Using partial-orders for detecting faults in concurrent systems

A. Petrenko[a], A. Ulrich[b], and V. Chapenko[c]

a *Centre de Recherche Informatique de Montreal (CRIM), 1801 Av. McGill College, Suite 800, Montreal, H3A 2N4, Canada, petrenko@crim.ca*

b *Department of Computer Science, University of Magdeburg, PF 4120, 39016 Magdeburg, Germany, ulrich@cs.uni-magdeburg.de*

c *Institute of Electronics and Computer Science, University of Latvia, 14 Dzerbenes street, Riga, LV-1006, Latvia, chapenko@edzi.lza.lv*

Abstract

The paper suggests test derivation approaches to obtain test suites for concurrent systems based on the concept of fault coverage criteria in opposition to structural test coverage criteria. Using a partial-order model, called *Mazurkiewicz Trace Machine* (MTM), for test derivation, the state explosion problem can be alleviated. The derived test suites are characterized by their small size compared to test suites from traditional test derivation approaches and exhibit a defined degree of fault coverage according to certain fault models. The fault models of concurrent systems considered in the paper are based on the most common faults, acceptance, refusal, and transfer faults. A scenario of test execution in concurrent systems, including a suitable test architecture, is discussed that explains the application of a test suite derived from an MTM in a test run.

Keywords

Concurrent systems, specification-based testing, test derivation, fault coverage.

1 INTRODUCTION

1.1 Motivation

Testing is an important means in the development cycle of software. It comprises the derivation of a *test suite* from a suitable specification of the software system and

A. Petrenko et al. (eds.), *Testing of Communicating Systems*

the application of the test suite in a *test run* to test whether an implementation of a system conforms to the specification. If the behaviors of the implementation and of the test suite differ from each other, an error probably exists in the implementation. In order to derive test suites from a formal specification of a system, two different approaches are known: (1) *structural testing* based on coverage criteria in the specification and (2) *fault testing* based on an assumed fault model of the implementation [3]. The first approach selects test suites according to a coverage criterion that is defined over the syntactical structure of the formal specification. The coverage criteria are based on empirical knowledge that have been proven some usefulness in practice. Examples of test coverage criteria are the statement coverage, edge coverage or path coverage criterion [9] or other coverage criteria defined in particular for concurrent systems [29].

On the other hand, fault testing uses a designated *fault model* of the implementation of a system that is decoupled from the syntactical structure of the system. Such fault models in the realm of labeled transition systems (LTSs) are, for instance, acceptance and refusal fault models or next-state fault models [3]. The advantage of fault testing over structural testing is that after a successful test run, one can conclude that a certain class of faults definitely will not appear in an implementation.

The exact knowledge about faults that may occur in a faulty implementation allows an effective reduction of test data to a minimum amount necessary to detect these faults. Test derivation approaches for fault testing have been introduced before mainly for testing telecommunication systems [2], [4], [7], [23], [25], [34], [35]. Results of this research are test derivation methods, like the transition tour method, W-method, or HSI-method.

In this paper we propose to apply fault testing also in the general context of testing concurrent systems consisting of a collection of sequential modules. A prerequisite of fault testing of concurrent systems is the existence of a suitable description model for them. The traditional model is the reachability graph that is obtained when the sequential modules of the concurrent system are combined together step-by-step in an interleaving framework. Latest developments in verification techniques for concurrent systems avoid the construction of a reachability graph in favor of a partial-order model though. One partial-oder model is the so-called *Mazurkiewicz Trace Machine* (MTM), first introduced in [11] and [12]. The MTM is a reduced reachability graph that still preserves the safety properties of the concurrent system.

So far the model of an MTM has been exploited only in the verification process of concurrent systems. Their application to test derivation, however, was underestimated. To end this situation, this paper exploits the MTM model for the purpose of test derivation. It shows that this partial-order model is also a favorable model in testing since test suites derived from an MTM are much smaller than ones from the traditional reachability graph under the assumption of an equal degree of fault coverage.

1.2 Related work

First research in the area of concurrent systems was done by developing debugging methods for them. Debugging focuses on the process of isolating an already known error in the concurrent system. This is different to testing where the emphasis lies in detecting errors first, which can be debugged later. The paper [21] addressed the problem of unpredictable system runs due to concurrency. In [28], an approach based on Ada was developed to provide a method that allows a deterministic replay

of a system run. The approach can be adopted to support a suitable test architecture for concurrent systems (see Section 2.2).

Test derivation methods for concurrent systems that systematically cover the behavior of a concurrent system and provide test suites of a defined fault coverage are still quite new. In [16], an hierarchy-based finite state machine (FSM) construction approach is used to perform a structural test ofa system consisting of several concurrent FSMs. This technique was refined in [17]. It describes an incremental, bottom-up-oriented approach and assumes that each subsystem considered can be tested separately. Test derivation is based on degrees of *test coverage* (opposite to *fault coverage* as considered in our paper). The selection of test suites, however, is left open by referring to the work in [29]. A further approach for structural test derivation presented in [6] is based on a specification-based selection of test data. The paper gives hints on the selection of suitable test data to provide a useful degree of test coverage. The selection process must be assisted by a test expert though.

Early test derivation approaches that systematically derive test suites according to a certain fault coverage and try to avoid state explosion during the generation of test suites are given in [15], [32], and [33]. A general drawback of these approaches is that they exploit a complicated concurrency model as the basis for test derivation that cannot, with exception of [33], be efficiently computed in all cases.

At this point we extend the work previously done in testing concurrent systems by suggesting test derivation methods that generate test suites with fault coverage guarantee. Based on the partial-order model of an MTM, our paper discusses fault models that supports acceptance, refusal, and transfer faults. As a result, test suites are generated which are quite short in many cases, but still able to detect all faults of the associated fault model. The application of an MTM in testing requires specific measures for test architectures of concurrent systems. This issue is a further discussion point in this paper.

The paper is organized as follows. First, Section 2 introduces necessary assumptions and definitions of a concurrent system and discusses aspects of a test architecture that can support test execution based on partial orders. Section 3 introduces a framework for fault detection used in this paper. Section 4 deals with test derivation algorithms that generate test suites according to a particular fault model. Finally, Section 5 concludes the paper.

2 PRELIMINARIES

2.1 Interleaved-based models vs. partial-order models of a concurrent system

A *concurrent system* $\Im$ is defined as a parallel composition $\Im = M_1 \parallel \ldots \parallel M_n$ of n finite labeled transition systems (LTSs) communicating synchronously. The same message can be exchanged between two or more component LTSs at a single synchronization (multi-rendezvous). Transmitting messages and their receipt through interaction points are referred to *actions* in an LTS.

Definition 1. A *labeled transition system* (LTS or *machine* for short) M is defined by a quadruple $(S, A, \rightarrow, s_0)$, where S is a finite set of states; A is a finite set of actions (the alphabet); $\rightarrow \subseteq S \times A \times S$ is a transition relation; and $s_0 \in S$ is the initial state.

A transition $(s_1, a, s_2) \in \rightarrow$ is also written as $s_1 - a \rightarrow s_2$. An LTS is *deterministic* if there are no two transitions $s - a \rightarrow s_1$ and $s - a \rightarrow s_2$ for any start state s and action a

with $s_1 \neq s_2$. With no loss of generality, we assume that each component LTS is initially connected. Furthermore, we consider concurrent systems to be composed of deterministic LTSs only.

