Conformance Relations for Labeled Event Structures

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. We propose a theoretical framework for testing concurrent systems from true concurrency models like Petri nets or networks of automata. The underlying model of computation of such formalisms are labeled event structures, which allow to represent concurrency explicitly. The activity of testing relies on the definition of a conformance relation that depends on the observable behaviors on the system under test, which is given for sequential systems by ioco type relations. However, these relations are not capable of capturing and exploiting concurrency of non sequential behavior. We study different conformance relations for labeled event structures, relying on different notions of observation, and investigate their properties and connections.

1 Introduction

This paper aims at laying the foundations of a systematic study of conformance relations for specifications that integrate features of concurrent behavior. Our ultimate goal is to lift conformance testing and its formal tools to the level of *event structure semantics*, where it currently focusses on *sequential* actions.

The present state of the art: a sequential picture. In fact, one of the most popular formalisms studied in conformance testing is that of *labeled transition systems* (LTS). A labeled transition system is a structure consisting of states and transitions labeled with actions from one state to another. This formalism is usually used for modeling the behavior of processes and as a semantical model for various formal languages such as CCS [1], CSP [2], SDL [3] and LOTOS [4]. Depending on the nature of the possible observations, different conformance relations have been defined for labeled transitions systems [5–11]; we will study how these lift to the "concurrent world". Several developments were built on the relation of *trace preorder* (trace inclusion). Firstly, it was refined into the *testing preorder*, that requires not only the inclusion of the implementation traces in those of the specification, but also that any action refused by the implementation should be refused by the specification [5, 12]. A practical modification of the testing preorder was presented in [7], which proposed to base the observations on the traces of the specification only, leading to a weaker relation

called **conf**. A further refinement concerns the inclusion of quiescent traces as a conformance relation (e.g. Segala [10]). Moreover, Tretmans [11] proposed the **ioco** relation: each output produced by the implementation on specified stimuli should corresponde to the specified ones, and the implementation is authorised to reach a state where it cannot produce any output only if this is the case in the specification too.

Shifting to concurrent specifications. However, this framework does not yet very well support the testing from *concurrent* specifications, in which some pairs of events can be specified to occur in arbitrary order, or jointly. The exhaustive testing of all interleavings for a set of concurrent transitions is prohibitively slow, and is also conceptually inadequate; for both reasons, our aim is to provide a generalized framework that handles *true concurrency* in partially ordered models. The first major steps in this direction had been made in [13, 14]: partially ordered patterns of input/output events were admitted as transition labels in a generalized I/O-automaton model, leading to a generalization of the basic notions and techniques of I/O-sequence based conformance testing. An important *practical* benefit of true-concurrency models here is an overall *complexity reduction*, despite the fact that checking partial orders requires in general multiple passes through the same labelled transition, so as to check for presence/absence of specified order relations between input and output events. In fact, if the system has n parallel and interacting processes, the length of checking sequences increases by a factor that is polynomial in n. At the same time, the overall size of the automaton model (in terms of the number of its states and transitions) shrinks *exponentially* if the concurrency between the processes is explicitly modeled. This feature indicates that with increasing size and distribution of SUTs in practice, it is computationally wise to seek alternatives for the direct sequential modeling approach. We add that true concurrency models are not only promising for practical reasons, but also are more adequate in reflecting the actual structure of distributed systems, and tend to be more accessible for designers and implementers, in particular if modularity can be exploited.

As indicated above, the work presented in [13, 14] presents a first step towards a concurrency-based conformance theory. The partial-order I/O automata models developed there progress with respect to global state models such as multiport I/O-Automata by specifying dependence relations *across* processes explicitly, and allow to specify natural conditions that avoid e.g. controllability violations. However, the models of [13, 14] still force us to maintain a sequential automaton as the system's skeleton, and to include synchronization constraints (typically: that all events specified in the pattern of a transition must be completed before any other transition can start), which limit both the application domain and the benefits from concurrency modeling. In other work in progress, we abandon automata altogether and focus on *Petri nets* as system models, which allows to completely discard any *global* synchronizations, and to exploit existing theory of concurrent behavior for devising testing strategies.

The present article provides the *semantic* viewpoint which accompanies and complements that shift in *systems* modeling. We use throughout a canonical

semantic model for concurrent behavior, *labeled event structures*, providing a unifying semantic framework for system models such as Petri nets, communicating automata, or process algebras; we abstract away from the particularities of system specification models, to focus entirely on behavioral relations.

The underlying mathematical structure for the system semantics is given by *event structures* in the sense of Winskel et al [15]. Mathematically speaking, they are particular partially ordered sets, in which order between events e and e' indicates precedence, and where any two events e and e' that are *not* ordered maybe either

- in *conflict*, meaning that in any evolution of the system in which e occurs, e' *cannot* occur; or
- *concurrent*, in which case they may occur in the same system run, without a temporal ordering, i.e. e may occur before e', after e', or simultaneously.

The state reached after some execution is represented by a *configuration* of the event structure, that is a conflict-free, history-closed set. The use of partial order semantics provides richer information and finer system comparisons than the interleaved view.

Overview. The paper is organized as follows: Section 2 gives the fundamental definitions of the semantic model of labeled event structures and Sect. 3 gives two definitions of observation of processes. Then, Sect. 4 introduces and studies conformance relations for *general* labeled event structures, and Sect. 5 specializes to *I/O systems* in which the label set is split into *input* and *output* labels, and introduces a new, true-concurrency-enabled **ioco** relation. Section 6 discusses the advantages and drawbacks of the conformance relations presented, and concludes.

2 Labeled Event Structures

We shall be using event structures following Winskel et al [15] to describe the dynamic behavior of a system. In this paper we will consider only prime event structures [16], a subset of the original model which is sufficient to describe concurrent models (therefore we will simply call them event structures), and we label their events with actions over a fixed alphabet L.

Definition 1 (Labeled event structure). *A* labeled event structure *over an alphabet L is a 4-tuple $\mathcal{E} = (E, \leq, \#, \lambda)$ such that*

- *E is a set of events,*
- *$\leq \subseteq E \times E$ is a partial order (called* causality*) satisfying the property of* finite causes*, i.e. $\forall e \in E : |\{e' \in E \mid e' \leq e\}| < \infty$,*
- *$\# \subseteq E \times E$ is an irreflexive symmetric relation (called* conflict*) satisfying the property of* conflict heredity*, i.e. $\forall e, e', e'' \in E : e \# e' \wedge e' \leq e'' \Rightarrow e \# e''$,*
- *$\lambda : E \to L$ is a labeling mapping.*

We denote the class of all labeled event structures over L by $\mathcal{LES}(L)$.

