occurred (recall that we consider processes as trees), then we can ensure that $\checkmark$ is reachable with z by adding to any such a state r of t a τ-labeled edge to such a state r'. $\square$

For the example process $x = a \cdot (b + b \cdot \checkmark + c \cdot \checkmark)$, the successful covering test $t = a \cdot \circledast$ can be expanded using the partner $z' = a \cdot c \cdot \checkmark$ into the partner $z = a \cdot \circledast \cdot \tau \cdot c \cdot \checkmark$.

Lemma 2. *Let x be a process that does not contain the action $\circledast$. Every restricted test that can be expanded into a partner for process x is a successful covering test on x.*

Proof. Let x be a process that does not contain the action $\circledast$. Let t be a restricted test, and let z be an expansion of t that is a partner for process x. What remains to be proved is that t is a successful covering test on x.

Partner z is a covering process for x, even if some branches of z are removed. So every test that can be expanded into z is a covering test on x.

To show that t is a successful test on x, we distinguish between two kinds of occurrences of $\checkmark$ in z. Each action $\checkmark$ that is also present in the restricted test t is directly followed by a trivial test. Each other action $\checkmark$ in z is in a branch that occurs after an action $\circledast$ in t. In both cases it holds that, as z is a partner for x, test t is a successful test on x. □

4.4. Accordance implies covering restricted testing

Using this theory on covering restricted testing, we can prove that accordance implies covering restricted testing.

Theorem 4 (Accordance implies covering restricted testing)**.** *For any two processes x and y that do not contain the action $\circledast$ holds*:

$$x \sqsubseteq_{acc} y \Rightarrow x \sqsubseteq_{crt} y.$$

Proof. Let x and y be processes that do not contain the action $\circledast$. Let $x \sqsubseteq_{acc} y$, i.e., each partner for y is a partner for x. To prove $x \sqsubseteq_{crt} y$, let t be a restricted test that is a successful covering test on y. What remains to be proved is that t is a successful covering test on x.

Using Lemma 1, there exists a partner z for y such that z is an expansion of t. Using $x \sqsubseteq_{acc} y$, z is also a partner for x. Using Lemma 2, restricted test t is a successful covering test on x. □

The example processes $x = a \cdot \checkmark + b \cdot \checkmark$ and $y = \tau \cdot a \cdot \checkmark + b \cdot \checkmark$ are related by $x \sqsubseteq_{acc} y$ and $x \sqsubseteq_{crt} y$. In the proof, the example test $t = a \cdot \circledast$ is replaced by the partner $z = a \cdot \circledast \cdot \tau \cdot \checkmark$.

From the two implications presented in Theorems 3 and 4 we immediately conclude that accordance is equivalent to covering restricted testing.

Corollary 2 (Accordance equals covering restricted testing)**.** *For any two processes x and y that do not contain the action $\circledast$ holds*: $x \sqsubseteq_{acc} y \Leftrightarrow x \sqsubseteq_{crt} y$.

5. Covering restricted testing versus restricted testing

In this section we study in which cases restricted testing and accordance coincide, or rather (based on Corollary 2), in which cases restricted testing and covering restricted testing coincide. Instead of restricting the considered tests, as we did in the previous section, we introduce two restrictions on the tested processes.

5.1. Restricted testing implies covering restricted testing

In Theorem 2 we proved that restricted testing implies accordance, and in Theorem 4 we proved that accordance implies covering restricted testing. Thus we conclude that restricted testing implies covering restricted testing.

Corollary 3 (Restricted testing implies covering restricted testing)**.** *For any two processes x and y that do not contain the action $\circledast$ holds*: $x \sqsubseteq_{rt} y \Rightarrow x \sqsubseteq_{crt} y$.

In the remainder of this section we focus on conditions under which we can prove, for any processes x and y, that $x \sqsubseteq_{crt} y \Rightarrow x \sqsubseteq_{rt} y$. In contrast to the pre-order $\sqsubseteq_{rt}$, the pre-order $\sqsubseteq_{crt}$ gives no information on uncontrollable processes. Therefore, we have to focus on controllable processes.