As usual, $s{=}\alpha{\Rightarrow}s'$ denotes a *trace* starting from state s and ending in state s'. The end state s' is reached by the sequence of actions a_i in the trace $\alpha \in A^*$ of M. The trace α traverses a set of intermediate states: $s{-}a_1{\rightarrow}s_1{-}a_2{\rightarrow}s_2{-}\ldots{\rightarrow}s'$. The set of traces of state s is defined as the set $\{\alpha \in A^* \mid \exists s' \in S\colon s{=}\alpha{\Rightarrow}s'\} = Tr(s)$. The set of traces of M is $L(M) = Tr(s_0)$, i.e. the language accepted by LTS M starting from its initial state [14].

Definition 2. State s_1 of LTS M_1 and state s_2 of LTS M_2 are *trace-equivalent*, $s_1 \approx_{tr} s_2$, if $Tr(s_1) = Tr(s_2)$; states s_1, s_2 are *distinguishable*, $s_1 \neq s_2$, if $Tr(s_1) \neq Tr(s_2)$. Trace equivalence (distinguishability) of LTSs is defined as trace equivalence (distinguishability) of their initial states s_{01} and s_{02}, i.e., $M_1 \approx_{tr} M_2$, if $Tr(s_{01}) = Tr(s_{02})$ and $M_1 \neq M_2$, if $Tr(s_{01}) \neq Tr(s_{02})$.

Henceforth, we assume that each component of a concurrent system is minimal, i.e., every two of its states are distinguishable.

The joint behavior of a concurrent system $\Im = M_1 \parallel \ldots \parallel M_n$ can be described by means of a *composite machine* defined over $A_\Im \subseteq A_1 \cup \ldots \cup A_n$, the (global) alphabet of system $\Im$. Components execute *shared* actions that require rendezvous of several component LTSs along with *local* actions that are executed by a component and its environment only. We assume that different rendezvous on the same event have distinct names in the concurrent system, i.e., names of local and shared actions in $A_\Im$ are globally unique. Let $ID = \{1, 2, \ldots, n\}$ be the set of indexes of the LTSs in $\Im$. For each $a \in A_\Im$, $\boldsymbol{id}(a)$ denotes the *occurrence set* $\{i \in ID \mid a \in A_i\}$, i.e. the set of components involved in a rendezvous over a.

Definition 3. A *composite machine* of a given concurrent system $\Im$ of n LTSs $M_i = (S_i, A_i, \rightarrow_i, s_{0i})$ is the quadruple $(S_\Im, A_\Im, \rightarrow_\Im, s_\Im)$, where $S_\Im$ is a global state space, $S_\Im \subseteq S_1 \times \ldots \times S_n$; $A_\Im \subseteq A_1 \cup \ldots \cup A_n$ is the set of actions (the global alphabet), $s_\Im = (s_{01}, \ldots, s_{0n})$ is the initial global state, and the transition relation $\rightarrow_\Im$ is defined as follows: Let $a \in A_\Im$ and $(s_1, \ldots, s_n) \in S_\Im$. $((s_1, \ldots, s_n), a, (s_1', \ldots, s_n')) \in \rightarrow_\Im$, where $s_j' = s_j$ for all $j \notin \boldsymbol{id}(a)$, if there exists $s_i' \in S_i$ such that $(s_i, a, s_i') \in \rightarrow_i$ for all $i \in \boldsymbol{id}(a)$.

We further assume that the global state space comprises only states reachable from the initial state. We use $C_\Im$ to denote the composite machine $(S_\Im, A_\Im, \rightarrow_\Im, s_\Im)$ of a given concurrent system $\Im$, and $s(i)$ to denote a local state of component M_i in a global state s of $C_\Im$. Let $L(C_\Im)$ denote the set of all traces executed by the composite machine from its initial state, i.e., $L(C_\Im)$ is the language of the concurrent system $\Im$. A way to represent the composite machine is by means of the reachability graph. It represents the behavior of a concurrent system $\Im$ in the interleaved-based semantics. In partial-order semantics, the same behavior may have a more compact representation. We define such a representation based on the trace theory of Mazurkiewicz [20], [24].

Definition 4. The *independence relation* over the global alphabet $A_\Im$ is the relation $I = \{(a, b) \in A_\Im \times A_\Im \mid \boldsymbol{id}(a) \cap \boldsymbol{id}(b) = \varnothing\}$.

$(A_\Im, I) = \Lambda$ is the *concurrent alphabet* of $\Im$, as defined in the trace theory of Mazurkiewicz. Let "." denote the concatenation of words over $A_\Im$. We define the

equivalence of traces over Λ as the least congruence relation $\equiv_\Lambda \subseteq A_\Im^* \times A_\Im^*$ w.r.t. the concatenation operator and including $\forall\, a, b \in A_\Im$: $(a, b) \in I \Rightarrow a.b \equiv_\Lambda b.a$.

Definition 5. A *Mazurkiewicz trace* (*M-trace*) over $\Lambda = (A_\Im, I)$ is an equivalence class of $\equiv_\Lambda$ of traces over $A_\Im^*$.

An M-trace is fully characterized by one of its traces α and the concurrent alphabet $(A_\Im, I)$ and is denoted by $[\alpha]$, where α is a representative trace of the class. By successively permuting adjacent independent actions in α, one can obtain all other traces in $[\alpha]$. An M-trace $[\alpha]$ is a partial order over $A_\Im^*$, traces in $[\alpha]$ are called *linearizations*. Since an M-trace is sufficiently represented by a single linearization over a concurrent alphabet, the behavior of a concurrent system can be represented by one linearization for each M-trace only, instead of giving all possible traces the system can perform in the composite machine. In fact, certain submachines of the composite machine contain the required linearizations.

Definition 6. Given two machines $M_1 = (S, A, \rightarrow_1, s_{01})$ and $M_2 = (S', A', \rightarrow_2, s_{02})$. M_2 is said to be a *submachine* of M_1 if $S' \subseteq S$, $s_{02} = s_{01}$, $A' \subseteq A$, and $\rightarrow_2 \subseteq \rightarrow_1$.

Clearly, $L(M_2) \subseteq L(M_1)$ for all submachines M_2 of M_1. We immediately extend this definition for machines with a state isomorphism. Namely, if a machine M_3 is isomorphic to M_2 that, in turn, is a submachine of M_1, then we say that M_3 is a submachine of M_1.

Definition 7. Let $C_\Im = (S_\Im, A_\Im, \rightarrow_\Im, s_\Im)$ be the composite machine of concurrent system $\Im$. A submachine $M_\Im = (S_M, A_\Im, \rightarrow_M, s_\Im)$ of $C_\Im$ is a *Mazurkiewicz trace machine* (MTM) of $\Im$, if for all traces $\alpha \in L(C_\Im)$, there exists a trace $\alpha' \in L(M_\Im)$ such that α' is a linearization of an M-trace defined by an extension of α, i.e., $\alpha \in Pref([\alpha'])$.

