Unfolding-based Test Selection for Concurrent Conformance

Hernán Ponce de León[1], Stefan Haar[1], and Delphine Longuet[2]

[1] INRIA and LSV, École Normale Supérieure de Cachan and CNRS, France
ponce@lsv.ens-cachan.fr , stefan.haar@inria.fr
[2] Univ Paris-Sud, LRI UMR8623, Orsay, F-91405
longuet@lri.fr

Abstract. Model-based testing has mainly focused on models where currency is interpreted as interleaving (like the **ioco** theory for labeled transition systems), which may be too coarse when one wants concurrency to be preserved in the implementation. In order to test such concurrent systems, we choose to use Petri nets as specifications and define a concurrent conformance relation named **co-ioco**. We propose a test generation algorithm based on Petri net unfolding able to build a complete test suite w.r.t our **co-ioco** conformance relation. In addition we propose a coverage criterion based on a dedicated notion of complete prefixes that selects a manageable test suite.

Model-based Testing. The aim of testing is to execute a software system, the *implementation*, on a set of input data selected so as to find discrepancies between actual behavior and intended behavior described by the *specification*. The testing process is usually decomposed into three phases: selection of relevant input data, called a *test suite*, among the possible inputs of the system; submission of this test suite to the implementation, its *execution*; and decision of the success or the failure of the test suite submission, known as the *oracle problem*. We focus here on the selection phase, crucial for relevance and efficiency of testing.

Model-based testing requires a behavioral description of the system under test. One of the most popular formalisms studied in conformance testing is that of *input output labeled transition systems* (IOLTS). In this framework, the correctness (or conformance) relation the system under test (SUT) and its specification must verify is formalized by the **ioco** relation [1]. This relation has become a standard, and is used as a basis in several testing theories for extended state-based models: restrictive transition systems [2, 3], symbolic transition systems [4, 5], timed automata [6], multi-port finite state machines [7].

Model-based Testing of Concurrent Systems. Systems composed of concurrent components are naturally modeled as a *network of finite automata*, a formal class of models that can be captured equivalently by *safe Petri nets*. Concurrency in a specification can arise for different reasons. First, two events may be physically localized on different components, and thus be "naturally" independent of one another; this distribution is then part of the system construction. Second, the specification may not care about the order in which two actions are performed *on the same component*, and thus leave the choice of their ordering to the implementation. Depending on the nature of the concurrency specified in a given case, and thus on the intention of the specification, the *implementation relations* have to allow or disallow ordering of concurrent events. The kind of systems that we consider is of the first type, where concurrency comes from the distribution of components. Therefore, we want concurrency of the specification to be preserved in the implementation.

Model-based testing of concurrent systems has been studied for a long time [8–10], however it is most of the time studied in the context of interleaving, or trace, semantics, which is known to suffer the state space explosion problem. While the passage to models with explicit concurrency has been successfully performed in other fields of formal analysis such as model checking or diagnosis, testing has embraced such models somewhat more recently. Ulrich and König propose in [11] a framework for testing concurrent systems specified by communicating labeled transition systems. The specification is translated into a Petri net, and a complete prefix of its unfolding is used to construct a *behavior machine*. The conformance relation proposed in [11] is a generalization of trace equivalence relation; their work does not include a test selection procedure, or how the choice of complete prefix impacts selection. Since our goal is to include *conflict* relations as well, we will use *event structures* and their properties.

Haar et al [12, 13] generalize the basic notions and techniques of I/O-sequence based conformance testing via a generalized I/O-automaton model where partially ordered patterns of input/output events are admitted as transition labels. However, these models still maintain a sequential automaton as the system's skeleton, and include synchronization constraints, e.g. all events in the course of a transition must be completed before any other transition can start.

Our Contribution. In order to enlarge the application domain, and at stronger benefits from concurrency modeling, we have introduced in [14] a concurrent conformance relation named **co-ioco**, as a generalization of **ioco**. In [15], we dropped the input enabledness assumption and enlarged the conformance relation in order to observe refusals. Extra causality between outputs specified as concurrent is also allowed.

This paper extends [14, 15] with a conformance relation where actions specified as concurrent must occur independently, on different processes, in any conformant implementation. While sufficient conditions for soundness and exhaustiveness of test suites have been given in [15], we need more: in practice, only a finite number of test cases can be executed; hence we need a method to select a finite set of relevant test cases covering as many behaviors as possible (thus finding as many anomalies as possible).

The main contributions of this paper are the following: an algorithm to construct a complete test suite; a selection criterion that stipulates which behaviors of the system should be tested in order to have a good coverage of the specification; and an algorithm to construct a sound test suite based on this criterion.

Outline. The paper is organized as follows. Section 1 recalls basic notions about Petri nets, occurrence nets and labeled event structures. Section 2 introduces our testing hypotheses and our **co-ioco** conformance relation. In Section 3, we define the notion of complete test suite, we give sufficient conditions for a test suite to be complete and an algorithm producing such a test suite. Finally, we define in Section 4 our notion of coverage criterion and we adapt the complete finite prefix algorithm of [16] to build a sound test suite satisfying this criterion.

1 I/O Petri Nets and their Semantics

We choose to use *Petri nets* as specifications to have explicit concurrency. The semantics associated to a Petri net is given by its unfolding to an *occurrence net*, which can also be seen as an *event structure*. We will present both notions since we use them in different contexts in the following. The execution traces for this semantics are not sequences but *partial orders*, which keep concurrency explicit. We recall here these basic notions.

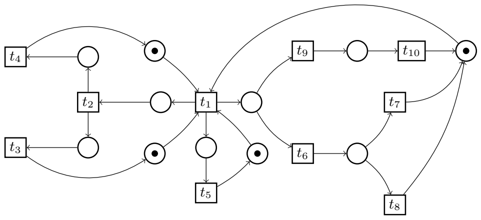
I/O Petri Nets. A *net* is a tuple $N = (P, T, F)$ where *(i)* $P \neq \emptyset$ is a set of *places*, *(ii)* $T \neq \emptyset$ is a set of *transitions* such that $P \cap T = \emptyset$, *(iii)* $F \subseteq (P \times T) \cup (T \times P)$ is a set of *flow arcs*. A *marking* is a multiset M of places, i.e. a map $M : P \to \mathbb{N}$. A *Petri net* is a tuple $\mathcal{N} = (P, T, F, M_0)$, where *(i)* (P, T, F) is a finite net, and *(ii)* $M_0 : P \to \mathbb{N}$ is an *initial marking*. Elements of $P \cup T$ are called the *nodes* of $\mathcal{N}$. For a transition $t \in T$, we call ${}^\bullet t = \{p \mid (p, t) \in F\}$ the *preset* of t, and $t^\bullet = \{p \mid (t, p) \in F\}$ the *postset* of t. In figures, we represent as usual places by empty circles, transitions by squares, F by arrows, and the marking of a place p by black tokens in p. A transition t is *enabled* in marking M, written $M \xrightarrow{t}$, if $\forall p \in {}^\bullet t,\ M(p) > 0$. This enabled transition can *fire*, resulting in a new marking $M' = M - {}^\bullet t + t^\bullet$. This firing relation is denoted by $M \xrightarrow{t} M'$. A marking M is *reachable* from M_0 if there exists a *firing sequence*, i.e. transitions $t_0 \dots t_n$ such that $M_0 \xrightarrow{t_0} M_1 \xrightarrow{t_1} \dots \xrightarrow{t_n} M$. The set of markings reachable from M_0 (in $\mathcal{N}$) is denoted $\mathbf{R}_{\mathcal{N}}(M_0)$ (we drop the subscript referring to $\mathcal{N}$ when it is clear from the context). A Petri net $\mathcal{N} = (P, T, F, M_0)$ is *(1-)safe* iff for all reachable markings $M \in \mathbf{R}(M_0)$, $M(p) \in \{0, 1\}$ for all $p \in P$.