Given a labeled event structure $\mathcal{E} = (E, \leq, \#, \lambda) \in \mathcal{LES}(L)$, two events $e, e' \in E$ are said to be *concurrent*, written e **co** e', iff neither $e \leq e'$ nor $e' \leq e$ nor $e \# e'$ hold.

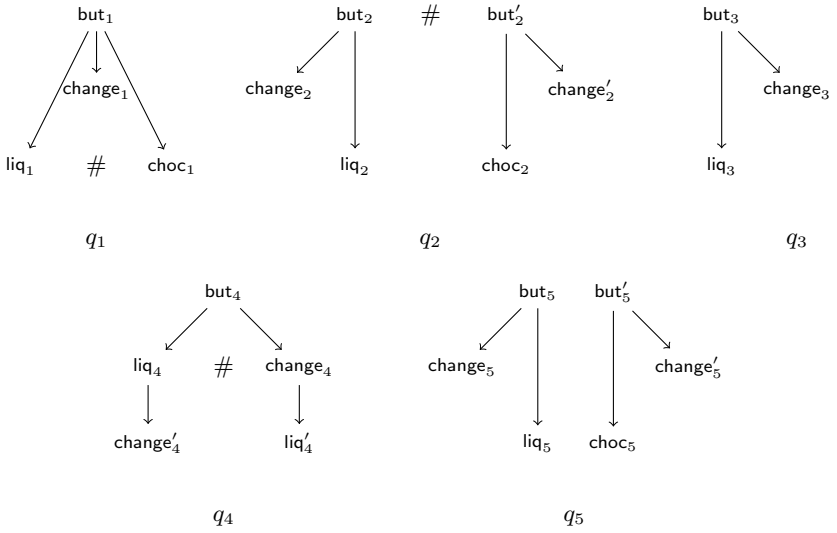

Fig. 1. Labeled event structures

Example 1. Fig. 1 presents different LES specifications of vending machines. The requirements are the following: when one pushes a button, the machine delivers chocolate bars or liquorices, and supplies change. We represent causality between events by the Hasse diagram of $\leq$, and direct conflict by $\#$. The labeling λ is such that $\lambda(e_i) = \lambda(e'_i) = e$.

Machine q_1 to q_4 have only one button while machine q_5 has two of them. In machines q_1 and q_2, a choice is made between supplying liquorice or chocolate after pressing the button, and concurrently, the machines supply change. The choice is made when the button is pushed in machine q_2 but internally after the pressing of the button in machine q_1. Machine q_3 only supplies liquorice and change concurrently while q_4 do both, but in a sequential way. We can press concurrently two different buttons in q_5, each of them producing liquorice or chocolate and supplying change.

A computation state of an event structure is called a configuration and is represented by the set of events that have occurred in the computation. If an event is present in a configuration, then so are all the events on which this event causally depends. Moreover, a configuration obviously does not contain conflicting events.

Definition 2 (Configuration). *Let* $\mathcal{E} = (E, \leq, \#, \lambda) \in \mathcal{LES}(L)$*, a* configuration *of* $\mathcal{E}$ *is a set of events* $C \subseteq E$ *such that*

- C *is causally closed:* $e \in C \Rightarrow \forall e' \leq e : e' \in C$*, and*
- C *is conflict-free:* $\forall e, e' \in C : \neg(e \# e')$.

The initial configuration of $\mathcal{E}$*, denoted by* $\bot_{\mathcal{E}}$*, is the empty set of events. We denote the set of all the configurations of* $\mathcal{E}$ *by* $\mathcal{C}(\mathcal{E})$.

Example 2. The configurations of the labeled event structure q_1 of Fig. 1 are $\bot_{q_1}$, $\{\mathsf{but}_1\}$, $\{\mathsf{but}_1, \mathsf{liq}_1\}$, $\{\mathsf{but}_1, \mathsf{change}_1\}$, $\{\mathsf{but}_1, \mathsf{choc}_1\}$, $\{\mathsf{but}_1, \mathsf{liq}_1, \mathsf{change}_1\}$, and $\{\mathsf{but}_1, \mathsf{choc}_1, \mathsf{change}_1\}$. It is worth noting that the configurations of q_1 and q_2 are different but their λ-images are the same.

A particular kind of event structures are those representing only sequential behaviors, i.e. without concurrency. A labeled event structure is called *sequential* iff there are no pairs of concurrent events in it: $\leq \cup \# = E \times E$. Sequential event structures can be seen as the computation trees obtained by unfolding labeled transition systems [17]. In Fig. 1, q_4 is a sequential labeled event structure.

3 Observing Event Structures

The next sections will present several conformance relations over labeled event structures. These relations are based on the chosen notion of observation of the system behavior in response to stimuli. The observations most studied in the literature for defining conformance relations are (execution) traces and refusals.

The definition of the notion of trace for a labeled event structure is not straightforward since it relies on the chosen semantics for concurrency [18]. The presence of explicit concurrency in a specification may be interpreted in several ways. In an early stage of specification, concurrency between events may be used as underspecification, leaving the choice of the actual order between events to the developper. The events specified as concurrent may then occur in any order in the implementation (maybe always the same one). In the specification of a distributed system however, concurrent events in the specification may be meant to remain concurrent in the implementation, because they are destined to occur in different components executed in parallel for instance.

We follow here two established semantics for concurrency, namely *interleaving* semantics where concurrent events may be executed in any order, and *partial order* semantics where no order is wanted or can be observed between concurrent events. In the first case, observing the behavior of the system action by action is sufficient since concurrent events will be observed sequentially. In the second case, several concurrent events may be observed together in one step, since they are not ordered. This leads to two definitions of traces for labeled event structures.

3.1 Single Action Observations

In this first setup, one considers *atomic* experiments on a system as single actions, and obtains an interleaving semantics for concurrency.