5.2. Pruned processes

In Section 4, covering tests were introduced to avoid distinguishing between different uncoverable states. To avoid this restriction on the tests, we introduce a normal form for the uncoverable states of the process being tested.

Uncontrollable processes have no partners or covering tests, whereas the smallest set of successful tests (in terms of fair testing) is the set of trivial tests. The idea is to introduce a normal form such that uncoverable states correspond to states for which only the trivial tests are successful.

The simplest uncontrollable process, viz., δ, is not a candidate for this normal form, as non-trivial tests like $a + \circledast$ are successful on process δ, but not on, e.g., the uncontrollable process a. The next lemma gives a normal form U for the uncoverable states, and proves that U is an uncontrollable process and that every successful test on U is trivial.

Lemma 3 (Normalized uncontrollable process)**.** *Let U be the process such that only $U \xrightarrow{a} U$, for every action a, and $U \xrightarrow{\tau} \delta$. Process U is uncontrollable, and all successful tests on U are trivial tests.*

Proof. Process U is uncontrollable because of $U \xrightarrow{\tau} \delta$. To show that all successful tests are trivial tests, consider any test t that is a successful test on U, and any process x that does not contain the action $\circledast$. Process U is such that it can mimic every execution of x and afterwards perform the τ-edge and block. Test t is a successful test on U, and hence t is a successful test on every process x. □

Instead of process U as defined in Lemma 3, we could also use the catastrophic divergent process *CHAOS* from CSP [5], but process *CHAOS* is more complicated than needed for our purposes.

Definition 15 (*Pruned process*)**.** A process is *pruned* iff every first uncoverable state is equal to U.

Note that the first uncoverable states are well-defined, as we consider processes as trees with one initial state. Any process can be transformed into an accordance-equivalent pruned process; *pruning* is the operation of replacing each first uncoverable state (including all outgoing edges) by U. Pruning the example process $x = a \cdot (b + b \cdot \checkmark + c \cdot \checkmark)$ yields the process $a \cdot (b \cdot U + b \cdot U + c \cdot \checkmark)$, which is bisimilar to $a \cdot (b \cdot U + c \cdot \checkmark)$. Moreover, every pruned uncontrollable process is equal to U. We can decide whether a finite-state process is pruned by computing its coverable and uncoverable states.

5.3. Vulnerable set of processes

To prove that $\sqsubseteq_{crt}$ implies $\sqsubseteq_{rt}$, it turns out to be insufficient to restrict the processes to pruned processes. For example, consider the two processes $x = a \cdot \checkmark$ and $y = a \cdot \checkmark + b \cdot U$, which are controllable and pruned processes containing the synchronized actions a and b. The term $b \cdot U$ in y can be interpreted as that no partner may synchronize on action b in this state; in contrast, a partner for x may offer synchronization on b. In this case $x \sqsubseteq_{acc} y$ holds, and hence $x \sqsubseteq_{crt} y$ holds by Theorem 4, but the restricted test $t = b \cdot \circledast$ is successful for y, but not for x.

The problem is the action on the edge from a coverable to an uncoverable state. Although the proposed kind of pruning does not change the semantics in terms of $\sqsubseteq_{acc}$ and $\sqsubseteq_{crt}$, removing this action would change the semantics, and hence we do not consider this to be feasible.

To find a sufficient condition on the processes, let us consider a typical proof attempt for $x \sqsubseteq_{crt} y$ implies $x \sqsubseteq_{rt} y$, for any two processes x and y. Such a proof starts with a successful $\sqsubseteq_{rt}$-test t on y. To use the $\sqsubseteq_{crt}$-relation, test t should be transformed into a $\sqsubseteq_{crt}$-test. The following lemma describes a way to do so.

Lemma 4. *Let x be a controllable and pruned process that does not contain the action $\circledast$. Any successful test on x can be transformed into a successful covering test on x in two steps. First, replace each trivial test by action $\circledast$. Second, replace some branches $a \cdot \circledast$, where action a can move x from a coverable to an uncoverable state, by action $\circledast$.*

Proof. Let x be a controllable and pruned process that does not contain the action $\circledast$. Let t be a successful test on x.

