Testing non-deterministic state machines with fault coverage

Susumu Fujiwara* and Gregor v. Bochmann

Département d'informatique et de recherche opérationnelle
Université de Montréal

Abstract
The selection of appropriate test cases is an important issue in the development of communication protocols. Various test case selection methods have been developed for the case that the protocol specification is given in the form of a deterministic finite state machine (FSM). This paper present a new method which applies in the case of non-deterministic specifications and implementations. The testing process is more complex if the specification, or even the implementation, is non-deterministic. Nevertheless, under appropriate assumptions, the described test case selection method leads to a finite set of finite test cases for a given specification which guarantees that any deviation of the implementation from the specification will be detected. The paper presents the new test selection method in a framework for testing non-deterministic systems and demonstrates its use with small examples.

1. Introduction

Testing plays an important role during the development of computer hardware and software. The selection of appropriate test cases is an important issue in this context. We assume in this paper that a specification of the desired behavior of the system component to be tested is available. Such a specification can be taken as the basis for the development of a suite of test cases, or for evaluating the coverage of a given test suite. This paper deals with the development of a test suite covering the behavior of a system component defined by a finite state machine specification. In contrast to most methods described in the literature, we allow for non-deterministic specifications and implementations.

The issue of testing implementations in respect to a specified behavior has recently received much attention in the area of communication protocols [Rayn 87, Sari 89]. In order to validate the protocol implementation, a set of test cases, usually called a "test suite", is needed to determine whether an implementation conforms to its specification. In the case that a formal specification of the protocol is available, the test selection and fault analysis can be based on this specification [Sari 89, Boch 89m].

This paper considers the case that the specification and its implementation may have non-deterministic behaviors. We assume that both the specification and the implementation can be modelled by finite labelled transition systems. In addition to finite state machines, there are many languages which are based on (in general infinite) labelled transition systems, such as CCS [Miln 80], CSP [Hoar 85], and LOTOS [Bolo 87]. The test method described in this paper can be adapted to (subsets of) these languages.

Most test selection methods for (deterministic) finite state machines [Nait 81, Chow 78, Gone 70, Sabn 88] assume that the purpose of testing is to demonstrate that the behavior of

---

* S. Fujiwara is with NTT Network Information Systems Laboratories, Musashino-shi, Japan, and was on leave at the Université de Montréal during 1989-1990.

the implementation under test (IUT) is equivalent to the behavior defined by the specification. In the case of non-deterministic machines, however, it is more appropriate to demonstrate that the IUT implements the specification, where "implements" is an ordering relation which, in general, allows several different implementation to satisfy a given specification. There are many "implements" relations that have been proposed in the literature, such as conformance [Brin 88], failure [Deni 84], reduction [Brin 87], extension [Brin 87], failure trace and generalized failure [Lang 89], and conformance based on acceptance [Henn 85, Boch 89f]. In this paper we are concerned with the failure preorder which is the intersection of the conformance relation and the preorder of trace inclusion [Tret 89]. The corresponding equivalence relation is testing equivalence.

In the case of testing for the conformance relation between the IUT and the specification, the concept of a "canonical tester" has been introduced [Brin 88] which can be derived from the specification which is assumed to be given in LOTOS [Bolo 87]. However, such a tester has in general an infinite behavior and is therefore not suitable as a test suite. The so-called CO-OP method [Weze 89] can be used to construct finite canonical testers in the case of finite behaviors.

The purpose of this paper is to describe a new method for developping a finite test suite which checks whether an implementation satisfies the failure preorder in respect to a given specification, which is given in the form of a non-deterministic FSM. Assuming that the behavior of the implementation can also be described by a non-deterministic FSM with a limited number of states, and assuming that a reset function is correctly implemented, the derived test suite guarantees that any deviation from the failure preorder relation will be detected.

Section 2 introduces the basic notation for labelled transition systems which is the theoretical framework in which the specifications and implementations to be tested are described. It also introduces the notation of "multi-state" which corresponds to the set of states in which a given system may be after a given sequence of interactions. This notion simplifies the treatment of the non-deterministic behavior of the specified systems.

Section 3 introduces an abstract framework for the testing of non-deterministic systems and discusses the failure preorder relation which is taken as the implementation relation to be verified through testing. Finally, an algorithm is presented which allows to verify the failure preorder relation if both the specification and the implementation are given in the form of (non-deterministic) finite state machines. Section 4 considers the case that the description of the implementation is not known, and its conformance to the specification must be checked through testing. It is shown that the above algorithm can be combined with methods for the identification of the states in the implementation and a systematic coverage of the possible behaviors in order to obtain an algorithm for the development of a test suite with guaranteed fault detection power.

This paper is a shortened version of [Fuji 91] which contains a more detailed discussion of these issues and the proofs of the mentioned theorems. It also includes the application of the same approach to the case of testing for testing equivalence.