Definition 3. *Let* $\mathcal{E} = (E, \leq, \#, \lambda) \in \mathcal{LES}(L)$, $a \in L$, $\sigma = \sigma_1 \cdot \sigma_2 \cdot \ldots \cdot \sigma_n \in L^+$ *and* $C, C' \in \mathcal{C}(\mathcal{E})$, *we define*

$$
\begin{array}{ll}
C \stackrel{a}{\Longrightarrow} C' & \triangleq \exists e \in E \backslash C : C' = C \cup \{e\} \text{ and } \lambda(e) = a \\
C \stackrel{a}{\Longrightarrow} & \triangleq \exists C' : C \stackrel{a}{\Longrightarrow} C' \\
C \stackrel{\sigma}{\Longrightarrow} C' & \triangleq \exists C_0, \ldots C_n : C = C_0 \stackrel{\sigma_1}{\Longrightarrow} C_1 \stackrel{\sigma_2}{\Longrightarrow} \ldots \stackrel{\sigma_n}{\Longrightarrow} C_n = C' \\
C \stackrel{\sigma}{\Longrightarrow} & \triangleq \exists C' : C \stackrel{\sigma}{\Longrightarrow} C'
\end{array}
$$

One goes from a configuration to another by performing only one action at a time, thus leading to a trace semantics where an execution is a sequence of single actions (obviously, the empty sequence leads to the same configuration, i.e. $C \stackrel{\epsilon}{\Longrightarrow} C$).[3] Possible observations of the system behavior are captured by the following definition.

Definition 4. *Let* $\mathcal{E} \in \mathcal{LES}(L)$, $A \subseteq L$, $\sigma \in L^*$, $S \subseteq \mathcal{C}(\mathcal{E})$ *and* $C, C' \in \mathcal{C}(\mathcal{E})$, *we define*

- $\mathit{traces}(\mathcal{E}) \triangleq \{\sigma \in L^* \mid \bot_{\mathcal{E}} \stackrel{\sigma}{\Longrightarrow}\}$
- C ***after*** $\sigma \triangleq \{C' \mid C \stackrel{\sigma}{\Longrightarrow} C'\}$
- C ***refuses*** $A \triangleq \forall a \in A : C \stackrel{a}{\not\Longrightarrow}$
- S ***refuses*** $A \triangleq \exists C \in S : C$ ***refuses*** A

The set traces($\mathcal{E}$) contains the full action sequences of $\mathcal{E}$, while C **after** σ contains the possible configurations reached from C when σ was observed. Refusal of an action set A means the impossibility of executing any transition with a label in A. In the next section we will use **refuses** together with **after**, and as the system can reach several configurations after σ, we extend **refuses** to sets of configurations.

Example 3. With this interleaving semantics, the traces of machine q_3 in Fig. 1 are $\{\epsilon, \mathsf{but}, \mathsf{but} \cdot \mathsf{liq}, \mathsf{but} \cdot \mathsf{change}, \mathsf{but} \cdot \mathsf{liq} \cdot \mathsf{change}, \mathsf{but} \cdot \mathsf{change} \cdot \mathsf{liq}\}$, since concurrent events may be seen in any order. Therefore, machines q_3 and q_4 have the same traces. Due to the inheritance of conflict, machines q_1 and q_2 also have the same traces since after but, one can perform liq and change in any order, or choc and change in any order.

Concerning refusals, one can see that machine q_1 cannot produce chocolate after producing liquorice, i.e. ($\bot_{q_1}$ **after** $\mathsf{but} \cdot \mathsf{liq}$) **refuses** $\{\mathsf{choc}\}$. Note that S **refuses** A is false when S is empty, therefore ($\bot_{q_3}$ **after** $\mathsf{but} \cdot \mathsf{choc}$) **refuses** $\emptyset$ is false since $\mathsf{but} \cdot \mathsf{choc}$ is not a trace of q_3.

[3] We denote by ϵ the empty word and by $\_ \cdot \_$ the concatenation of words in L^*.

3.2 Partially Ordered Observations

Since the event structure model is capable of explicitly distinguishing the causal structure of the model, it is natural to expect the observations of machines q_3 and q_4 of Fig. 1 to be different; in fact, actions liq and change are independent in q_3, but they are causally ordered in q_4.

In order to distinguish such behaviors, the notion of trace must keep concurrency explicit, i.e. must preserve the partial order of the events of an execution. We first recall the notion of labeled partial order, then we lift Def. 3 and Def. 4 to the partial order setting.

Definition 5 (Labeled partial order). *A* labeled partial order *over L is a tuple $\omega = (E_\omega, \leq_\omega, \lambda_\omega)$, where*

- *$(E_\omega, \leq_\omega)$ is a partial order, and*
- *$\lambda_\omega : E_\omega \to L$ is a labeling mapping.*

We denote the class of all labeled partial orders over L by $\mathcal{LPO}(L)$.

Labeled partial orders will be used to represent observations of executions containing concurrent events. Moreover, we will need the notion of *labeled concurrent set* to represent a set of concurrent events: we say that $\alpha \in \mathcal{LPO}(L)$ is a labeled concurrent set over L iff $<_\alpha = \emptyset$, and denote the class of such objects by $\mathcal{CO}(L)$.

In partial order semantics, a step of an execution from a given configuration may be a single action or a set of actions performed concurrently. This leads to the following definitions. We indicate by a subscript or superscript π the relations and sets to be interpreted in the partial order semantics.

Definition 6. *Let $\mathcal{E} = (E, \leq, \#, \lambda) \in \mathcal{LES}(L)$, $\alpha = (E_\alpha, \leq_\alpha, \lambda_\alpha) \in \mathcal{CO}(L)$, $\omega = (E_\omega, \leq_\omega, \lambda_\omega) \in \mathcal{LPO}(L)$ and $C, C' \in \mathcal{C}(\mathcal{E})$, we define*

$$
\begin{aligned}
C \overset{\alpha}{\Longrightarrow}_\pi C' &\triangleq \exists A \subseteq E \backslash C : C' = C \cup A, A = E_\alpha, \\
&\qquad <_{|A \times A} = \emptyset \text{ and } \lambda_{|A} = \lambda_\alpha \\
C \overset{\alpha}{\Longrightarrow}_\pi &\triangleq \exists C' : C \overset{\alpha}{\Longrightarrow}_\pi C' \\
C \overset{\omega}{\Longrightarrow}_\pi C' &\triangleq \exists A \subseteq E \backslash C : C' = C \cup A, A = E_\omega, \\
&\qquad \leq_{|A \times A} = \leq_\omega \text{ and } \lambda_{|A} = \lambda_\omega \\
C \overset{\omega}{\Longrightarrow}_\pi &\triangleq \exists C' : C \overset{\omega}{\Longrightarrow}_\pi C'
\end{aligned}
$$

The ability of making concurrent execution explicit is the key advantage in using partial order semantics.