As $\circledast$ is a trivial test, replacing in t every trivial test by action $\circledast$ yields a successful test t' on x. Consider in the composition $x \| t'$ each first reachable state $q' \| r'$, such that q' is an uncoverable state of x and r' is a state of t'. As x is controllable, there is also a reachable state $q \| r$, in which q is a coverable state of x and r is a state of t, such that there is an edge $q \| r \xrightarrow{a} q' \| r'$ for a synchronized action a different from $\checkmark$. As x is pruned, q' is equal to U. As every trivial test in t' is equal to $\circledast$, we have $r' = \circledast$. By replacing in t' each such branch $a \cdot \circledast$ by $\circledast$ we obtain a successful covering test on x. □

For the example process $x = a \cdot (b + b \cdot \checkmark + c \cdot \checkmark)$, the successful test $a \cdot b \cdot (\circledast + a \cdot \circledast)$ is first transformed into the successful test $a \cdot b \cdot \circledast$, and then in the successful covering test $a \cdot \circledast$.

Using Lemma 4, a successful $\sqsubseteq_{rt}$-test t on y can be transformed into a successful $\sqsubseteq_{crt}$-test t' on y. Using the given $\sqsubseteq_{crt}$-relation, test t' is a successful $\sqsubseteq_{crt}$-test on x. To conclude that t is a successful $\sqsubseteq_{rt}$-test on x, we have to replace in t' some actions $\circledast$ by a branch $a \cdot \circledast$, for some synchronized action a. However, it is not guaranteed that a can be accepted, and hence a deadlock may be introduced.

For the counter-example in the beginning of this section, test $t = b \cdot \circledast$ is transformed into the successful covering test $t' = \circledast$ on x. To conclude that t is successful on x, action b should be accepted by x, but this is not the case.

To solve this issue, we assume not only that the processes have been pruned, but also that the considered set of processes is vulnerable.

Definition 16 (*Vulnerable set of processes*)**.** A set of processes is *vulnerable* iff every action that can move some of the processes from a coverable state (where $\surd$ has not yet occurred) to an uncoverable state, is an action that can be accepted in every coverable state of each of the processes. A process x is vulnerable iff $\{x\}$ is a vulnerable set of processes.

This may sound like a severe restriction, but in Section 6.1 we will see that vulnerable sets of processes are quite common in practice. We can decide whether a finite-state process is vulnerable by computing its coverable and uncoverable states.

In general there exists no accordance-equivalent process that is vulnerable. For example, consider the process $x = a + b \cdot a \cdot \surd$. In what follows, we collect necessary conditions on a vulnerable process y that is accordance-equivalent to x, and show a contradiction. Process x and y have $b \cdot a \cdot \surd$ as a partner; hence all the states in y that can be visited with this partner are coverable. However, $b \cdot a \cdot \surd + a \cdot z'$, for any process z', is not a partner for x and y; hence a is an action that can move y from a coverable state to an uncoverable state. Finally, $b \cdot a \cdot (\surd + a)$ is a partner for x and y; hence y cannot accept a in a certain coverable state. So there exists no vulnerable process y that is accordance-equivalent to x.

Pruning behaves nicely in terms of a vulnerable set of processes as it only normalizes uncoverable states.

Lemma 5. *Pruning each process in a vulnerable set of processes yields a vulnerable set of pruned processes.*

5.4. Covering restricted testing implies restricted testing

Using this theory on pruning and vulnerable processes, we can prove that covering restricted testing implies restricted testing.

Theorem 5 (Covering restricted testing implies restricted testing)**.** *Let x and y be two controllable and pruned processes that do not contain the action $\circledast$. If x and y are in a vulnerable set of processes, then the following holds*:

$$x \sqsubseteq_{crt} y \;\Rightarrow\; x \sqsubseteq_{rt} y.$$

Proof. Let x and y be controllable and pruned processes that do not contain the action $\circledast$, and that are in a vulnerable set of processes. Let $x \sqsubseteq_{crt} y$, i.e., for each restricted test t'' it holds that if t'' is a successful covering test on y, then t'' is a successful covering test on x. To prove $x \sqsubseteq_{rt} y$, let t be a restricted test that is successful on y. What remains to be proved is that t is successful on x.