Let $\mathcal{I}$ and $\mathcal{O}$ be two disjoint non-empty sets of *input* and *output* labels, respectively. For a net $N = (P, T, F)$, a mapping $\lambda : T \to (\mathcal{I} \uplus \mathcal{O})$ is called an *I/O-labeling*. Denote by $T^{\mathcal{I}}$ and $T^{\mathcal{O}}$ the input and output transition sets, respectively; that is, $T^{\mathcal{I}} \triangleq \lambda^{-1}(\mathcal{I})$ and $T^{\mathcal{O}} \triangleq \lambda^{-1}(\mathcal{O})$. An *I/O Petri net* is a pair $\Sigma = (\mathcal{N}, \lambda)$, where $\mathcal{N} = (P, T, F, M_0)$ is a 1-safe Petri net and $\lambda : T \to (\mathcal{I} \uplus \mathcal{O})$ an I/O-labeling. Σ is called *deterministically labeled* iff no two transitions with the same label are simultaneously enabled, i.e. for all $t_1, t_2 \in T$ and $M \in \mathbf{R}(M_0)$:

$$(M \xrightarrow{t_1}\ \wedge\ M \xrightarrow{t_2}\ \wedge\ \lambda(t_1) = \lambda(t_2)) \Rightarrow t_1 = t_2$$

Note that 1-safeness of the Petri net is not sufficient for guaranteeing deterministic labeling. Deterministic labeling ensures that the system behavior is locally discernible through labels, either through distinct inputs or through observation of different outputs.

When testing reactive systems, we need to differentiate situations where the system can still produce some outputs and those where the system can not evolve without an input from the environment. Such situations are captured by the notion of *quiescence* [17]. A marking is said quiescent if it does not enable output transitions, i.e. $M \xrightarrow{t}$ implies $t \in T^{\mathcal{I}}$. The observation of quiescence is usually instrumented by timers. Jard and Jéron [18] present three different kinds of quiescence: *output quiescence* when the



t	$\lambda(t)$
t_1	?*login*
t_2	?*insurance*
t_3	!*ins_price*
t_4	!*ins_data*
t_5	!*us_data*

t	$\lambda(t)$
t_6	?*train*
t_7	!*tr_price 1*
t_8	!*tr_price 2*
t_9	?*plane*
t_{10}	!*pl_price*

Fig. 1. A travel agency specified by an I/O Petri net

system is waiting for an input from the environment, *deadlock* when the system can not evolve anymore, and *livelock* when the system diverges by an infinite sequence of silent actions.

Occurrence Nets and Unfoldings. Occurrence nets can be seen as Petri nets[3] with a special acyclic structure that highlights *conflict* between transitions that compete for resources. Formally, let $N = (P, T, F)$ be a net, $<_N$ the transitive closure of F, and $\leqslant_N$ the reflexive closure of $<_N$. We say that transitions t_1 and t_2 are in *structural conflict*, written $t_1 \#^\omega t_2$, if and only if $t_1 \neq t_2$ and ${}^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset$. *Conflict* is inherited along $<_N$, that is, the conflict relation $\#$ is given by

$$a \# b \Leftrightarrow \exists t_a, t_b \in T : t_a \#^\omega t_b \wedge t_a \leqslant_N a \wedge t_b \leqslant_N b$$

Finally, the *concurrency relation* **co** holds between nodes $a, b \in P \cup T$ that are neither ordered nor in conflict, i.e. a **co** $b \Leftrightarrow \neg (a \leqslant b) \ \wedge \ \neg (a \# b) \ \wedge \ \neg (b < a)$.

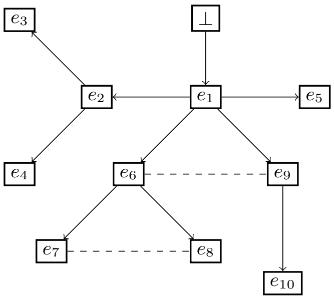
Definition 1. *A net* $ON = (B, E, G)$ *is an* occurrence net *if and only if*

1. $\leqslant_{ON}$ *is a partial order;*
2. *for all* $b \in B$, $|{}^\bullet b| \in \{0, 1\}$*;*
3. *for all* $x \in B \cup E$*, the set* $[x] = \{y \in E \mid y \leqslant x\}$ *is finite;*
4. *no self-conflict, i.e. there is no* $x \in B \cup E$ *such that* $x \# x$*;*
5. $\bot \in E$ *is the only* $\leq$*-minimal node (event* $\bot$ *creates the initial conditions)*

Call the elements of E *events*, those of B *conditions*. An ON can also be given as a tuple $(B, E \backslash \{\bot\}, F, cut_0)$, where $cut_0 = \bot^\bullet$ is the set of minimal conditions. Occurrence nets are the mathematical form of the *partial order unfolding semantics* [16]. A *branching process* of a 1-safe Petri net $\mathcal{N} = (N, M_0)$ is given by a pair $\Phi = (ON, \varphi)$, where $ON = (B, E, G)$ is an occurrence net, and $\varphi : B \cup E \to P \cup T$ is such that:

1. it is a homomorphism from ON to N, i.e.
   - $\varphi(B) \subseteq P$ and $\varphi(E) \subseteq T$, and
   - for every $e \in E$, the restriction of φ to ${}^\bullet e$ is a bijection between the set ${}^\bullet e$ in ON and the set ${}^\bullet \varphi(e)$ in N, and similarly for $e^\bullet$ and $\varphi(e)^\bullet$;

[3] when one allows Petri nets to be *infinite*



e	$\varphi(e)$	$\lambda(e)$
e_1	t_1	?*login*
e_2	t_2	?*insurance*
e_3	t_3	!*ins_price*
e_4	t_4	!*ins_data*
e_5	t_5	!*us_data*
e_6	t_6	?*train*
e_7	t_7	!*tr_price 1*
e_8	t_8	!*tr_price 2*
e_9	t_9	?*plane*
e_{10}	t_{10}	!*pl_price*

Fig. 2. Part of the unfolding of the PN from Figure 1 represented as an IOLES. Causality is represented by arrows and immediate conflict by dashed lines.