Definition 7. *Let $\mathcal{E} \in \mathcal{LES}(L)$, $A \subseteq \mathcal{CO}(L)$, $\omega \in \mathcal{LPO}(L)$, $S \subseteq \mathcal{C}(\mathcal{E})$ and $C, C' \in \mathcal{C}(\mathcal{E})$, we define*

- *$traces_\pi(\mathcal{E}) \triangleq \{\omega \in \mathcal{LPO}(L) \mid \bot_\mathcal{E} \overset{\omega}{\Longrightarrow}_\pi\}$*
- *C **after**$_\pi$ $\omega \triangleq \{C' \mid C \overset{\omega}{\Longrightarrow}_\pi C'\}$*

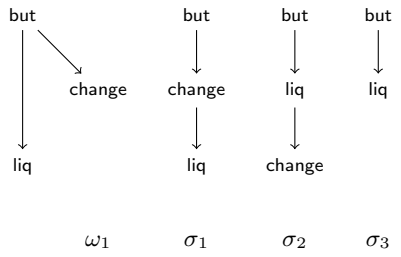

Fig. 2. Partially ordered observations vs. sequential observations

- C ***refuses***$_\pi$ $A \triangleq \forall \alpha \in A : C \stackrel{\alpha}{\not\Longrightarrow}_\pi$
- S ***refuses***$_\pi$ $A \triangleq \exists C \in S : C$ ***refuses***$_\pi$ A

Example 4. We consider labeled partial orders of Fig 2. We can observe liq and change concurrently after but in machine q_1 of Fig. 1, so we have $\omega_1 \in \text{traces}_\pi(q_1)$, but we cannot observe them concurrently in q_4 because the system only allows to see them ordered, thus $\omega_1 \notin \text{traces}_\pi(q_4)$.

In the other way round, causality between liq and change is desired in q_4 but cannot be observed or is not wanted in q_1, so $\sigma_1 \in \text{traces}_\pi(q_4)$ and $\sigma_1 \notin \text{traces}_\pi(q_1)$.

Prefixes are also allowed, we have for instance $\sigma_3 \in \text{traces}_\pi(q_1)$.

Note that in a sequential labeled event structure $\mathcal{E}$, since there is no concurrency, we have $\text{traces}_\pi(\mathcal{E}) = \text{traces}(\mathcal{E})$. Since $L \subseteq \mathcal{CO}(L)$ and $L^* \subseteq \mathcal{LPO}(L)$, Def. 6 and Def. 7 are strict generalizations of Def. 3 and Def. 4.

Remark 1. By abuse of notation, we will use indistinctly σ_1 and but $\cdot$ change $\cdot$ liq or ω_1 and but $\cdot$ (liq **co** change).

4 Conformance Relations for Concurrent Systems

The objective of this paper is to propose a generalization of the **ioco** relation [11]. We first propose generalizations of the conformance relations defined in the literature for systems with symmetric interactions, i.e. where inputs and outputs are not differentiated. We follow the presentation and notations adopted in [11].

The first relation proposed in the literature, called *trace preorder*, is based on the inclusion of the executions of the system under test in those allowed by the specification. The intuition is that an implementation i should not exhibit any unspecified sequence of actions, i.e. not present in the specification s.

Definition 8 (Trace preorder for single action observation). *Let* $i, s \in \mathcal{LES}(L)$*, then*

$$i \leq_{tr} s \Leftrightarrow traces(i) \subseteq traces(s)$$

Example 5. With the interleaving semantics, the traces of q_1 and q_2 are the same, and we have $q_1 \leq_{\text{tr}} q_2$ and $q_2 \leq_{\text{tr}} q_1$. Analogously, $q_3 \leq_{\text{tr}} q_4$ and $q_4 \leq_{\text{tr}} q_3$. The systems q_3 and q_4 implement part of what is specified in q_1 and q_2, therefore $q_3, q_4 \leq_{\text{tr}} q_1, q_2$.

This relation is very similar to the trace preorder for labeled transition systems since it is based on the observation of sequences of actions. On the contrary, the adaptation of the trace preorder to labeled event structures with the partial order semantics leads to a new conformance relation.

Definition 9 (Trace preorder for partially ordered observation). *Let $i, s \in \mathcal{LES}(L)$, then*

$$i \leq^{\pi}_{tr} s \Leftrightarrow \mathit{traces}_{\pi}(i) \subseteq \mathit{traces}_{\pi}(s)$$

Example 6. Since $\text{traces}_{\pi}(q_1) = \text{traces}_{\pi}(q_2)$, we have $q_1 \leq^{\pi}_{\text{tr}} q_2$ and $q_2 \leq^{\pi}_{\text{tr}} q_1$. We also have $q_3 \leq^{\pi}_{\text{tr}} q_1$ and $q_3 \leq^{\pi}_{\text{tr}} q_2$, but $q_4 \not\leq^{\pi}_{\text{tr}} q_3$ because $\leq^{\pi}_{\text{tr}}$ requires concurrent events in the specification to remain truly concurrent in the implementation and does not accept any order between them as it is the case of $\leq_{\text{tr}}$. The traces of q_2 are observable in q_5, but q_5 accepts more behaviors since the second button can still be pushed after the first one was, so $q_2 \leq^{\pi}_{\text{tr}} q_5$ but $q_5 \not\leq^{\pi}_{\text{tr}} q_2$.

With both relations, we have that q_2 correctly implements q_1, but q_1 specifies that after pressing a button the user has a choice between liquorice and chocolate, while q_2 may refuse one of these. The reason of this is that both $\leq_{\text{tr}}$ and $\leq^{\pi}_{\text{tr}}$ only consider sequences (resp. partial order) of actions as observations, and not whether conflicts are resolved internally by the machine, or externally by the environment.

Therefore, we propose a stronger relation to refine $\leq^{\pi}_{\text{tr}}$. In addition to requiring that any execution of the implementation is allowed in the specification, we require that any time the implementation refuses to perform a new action, that action cannot be performed in the specification either. This new conformance relation generalizes the testing preorder of [5].[4]

Definition 10 (Testing preorder for partially ordered observation). *Let $i, s \in \mathcal{LES}(L)$, then*

$$\begin{aligned} i \leq^{\pi}_{te} s \Leftrightarrow \forall \omega \in \mathcal{LPO}(L), A \subseteq \mathcal{CO}(L) : \\ \bot_i \ \textbf{\textit{after}}_{\pi}\ \omega\ \textbf{\textit{refuses}}_{\pi}\ A \Rightarrow \bot_s \ \textbf{\textit{after}}_{\pi}\ \omega\ \textbf{\textit{refuses}}_{\pi}\ A \end{aligned}$$