It follows from the definition that an MTM completely characterizes the traces of the concurrent system, i.e.

$$L(C_\Im) = \bigcup_{\alpha \in L(M_\Im)} Pref([\alpha]) \ ,$$

where $Pref(\Gamma)$ denotes the set $\{\alpha \in A_\Im^* \mid \exists \beta \in A_\Im^*: \alpha\beta \in \Gamma\}$ for a given $\Gamma \subseteq A_\Im^*$. The definition of an MTM is inspired by [12]. The main advantage of an MTM over the model of a composite machine is that the number of states in the MTM is usually much less than in the composite machine. Results e.g. from [8], [13], and [36] show that the saving rate in the number of states can be huge for many examples (27 to 90 per cent of reduction), although the state space size is still exponential in the worst case since the overall complexity of state space exploration remains PSPACE-complete. Also, the computational complexity of the MTM construction from a set of concurrent modules is mostly smaller compared to the construction of the reachability graph [13].

The construction of an MTM is similar to that of a composite machine, except that in each global state, we select among all independent actions only one action for inclusion into the MTM, see [12]. Since such a selection is an arbitrary process, several MTMs, probably with a different number of states, can be obtained. An MTM possesses a number of attractive properties, which are used to improve verification techniques and, as we shall demonstrate in this paper, can also be exploited for test derivation purposes.

Proposition 1. Let $C_{\Im}$ be the composite machine of concurrent system $\Im$, and let $M_{\Im}$ be an MTM for this system. For all components M_i, for all states $s_{ij} \in S_i$, where S_i is the set of states in M_i, s_{ij} is reachable from the initial state in $C_{\Im}$ iff s_{ij} is reachable from the initial state in $M_{\Im}$.

In short, all local states s_{ij} reachable in $C_{\Im}$ are also reachable in $M_{\Im}$ and vice versa. The proof is given in [12]. We can also easily prove the following.

Proposition 2. Let $C_{\Im} = (S_{\Im}, A_{\Im}, \rightarrow_{\Im}, s_{\Im})$ be the composite machine of concurrent system $\Im$, and let $M_{\Im} = (S_M, A_{\Im}, \rightarrow_M, s_{\Im})$ be an MTM for this system. For all actions $a \in A_{\Im}$, and all transitions $(s, a, s') \in \rightarrow_{\Im}$, there exists a transition $(p, a, p') \in \rightarrow_M$ such that $p(i) = s(i)$ for all $i \in \mathbf{id}(a)$.

This proposition indicates that all unexecutable transitions in a concurrent system can be already detected when an MTM is constructed. For the purpose of test derivation, we may well assume that these transitions have been deleted from the given system $\Im$, henceforth, $A_{\Im} = A_1 \cup \ldots \cup A_n$ is taken for granted.

As an example, we consider the concurrent system $\Im = A \parallel B$, which consists of the two component machines A and B (Figure 1). Actions a and c are shared, the remaining actions are local. Figure 1 also shows the composite machine $C_{\Im}$. Note that, in this example, the number of states of the composite machine reaches its maximum, and all global states are reachable from the initial state. The independence relation of this system is $I = \{(b, e), (b, e), (d, e), (e, d)\}$. Given the trace *abed*, for example, one can, by successively permuting adjacent independent actions in *abed*, obtain all other sequences in [*abed*], i.e. [*abed*] = {*abed*, *abde*, *aebd*}. Two MTMs of the example system are shown in Figure 2, they use different numbers of states to represent the same M-traces.

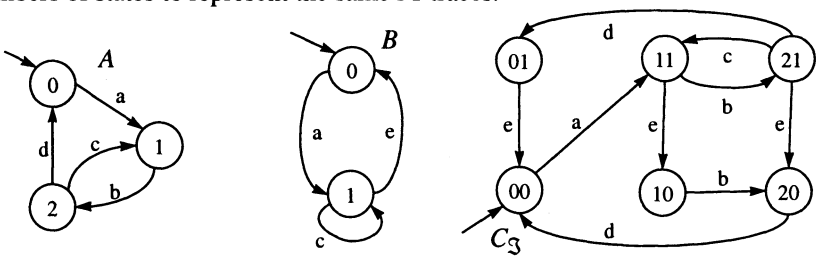

Figure 1. The concurrent system $\Im = A \parallel B$ and its composite machine $C_{\Im}$.

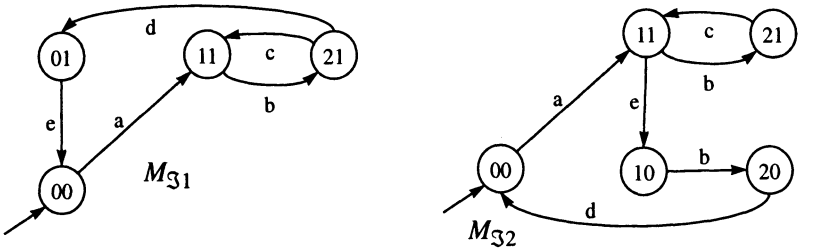

Figure 2. Two MTMs from the composite machine $C_{\Im}$.

2.2 Assumptions on a test architecture

A *test architecture* is defined as a parallel, synchronous composition of the *implementation under test* (*IUT*) of the concurrent system $\mathfrak{I}$ with its tester. If a test suite comprises several test cases, a separate tester is built for each test case. It is required that the number of LTSs in the IUT of the concurrent system is constant when the test run takes place (static system). Furthermore, the structure of the system $\mathfrak{I}$, i.e. its composition of n LTSs, is preserved in the implementation. From the testing perspective, it should be stated what actions can be observed and controlled. To avoid nondeterministic execution of the IUT due to nonobservability, we assume that *all* local and shared actions are observable during testing (grey-box testing approach).

In testing concurrent systems the information about action names observed during a test run is not sufficient to assess conformance between specification and IUT. Due to the existence of multi-rendezvous among component LTSs of the IUT, the tester must also know what components participate in a specific multi-rendezvous. Furthermore, the issue of true concurrency among actions of the IUT requires that the tester has the power not only to observe an action of the IUT, but must also control each occurrence of a multi-rendezvous. Thus, the crucial point in testing concurrent systems is to perform a *deterministic test run.*

The problem can be solved by applying *instant replay techniques* used for debugging concurrent systems [28]. The proposal in [28] assumes a global controller that is asked for permission by the components of the IUT before they are allowed to interact with each other. To achieve this, control code, called *probes*, is added into the source code of the components before each interaction invocation. Only after the tester received all requests to execute a certain interaction from participating components, it grants this interaction. If not or if a wrong component asks for permission, an error in the IUT has occurred.

Adopting this technique to our purposes, it means that the tester collects the action names together with their independence information from the IUT, i.e., it controls and observes the global alphabet and the occurrence of the multi-rendezvous of the system. Only if the action name *and* its occurrence set observed from the IUT equal to the corresponding ones in the specification, a correct multi-rendezvous happened. Otherwise, the IUT does not conform to the specification. In Section 3.2 we give a model of such a tester in the context of a conformance relation between the IUT and its corresponding specification.