2. Notations for non-deterministic state machines

2.1 Labelled transition systems

We use a labelled transition system to model state machines which represent the

specification or an implementation. A labelled transition system (LTS) is defined as a 4-tuple <St, L, T, S0> where:
- St is a (countable) non-empty set of states;
- L is a (countable) set of observable actions;
- $T = \{-\mu\rightarrow \subseteq St \times St \mid \mu \in L \cup \{\tau\}\}$ is a set of binary relations on St;
- $S0 \in St$ is the initial state of the system.

We write $P\text{-}\mu\rightarrow P'$ for a pair of states P and P' that belongs to the relatiøn $\text{-}\mu\rightarrow$; it is also called a transition. $\text{-}\tau\rightarrow$ represents internal, non-observable transitions. In this paper, we use a LTS with finite number of states (St), actions (L) and no t-transitions to model the system behaviors. We often make no notational distinction between a state S and a transition system consisting of all states accessible from S. In the later sections, we often consider a transition system consisting of a component S, representing the specification, and a component I representing an implementation. Further notations are defined in the tables 2A and 2B.

Table 2.A. Notation for labelled transition systems

notation	**meaning**
L	set of observable actions; a,b,c,... denote elements of L
L*	set of strings over L; σ denotes such strings; ε is the empty string
L'	$L \cup \{\tau\}$; μ denotes elements of L'
St	set of states; P,Q,S, and I denote such states
$\hat{S}$	set of states reachable from S
$P\text{-}\mu_1 \ldots \mu_n \rightarrow Q$	there exist P_i for $0 \leq i \leq n$ such that $P = P_o - \mu_1 \rightarrow P_1 \ldots - \mu_n \rightarrow P_n = Q$
$P\text{-}\mu_1 \ldots \mu_n \rightarrow$	there exists Q such that $P\text{-}\mu_1 \ldots \mu_n \rightarrow Q$
$P+\mu_1 \ldots \mu_n \rightarrow$	no Q exists such that $P\text{-}\mu_1 \ldots \mu_n \rightarrow Q$
$P = \varepsilon \Rightarrow Q$	$P - \tau^n \rightarrow Q$ $(1 \leq n)$ or $P = Q$
$P = a \Rightarrow Q$	there exist P_1, P_2 such that $P = \varepsilon \Rightarrow P_1 - a \rightarrow P_2 = \varepsilon \Rightarrow Q$
$P = a_1 \ldots a_n \Rightarrow Q$	there exist Pi for $0 \leq i \leq n$ such that $P = P_o = a_1 \Rightarrow P_1 \ldots = a_n \Rightarrow P_n = Q$
$P = \sigma \Rightarrow Q$	$P = a_1 \ldots a_n \Rightarrow Q$ with $\sigma = a_1 \ldots a_n$
$P = \sigma \Rightarrow$	there exists Q such that $P = \sigma \Rightarrow Q$
$P \neq \sigma \Rightarrow$	no Q exists such that $P = \sigma \Rightarrow Q$
Tr(P)	$\{ \sigma \in L^* \mid P = \sigma \Rightarrow \}$, i.e. the set of traces accepted starting in state P
out(P)	$\{a \in L \mid P = a \Rightarrow \}$, i.e. the possible next actions from state P

Table 2.B: Notation related to multi-states and transitions between them

notation	**meaning**
$\hat{S}$, $\hat{I}$	set of all multi-states reachable from S_0 and I_0, respectively

$\mathbf{S}$, $\mathbf{S_i}$, $\mathbf{S_j}$	multi-states included in $\hat{S}$, in particular, $\mathbf{S_0}$={S0}
$\mathbf{I}$, $\mathbf{I_k}$, $\mathbf{I_l}$	multi-states included in $\hat{I}$; $\mathbf{I_0}$={I0} in particular
$\mathbf{S_i = \sigma \Rightarrow S_j}$	$\mathbf{S}_j = \{S_j \mid S_i \in \mathbf{S}_i,\ S_i = \sigma \Rightarrow S_j\}$, $\mathbf{S}_j \neq \varnothing$
$\mathbf{S_i = \sigma \Rightarrow}$	there exists $\mathbf{S_j}$ such that $\mathbf{S}_i = \sigma \Rightarrow \mathbf{S}_j$
$\mathbf{S_i \neq \sigma \Rightarrow}$	no $\mathbf{S_j}$ exists such that $\mathbf{S}_i = \sigma \Rightarrow \mathbf{S}_j$

2.2. Example specification

Figure 1 shows a transition diagram defining a non-deterministic FSM. The initial state is S0. In this state, the action a is possible and may lead to the states S1 or S2. In these states the action b is possible and leads deterministically to states S3 and S0, respectively. The action c is also possible in state S2, etc. We use the notation where fat arrows represent deterministic transitions, that is, there is only one transition for the given action in the given state.