Example 7. Consider again Fig. 1, we have $q_1 \leq^{\pi}_{\text{te}} q_2$: there are no trace ω and no set of events A such that $\bot_{q_1}$ $\textbf{after}_{\pi}$ ω $\textbf{refuses}_{\pi}$ A and $\neg(\bot_{q_2}$ $\textbf{after}_{\pi}$ ω $\textbf{refuses}_{\pi}$ $A)$. However, $q_2, q_3 \not\leq^{\pi}_{\text{te}} q_1$, since for instance $\bot_{q_2}$ $\textbf{after}_{\pi}$ but $\textbf{refuses}_{\pi}$ $\{\mathsf{choc}\}$ and $\neg(\bot_{q_1}$ $\textbf{after}_{\pi}$ but $\textbf{refuses}_{\pi}$ $\{\mathsf{choc}\})$. The

[4] From now on, we will present the conformance relations for the partial order semantics only. The corresponding conformance relations for the interleaving semantics can be straightforwardly deduced.

button can be pressed twice concurrently in q_5, but not in q_2, so ($\perp_{q_2}$ **after** but) **refuses** {but} and $q_2 \not\leq^{\pi}_{\text{te}} q_5$.

Note that the relation $\leq^{\pi}_{\text{te}}$ does not allow extra traces in the implementation. In fact, $q_1 \not\leq^{\pi}_{\text{te}} q_3$ since $\perp_{q_1}$ **after**$_\pi$ but $\cdot$ choc **refuses**$_\pi$ $\emptyset$, yet $\perp_{q_3}$ **after**$_\pi$ but $\cdot$ choc $= \emptyset$, hence $\neg(\perp_{q_3}$ **after**$_\pi$ but $\cdot$ choc **refuses**$_\pi$ $\emptyset)$. As but **co** but is a trace of q_5 yet not one of q_2, we have $q_5 \not\leq^{\pi}_{\text{te}} q_2$. As this relation checks for trace inclusion, it still differentiates between q_3 and q_4, i.e. $q_4 \not\leq^{\pi}_{\text{te}} q_3$

We propose a weaker relation $\mathbf{conf}_\pi$ restricting all the traces to only the ones contained in the specification. This relation requires that the implementation does what it has to do, not that it does not what it is not allowed to do. It allows underspecification, i.e. that only a subset of the functionalities of the actual system are specified.

Definition 11 (Relation conf for partially ordered observation). *Let $i, s \in \mathcal{LES}(L)$, then*

$$i \ \textbf{\textit{conf}}_\pi \ s \Leftrightarrow \forall \omega \in \mathit{traces}_\pi(s), A \subseteq \mathcal{CO}(L) : \\ \perp_i \textbf{\textit{after}}_\pi \ \omega \ \textbf{\textit{refuses}}_\pi \ A \Rightarrow \perp_s \textbf{\textit{after}}_\pi \ \omega \ \textbf{\textit{refuses}}_\pi \ A$$

Example 8. We saw in Ex. 7 that $q_1 \leq^{\pi}_{\text{te}} q_2$, and since $\mathbf{conf}_\pi$ considers the traces of q_2 only, we have q_1 $\mathbf{conf}_\pi$ q_2.

Since the relation $\mathbf{conf}_\pi$ is based on the traces of the specification only, it allows extra traces in the implementation. So even if $q_1 \not\leq^{\pi}_{\text{te}} q_3$, we have q_1 $\mathbf{conf}_\pi$ q_3. In the same way, $q_5 \not\leq^{\pi}_{\text{te}} q_2$ but q_5 $\mathbf{conf}_\pi$ q_2.

However, if all traces of the implementation are also traces of the specification, then the testing preorder is equivalent to $\mathbf{conf}_\pi$. We have $\neg(q_2$ $\mathbf{conf}_\pi$ $q_1)$ since $q_2 \not\leq^{\pi}_{\text{te}} q_1$ and $\text{traces}_\pi(q_1) = \text{traces}_\pi(q_2)$. Moreover, we have $\neg(q_3$ $\mathbf{conf}_\pi$ $q_1)$ since $q_3 \not\leq^{\pi}_{\text{te}} q_1$ and $\text{traces}_\pi(q_3) \subseteq \text{traces}_\pi(q_1)$, and also $\neg(q_2$ $\mathbf{conf}_\pi$ $q_5)$ since $q_2 \not\leq^{\pi}_{\text{te}} q_5$ and $\text{traces}_\pi(q_2) \subseteq \text{traces}_\pi(q_5)$.

The following result relates the different implementation relations in the partial order semantics.

Proposition 1

1. $\leq^{\pi}_{tr}$ *and* $\leq^{\pi}_{te}$ *are preorders;* $\textbf{\textit{conf}}_\pi$ *is reflexive.*
2. $\leq^{\pi}_{te} = \leq^{\pi}_{tr} \cap \textbf{\textit{conf}}_\pi$

Proof. Point 1 being obvious, we only show point 2, by proving that the inclusion holds in both directions. Suppose $i \not\leq^{\pi}_{\text{tr}} s$, then there exists $\omega \in \mathcal{LPO}(L)$ such that $\perp_i \stackrel{\omega}{\Longrightarrow}_\pi$, but $\perp_s \not\stackrel{\omega}{\Longrightarrow}_\pi$, thus $\perp_s$ **after**$_\pi$ $\omega = \emptyset$ and $\neg(\perp_s$ **after**$_\pi$ ω **refuses**$_\pi$ $\emptyset)$ while $\perp_i$ **after**$_\pi$ ω **refuses**$_\pi$ $\emptyset$, $i \not\leq^{\pi}_{\text{te}} s$ and finally $\leq^{\pi}_{\text{te}} \subseteq \leq^{\pi}_{\text{tr}}$. As $\mathbf{conf}_\pi$ is a restriction of $\leq^{\pi}_{\text{te}}$ to the traces of s, it is easy to prove that $\leq^{\pi}_{\text{te}} \subseteq \mathbf{conf}_\pi$.

Suppose $i \not\leq^{\pi}_{\text{te}} s$, then there exist $\omega \in \mathcal{LPO}(L), A \subseteq \mathcal{CO}(L)$ such that $\perp_i$ **after**$_\pi$ ω **refuses**$_\pi$ A and $\neg(\perp_s$ **after**$_\pi$ ω **refuses**$_\pi$ $A)$. If $\omega \in \text{traces}_\pi(s)$ we have that $\neg(i$ $\mathbf{conf}_\pi$ $s)$. If $\omega \notin \text{traces}_\pi(s)$, we know by $\perp_i$ **after**$_\pi$ ω **refuses**$_\pi$ A that $\omega \in \text{traces}_\pi(i)$ and therefore $i \not\leq^{\pi}_{\text{tr}} s$.