3 A FRAMEWORK FOR FAULT DETECTION

3.1 General fault model

Fault models are usually required to guarantee the detection of certain types of faults by means of a finite tester [4]. As usual in specification-based testing, a tester is derived from a given specification of the system. In addition, the tester requires a certain criterion to assess a test run over the IUT. This criterion is customary defined as *conformance relation* between the specification and an implementation of a certain domain [5], [30], [31]. The conformance relation classifies all possible implementations into a class of implementations conforming to the specification, i.e., a test run of an implementation with a test case derived according to the conformance relation yields the verdict *pass*, and into a class of non-conforming implementations, i.e., a test run yields the verdict *fail*. The implementation domain is usually a finite set of implementations that can be derived from the specification by performing a number of mutations representing faults of a certain type, i.e., the

implementation domain is defined implicitly by properties common to all implementations. Similar to the realm of testing sequential systems, we assume as a minimal prerequisite for the fault domain that all actions an IUT submits during testing are known in advance, i.e. that the global alphabet of an IUT is a subset of the given global alphabet. This assumption appears fundamentally in the context of fault-driven testing of sequential systems [10]. We believe that the above concepts are pertinent to a parallel composition of sequential systems as well, and following [25] we define a fault model for a concurrent system as the triple

<*specification, conformance relation, implementation domain*>.

Based on this concept, we are ready to state what constitutes a sound and complete test suite: A test suite is a finite set of finite test cases consisting of actions accessible for testing. A test suite is *sound* w.r.t. a fault model if any conforming implementation passes the test suite. A sound test suite is *complete* w.r.t. a fault model if any non-conforming implementation from the implementation domain fails it.

A general fault model and a complete test suite for a concurrent system corresponding to it might be derived by the following approach originally developed for sequential systems. In particular, given a concurrent system $\Im = M_1 \| \ldots \| M_n$, we may treat a composite machine $C_\Im$ as the specification and the trace-equivalence of composite machines as a conformance relation. The implementation domain could take several forms according to the existing test derivation methods for LTS specifications with guaranteed fault coverage [23]. As an example, the implementation domain is often defined as the universe $C(A_\Im, N_{max})$ of all LTSs defined over some actions of the given alphabet $A_\Im$ with at most N_{max} states. The fault model $<C_\Im, \approx_{tr}, C(A_\Im, N_{max})>$, where $N_{max} \geq |S_\Im|$, is a classical fault model most frequently used for fault detection in state machines [4], [7], [23], [34], [35]. It corresponds to the most general type of faults that may occur in a sequential system. Following this approach, a test suite for the concurrent system $\Im$ that is complete with respect to this fault model can be obtained by applying, for example, the method presented in [30].

Trace-equivalence of composite machines is, however, not sufficiently strong enough for testing concurrent systems, even for deterministic ones as considered here. It does not discriminate between a sequential system M and the concurrent system $M \| M$ composed of two instances of M, since they have isomorphic composite machines. However, any tester with control over multi-rendezvous, as outlined in Section 2.2, does. Speaking more generally, we wish to distinguish two instances of the same action with different occurrence sets. Apparently, the problem can be easily fixed by decorating each action in composite machines with the names of component LTSs executing it.

Given the composite machine $C_\Im = (S_\Im, A_\Im, \rightarrow_\Im, s_\Im)$ of concurrent system $\Im = M_1 \| \ldots \| M_n$, where $A_\Im = A_1 \cup \ldots \cup A_n$, and the set of names of components *ID*, we use $\hat{a}$ to denote the pair $<a, \mathbf{id}(a)>$. Replacing the global alphabet $A_\Im$ by the set $\hat{A}_\Im = \{ \hat{a} \mid a \in A_\Im \}$ in the composite machine, we obtain an isomorphic LTS $\hat{C}_\Im = (S_\Im, \hat{A}_\Im, \rightarrow_\Im, s_\Im)$, called the *augmented composite machine* of $\Im$. Now, the trace equivalence can immediately be employed to define the conformance relation for (deterministic) concurrent systems.

Definition 8. Given two concurrent systems $\Im$ and $\Re$, let $\hat{C}_\Im$ and $\hat{C}_\Re$ be the augmented composite machines of $\Im$ and $\Re$, respectively. $\Im$ and $\Re$ are *trace-equivalent*, denoted $\Im \approx \Re$, if $C_\Im \approx_{tr} C_\Re$. $\Im$ and $\Re$ are *distinguishable*, $\Im \neq \Re$, if $L(\hat{C}_\Im) \neq L(\hat{C}_\Re)$.

In other words, two systems $\Im$ and $\Re$ are trace-equivalent if their composite machines are trace-equivalent, and for all actions $a \in A_\Im$ in all traces of $L(C_\Im)$, the names of components involved in the execution of a are identical. Henceforth, we will use the notions of a composite machine and an augmented composite machine interchangeably whenever no confusion arises. Similarly, we define an augmented MTM $M_\Im$ for system $\Im$.

It remains now to show how an implementation domain for concurrent systems can be defined. We assume that an implementation $\Re$ is modeled as parallel composition of a number of component LTSs. The number of components is exactly the same as in the specification $\Im$, since the tester establishes a communication link to every component to perform test runs (Section 2.2). This implies that the underlying communication subsystem is assumed to be perfect and its possible faults are modeled by faults inside the components of the IUT.

As mentioned earlier, we assume that $A_\Re \subseteq A_\Im$ for all possible implementations $\Re$. It implies that in the general case, the local alphabet of any component LTS is also a subset of $A_\Im$, but does not necessarily coincide with the local alphabet in the specification. Some components might be assumed to be fault-free, and their alphabets remain intact.

3.2 The model of a tester

Next, we present a model of testers used to verify whether or not a system $\Re$ is trace-equivalent to the system $\Im$. A test run should stop as soon as deadlock occurs. Thus, test cases for $\Im$ can be defined as valid traces $\hat{\sigma} \in L(C_\Im)$ and invalid traces $\hat{\gamma}.\hat{a}$, where $\hat{\gamma} \in L(C_\Im) \cup \{\hat{\varepsilon}\}$, $\hat{\varepsilon} = <\varepsilon, \varnothing>$, an empty symbol; $\gamma.a \notin L(C_\Im)$, $\hat{a} = <a, id>$, $a \in A_\Im$, and id is a non-empty subset of the set ID. Intuitively, the valid traces are used to verify whether or not an IUT possesses traces of $\Im$ (valid behavior), whereas the invalid traces check the IUT for invalid behavior.

Given a valid trace $\hat{\sigma}$, we define the corresponding tester $T(\hat{\sigma})$ as an acyclic LTS $T(\hat{\sigma}) = (S_T, A_\Im, \rightarrow_T, s_T)$ such that its language is the set $Pref(\hat{\sigma}) \subseteq A_\Im$ and the set of its states S_T contains $|\hat{\sigma}| + 1$ states. A test run of $\hat{\sigma}$ against the IUT $\Re$ can be modeled as a parallel composition $T(\hat{\sigma}) \parallel C_\Re$. The system $T(\hat{\sigma}) \parallel C_\Re$ is used to assign the verdict of a test run, *pass* or *fail*, to states of the tester according to a labeling function ***ver***: $S_T \rightarrow \{pass, fail\}$. Note that we are dealing with deterministic composite machines, for which there is no need to use the verdict *inconclusive*. The system $T(\hat{\sigma}) \parallel C_\Re$ executing trace $\hat{\sigma}$ comes to a deadlock since the tester $T(\hat{\sigma})$ reaches its final state; this state is the only one labeled with *pass*, the remaining states are labeled with *fail*. The IUT passes test case $\hat{\sigma}$ if the system $T(\hat{\sigma}) \parallel C_\Re$ deadlocks only when the state *pass* of the tester is reached.