2. the restriction of φ to cut_0 is a bijection from cut_0 to M_0; and
3. for every $e_1, e_2 \in E$, if ${}^\bullet e_1 = {}^\bullet e_2$ and $\varphi(e_1) = \varphi(e_2)$ then $e_1 = e_2$.

The unique (up to isomorphism) maximal branching process $\mathcal{U} = (ON_{\mathcal{U}}, \varphi_{\mathcal{U}})$ of $\mathcal{N}$ is called the *unfolding* of $\mathcal{N}$.

Input/Output Labeled Event Structures. Occurrence nets give rise to event structures in the sense of Winskel et al [19]; as usual, we will use both the event structure and the occurrence net formalism, whichever is more convenient. An *input/output labeled event structure (IOLES)* over an alphabet $L = \mathcal{I} \uplus \mathcal{O}$ is a 4-tuple $\mathcal{E} = (E, \leq, \#, \lambda)$ where *(i)* E is a set of events, *(ii)* $\leq\ \subseteq E \times E$ is a partial order (called *causality*) satisfying the property of *finite causes*, i.e. $\forall e \in E : |\{e' \in E \mid e' \leq e\}| < \infty$, *(iii)* $\#\ \subseteq E \times E$ is an irreflexive symmetric relation (called *conflict*) satisfying the property of *conflict heredity*, i.e. $\forall e, e', e'' \in E : e \# e' \wedge e' \leq e'' \Rightarrow e \# e''$, *(iv)* $\lambda : E \rightarrow (\mathcal{I} \uplus \mathcal{O})$ is a labeling mapping. In addition, we assume every IOLES $\mathcal{E}$ has a unique minimal event $\perp_{\mathcal{E}}$. We denote the class of all input/output labeled event structures over L by $\mathcal{IOLES}(L)$. Given event e, its *local configuration* is $[e] \triangleq \{e' \in E \mid e' \leq e\}$, and its set of *causal predecessors* is $\langle e \rangle \triangleq [e] \backslash \{e\}$. Two events $e, e' \in E$ are said to be concurrent (e **co** e') iff neither $e \leq e'$ nor $e' \leq e$ nor $e \# e'$ hold; $e, e' \in E$ are in *immediate conflict* $(e_1 \#^\mu e_2)$ iff $[e_1] \times [e_2] \cap \# = \{(e_1, e_2)\}$. A *configuration* of an IOLES is a non-empty set $C \subseteq E$ that is *(i) causally closed*, i.e. $e \in C$ implies $[e] \subseteq C$, and *(ii) conflict-free*, i.e. $e \in C$ and $e \# e'$ imply $e' \notin C$. Note that we define, for technical convenience, all configurations to be non-empty; the initial configuration of $\mathcal{E}$, containing only $\perp_{\mathcal{E}}$ and denoted by $\perp_{\mathcal{E}}$, is contained in every configuration of $\mathcal{E}$. We denote the set of all the configurations of $\mathcal{E}$ by $\mathcal{C}(\mathcal{E})$.

Labeled Partial Orders. We are interested in testing distributed systems where concurrent actions occur in different components of the system. For this reason, we want to keep concurrency explicit, i.e. specifications do not impose any order of execution between concurrent events. Labeled partial orders can then be used to represent executions of such systems. A *labeled partial order* (lpo) is a tuple $lpo = (E, \leq, \lambda)$ where E is a set of events, $\leq$ is a reflexive, antisymmetric, and transitive relation, and $\lambda : E \rightarrow L$ is a labeling mapping to a fix alphabet L. We denote the class of all labeled partial orders over

L by $\mathcal{LPO}(L)$. Consider $lpo_1 = (E_1, \leq_1, \lambda_1)$ and $lpo_2 = (E_2, \leq_2, \lambda_2) \in \mathcal{LPO}(L)$. A bijective function $f : E_1 \to E_2$ is an isomorphism between lpo_1 and lpo_2 iff *(i)* $\forall e, e' \in E_1 : e \leq_1 e' \Leftrightarrow f(e) \leq_2 f(e')$ and *(ii)* $\forall e \in E_1 : \lambda_1(e) = \lambda_2(f(e))$. Two labeled partial orders lpo_1 and lpo_2 are isomorphic if there exists an isomorphism between them. A *partially ordered multiset* (pomset) is an isomorphism class of lpos. We will represent such a class by one of its objects. Denote the class of all non empty pomsets over L by $\mathcal{POMSET}(L)$. The evolution of the system is captured by the following definition: pomsets are observations.

Definition 2. *For* $\mathcal{E} = (E, \leq, \#, \lambda) \in \mathcal{IOLES}(L)$, $\omega \in \mathcal{POMSET}(L)$ *and* $C, C' \in \mathcal{C}(\mathcal{E})$, *define*

$$\begin{aligned} C \overset{\omega}{\Longrightarrow} C' &\triangleq \exists lpo = (E_\omega, \leq_\omega, \lambda_\omega) \in \omega : E_\omega \subseteq E \backslash C, C' = C \cup E_\omega, \\ &\quad \leq \cap \, (E_\omega \times E_\omega) = \leq_\omega \textit{ and } \lambda_{|E_\omega} = \lambda_\omega \\ C \overset{\omega}{\Longrightarrow} \quad &\triangleq \exists C' : C \overset{\omega}{\Longrightarrow} C' \end{aligned}$$

We can now define the notions of traces and of configurations reachable from a given configuration by an observation. Our notion of traces is similar to the one of Ulrich and König [11].

Definition 3. *For* $\mathcal{E} \in \mathcal{IOLES}(L)$, $\omega \in \mathcal{POMSET}(L)$, $C, C' \in \mathcal{C}(\mathcal{E})$, *define*

$$\begin{aligned} &\textit{traces}(\mathcal{E}) \triangleq \{\omega \in \mathcal{POMSET}(L) \mid \perp_\mathcal{E} \overset{\omega}{\Longrightarrow}\} \\ &C \textbf{ \textit{after} } \omega \triangleq \{C' \mid C \overset{\omega}{\Longrightarrow} C'\} \end{aligned}$$

Note that for deterministically labeled I/O Petri nets, the corresponding IOLES is deterministic and the set of reachable configurations is a singleton.