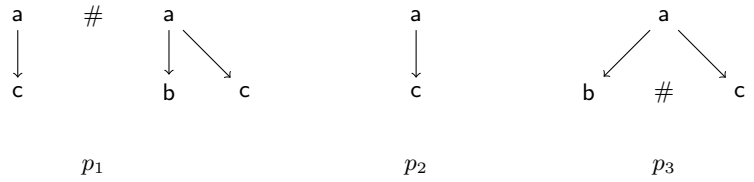

Fig. 3. The $\mathbf{conf}_\pi$ relation is not transitive.

Example 9. In Fig. 3 we can see that p_2 $\mathbf{conf}_\pi$ p_1. If we denote the set $\{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$ by L, we have $\bot_{p_2}$ $\mathbf{after}_\pi$ a $\mathbf{refuses}_\pi$ $\{\mathsf{a}, \mathsf{b}\}$ and $\bot_{p_1}$ $\mathbf{after}_\pi$ a $\mathbf{refuses}_\pi$ $\{\mathsf{a}, \mathsf{b}\}$; we also have $\bot_{p_2}$ $\mathbf{after}_\pi$ $\mathsf{a} \cdot \mathsf{c}$ $\mathbf{refuses}_\pi$ L and $\bot_{p_1}$ $\mathbf{after}_\pi$ $\mathsf{a} \cdot \mathsf{c}$ $\mathbf{refuses}_\pi$ L; finally $\bot_{p_2}$ $\mathbf{after}_\pi$ $\mathsf{a} \cdot \mathsf{b}$ $\mathbf{refuses}_\pi$ S is false for any set S. We can see that p_3 $\mathbf{conf}_\pi$ p_2 since p_3 is p_2 with an additional branch. Nevertheless we do not have p_3 $\mathbf{conf}_\pi$ p_1: we have $\bot_{p_3}$ $\mathbf{after}_\pi$ $\mathsf{a} \cdot \mathsf{b}$ $\mathbf{refuses}_\pi$ $\{\mathsf{c}\}$ but $\neg(\bot_{p_1}$ $\mathbf{after}_\pi$ $\mathsf{a} \cdot \mathsf{b}$ $\mathbf{refuses}_\pi$ $\{\mathsf{c}\})$. This shows that $\mathbf{conf}_\pi$ is not transitive.

The interested reader may verify that the relations presented in Ex. 6, Ex. 7 and Ex. 8 satisfy Prop. 1.

5 Conformance Relations for Input/Output Concurrent Systems

As usual when testing reactive systems, we want to distinguish between the controllable and observable actions of the system under test. We extend the model of labeled event structures to make a distinction between input actions (proposed by the environment) and output actions (produced by the system) of the system, leading input-output labeled event structures (IOLES).

Definition 12 (Input-output labeled event structure). *An* input-output labeled event structure *is a labeled event structure over the alphabet* $L = L_i \uplus L_o$*. The class of input-output labeled event structures over* L *is denoted by* $\mathcal{IOLES}(L)$*.*

As usual, let $?a$ denote an input action in L_i and $!a$ an output action in L_o. Examples of IOLES are given in Fig. 4.

5.1 The ioco Relation for the Interleaving Semantics

We first present the definition of the **ioco** conformance relation for labeled event structures with the interleaving semantics.

The **ioco** relation requires that, after a trace of the specification, the outputs produced by the implementation are authorized by the specification, but also the absence of outputs. A state where the system cannot produce outputs is called quiescent in the labeled transition system framework. Similarly, in the labeled event structure framework, a configuration where the system cannot produce outputs will be called quiescent.

Definition 13 (Quiescent configuration for single action observation). *Let $\mathcal{E} \in \mathcal{IOLES}(L)$. A configuration $C \in \mathcal{C}(\mathcal{E})$ is* quiescent *iff $\forall a \in L_o : C \stackrel{a}{\not\Longrightarrow}$.*

In our framework, a system is in a quiescent configuration if it is waiting for an input from the environment or it deadlocks.

As it is now standard in the LTS framework, we assume that quiescent configurations are observable by a special output action $\delta \in L_o$. The event corresponding to a δ action should be unique in the given configuration (1), and it should be in conflict with all the other possible events from the same configuration (2). Additionally, since the δ action captures the notion of not observing anything but the absence of reaction of the system, observing δ should not change the behavior of the system (3). We denote by $\Delta_\mathcal{E}$ an IOLES $\mathcal{E}$ enriched by δ such that these properties hold. Formally, we assume that every quiescent configuration $C \in \mathcal{C}(\Delta_\mathcal{E})$ has the following properties:

$$(\exists!\ e_\delta \in E \setminus C : \lambda(e_\delta) = \delta \wedge C \stackrel{\delta}{\Longrightarrow}) \tag{1}$$

$$\wedge\ (\forall e' \in E \setminus (C \cup \{e_\delta\}), C \stackrel{\lambda(e')}{\Longrightarrow}\ \Rightarrow\ e_\delta \# e') \tag{2}$$

$$\wedge\ (\forall \sigma \in L^* : C \stackrel{\sigma}{\Longrightarrow}\ \Rightarrow\ C \cup \{e_\delta\} \stackrel{\sigma}{\Longrightarrow}) \tag{3}$$

In the interleaving semantics, the way to observe the outputs of the system in response to stimuli is the same as in the LTS framework: the set of possible outputs from a given configuration is the set of every single possible output.

Definition 14 (Expected outputs for single action observation). *Let $\mathcal{E} \in \mathcal{IOLES}(L)$ and $S \subseteq \mathcal{C}(\mathcal{E})$, then*

$$out(S) \triangleq \bigcup_{C \in S} \{a \in L_o \mid C \stackrel{a}{\Longrightarrow}\}$$

We obtain the following adaptation of the **ioco** conformance relation to the labeled event structure framework with the interleaving semantics: for any trace of the specification enriched with δ actions, every single output produced by the implementation after this trace (including δ) is authorised by the specification.

Definition 15 (ioco for single action observation). *Let $i, s \in \mathcal{IOLES}(L)$, then*

$$i\ \textbf{\textit{ioco}}\ s \Leftrightarrow \forall \sigma \in traces(\Delta_s) : out(\bot_{\Delta_i}\ \textbf{\textit{after}}\ \sigma) \subseteq out(\bot_{\Delta_s}\ \textbf{\textit{after}}\ \sigma)$$

Example 10. We consider the labeled event structures of Fig. 4, s and s' being two specifications and i a possible implementation.