Similarly, we define the tester $T(\hat{\gamma}.\hat{a})$ corresponding to an invalid trace $\hat{\gamma}.\hat{a} \notin L(C_\Im)$. The tester is modeled with $|\hat{\gamma}.\hat{a}| + 1$ states. Since the system $T(\hat{\gamma}.\hat{a}) \parallel C_\Im$ must deadlock after $\hat{\gamma}$ is executed, the state of the tester reached after $\hat{\gamma}$ is labeled with verdict *pass*, whereas all others are labeled with *fail*. In both cases, the test verdict of a test run is obtained from the label of the last state reached by the tester.

4 TEST DERIVATION

4.1 Detecting missing transitions

We assume here that certain transitions possibly not implemented at some states of component LTSs are the only implementation faults. That means, an action caused

by a transition in the specification of a component may be refused in a faulty implementation. At the same time, we assume that an accepted action causes a transition into a correct state. Thus, whatever an implementation of each component does it conforms to the specification; it may reduce the specification though.

Formally, we define the acceptance fault model as $<\Im, \approx, \Re_{acc}>$, where $\Re_{acc}$ is the set of all implementation systems $\Re$ consisting of n initialized LTSs such that they are submachines of the corresponding machines in specification $\Im$. The *acceptance set* [18] of any local state in $\Re$ is a subset of that set of the corresponding local state in $\Im$. The action set $A_{\Re}$ of any concurrent system $\Re \in \Re_{acc}$ is a subset of the action set $A_{\Im}$ of $\Im$. Intuitively, the detection of acceptance faults can be achieved by using a transition cover of a machine [22].

Definition 9. Given an initially connected LTS $M = (S, A, \rightarrow, s_0)$, a set of traces $TC(M)$ is said to be a *transition cover* of M if for all transitions $(s, a, s') \in \rightarrow$, there exists $\beta a \in Pref(TC(M))$ such that $s_0 = \beta \Rightarrow s$.

By definition, the global states of composite machine $C_{\Im}$ contain all local states of all component machines of $\Im$. Moreover, all executable transitions between local states are present in the composite machine (it has no unexecutable transition by our assumption). To check whether or not all transitions at a particular local state are preserved in the IUT, we should try to execute all actions accepted at a corresponding global state. If an action a is executed in a global state and if the observed occurrence set coincide with the one in the augmented composite machine $\hat{C}_{\Im}$, then the action causes transitions at all local states defined by $\boldsymbol{id}(a)$. Consequently, a transition cover of $\hat{C}_{\Im}$ is already a complete test suite w.r.t. the fault model $<\Im, \approx, \Re_{acc}>$. We demonstrate that this statement holds even when the augmented composite machine $\hat{C}_{\Im}$ is replaced by an augmented MTM $\hat{M}_{\Im}$. A shorter test suite detecting acceptance faults is then obtained, provided that we can find a transition cover of $\hat{M}_{\Im}$ shorter than that of $\hat{C}_{\Im}$.

Proposition 3. Given a concurrent system $\Im$, an augmented MTM $\hat{M}_{\Im}$, and the fault model $<\Im, \approx, \Re_{acc}>$, let $TC(\hat{M}_{\Im})$ be a transition cover of $\hat{M}_{\Im}$. Then $TC(\hat{M}_{\Im})$ is a test suite complete w.r.t. $<\Im, \approx, \Re_{acc}>$.

Proof. Consider a transition cover $TC(\hat{M}_{\Im})$ of the augmented MTM $\hat{M}_{\Im}$. By assumption that the system $\Im$ has no unexecutable transitions and $A_{\Im} = A_1 \cup \ldots \cup A_n$, and by virtue of Proposition 2, the corresponding transition cover $TC(M_{\Im})$ of MTM $M_{\Im}$ executes all transitions in each component LTS in $\Im$. In other words, the projection of all sequences of $TC(\hat{M}_{\Im})$ into a component M_i of $\Im$ is a transition cover $TC(M_i)$ of this component. The system $\Re \in \Re_{acc}$ consists of exactly n component LTSs, as the system $\Im$ does. If $\Re$ passes the test suite $TC(\hat{M}_{\Im})$, it means that for all components in $\Re$, the traces of $TC(M_i)$ are valid traces of the i-th component in $\Re$. Moreover by definition of $\Re_{acc}$, each component in $\Re$ is a submachine of the corresponding component machine in $\Im$. It means that each component machine of $\Re$ is isomorphic to the corresponding machine of $\Im$. As result, $\Re \approx \Im$.

◊

Thus, the problem of deriving a minimal test suite for a given concurrent system complete w.r.t. acceptance faults can be reduced to that of finding a minimal transition cover of an MTM of the given system. Consider our example (Figure 1). Each MTM (Figure 2) has a minimal transition tour of 6 actions although they have different number of states. Take one of them, e.g. *a.b.c.b.d.e*. Decorating actions with occurrence sets, we obtain the test sequence $<a, \{1, 2\}>.<b, \{1\}>.<c, \{1, 2\}>.<b,$

{1}>.<*d*, {1}>.<*e*, {2}>. The tester has seven states, the seventh state is labeled with *pass*, the other ones with *fail* (see Figure 3).

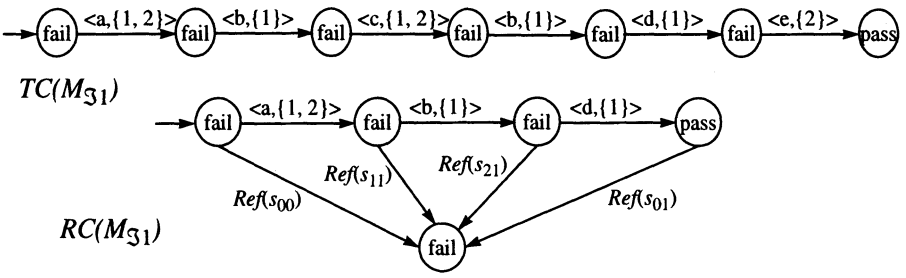

Figure 3. Test suites $TC(\hat{M}_{\Im 1})$ and $RC(\hat{M}_{\Im 1})$ for acceptance and refusal testing of the system $\Im$.

4.2 Detecting superfluous transitions

Assume now that any faulty implementation $\Re$ of a given specification $\Im$ extends the specification. In this case, we assume that superfluous transitions implemented at some local states of component LTSs are the only faults and that the specified transitions are correctly implemented in each component of the IUT.

We define the refusal fault model as $<\Im, \approx, \Re_{ref}>$, where $\Re_{ref}$ is defined as the set of all implementations consisting of n initialized LTSs such that the components of specification $\Im$ are submachines of the corresponding components in implementation $\Re$. The action set of any component of $\Re \in \Re_{ref}$ is assumed to be a subset of the global alphabet $A_\Im$. Note that the number of states in a component may exceed that in the specification, however, we do not bound it here.