Based on Lemma 4, we replace in t each trivial test by action $\circledast$, yielding a successful test t' on y, and afterwards replace some branches $a \cdot \circledast$ (where a can move y from a coverable state to an uncoverable state) by action $\circledast$, yielding a successful covering test t'' on y. As all trivial tests in t' had been replaced by $\circledast$, no $\circledast$ occurs before such an a in t'. As test t is restricted, tests t' and t'' are also restricted, and hence no $\surd$ occurs before such an a in t'.

Using $x \sqsubseteq_{crt} y$, test t'' is a successful covering test on x. To obtain t again, we first replace some actions $\circledast$ by a branch $a \cdot \circledast$. For the actions $\circledast$ that are not reachable in the composition $x \| t''$, this cannot harm success on x. Every action $\circledast$ that is reachable in the composition $x \| t''$ starts in a coverable state of x, as t'' is a covering test on x. Action a can be accepted in this state, as x and y are in a vulnerable set of processes and action a could move y from a coverable state (where $\surd$ had not yet occurred) to an uncoverable state. Finally we replace some actions $\circledast$ in successful test t' on x by a trivial test, yielding test t, which is guaranteed to be successful on x. $\square$

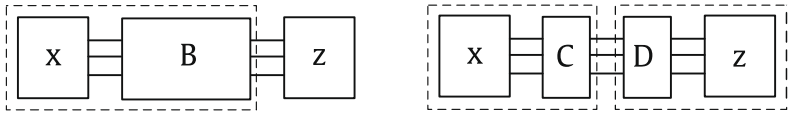
From Corollary 3 and Theorem 5 we immediately conclude that, for certain processes, covering restricted testing is equivalent to restricted testing.

Corollary 4 (Covering restricted testing equals restricted testing)**.** *Let x and y be two controllable and pruned processes that do not contain the action $\circledast$. If x and y are in a vulnerable set of processes, then $x \sqsubseteq_{crt} y \;\Leftrightarrow\; x \sqsubseteq_{rt} y$.*

As the notions of accordance and covering restricted testing are equivalent (see Corollary 2), we can relate accordance and restricted testing.

Corollary 5 (Accordance equals restricted testing)**.** *Let x and y be two controllable and pruned processes that do not contain the action $\circledast$. If x and y are in a vulnerable set of processes, then $x \sqsubseteq_{acc} y \;\Leftrightarrow\; x \sqsubseteq_{rt} y$.*

These restrictions are not yet enough to prove $x \sqsubseteq_{acc} y \Rightarrow x \sqsubseteq_{ft} y$. Consider the counter-example $x = a \cdot \surd + \surd$ and $y = a \cdot \surd$, where x and y are in a vulnerable set of pruned processes. In relation to process y, process x only makes it easier to terminate properly, so $x \sqsubseteq_{acc} y$ holds. However, the (non-restricted) test $t = \circledast + \surd$ is a successful test on y, but not on x. In particular, this shows that fair testing and accordance do not coincide for processes in which all states are coverable, although this was claimed by [13].



(a) One buffer (B) (b) Two buffers (C and D)

Fig. 3. Buffering schemes.

6. Asynchronous communication

In this section we show that all processes that only communicate asynchronously are in a vulnerable set. We first describe how to model such processes, and then derive some useful properties of them.

6.1. Modeling asynchronous communication

Processes that communicate asynchronously can be modeled as processes that communicate synchronously, by explicitly introducing the communication buffers between them as additional processes (see also a brief remark in [11]). In line with [19,3], we consider unbounded uni-directional buffers that are empty in the initial state and that must be empty upon termination.

Let M be a set of message types. Let the asynchronous processes use synchronized actions (apart from $\checkmark$) of the shape $?m$ and $!m$, for $m \in M$, to denote respectively receiving and sending a message of type m. For each message type m, a buffer can be modeled as $B_m(0)$, where B_m is defined as:

$$B_m(0) = \checkmark + !m \cdot B_m(1), \quad B_m(n+1) = ?m \cdot B_m(n) + !m \cdot B_m(n+2)$$

In $B_m(n)$, the natural number n denotes the number of messages in the buffer. Synchronization on action $!m$ puts a message in the buffer, whereas synchronization on $?m$ gets a message from the buffer. The buffer can only terminate properly when it is empty.