As we saw in previous examples, with the interleaving semantics, s and s' have the same traces: traces(s) = traces(s') = $\{\epsilon, ?\mathsf{but}, ?\mathsf{but} \cdot !\mathsf{liq}, ?\mathsf{but} \cdot !\mathsf{change}, ?\mathsf{but} \cdot !\mathsf{liq} \cdot !\mathsf{change}, ?\mathsf{but} \cdot !\mathsf{change} \cdot !\mathsf{liq}\}$ and we obtain:[5]

[5] To lighten the examples, we consider the traces of s and i only, and not all the traces of Δ_s and Δ_i, since it makes no difference in these cases.

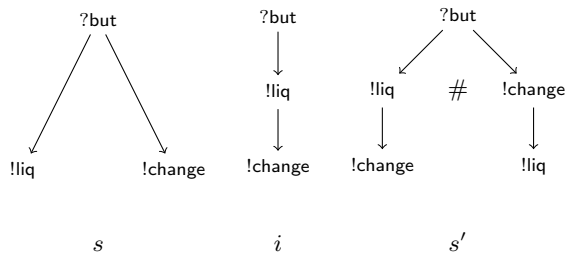

Fig. 4. Difference between **ioco** and **co-ioco**

σ	out($\perp_{\Delta_s}$ **after** σ)	out($\perp_{\Delta'_s}$ **after** σ)	out($\perp_{\Delta_i}$ **after** σ)
ϵ	$\{\delta\}$	$\{\delta\}$	$\{\delta\}$
?but	{!liq, !change}	{!liq, !change}	{!liq}
?but · !liq	{!change}	{!change}	{!change}
?but · !change	{!liq}	{!liq}	$\emptyset$
?but · !liq · !change	$\{\delta\}$	$\{\delta\}$	$\{\delta\}$
?but · !change · !liq	$\{\delta\}$	$\{\delta\}$	$\emptyset$

We conclude that both i **ioco** s and i **ioco** s' hold. Note that the fact of observing the empty set is different from observing δ. Observing δ after executing a trace σ means that the system performed σ and reached a quiescent configuration, while observing the empty set formally denotes the fact of not being able to execute the experiment σ as in the case of i for the trace ?but · !change.

5.2 The ioco Relation for the Partial Order Semantics: co-ioco

We define a new conformance relation **co-ioco** for labeled event structures with the partial order semantics.

We first need to define the notion of quiescent configuration in this semantics: here, the possible actions in a given configuration are not only single actions but also sets of concurrent events.

Definition 16 (Quiescent configuration for partially ordered observation). *Let* $\mathcal{E} \in \mathcal{IOLES}(L)$. *A* quiescent *configuration* $C \in \mathcal{C}(\mathcal{E})$ *is such that* $\forall \alpha \in \mathcal{CO}(L_o) : C \stackrel{\alpha}{\not\Longrightarrow}_\pi$.

We also need to redefine the properties that the enhancement of an IOLES by δ actions must verify. The conflict with other possible events in the given configuration expressed by property (2) extends to sets of concurrent events. Property (3) naturally extends to partial order semantics, considering partially ordered trace instead of sequential ones. Therefore, denoting by $\Delta_\mathcal{E}$ the enhancement by δ actions of an IOLES $\mathcal{E}$, we assume that every quiescent configuration

$C \in \mathcal{C}(\Delta_{\mathcal{E}})$ has the following properties:

$$\begin{aligned}
&(\exists!\ e_\delta \in E \setminus C : \lambda(e_\delta) = \delta \wedge C \stackrel{\delta}{\Longrightarrow}_\pi) \\
\wedge\ &(\forall \alpha = (E_\alpha, \leq_\alpha, \lambda_\alpha) \in \mathcal{CO}(L) : C \stackrel{\alpha}{\Longrightarrow}_\pi \ \Rightarrow\ \forall e' \in E_\alpha : e_\delta \# e') \\
\wedge\ &(\forall \omega \in \mathcal{LPO}(L) : C \stackrel{\omega}{\Longrightarrow}_\pi \ \Rightarrow\ C \cup \{e_\delta\} \stackrel{\omega}{\Longrightarrow}_\pi)
\end{aligned}$$

In the partial order semantics, the outputs of the system under test in response to stimuli may be single outputs as well as sets of concurrent outputs. We need any set of concurrent outputs to be entirely produced by the system under test, so we define the set of expected outputs from a set of configurations S as the set of maximal sets of concurrent outputs.

Definition 17 (Expected outputs for partially ordered observation). *Let $\mathcal{E} \in \mathcal{IOLES}(L)$ and $S \subseteq \mathcal{C}(\mathcal{E})$, then*

$$out_\pi(S) \triangleq \bigcup_{C \in S} \{\alpha \in \mathcal{CO}(L_o) \mid C \stackrel{\alpha}{\Longrightarrow}_\pi \textit{ and } E_\alpha \textit{ is maximal w.r.t} \subseteq\}$$

We obtain the following formulation of the **co-ioco** conformance relation, in the case of the partial order semantics: for any partially ordered trace of the specification enriched with δ actions, the sets of concurrent outputs produced by the implementation after this trace (including δ) are among those authorised by the specification.

Definition 18 (co-ioco). *Let $i, s \in \mathcal{IOLES}(L)$, then*

$$i \ \textbf{\textit{co-ioco}}\ s \Leftrightarrow \forall \omega \in traces_\pi(\Delta_s) : out_\pi(\perp_{\Delta_i} \textbf{\textit{after}}_\pi\ \omega) \subseteq out_\pi(\perp_{\Delta_s} \textbf{\textit{after}}_\pi\ \omega)$$

Example 11. We have $\text{traces}_\pi(s) = \{\epsilon, ?\mathsf{but}, ?\mathsf{but} \cdot !\mathsf{liq}, ?\mathsf{but} \cdot !\mathsf{change}, ?\mathsf{but} \cdot (!\mathsf{liq}\ \textbf{co}\ !\mathsf{change})\}$ and $\text{traces}_\pi(s') = \{\epsilon, ?\mathsf{but}, ?\mathsf{but} \cdot !\mathsf{liq}, ?\mathsf{but} \cdot !\mathsf{change}, ?\mathsf{but} \cdot !\mathsf{liq} \cdot !\mathsf{change}, ?\mathsf{but} \cdot !\mathsf{change} \cdot !\mathsf{liq}\}$ and we can observe:

ω	$\text{out}_\pi(\perp_{\Delta_s}\ \textbf{after}_\pi\ \omega)$	$\text{out}_\pi(\perp_{\Delta'_s}\ \textbf{after}_\pi\ \omega)$	$\text{out}_\pi(\perp_{\Delta_i}\ \textbf{after}_\pi\ \omega)$
ϵ	$\{\delta\}$	$\{\delta\}$	$\{\delta\}$
?but	{!liq **co** !change}	{!liq, !change}	{!liq}
?but · !liq	{!change}	{!change}	{!change}
?but · !change	{!liq}	{!liq}	$\emptyset$
?but · (!liq **co** !change)	$\{\delta\}$	$\emptyset$	$\emptyset$
?but · !liq · !change	$\emptyset$	$\{\delta\}$	$\{\delta\}$
?but · !change · !liq	$\emptyset$	$\{\delta\}$	$\emptyset$

We conclude that i **co-ioco** s' but $\neg(i$ **co-ioco** $s)$. This is due to the fact that the **co-ioco** relation requires any concurrent set of outputs depending on the same input to remain concurrent.

6 Conclusion, Discussion and Future Work

We have laid several cornerstones for conformance testing under true concurrency. Four well-established conformance relations over labeled transition systems [11] (trace preorder, testing preorder, **conf** and **ioco**) have been extended

to concurrency-enabled relations over labeled event structures, and illustrated in several examples. The next steps will encompass test case generation and the formalization of adequate centralized and distributed test architectures.

With the interleaving semantics, the relations we obtain boil down to the same relations defined for LTS, since they focus on sequences of actions. The only advantage of using labeled event structures as a specification formalism for testing remains in the conciseness of the concurrent model with respect to a sequential one. As far as testing is concerned, the benefit is low since every interleaving has to be tested.

By contrast, under the partial order semantics, the relations we obtain allow to distinguish explicitly implementations where concurrent actions are implemented concurrently, from those where they are interleaved, i.e. implemented sequentially. Therefore, these relations will be of interest when designing distributed systems, since the natural concurrency between actions that are performed in parallel by different processes can be taken into account. In particular, the fact of being unable to control or observe the order between actions taking place on different processes will not be considered as an impediment for testing. This opens interesting perspectives for a distributed test architecture where testers would scarcely have to synchronize. If the specification allows it, one could even think of a local test architecture where testers are completely independent from each other.

It should be noted that the **co-ioco** relation allows for further refinement that we do not discuss here. As defined in Def. 7, this relation deals with concurrent inputs and concurrent outputs in the same way: it requires concurrency to be preserved by the implementation. This is exactly what should be the case when the specification requires these concurrent inputs to be processed by different entities, and the concurrent outputs to be issued from different processes. That is, the *distribution* or attribution of events assigned to concurrent processes is then part of the specification, and conformance requires the implementation to have exactly this distribution. An analogous idea is captured in the concurrency-preserving bisimulation relation developed in [19]. It is natural to look also for means of dealing with concurrency in specifications in a different way, namely with a *"don't care"*-type approach: that is, for some events a and b that are specified as concurrent, one may accept implementations that order a before b OR that order b before a (provided of course they conform otherwise to the specification). Care must then be taken to distinguish different types of dependency (output on output? input on output? output on input?) and to study which kind of dependency may be added without compromising required properties. A discussion of some of these aspects, plus the question when dependencies might be *dropped* in the implementation, can be found in [14]. In the present context, such generalization requires the modification of Def. 6: the requirement $\leq_{|A\times A} = \leq_\omega$ is then replaced by an inclusion relation, with additional constraints such as preservation of immediate input-output-orders etc. The discussion and development of these points, which are at the heart of work in progress, would have taken us too far afield here.

Acknowledgment: The research related here was funded by the DIGITEO/DIM-LSC project TECSTES, convention DIGITEO Number 2011-052D - TECSTES.

References

1. Milner, R.: Communication and concurrency. PHI Series in computer science. Prentice Hall (1989)
2. Hoare, T.: Communicating Sequential Processes. Prentice-Hall (1985)
3. ITU-TS: Recommendation Z.100: Specification and Description Language (2002)
4. Brinksma, E., Scollo, G., Steenbergen, C.: Lotos specifications, their implementations and their tests. In Linn, R.J., Uyar, M.U., eds.: Conformance testing methodologies and architectures for OSI protocols. IEEE Computer Society Press (1995) 468–479
5. De Nicola, R., Hennessy, M.: Testing equivalences for processes. Theoretical Computer Science **34** (1984) 83–133
6. Abramsky, S.: Observation equivalence as a testing equivalence. Theoretical Computer Science **53** (1987) 225–241
7. Brinksma, E.: A theory for the derivation of tests. In: Protocol Specification Testing and Verification VIII, North-Holland (1988) 63–74
8. Phillips, I.: Refusal testing. Theoretical Computer Science **50** (1987) 241–284
9. Langerak, R.: A testing theory for LOTOS using deadlock detection. In: Protocol Specification, Testing and Verification IX, North-Holland (1990) 87–98
10. Segala, R.: Quiescence, fairness, testing, and the notion of implementation. Information and Computation **138**(2) (1997) 194–210
11. Tretmans, J.: Test generation with inputs, outputs and repetitive quiescence. Software - Concepts and Tools **17**(3) (1996) 103–120
12. De Nicola, R.: Extensional equivalences for transition systems. Acta Informatica **24**(2) (1987) 211–237
13. 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 Lecture Notes in Computer Science., Springer (2008) 169–183
14. Haar, S., Jard, C., Jourdan, G.V.: Testing input/output partial order automata. In: Testing of Software and Communicating Systems. Volume 4581 of Lecture Notes in Computer Science., Springer (2007) 171–185
15. Nielsen, M., Plotkin, G.D., Winskel, G.: Petri nets, event structures and domains, part I. Theoretical Computer Science **13** (1981) 85–108
16. Winskel, G.: Event structures. In: Advances in Petri Nets. Volume 255 of Lecture Notes in Computer Science. (1986) 325–392
17. Nielsen, M., Sassone, V., Winskel, G.: Relationships between models of concurrency. In: REX School/Symposium. (1993) 425–476
18. Aceto, L., De Nicola, R., Fantechi, A.: Testing equivalences for event structures. In: Mathematical Models for the Semantics of Parallelism. Volume 280 of Lecture Notes in Computer Science. (1986) 1–20
19. Balaguer, S., Chatain, Th., Haar, S.: A concurrency-preserving translation from time Petri nets to networks of timed automata. In: International Symposium on Temporal Representation and Reasoning, IEEE Computer Society Press (2010) 77–84