Similar to [18], we define the *refusal set* of a global state s in the augmented composite machine $\hat{C}_\Im$ as the set $Ref_\Im(s) = \{<a, id> \mid s \not\rightarrow a \rightarrow_\Im \vee id \neq \boldsymbol{id}(a)\}$, where $s \not\rightarrow a \rightarrow_\Im$ denotes the fact that action $a \in A_\Im$ is refused at state s and $id \subseteq ID$, $id \neq \emptyset$. For any $<a, id> \in Ref_\Im(s)$, the action a accepted at a corresponding global state in an IUT means that each component in the occurrence set id observed by the tester has a superfluous transition labeled with a. Intuitively, to detect implementation faults defined by the fault model $<\Im, \approx, \Re_{ref}>$, it is sufficient to check whether or not any action from $Ref_\Im(s)$ causes an invalid transition at a corresponding state of the IUT. This must be done for each global state s of the augmented composite machine $\hat{C}_\Im$. For this purpose, we use invalid traces as test cases. Alternatively, to save a number of test runs, we can define tests of a set of refusal traces if we assume that the tester is able to proceed after a deadlock (as in [18]).

Minimal invalid traces are constructed based on a minimal state cover. Given an initially connected LTS $M = (S, A, \rightarrow, s_0)$, we define a *state cover*, denoted $SC(M)$, of the LTS as a prefix-closed set of traces such that for all $s \in S$, there exists a transfer sequence $\alpha \in Pref(SC)$ such that $s_0 = \alpha \Rightarrow s$. For a strongly connected machine, a state cover can be constructed as a state tour of the machine.

Definition 10. Given the augmented composite machine $\hat{C}_\Im = (S_\Im, A_\Im, \rightarrow_\Im, s_\Im)$ and its state cover $SC(\hat{C}_\Im)$, we define a *refusal cover* of $\hat{C}_\Im$, denoted $RC(\hat{C}_\Im)$, as

the set
$\{\hat{\gamma}.\hat{a} \mid \hat{\gamma} \in Pref(SC(\hat{C}_{\Im})) \wedge s_{\Im}=\hat{\gamma} \Rightarrow s \wedge \hat{a} \in Ref_{\Im}(s)\}$.

Similar to the case of acceptance faults, a complete test suite w.r.t. fault model $<\Im, \approx, \Re_{ref}>$ can be obtained from an augmented MTM $\hat{M}_{\Im}$. A refusal cover of $\hat{M}_{\Im}$ is defined as in Definition 10, provided that the refusal set $Ref_M(s)$ of each state s of $\hat{M}_{\Im}$ is the one of state s in the augmented composite machine $\hat{C}_{\Im}$.

Proposition 4. Given concurrent system $\Im$, an MTM $\hat{M}_{\Im}$, and the fault model $<\Im, \approx, \Re_{ref}>$, let $RC(\hat{M}_{\Im})$ be a refusal cover of $\hat{M}_{\Im}$. Then the set $RC(\hat{M}_{\Im})$ is a test suite complete w.r.t. $<\Im, \approx, \Re_{ref}>$.

Proof. Consider an arbitrary concurrent system $\Re \in \Re_{ref}$. Similar to Proposition 3, we claim that each component of $\Re$ is isomorphic to the corresponding one of $\Im$. We prove Proposition 4 by contradiction.

Assume that $\Re$ is not a conforming implementation of $\Im$, i.e. $L(\hat{C}_{\Im}) \neq L(\hat{C}_{\Re})$, but $\Re$ passes $RC(\hat{M}_{\Im})$. In this case, the composite machine $\hat{C}_{\Re}$ has at least one trace $\hat{\sigma}.\hat{a}$ that is not a valid trace of $C_{\Im}$, i.e. $\hat{\sigma} \in L(\hat{C}_{\Re}) \cap L(\hat{C}_{\Im})$ and $\hat{\sigma}.\hat{a} \in L(\hat{C}_{\Re}) \setminus L(\hat{C}_{\Im})$. Let s be a global state of $\hat{C}_{\Im}$ such that $s_0=\hat{\sigma} \Rightarrow s$, $\hat{a} \in Ref_{\Im}(s)$ and $\hat{a} = <a, id>$. It means that in $\Re$ for all $i \in id$, the action a causes a supplementary transition from a local state that corresponds to $s(i)$ in $\Im$. By virtue of Proposition 1, if state $s(i)$ is reachable in LTS $C_{\Im}$, then it is also reachable in LTS $M_{\Im}$. Specifically, let s' be a global state of $\hat{M}_{\Im}$ such that $s'(i) = s(i)$. As a result, we obtain $\hat{a} \in Ref_M(s')$. The MTM $\hat{M}_{\Im}$ is initially connected, hence there exists a trace $\beta \in SC(\hat{M}_{\Im})$ such that $s_0=\beta \Rightarrow s'$. By construction, the test suite $RC(\hat{M}_{\Im})$ has an invalid trace $\beta.\hat{a}$, where $\hat{a} \in Ref_M(s')$. Thus if $\Re \in \Re_{ref}$ and $L(C_{\Re}) \neq L(C_{\Im})$, then $\Re$ cannot pass the test case $\beta.\hat{a} \in RC(\hat{M}_{\Im})$.

◊

We illustrate the process of test derivation for refusal faults using our example again (Figure 1 and Figure 2). The sequence $<a, \{1, 2\}>.<b, \{1\}>.<d, \{1\}>$ can be used as a state cover of the augmented MTM $\hat{M}_{\Im}$. The corresponding refusal cover is $Ref(s_{00}).<a, \{1, 2\}>.Ref(s_{11}).<b, \{1\}>.Ref(s_{21}).<d, \{1\}>.Ref(s_{01})$, where for example the refusal set of state s_{21} is $Ref(s_{21}) = \{<a, \{1\}>, <a, \{2\}>, <a, \{1, 2\}>, <b, \{1\}>, <b, \{2\}>, <b, \{1, 2\}>, <c, \{1\}>, <c, \{2\}>, <d, \{1\}>, <d, \{2\}>, <d, \{1, 2\}>, <e, \{1\}>, <e, \{1, 2\}>\}$ (see Figure 3).

4.3 Detecting both missing and superfluous transitions

In a more general case, it is likely that both faults, missing and superfluous transitions, may occur in an IUT of a concurrent system simultaneously. The two types of faults are independent from each other. Assume that $Acc(s)$ and $Ref(s)$ are the acceptance and refusal sets of state s of an LTS, respectively. Clearly, $Acc(s) \cap Ref(s) = \varnothing$. Thus, a test suite can be derived to cover both faults by combining test suites for accepting and refusal faults.

We define the acceptance/refusal fault model as $<\Im, \approx, \Re_{mix}>$, where $\Re_{mix}$ is the set of all implementation systems $\Re$ consisting of n initialized LTSs according to specification $\Im$ such that the components of an implementation $\Re$ contain zero or more missing transitions as well as zero or more superfluous ones. The set of local actions of a component in $\Re$ is a subset of the global alphabet $A_{\Im}$.