2 Testing Framework for IOPNs

Testing Hypotheses. We assume that the specification of the system under test is given as a 1-safe and deterministically labeled I/O Petri net $\Sigma = (\mathcal{N}, \lambda)$ over alphabet $L = \mathcal{I} \uplus \mathcal{O}$ of input and output labels. To be able to test an implementation against such a specification, we make a set of testing assumptions. First of all, we make the usual testing assumption that the behavior of the SUT itself can be modeled by a 1-safe I/O Petri net over the same alphabet of labels. We also assume as usual that the specification does not contain cycles of outputs actions, so that the number of expected outputs after a given trace is finite.

Assumption 1 *The net* $\mathcal{N}$ *has no cycle containing only output transitions.*

Third, in order to allow the observation of both the outputs produced by the system and the inputs it can accept, markings where conflicting inputs and outputs are enabled should not be reachable. As a matter of fact, if conflicting input and output are enabled in a given marking, once the output is produced, the input is not enabled anymore, and vice versa. Such markings prevent from observing the inputs enabled in a given configuration, which we will see is one of the key points of our conformance relation. For this reason, we restrict the form of the nets we consider via the following assumption on the unfolding:[4]

Assumption 2 *The unfolding of the net $\mathcal{N}$ has no immediate conflict between input and output events, i.e.* $\forall e_1 \in E^{\mathcal{I}}, e_2 \in E^{\mathcal{O}} : \neg(e_1 \#^{\mu} e_2)$.

Conformance Relation. A formal testing framework relies on the definition of a conformance relation to be satisfied by the SUT and its specification. In the LTS framework, the **ioco** conformance relation compares the outputs and blockings in the implementation after a trace of the specification to the outputs and blockings authorised after this trace in the specification. Classically, the produced outputs of the system under test are elements of $\mathcal{O}$ (single actions) and blockings are observable by a special action $\delta \notin L$ which represents the expiration of a timer.

By contrast, in partial order semantics, we need any set of outputs to be entirely produced by the system under test before we send a new input; this is necessary to detect outputs depending on extra inputs. Suppose two concurrent outputs o_1 and o_2 depending on input i_1 and another input i_2 depending on both outputs. Clearly, an implementation that accepts i_2 before o_2 should not be considered as correct, but if i_2 is sent too early to the system, we may not know if the occurrence of o_2 depends or not on i_2. For this reason we define the expected outputs from a configuration C as the pomset of outputs leading to a quiescent configuration. Such a configuration always exists, and must be finite by Assumption 1.

The notion of quiescence is inherited from nets, i.e. a configuration C is quiescent iff $C \overset{\omega}{\Longrightarrow}$ implies $\omega \in \mathcal{POMSET}(\mathcal{I})$. We assume as usual that quiescence is observable by a special δ action, i.e. C is quiescent iff $C \overset{\delta}{\Longrightarrow}$.

Definition 4. *For $\mathcal{E} \in \mathcal{IOLES}(L)$, $C \in \mathcal{C}(\mathcal{E})$, the outputs produced by C are*

$$out_{\mathcal{E}}(C) \triangleq \{!\omega \in \mathcal{POMSET}(\mathcal{O}) \mid C \overset{!\omega}{\Longrightarrow} C' \wedge C' \overset{\delta}{\Longrightarrow}\} \cup \{\delta \mid C \overset{\delta}{\Longrightarrow}\}$$

The **ioco** theory assumes the input enabledness of the implementation [1], i.e. in any state of the implementation, every input action is enabled. This assumption is made to ensure that no blocking can occur during the execution of the test until its end and the emission of a verdict. However, as explained by Heerink, Lestiennes and Gaudel in [2, 3] even if many realistic systems can be modeled with such an assumption, there remains a significant portion of realistic systems that can not be modeled as such. In order to overcome these difficulties, Lestiennes and Gaudel enrich the system model by refused transitions and a set of possible actions is defined in each state. Any possible input in a given state of the specification should be possible in a correct implementation.

Definition 5. *For $\mathcal{E} \in \mathcal{IOLES}(L)$ and $C \in \mathcal{C}(\mathcal{E})$, the possible inputs in C are*

$$poss_{\mathcal{E}}(C) \triangleq \{?\omega \in \mathcal{POMSET}(\mathcal{I}) \mid C \overset{?\omega}{\Longrightarrow}\}$$

[4] Gaudel et al [3] assume a similar property called *IO-exclusiveness*.

Our **co-ioco** conformance relation for labeled event structures can be informally described as follows. The behavior of a correct **co-ioco** implementation after some observations (obtained from the specification) should respect the following restrictions: (1) the outputs produced by the implementation should be specified; (2) if a quiescent configuration is reached, this should also be the case in the specification; (3) any time an input is possible in the specification, this should also be the case in the implementation. These restrictions are formalized by the following conformance relation.

Definition 6. *Let $\mathcal{E}_i, \mathcal{E}_s \in \mathcal{IOLES}(L)$, then*

$$\begin{array}{rl} \mathcal{E}_i \ \textbf{\textit{co-ioco}}\ \mathcal{E}_s \Leftrightarrow \forall \omega \in \mathit{traces}(\mathcal{E}_s) : & \\ \mathit{poss}_s(\perp \textbf{\textit{ after }} \omega) & \subseteq \mathit{poss}_i(\perp \textbf{\textit{ after }} \omega) \\ \mathit{out}_i(\perp \textbf{\textit{ after }} \omega) & \subseteq \mathit{out}_s(\perp \textbf{\textit{ after }} \omega) \end{array}$$

When several outputs in conflicts are possible, our conformance relation allows implementations where at least one of them is implemented. Extra inputs are allowed in any configuration, but extra outputs, extra quiescence and extra causality between events specified as concurrent are forbidden.

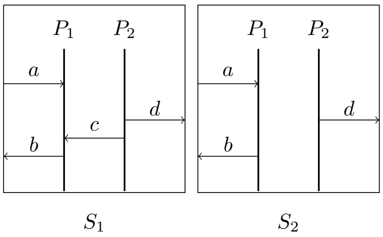


Fig. 3. Message sequence charts showing two implementations of concurrency.

Consider Figure 3. In the **ioco** theory where concurrency is interpreted as interleaving, the concurrency between outputs $!b$ and $!d$ of system S_2 would be described allowing either $!b$ before $!d$ or $!d$ before $!b$. S_1 would be a correct implementation w.r.t **ioco** because one of the two possible orders between the outputs is observed, even if process P_2 interferes in the behavior of process P_1 ($!b$ depends on $!d$). We want to prevent implementations like S_1 introducing extra dependency between events specified as concurrent. Therefore actions specified as concurrent must be implemented as such, meaning that they must occur on different processes and must be independent from each other.

3 Complete Test Suites

A test case is a specification of the tester's behavior during an experiment carried out on the SUT. It must be controllable, i.e. the tester must not have choices to make during the execution of the test. That is, tests must be deterministic, and at any stage, the next input to be proposed by the tester must be unique, i.e. there are no immediate conflicts between inputs. Finally, we require the experiment to terminate, i.e. the resulting event structure to be finite.