Thus, asynchronous communication using a set of message types M can be modeled as follows. Let x be a given asynchronous process and z be an asynchronous partner, that use disjoint sets of synchronized actions for the message types M. Let B denote the merge $\|_{\{\checkmark\}}$ of B_m, for every $m \in M$. The buffered composition of x and z can be modeled as $x \parallel B \parallel z$; in addition to synchronization over $\checkmark$, the first merge synchronizes over the synchronized actions of the buffer for x, and the second merge synchronizes over the synchronized actions of the buffer for z.

To keep the binary setting of a given process x and a partner z, and to avoid imposing restrictions on the set of all partners, we integrate the buffers with each given process. The resulting process is $x \parallel B$; see Fig. 3(a). Thus, the partner z can send a message of type m to the process x using a synchronized action $!m$ of the buffer; receiving a message from z is an internal action of the composition of the process x and the buffer B. Similarly, the partner z can receive a message of type m from the process x using a synchronized action $?m$ of the buffer; sending a message to z is an internal action of the composition of the process x and the buffer B. We call such an integrated process $x \parallel B$ a buffered process.

Definition 17. Let I and O be two disjoint sets of message types. An (I, O)-*buffered process* is a process where each action is either

1. an internal (unsynchronized) action;
2. the synchronized action $\checkmark$;
3. a synchronized action that gets a message from an output buffer for type $m \in O$; or
4. a synchronized action that puts a message in an input buffer for type $m \in I$.

In Section 2.2 we have seen that, in general, synchronous processes do not become smaller in the accordance pre-order by increasing some of their external choices. However, to some extent, this does hold for (I, O)-buffered processes in terms of the asynchronous processes they are based on. For example, consider the following extension of an asynchronous process (before introducing the buffers for $I = \{a, b\}$ and $O = \emptyset$):

$$?a \cdot \checkmark \quad \rightarrow \quad ?a \cdot \checkmark + ?b$$

In both cases, after introducing the buffers it is possible for a partner to synchronize on $!b$. Moreover, in both cases after this behavior proper termination cannot be guaranteed anymore. However, the following similar extension is not valid:

$$?a \cdot ?b \cdot \checkmark \quad \not\rightarrow \quad ?a \cdot ?b \cdot \checkmark + ?b$$

In this case a controllable process is extended to an uncontrollable process, and this makes the process bigger in the accordance pre-order.

6.2. Properties of asynchronous services

It is well known that two (unbounded) buffers in series behave as one (unbounded) buffer, i.e., they are branching bisimilar after hiding the synchronized actions in between them. So, B_m can be replaced by $C_m \parallel_{\{\checkmark,m\}} D_m$, where C_m and D_m are buffers that are obtained from B_m by renaming some of the actions:

$$C_m(0) = \checkmark + !m \cdot C_m(1), \quad C_m(n+1) = m \cdot C_m(n) + !m \cdot C_m(n+2)$$

$$D_m(0) = \checkmark + m \cdot D_m(1), \quad D_m(n+1) = ?m \cdot D_m(n) + m \cdot D_m(n+2)$$

Thus, using the associativity of the merge, any partner z of an (I,O)-buffered process $x \| B$ can be transformed into a buffered partner $D \| z$ for $x \| C$ (or vice versa), without affecting weak termination; see Fig. 3(b). As $x \| B$ and $x \| C$ are identical up to renaming some synchronized actions (like m), $D \| z$ can be transformed into a (O,I)-buffered partner for $x \| B$ by some renaming.

Let us investigate which actions can move a buffered process from a coverable to an uncoverable state.

Lemma 6. *Let x be an (I,O)-buffered process, let q be a coverable state of x, and let a be an action such that $q \xrightarrow{a} q'$, for some state q' of x. State q' is guaranteed to be coverable if action a is*

1. *an internal (unsynchronized) action;*
2. *the synchronized action $\checkmark$;*
3. *a synchronized action $?m$ of type $m \in O$ that gets a message from an output buffer.*

Proof. Let x be a buffered process, let q be a coverable state of x, and let a be an action such that $q \xrightarrow{a} q'$, for some state q' of x.