Due to the independence property between acceptance and refusal faults, a test suite that is complete w.r.t. the fault model can now be obtained by combining a transition cover and a refusal cover of the augmented composite machine, $TC(\hat{C}_{\Im})$

$\cup RC(\hat{C}_{\Im})$. As carried out above, a shorter test suite for the combined fault model can be obtained if the augmented MTM $\hat{M}_{\Im}$ is used instead, $TC(\hat{M}_{\Im}) \cup RC(\hat{M}_{\Im})$.

Both test suites $TC(\hat{M}_{\Im})$ and $RC(\hat{M}_{\Im})$ can be merged into a single test suite $TRC(\hat{M}_{\Im})$ under the assumption that the transition cover used in $TC(\hat{M}_{\Im})$ is also a state cover in $RC(\hat{M}_{\Im})$. Since the composite machine is initially connected, this property is always fulfilled. Taking our example of Figure 1 and Figure 2, a test suite complete w.r.t. $<\Im, \approx, \Re_{mix}>$ can be given, for example, as a single test sequence $TRC(\hat{M}_{\Im}) = Ref(s_{00}).<a,\{1, 2\}>. Ref(s_{11}).<b,\{1\}>.Ref(s_{21}).<c, \{1, 2\}>.<b,\{1\}>.<d, \{1\}>.Ref(s_{01}).<e, \{2\}>$, where $Ref(s_x)$ is the refusal set of the global state s_x in $M_{\Im_1}$.

4.4 Detecting transfer faults

Faults considered in the previous sections do not exhaust the variety of potential faults which may occur in implementations of a given concurrent system. We demonstrate that an MTM can also be useful to derive tests detecting other types of faults, in particular, transfer faults in component machines.

An implementation $\mathcal{M}_i$ of a component machine M_i of the concurrent system $\Im = M_1 \| \ldots \| M_n$ is said to have *transfer faults* if $\mathcal{M}_i$ can be obtained from M_i by changing only the tail state of some transitions. We denote with $\aleph(M_i)$ the set of all possible implementations with transfer faults. The set $\aleph(M_i)$ is finite since any implementation $\mathcal{M}_i$ has at most $|S_i|$ states. In testing compound systems, one often assumes that faults are located in a single component only. These assumptions lead us to the fault model $<\Im, \approx, \Re(M_i)>$, where $\Re(M_i)$ is the set of all concurrent systems $\Re = M_1 \| \ldots \| \mathcal{M}_i \| \ldots \| M_n$, with $\mathcal{M}_i \in \aleph(M_i)$, including the system $\Im$ itself.

Transfer faults that are considered when a component is tested in isolation require a state identification facility. In the realm of I/O-FSMs, a characterization set, harmonized state identifiers, a distinguishing sequence, or UIO-sequences serve as examples of this facility [4], [23]. These notions were also redefined for the LTS model assuming trace semantics [30]. In particular, a *characterization set* for machine $M = (S, A, \rightarrow, s_0)$ is the set $W(M) \subseteq A^*$ such that for all $s_k, s_l \in S, k \neq l$, there exists a sequence $\alpha \in Pref(W(M)) \cap (Tr(s_k) \oplus Tr(s_l))$. The idea in constructing a test suite complete w.r.t. transfer faults of M in isolation with the remaining machines in $\Im$ is to concatenate a sequence of actions covering a given transition with every sequence of the characterization set $W(M)$ and repeat this process for each transition of M (this is the W-method). The characterization set $W(M)$ allows to identify the tail state of each transition in the IUT.

Adopting this approach to our needs of testing a concurrent system, we define a set of transfer sequences of system $\Im$ that cover all transitions in the component M_i as follows. Given an augmented MTM $\hat{M}_{\Im}$, let (p, a, p') be a local transition of M_i and $(s, \hat{a}, s')$ be the corresponding global transition, i.e. $s(i) = p$, $s'(i) = p'$, and $\hat{a} = <a, \mathbf{id}(a)>$. To cover the local transition (p, a, p'), a sequence $\hat{\gamma}.\hat{a}$ must be used, where $\hat{\gamma} \in Pref(SC(\hat{M}_{\Im}))$ such that $s_{\Im} = \hat{\gamma} \Rightarrow s$. The union of such sequences over all local transitions of M_i constitutes a *cover of local transitions* of M_i in $\hat{M}_{\Im}$, denoted $TC_i(\hat{M}_{\Im})$. Thanks to Proposition 1 and Proposition 2, we can always determine such a cover for a concurrent system without unexecutable local transitions directly from an MTM, avoiding thus the exploitation of the composite machine.

Proposition 5. Given a concurrent system $\Im$, an augmented MTM $\hat{M}_{\Im}$, and the fault model $<\Im, \approx, \Re(M_i)>$. Let $TC_i(\hat{M}_{\Im})$ be a cover of local transitions of M_i in

$\hat{M}_{\Im}$. If a characterization set $W(M_i)$ comprises only local actions of M_i, then the set $\{\hat{\gamma}.\hat{a}.\hat{\sigma} \mid \hat{\gamma}.\hat{a} \in TC_i(\hat{M}_{\Im}) \wedge \sigma \in W(M_i)\}$ is a test suite complete w.r.t. $<\Im, \approx, \Re(M_i)>$.

The case when a characterization set $W(M_i)$ uses shared actions is a bit more involved and we will report on it in an extended version of this paper. In our example, we use the augmented MTM $\hat{M}_{\Im 1}$ to derive a test suite to detect transfer faults in module *B*. There are three global transitions that represent all local transitions of *B* labeled with *a*, *c*, and *e* in Figure 2. In the MTM $M_{\Im 1}$, we use the three transfer sequences, namely *a*, *abc*, and *abde*. The two states of *B* are distinguished by the local action *e*, i.e. $W(B) = \{e\}$. The set of sequences $\{<a, \{1, 2\}>.<e, \{2\}>, <a, \{1, 2\}>.<b, \{1\}>.<c, \{1, 2\}>.<e, \{2\}>, <a, \{1, 2\}>.<b, \{1\}>.<d, \{1\}>.<e, \{2\}>.<e, \{2\}>\}$ yields a test suite that detects any transfer fault in implementations of module *B*.

5 Conclusions

The paper proposed the application of a partial-order model MTM for test derivation according to a chosen fault model. It was shown that test suites derived from an MTM exhibit the same degree of fault coverage as test suites from the composite machine of a concurrent system assuming the fault models of acceptance, refusal, and transfer faults. The advantage of an MTM over the composite machine is due to the fact that an MTM has a largely reduced state space in many cases resulting in much smaller test suites. However, testers are required that possess a higher degree of controllability to perform a deterministic execution of a specific test case. It was shown how such testers can be constructed using the concept of occurrence sets of actions.

We considered a tester that exercises global control over all actions in a concurrent system, shared and local ones. It would be interesting to consider somewhat more restrained testers which have no control over certain actions. It seems that the results of this paper can easily be applied to the case when certain local actions cannot be observed. We could simply replace these actions by a non-observable action and determinize the LTSs obtained. The situation with non-observable shared actions seems a bit more complicated. Once certain actions become invisible, a tester cannot directly observe a fault, which in turn might later be tolerated by other components. The problem seems similar to testing in context considered in the realm of I/O-FSMs [25], [26], but more research is required in this direction.