Definition 7. *A test case is a finite deterministic IOLES $\mathcal{E}_t = (E_t, \leq_t, \#_t, \lambda_t)$ where $(E_t^{\mathcal{I}} \times E_t^{\mathcal{I}}) \cap \#_t^{\mu} = \emptyset$. A test suite is a set of test cases.*

The *success* of a test is determined by the verdict associated to the result of its *execution* on the system, *pass* or *fail*, the pass verdict meaning that the result of the test is consistent with the specification according to the conformance relation. As IOLES can be seen as occurrence nets, we can model the test execution as the parallel composition of labeled nets [20]. This execution leads to a fail verdict in the following situations: (1) the implementation produces a pomset of outputs that the test case can not accept, (2) the test case can accept such a pomset of outputs, but the reached configuration is not quiescent, (3) a quiescent configuration is reached in the implementation, but not in the test case, or (4) the test case proposes an input that the implementation is not prepared to accept. These situations corresponds to a deadlock in the parallel composition, but not in the test case. If the test case deadlocks (and therefore the execution), the SUT passes the test case.

We expect our test suite to be *sound*, i.e. if the implementation fails the test, then it does not conform to the specification. A test suite is *exhaustive* iff it contains, for every non conforming implementation, a test that detects it. The existence of a *complete* (sound and exhaustive) test suite ensures *testability* of the conformance relation, since success of the SUT under such a test suite proves the SUT's conformance. For obtaining sound and exhaustive test suites, we give in [15] the following sufficient conditions. First, for a test suite to be sound, each test must produce only traces of the specification, and preserve all possible outputs for each such trace.

Theorem 1 ([15]). *Let $\mathcal{E}_s \in \mathcal{IOLES}(L)$ and T a test suite such that* [5]

1. $\forall \mathcal{E}_t \in T : \mathit{traces}(\mathcal{E}_t) \subseteq \mathit{traces}(\mathcal{E}_s)$
2. $\forall \mathcal{E}_t \in T, \omega \in \mathit{traces}(\mathcal{E}_t) : \mathit{out}_t(\perp \textbf{\textit{after}}\ \omega) = \mathit{out}_s(\perp \textbf{\textit{after}}\ \omega)$

then T is sound for $\mathcal{E}_s$ w.r.t **co-ioco***.*

A test suite is exhaustive if each trace of the specification appears in at least one test and if tests preserve quiescence.

Theorem 2 ([15]). *Let $\mathcal{E}_s \in \mathcal{IOLES}(L)$ and T a test suite such that*

1. $\forall \omega \in \mathit{traces}(\mathcal{E}_s), \exists \mathcal{E}_t \in T : \omega \in \mathit{traces}(\mathcal{E}_t)$*;*
2. $\forall \mathcal{E}_t \in T, \omega \in \mathit{traces}(\mathcal{E}_t) : (\perp_t \textbf{\textit{after}}\ \omega)$ *is quiescent implies* $(\perp_s \textbf{\textit{after}}\ \omega)$ *is quiescent;*

then T is exhaustive for $\mathcal{E}_s$ w.r.t **co-ioco***.*

The algorithm below builds a test case from an IOLES by resolving immediate conflicts between inputs, while accepting several branches in case of conflict between outputs (note that “mixed” immediate conflicts between inputs and outputs have been ruled out by Assumption 2). At the end of the algorithm, all such conflicts have been resolved in one way, following one fixed strategy of resolution of immediate input conflicts; the resulting object, the test case, is thus one branching prefix of the IOLES. In order to cover the other branches, the algorithm must be run several times with *different* conflict resolution schemes, to obtain a test suite that represents every possible event in at least one test case. Each such scheme can be represented as a linearization of the causality relation that specifies in which order the events are selected by the algorithm.

[5] The inclusion of possible inputs follows from point 1.

Algorithm 1 Constructs a test case from $\mathcal{E}$

```
Require: A finite and deterministically labeled $\mathcal{E} = (E, \leq, \#, \lambda) \in \mathcal{IOLES}(L)$ such that
    $\forall e \in E^{\mathcal{I}}, e' \in E^{\mathcal{O}} : \neg(e \#^{\mu} e')$ and a linearization $\mathcal{R}$ of $\leq$
Ensure: A test case $\mathcal{E}_t$ such that
    $\forall \omega \in \text{traces}(\mathcal{E}_t) : \text{out}_{\mathcal{E}_t}(\perp \textbf{ after } \omega) = \text{out}_{\mathcal{E}}(\perp \textbf{ after } \omega)$
$E_t := \emptyset$
$E_{temp} := E$
while $E_{temp} \neq \emptyset$ do
    $e_m := \min_{\mathcal{R}}(E_{temp})$
    $E_{temp} := E_{temp} \setminus \{e_m\}$
    if $(\{e_m\} \times E_t^{\mathcal{I}}) \cap \#^{\mu} = \emptyset \wedge [e_m] \subseteq E_t$ then
        $E_t := E_t \cup \{e_m\}$
    end if
end while
$\leq_t := \leq \cap (E_t \times E_t)$
$\#_t := \# \cap (E_t \times E_t)$
$\lambda_t := \lambda_{|E_t}$
return $\mathcal{E}_t = (E_t, \leq_t, \#_t, \lambda_t)$
```

By the above, we need to be sure that the collection of linearizations that we use considers all resolutions of immediate input conflict, i.e. is rich enough such that there is a pair of linearizations that reverses the order in a given immediate input conflict.

Definition 8. *Fix $\mathcal{E} \in \mathcal{IOLES}(L)$, and let $\mathcal{L}$ be a set of linearizations of $\leq$. Then $\mathcal{L}$ is an* immediate input conflict saturated *set, or* iics *set, for $\mathcal{E}$ iff for all $e_1, e_2 \in E^{\mathcal{I}}$ such that $e_1 \#^{\mu} e_2$, there exist $\mathcal{R}_1, \mathcal{R}_2 \in \mathcal{L}$ with $\forall e \in [e_1] : e \mathcal{R}_1 e_2$ and $\forall e \in [e_2] : e \mathcal{R}_2 e_1$.*

Proposition 1. *Let $\mathcal{L}$ be an iics set for $\mathcal{E}$, and T the test suite obtained using Algorithm 1 with $\mathcal{L}$. Then every event $e \in E$ is represented by at least one test case $\mathcal{E}_t \in T$.*