As q is coverable, there is a partner z that can visit q while being in a state r. To show that q' is coverable, we show how to construct a partner that can visit state q' in case partner z does not:

1. As internal action a is unsynchronized, partner z can also visit q'.
2. Adding in r a $\checkmark$-labeled outgoing edge, gives a partner that can visit q'.
3. Replacing partner z by an (O,I)-buffered partner as described before, gives a partner that can synchronize on $?m$ in every state, and hence it can visit q'. □

So, the only actions that can move an (I,O)-buffered process from a coverable state to an uncoverable state are actions that put a message into an input buffer. As the buffers are unbounded, these actions can be accepted in every state before any occurrence of action $\checkmark$. Combining these ingredients, we can conclude that certain sets of buffered processes are vulnerable.

Lemma 7. *Let I and O be two disjoint sets of message types. Any set of (I,O)-buffered processes is a vulnerable set of processes.*

Combining our previous results, we obtain the following result for asynchronous processes.

Corollary 6. *Let I and O be two disjoint sets of message types. Let x and y be two asynchronous processes that only contain the synchronized actions $\checkmark$, $?m$ for $m \in I$, and $!m$ for $m \in O$, but that do not contain the action $\circledast$. Let B be the buffer that corresponds to I and O. Let x' be the result of pruning the (I,O)-buffered process $x \| B$, and let y' be the result of pruning the (I,O)-buffered process $y \| B$. If $x \| B$ and $y \| B$ are controllable , then $x' \sqsubseteq_{acc} y' \Leftrightarrow x' \sqsubseteq_{rt} y'$.*

7. Conclusions and further work

We have studied the replaceability of services in terms of the accordance pre-order. Accordance is defined in terms of the partners for a process, which are the processes that can properly use the process, i.e., their composition is free of deadlocks and non-terminating loops. An uncoverable state of a process is a state that cannot be visited using any partner for the process.

We have identified the position of accordance in relation to traditional process equivalences and pre-orders. We have proved that fair testing is the coarsest traditional pre-order that implies accordance, and analyzed under which conditions accordance implies (restricted) fair testing. Thus we have identified the differences between accordance and fair testing:

- *Success and termination*: In case of accordance, the composed processes must terminate together (using a synchronized action $\checkmark$). In case of fair testing, only the test needs to perform an (unsynchronized) "success" action $\circledast$.
- *Testing for termination*: In case of accordance, there is no information about the states of the process after any occurrence of action $\checkmark$. In case of fair testing, there is information about each state of the process.
- *Uncoverable states*: In case of accordance, there is no information about uncoverable states of the process. In case of fair testing, there is information about each state of the process.

We have introduced a covering restricted testing pre-order as a variant of fair testing. This pre-order restricts the tests to those that can only test positively for termination, and that do not visit uncoverable states. We have proved that accordance is equivalent to this novel variant of fair testing.

We have also shown that we can allow the tests to visit uncoverable states after imposing two restrictions on the controllable processes being tested: processes have to be

- *pruned*: all uncoverable states must be in a normal form for which only trivial tests are successful.
- *vulnerable*: every action that can move a process from a coverable state to an uncoverable state should be acceptable in every coverable state.

This study is not just theoretical. In practice, most services only communicate asynchronously, and such services are guaranteed to be vulnerable. Services are typically not pruned, but the required identification of the uncoverable states is useful in terms of software engineering.

The covering nature of accordance is absent in most traditional process equivalences and pre-orders. Basically, accordance largely ignores the parts of the processes that cannot be used reliably. This corresponds to many situations in practice, where parts of services are known to be unreliable and hence are not (yet) used. Therefore, an interesting line of further work is to investigate how to integrate this concept in other process equivalences and pre-orders.

Acknowledgements

The authors like to thank the anonymous referees for their insightful comments. Authors Mooij and Voorhoeve participate in the Poseidon project at Thales under the responsibilities of the Embedded Systems Institute (ESI). This project is partially supported by the Dutch Ministry of Economic Affairs under the BSIK program.