Acknowledgment. This work was in part supported by the NSERC grant OGP0194381.

6 REFERENCES

[1] A. V. Aho, A. T. Dahbura, D. Lee, M. Ü. Uyar: *An optimization technique for protocol conformance test generation based on UIO sequences and rural Chinese postman tours*; IEEE Transactions on Communications, vol. 39, no. 11, 1991; pp. 1604-1615.

[2] B. S. Bosik, M. Ü. Uyar: *Finite state machine based formal methods in protocol conformance testing: from theory to implementation*; Computer Networks and ISDN Systems 22 (1); 1991; pp. 7-33.

[3] G. v. Bochmann, A. Petrenko: *Protocol testing: Review of methods and relevance for software testing*; Invited Paper, ACM International Symposium on

Software Testing and Analysis (ISSTA'94), USA, 1994.
[4] G. v. Bochmann, A. Petrenko, M. Yao: *Fault coverage of tests based on finite state machines*; 7th IWPTS; Tokyo, Japan; 1994; pp. 55-74.
[5] E. Brinksma: *A theory for the derivation of tests*; 8th IFIP Int'l Conference on Protocol Specification, Testing, and Verification (PSTV'88); 1988, pp. 63-74.
[6] R. H. Carver, K. C. Tai: *Test sequence generation from formal specifications of distributed programs*; 15th ICDCS; Vancouver, Canada; 1995; pp. 360-367.
[7] T. S. Chow: *Test design modeled by finite-state machines*; IEEE Transactions on Software Engineering, SE-4, No.3, 1978, pp. 178-187.
[8] C. Corbett: *An empirical evaluation of three methods for deadlock analysis of Ada tasking programs*; International Symposium on Software Testing and Analysis (ISSTA'94); Seattle, USA; 1994.
[9] C. Ghezzi, M. Jazayeri, D. Mandrioli: *Fundamentals of software Engineering*; Prentice-Hall, 1991.
[10] A. Gill: *Introduction to the theory of finite state machines*; McGraw-Hill, NY; 1962.
[11] P. Godefroid, P. Wolper: *Using partial orders for the efficient verification of deadlock freedom and safety properties*; Formal Methods in System Design, vol. 2, no. 2 (April 1993); pp. 149-164.
[12] P. Godefroid: *Partial-order methods for the verification of concurrent systems*; LNCS 1032; Springer, 1996.
[13] P. Godefroid, D. Peled, M Staskauskas: *Using partial-order methods in the formal validation of industrial concurrent programs*; International Symposium on Software Testing and Analysis (ISSTA'96); San Diego, USA; 1996; pp. 261–269.
[14] J. E. Hopcroft, J. D. Ullman: *Introduction to automata theory, languages, and computation*; Addison-Wesley; 1979.
[15] M. C. Kim, S. T. Chanson, S. W. Kang, J. W. Shin: *An approach for testing asynchronous communicating systems*; 9th IWTCS 1996; Germany; Sep. 1996.
[16] P. V. Koppol, K. C. Tai: *Conformance testing of protocols specified as labeled transition systems*; 8th International Workshop on Protocol Test Systems (IWPTS'95); Paris, France; 1995; pp. 143-158.
[17] P. V. Koppol, K. C. Tai: *An incremental approach to structural testing*; International Symposium on Software Testing and Analysis (ISSTA'96); San Diego, California, USA; 1996; pp. 14-23.
[18] R. Langerak: *A testing theory for LOTOS using deadlock detection*; 9th PSTV 1989; Enschede, The Netherlands; North Holland, 1990.
[19] D. Lee, K. K. Sabnani, D. M. Kristol, S. Paul: *Conformance testing of protocols specified as communicating FSMs*; IEEE INFOCOM'93; USA; 1993.
[20] A. Mazurkiewicz: *Trace theory*; Advances in Petri Nets (part II); LNCS 255; Springer, 1986; pp. 279-324.
[21] Ch. E. McDowell, D. P. Helmbold: *Debugging concurrent programs*; ACM Computing Surveys, vol. 21, no. 4 (Dec. 1989); pp. 593-622.
[22] S. Naito, M. Tsunoyama, Fault detection for sequential machines by transition tours; Fault Tolerant Computing Systems, 1981, pp. 238-243.
[23] A. Petrenko, G. v. Bochmann and M. Yao: *On fault coverage of tests for finite state specifications*; Special issue on Protocol Testing, Computer Networks and ISDN Systems, vol. 29, 1996, pp. 81-106.
[24] D. Peled, W. Penczek: *Using asynchronous Büchi automata for efficient automatic verification of concurrent systems*; 15th PSTV 1995; Warsaw, Poland; Chapman & Hall, 1995; pp. 305-321.
[25] A. Petrenko, N. Yevtushenko, G. v. Bochmann: *Fault models for testing in*

context; 9th FORTE 1996; Chapman & Hall, 1996; pp. 163-178.

[26] A. Petrenko, N. Yevtushenko, G. v. Bochmann and R. Dssouli: *Testing in context: framework and test derivation*; Computer Communications, 19, 1996, pp. 1236-1249.

[27] K. C. Tai, R. H. Carver: *Testing of distributed programs*; in A. Zomaya (ed.): *Handbook of Parallel and Distributed Computing*; McGraw Hill; 1995; pp. 956-979.

[28] K. C. Tai, R. H. Carver, E. E. Obaid: *Debugging concurrent Ada programs by deterministic execution*; IEEE Transactions on Software Engineering, vol. 17, no. 1 (Jan. 1991); pp. 45-63.

[29] R. N. Taylor, D. L. Levine, Ch. D. Kelly: *Structural testing of concurrent programs*; IEEE Transactions on Software Engineering, vol. 18, no. 3; 1992; pp. 206-215.

[30] Q. M. Tan, A. Petrenko, G. v. Bochmann: *Checking experiments with labeled transition systems for trace equivalence*; 10th IFIP Workshop on Testing Communicating Systems (IWTCS'97), Korea, 1997.

[31] J. Tretmans: *Conformance testing with labelled transition systems: Implementation relations and test generation*; Special issue on Protocol Testing, Computer Networks and ISDN Systems, Vol.29, 1996.

[32] A. Ulrich, S. T. Chanson: *An approach to testing distributed software systems*; 15th PSTV 1995; Warsaw, Poland; pp. 107-122; 1995.

[33] A. Ulrich, H. König: *Specification-based testing of concurrent systems*; IFIP Joint Int'l Conference on Formal Description Techniques, and Protocol Specification, Testing, and Verification (FORTE/PSTV'97); Osaka, Japan; 1997.

[34] M. P. Vasilevski: *Failure diagnosis of automata*; Cybernetics, Plenum, New York; No. 4, 1973, pp. 653-665.

[35] M. Yannakakis and D. Lee: *Testing finite state machines*; in Proceedings of the 23d Annual ACM Symposium on Theory of Computing, 1991, pp. 476-485.

[36] S. Duri, U. Buy, R. Devarapalli, and S. M. Shatz: *Application and experimental evaluation of state space reduction methods for deadlock analysis in Ada;* ACM Trans. Software Engineering and Methodology, Vol.3, No. 4, Oct. 1994, pp.340-380.