Proof. Let T be the test suite obtained by the algorithm and $\mathcal{L}$ and suppose e is not represented by any test case in T. We have then that for every $\mathcal{E}_t \in T$ either *(i)* $e \in E^{\mathcal{I}}$ and $\{e\} \times E_t^{\mathcal{I}} \cap \#^{\mu} \neq \emptyset$ or *(ii)* $[e] \not\subseteq E_t$. If *(i)*, we have that there exists $e' \in E_t^{\mathcal{I}}$ such that $e \#^{\mu} e'$ and $e' \mathcal{R}_1 e$ (where $\mathcal{R}_1$ is the linearization used to build $\mathcal{E}_t$). By Proposition 1 we know there exist $\mathcal{R}_2 \in \mathcal{L}$ such that $\forall e'' \in [e] : e'' \mathcal{R}_2 e'$ and then we can use $\mathcal{R}_2$ to construct $\mathcal{E}'_t \in T$ such that e is represented by $\mathcal{E}'_t$ which leads to a contradiction. If *(ii)*, then there exists $e' \in [e]$ such that $\{e'\} \times E_t^{\mathcal{I}} \cap \neq \emptyset$ and the analysis is analogous to the one in *(i)*. □

Note that the size of $\mathcal{L}$ and hence of T can be bounded by the number of input events in immediate conflict, i.e. $|T| \leq 2^{\mathcal{K}}$, where $\mathcal{K} = |\#^{\mu} \cap (E^{\mathcal{I}} \times E^{\mathcal{I}})|$. Note that in the case where several input events are two by two in immediate conflict, we need fewer test cases than one per pair. For example if $e_1 \#^{\mu} e_2, e_2 \#^{\mu} e_3$ and $e_3 \#^{\mu} e_1$, we only need three linearizations, each having a different event e_i preceding the two others whose order does not matter, and therefore only three cases. Moreover, for any pair of concurrent events e **co** e', the order in which they appear in any $\mathcal{R} \in \mathcal{L}$ is irrelevant; it suffices therefore to have in $\mathcal{L}$ only one representative for any class of permutations of some set of pairwise concurrent events in $\mathcal{E}$. Therefore, the size of $\mathcal{L}$ and thus of T depends on the degree of input conflict in $\mathcal{E}$ and not on the degree of concurrency. It is known that such a performance is characteristic of methods based on partial order unfoldings.

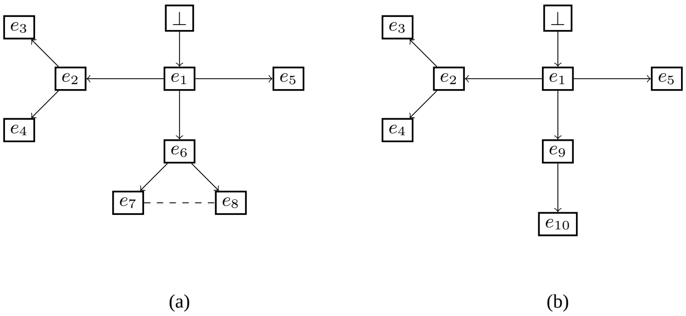


Fig. 4. Two test cases build using the IOLES in Figure 2 and Algorithm 1.

Example 1. The test cases (a) and (b) in Figure 4 can be obtained using Algorithm 1 and any linearizations $\mathcal{R}_1, \mathcal{R}_2$ such that $e_6\mathcal{R}_1e_9$ and $e_9\mathcal{R}_2e_6$.

Let $\mathcal{PREF}(\mathcal{E})$ be the set of all prefixes of $\mathcal{E}$, we show now that Algorithm 1 is general enough to produce a complete test suite from it.

Theorem 3. *From $\mathcal{PREF}(\mathcal{E})$ and an iics set $\mathcal{L}$ for $\mathcal{E}$, Algorithm 1 yields a complete test suite T.*

Proof. Soundness: By Theorem 1 we need to prove: (1) the traces of every test case are traces of the specification; (2) the outputs following a trace of the test case are preserved. (1) Trace inclusion is immediate as the test case is a prefix of the unfolding of the specification. (2) For a test $\mathcal{E}_t$ and a trace $\omega \in$ traces$(\mathcal{E}_t)$, if an output in out$_s(\perp$ **after** $\omega)$ is not in out$_t(\perp$ **after** $\omega)$, it means either that it is in conflict with an input in $\mathcal{E}_t$, which is impossible by Assumption 2, or that its past is not already in $\mathcal{E}_t$, which is impossible since ω is a trace of $\mathcal{E}_t$.

Exhaustiveness: By Theorem 2 we need to prove that every trace is represented in at least one test case, and that the algorithm does not introduce extra quiescence. Clearly, for all $\omega \in$ traces$(\mathcal{E}_s)$ there exists at least one complete prefix $c \in \mathcal{PREF}(\mathcal{E})$ such that $\omega \in$ traces(c). By Proposition 1 we can find $\mathcal{R} \in \mathcal{L}$ such that this trace remains in the test case obtain by the algorithm, i.e. $\exists t \in T : \omega \in$ traces(t). If we only consider the prefixes $c \in \mathcal{PREF}(\mathcal{E})$ such that $(\perp_c$ **after** $\omega)$ is quiescent implies $(\perp_s$ **after** $\omega)$ is quiescent, it follows that any test case built with the algorithm from c inherits this property. □

4 Coverage Criteria for Labeled Event Structures

In the **ioco** framework and its extensions, the selection of test suites is achieved by different methods. Tests can be built in a randomized way from a canonical tester, which is a completion of the specification representing all the authorized and forbidden behaviors [1]. Closer to practice is the selection of tests according to test purposes, which represent a set of behaviors one wants to test. [18]. Another method, used for symbolic transition systems for instance, is to unfold the specification until a certain testing criterion is fulfiled, and then to build a test suite covering this unfolding. Criteria for stopping the unfolding can be a given depth or state inclusion for instance [21].

The behavior of the system described by the specification consists usually of infinite traces. However, in practice, these long traces can be considered as a sequence of (finite) "basic" behaviors. For example, the travel agency offers few basic behaviors: (1) interaction with the server; (2) selection of insurance; and (3) selection of tickets. Any "complex" behavior of the agency is built from such basic behaviors. We choose a criterion allowing to cover each basic behavior described by the specification once, using a proper notion of *complete prefixes*.

Complete Prefixes as Testing Criteria. The dynamic behavior of a Petri net is entirely captured by its unfolding, but this unfolding is usually infinite. There are several different methods of truncating an unfolding. The differences are related to the kind of information about the original unfolding one wants to preserve in the prefix. Our aim is to use such a prefix to build test cases, therefore obtaining a finite prefix can be seen as defining a testing criterion.

As it is shown above, if the information about the produced outputs (and quiescence) is preserved in the test cases, we can prove the soundness of the test suite. Hence we aim at truncating the unfolding following an inclusion criterion, while preserving information about outputs and quiescence.

We say that a branching process β of an I/O Petri net Σ is *complete* if for every reachable marking M there exists a configuration C in β such that

1. $Mark(C) = M$ (i.e. M is represented in β), and
2. for every transition t enabled by M there exists $C \cup \{e\} \in \mathcal{C}(\beta)$ such that e is labeled by t.