References

[1] J. Baeten, W. Weijland, Process Algebra, Cambridge University Press, 1990.
[2] M. Bravetti, G. Zavattaro, Contract based multi-party service composition, in: FSEN'07, LNCS, vol. 4767, 2007, pp. 207–222.
[3] M. Bravetti, G. Zavattaro, Contract compliance and choreography conformance in the presence of message queues, in: WS-FM'08, LNCS, vol. 5387, 2009, pp. 37–54.
[4] E. Brinksma, A. Rensink, W. Vogler, Fair testing, in: CONCUR'95, LNCS, vol. 962, 1995, pp. 313–327.
[5] S. Brookes, C. Hoare, A. Roscoe, A theory of communicating sequential processes, Journal of the ACM 31 (3) (1984) 560–599.
[6] M. Boreale, R. De Nicola, R. Pugliese, Basic observables for processes, Information and Computation 149 (1999) 95–109.
[7] G. Castagna, N. Gesbert, L. Padovani, A theory of contracts for web services, ACM Transactions on Programming Languages and Systems 31 (2009).
[8] E. Clarke, E. Emerson, Design and synthesis of synchronization skeletons using branching-time temporal logic, in: Logics of Programs '81, LNCS, vol. 131, 1982, pp. 52–71.
[9] R. De Nicola, Extensional equivalences for transition systems, Acta Informatica 24 (2) (1987) 211–237.
[10] R. De Nicola, M. Hennessy, Testing equivalences for processes, Theoretical Computer Science 34 (1984) 83–133.
[11] C. Fournet, T. Hoare, S. Rajamani, J. Rehof, Stuck-free conformance, in: CAV'04, LNCS, vol. 3114, 2004, pp. 242–254.
[12] C. Laneve, L. Padovani, The must preorder revisited, in: CONCUR'07, LNCS, vol. 4703, 2007, pp. 212–225.
[13] R. Malik, D. Streader, S. Reeves, Conflicts and fair testing, Journal of Foundations of Computer Science 17 (4) (2006) 797–813.
[14] A. Mooij, M. Voorhoeve, Proof techniques for adapter generation, in: WS-FM'08, LNCS, vol. 5387, 2009, pp. 207–223.
[15] V. Natarajan, R. Cleaveland, Divergence and fair testing, in: ICALP'95, LNCS, vol. 944, 1995, pp. 648–659.
[16] I. Phillips, Refusal testing, Theoretical Computer Science 50 (1987) 241–284.
[17] A. Rensink, W. Vogler, Fair testing, Information and Computation 205 (2) (2007) 125–198.
[18] C. Stahl, P. Massuthe, J. Bretschneider, Deciding substitutability of services with operating guidelines, LNCS ToPNoC 5460 (II) (2009) 172–191.
[19] W.M.P. van der Aalst, N. Lohmann, P. Massuthe, C. Stahl, K. Wolf, Multiparty contracts: Agreeing and implementing interorganizational processes, The Computer Journal 53 (1) (2010) 90–106.
[20] R. van Glabbeek, The linear time–branching time spectrum II; the semantics of sequential processes with silent moves, in: CONCUR'93, LNCS, vol. 715, 1993, pp. 66–81.
[21] R. van Glabbeek, The linear time–branching time spectrum I. The semantics of concrete, sequential processes, in: Handbook of Process Algebra, Elsevier, 2001, pp. 3–99.
[22] R. van Glabbeek, M. Voorhoeve, Liveness, fairness and impossible futures, in: CONCUR'06, LNCS, vol. 4137, 2006, pp. 126–141.
[23] W. Vogler, in: Modular Construction and Partial Order Semantics of Petri Nets, LNCS, vol. 625, Springer-Verlag, 1992.
[24] K. Wolf, Does my service have partners? LNCS ToPNoC 5460 (II) (2009) 152-171.
[25] K. Wolf, C. Stahl, J. Ott, R. Danitz, Verifying livelock freedom in an SOA scenario, in: ACSD'09, IEEE Computer Society, July 2009, pp. 168–177.