A complete prefix Fin can be obtained modifying the unfolding algorithm. The *complete finite prefix algorithm* is presented in [16] and depends on the notion of *cut-off event*: how long the net is unfolded. The following notion corresponds to our inclusion criterion: every cycle is unfolded once.

Definition 9. *Let Fin be a branching process. An event e is a cut-off event iff Fin contains an event $e' \leq e$ such that $Mark([e']) = Mark([e])$.*

Nevertheless, as explained in Example 2, completeness does not imply that the information about outputs and quiescence is preserved.

Example 2. Consider Figure 5, we have that Fin is complete, but the expected outputs are not part of the prefix. We expect that o_1 is produced by the system after i_2 and i_4,

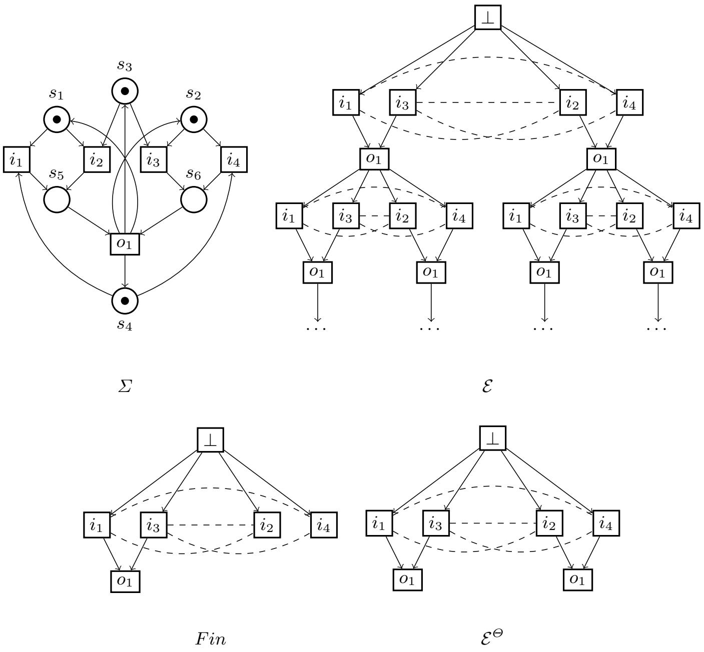


Fig. 5. I/O Petri net Σ, part of its unfolding $\mathcal{E}$, a complete finite prefix Fin and its quiescent closure $\mathcal{E}^{\Theta}$.

i.e. $\text{out}_{\mathcal{E}}(\bot$ **after** $(i_2 \cdot i_4)) = \{o_1\}$, but this is not the case in Fin, i.e. $o_2 \notin \text{out}_{Fin}(\bot$ **after** $(i_2 \cdot i_4)) = \{\delta\}$.

In order to preserve this information, we follow [21] and modify the complete finite prefix algorithm adding all the outputs from the unfolding that the complete prefix enables. As there exists no cycles of outputs in the original net, this procedure terminates, yielding a finite prefix. The procedure to compute the *quiescent closure* $\mathcal{E}^{\Theta}$ of the complete finite prefix is described by Algorithm 2.

As in [16], we implement a branching process of an I/O Petri net Σ as a list of nodes. A node is either a condition or an event. A condition is a pair (s, e), where s is a place of Σ and e its preset. An event is a pair (t, B), where t is a transition in Σ, and B is its preset. The possible extensions of a branching process β are the pairs (t, B) where the elements of B are pairwise in **co** relation, t is such that $\varphi(B) = {}^{\bullet}t$ and β contains no event e satisfying $\varphi(e) = t$ and ${}^{\bullet}e = B$. We denote the set of possible extensions of β by $PE(\beta)$. The following result is central and will help proving soundness of the test suites proposed below.

Theorem 4. *Let $\mathcal{E} \in \mathcal{IOLES}(L)$ and $\mathcal{E}^{\Theta}$ the quiescent closure of its complete finite prefix. Then*

1. *traces*$(\mathcal{E}^{\Theta}) \subseteq$ *traces*$(\mathcal{E})$
2. $\forall \omega \in$ *traces*$(\mathcal{E}^{\Theta})$: *out*$_{\mathcal{E}^{\Theta}}(\bot$ ***after*** $\omega) =$ *out*$_{\mathcal{E}}(\bot$ ***after*** $\omega)$

Algorithm 2 The quiescent closure of the complete finite prefix algorithm

```
Require: A 1-safe I/O Petri net $\Sigma = (T, P, F, M_0, \lambda)$ where $M_0 = \{s_1, \ldots, s_k\}$.
Ensure: A complete finite prefix $\mathcal{E}^\Theta$ of the unfolding $\mathcal{E}$ of $\Sigma$ such that
    $\forall \omega \in \text{traces}(\mathcal{E}^\Theta) : \text{out}_{\mathcal{E}^\Theta}(\bot \textbf{ after } \omega) = \text{out}_{\mathcal{E}}(\bot \textbf{ after } \omega)$
$\mathcal{E}^\Theta := (s_0, \emptyset), \ldots, (s_k, \emptyset)$
$pe := PE(\mathcal{E}^\Theta)$
cut-off := $\emptyset$
while $pe \neq \emptyset$ do
    choose an event $e = (t, B)$ in $pe$ such that $e$ is minimal w.r.t $\leq$;
    if $[e] \cap$ cut-off $= \emptyset$ then
        append to $\mathcal{E}^\Theta$ the event $e$ and a condition $(s, e)$ for every place $s$ in $t^\bullet$
        $pe := PE(\mathcal{E}^\Theta)$;
        if $e$ is a cut-off event of $\mathcal{E}^\Theta$ then
            cut-off := cut-off $\cup \{e\}$
        end if
    else
        $pe := pe \backslash \{e\}$
    end if
end while
$pe := PE(\mathcal{E}^\Theta)$
while $pe \cap T^\mathcal{O} \neq \emptyset$ do
    choose an event $e = (t, B)$ in $pe \cap T^\mathcal{O}$ such that $e$ is minimal w.r.t $\leq$;
    append to $\mathcal{E}^\Theta$ the event $e$ and a condition $(s, e)$ for every place $s$ in $t^\bullet$
    $pe := PE(\mathcal{E}^\Theta)$;
end while
return $\mathcal{E}^\Theta$
```

Proof. 1) is immediate since $\mathcal{E}^\Theta$ is a prefix of $\mathcal{E}$. Since only the outputs produced after the traces of $\mathcal{E}^\Theta$ are considered, 2) follows by its construction. □

The test suite build based on the inclusion criteria is sound:

Theorem 5. *Let Σ_s be the specification of a system and $\mathcal{E}_s$ the IOLES of its unfolding. Any test suite constructed using Algorithm 1 and $\mathcal{E}_s^\Theta$ as an input is sound for $\mathcal{E}_s$ w.r.t* **co-ioco**.

Proof. By Theorem 1 we need to prove that any trace of a test case $\mathcal{E}_t$ is a trace of $\mathcal{E}_s$ (which is trivial as $\mathcal{E}_t$ is a prefix of $\mathcal{E}_s^\Theta$ and therefore of $\mathcal{E}_s$) and that outputs and quiescence produced after any trace ω of such a test are preserved. The events of $\mathcal{E}_s^\Theta$ that are added to $\mathcal{E}_t$ are all the events whose past is already in $\mathcal{E}_t$ and which are not in immediate conflict with an input. An output cannot be in immediate conflict with an input by Assumption 2, so all the outputs whose past is already in $\mathcal{E}_t$ are added. So all the outputs from $\mathcal{E}_s^\Theta$ after a trace ω are preserved and by Theorem 4 we have $\forall \omega \in \text{traces}(\mathcal{E}_t) : \text{out}_t(\bot \textbf{ after } \omega) = \text{out}_s(\bot \textbf{ after } \omega)$. □

Example 3. The IOLES of Figure 2 is a complete prefix of the unfolding of the net in Figure 1 and can be obtained using Algorithm 2. We saw in Example 1 how to build test cases that cover such a complete prefix. Thus the test cases of Figure 4 form a sound test suite that covers the specification according to our inclusion criterion.

5 Conclusion and Future Work

We have presented a testing framework and a test generation algorithm for true concurrency specifications of distributed and concurrent systems. Our test selection criterion is based on the quiescent closure of the complete finite prefix of the unfolding of the specification; it allows to select, among all possible test cases, those covering the behaviors traversing each cycle once. As in the case of McMillan's complete prefixes, the size of our prefixes can be exponential in the number of reachable markings in worst case (see for example [16]). However, for several families of nets, the resulting prefix is smaller than the reachability graph. Full information about the behavior of the net can be reconstructed with only a finite marking-complete prefix whose size is bounded by the number of states in the reachability graph. However such reconstruction is not straight forwards.

Future technical studies include the question whether it is possible to drop assumptions 1 and 2 under a fairness assumption, meaning that in a given configuration, all the different events will eventually occur if the experiment is repeated enough times. However under such an assumption, controllability of test cases must be ensured during their construction.

The present testing approach here is global, meaning that a global control and observation of the distributed system is assumed, and tests are performed in a centralized way. The next step of our work is to distribute control and observation over several concurrent components. This will necessarily weaken the conformance relation, since dependencies between events occurring on different components cannot be observed anymore. The local test cases should, roughly speaking, be projections of the global test cases onto the different components, since concurrency of the specification was preserved in the test cases. We still have to investigate how distribution affects the power of testing, and how the resulting methods compares to others, such as the **dioco** framework of Hierons et al. [7] for multi-port IOTS.

Acknowledgment: This work was funded by the DIGITEO / DIM-LSC project TECSTES, convention DIGITEO Number 2011-052D - TECSTES.

References

1. Tretmans, J.: Test generation with inputs, outputs and repetitive quiescence. Software - Concepts and Tools **17**(3) (1996) 103–120
2. Heerink, L., Tretmans, J.: Refusal testing for classes of transition systems with inputs and outputs. In: Formal Description Techniques for Distributed Systems and Communication Protocols. Volume 107 of IFIP Conference Proceedings., Chapman & Hall (1997) 23–38
3. Lestiennes, G., Gaudel, M.C.: Test de systèmes réactifs non réceptifs. Journal Européen des Systèmes Automatisés **39**(1-2-3) (2005) 255–270
4. Faivre, A., Gaston, C., Le Gall, P., Touil, A.: Test purpose concretization through symbolic action refinement. In: Testing of Software and Communicating Systems. Volume 5047 of LNCS., Springer (2008) 184–199
5. Jéron, T.: Symbolic model-based test selection. Electronic Notes in Theoretical Computer Science **240** (2009) 167–184

6. Krichen, M., Tripakis, S.: Conformance testing for real-time systems. Formal Methods in System Design **34**(3) (2009) 238–304
7. Hierons, R.M., Merayo, M.G., Núñez, M.: Implementation relations for the distributed test architecture. In: Testing of Software and Communicating Systems. Volume 5047 of LNCS., Springer (2008) 200–215
8. Hennessy, M.: Algebraic Theory of Processes. MIT Press (1988)
9. Peleska, J., Siegel, M.: From testing theory to test driver implementation. In: Formal Methods Europe. Volume 1051 of LNCS., Springer (1996) 538–556
10. Schneider, S.: Concurrent and Real Time Systems: The CSP Approach. 1st edn. John Wiley & Sons, Inc., New York, NY, USA (1999)
11. Ulrich, A., König, H.: Specification-based testing of concurrent systems. In: Formal Description Techniques for Distributed Systems and Communication Protocols. Volume 107 of IFIP Conference Proceedings., Chapman & Hall (1998) 7–22
12. von Bochmann, G., Haar, S., Jard, C., Jourdan, G.V.: Testing systems specified as partial order input/output automata. In: Testing of Software and Communicating Systems. Volume 5047 of LNCS., Springer (2008) 169–183
13. Haar, S., Jard, C., Jourdan, G.V.: Testing input/output partial order automata. In: Testing of Software and Communicating Systems. Volume 4581 of LNCS., Springer (2007) 171–185
14. Ponce de León, H., Haar, S., Longuet, D.: Conformance relations for labeled event structures. In: Tests and Proofs. Volume 7305 of LNCS., Springer (2012) 83–98
15. Ponce de León, H., Haar, S., Longuet, D.: Model-based testing for concurrent systems with labeled event structures. http://hal.inria.fr/hal-00796006 (2012)
16. Esparza, J., Römer, S., Vogler, W.: An improvement of McMillan's unfolding algorithm. In: Tools and Algorithms for Construction and Analysis of Systems. Volume 1055 of LNCS., Springer (1996) 87–106
17. Segala, R.: Quiescence, fairness, testing, and the notion of implementation. Information and Computation **138**(2) (1997) 194–210
18. Jard, C., Jéron, T.: TGV: theory, principles and algorithms. International Journal on Software Tools for Technology Transfer **7** (2005) 297–315
19. Nielsen, M., Plotkin, G.D., Winskel, G.: Petri nets, event structures and domains, part I. Theoretical Computer Science **13** (1981) 85–108
20. Winskel, G.: Petri nets, morphisms and compositionality. In: Applications and Theory in Petri Nets. (1985) 453–477
21. Gaston, C., Le Gall, P., Rapin, N., Touil, A.: Symbolic execution techniques for test purpose definition. In: Testing of Software and Communicating Systems. Volume 3964 of LNCS., Springer (2006) 1–18