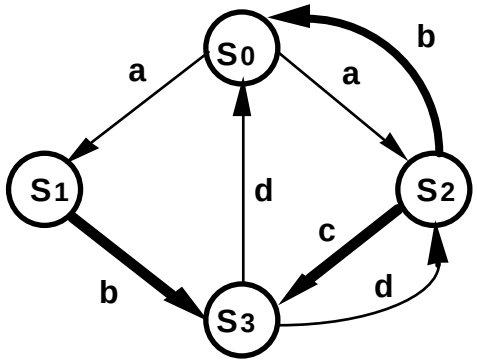

Figure 1: Specification S

2.3. Describing non-determinism by multi-states

In the case of non-determinism, the state machine may reach one out of several states for a given sequence of actions. In order to simplify the notation concerning all these possibilities, we introduce the notion of a multi-state which is a set of state in which the transition system may be after a given sequence of actions. This approach is similar to the classical proof that for each non-deterministic FSM, there is an equivalent deterministic FSM that accepts the same language of input sequences.

We define (S **after** σ) to be the set of states which can be reached by applying sequence σ starting from state S, that is, S **after** $\sigma = \{S_i \mid S = \sigma \Rightarrow S_i\}$. For the example of Figure 1, for instance, we have (S0 **after** a)={S1,S2}, (S0 **after** a.b) ={S0,S3}, (S0 **after** a.d)= $\varnothing$. The set of states (S0 **after** s) may be considered, in some sense, as a single "state" from the system behavior point of view. If we explore the LTS starting in S0 by applying the test sequence s, we will reach one state of (S0 **after** s). The behavior which will be observed after s is determined by the set of possible states. The notion (S0 **after** s) contains the non-determinism because it includes all possible states that can be reached. We use notations analogous to those for states, as given in the tables 2A and 2B.

We can define a testing tree for a non-deterministic machine in terms of multi-states which are reached after a given sequence of interactions, similar as described for deterministic FSM's [Chow 78]. It is important to note that this tree is deterministic (at each node, there is only one branch with a given action label), however, in contrast to the case of

deterministic machines, each node represents a set of possible states (a multi-state). As an example, Figure 2 shows a testing tree for the specifications of Figure 1. A testing tree can be derived using the following algorithms, which proceeds in several stages:

Algorithm 2.A (testing tree)

The algorithm constructs the tree in a breadth-first manner. We write G_k for the set of multi-states (i.e. nodes) derived up to the depth k.

(a) $k := 0$; $G_0 = \{ \{S0\} \}$

(b) $k := k+1$; for each multi-state of G_k which is not included in $G_{k-1} \cup ... \cup G_0$, derive all possible next multi-states by using the transition rule $S_i = a \Rightarrow S_j$; add the derived multi-states to G_{k+1} .

(c) If $G_{k+1} \subseteq G_0 \cup G_1 \cup ... \cup G_k$ (no new multi-set was included), then stop. Otherwise go back to step (b) and continue.

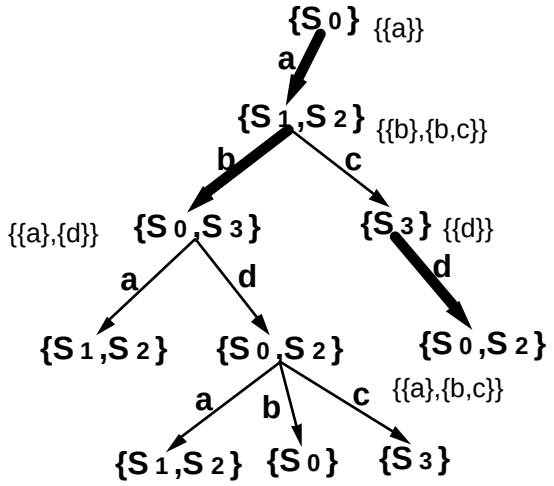

Figure 2: Testing tree for specification S (shown in Figure 1)

3. Testing non-deterministic state machines

3.1 Abstract testing framework

We use a framework for the discussion of the testing process described in [Brin 89] (see also [Fuji 91] for more details). In particular, we use the concepts of a test process t which is a LTS with a finite set of traces $Tr(t)$. Ω denotes the set of all test processes over the set of actions L. A test run of a LTS P with t is a derivation $P \| t = \sigma \Rightarrow P' \| t'$ which corresponds to the coupled execution of P and t yielding the interaction sequence σ. A test run is completed if the derived $P' \| t'$ satisfies $\forall \mu \in L^{'} : P^{'} \| t^{'} + \mu \rightarrow$. The observations of a test run yield a result which is either successful (also written as T) or deadlock (also written as F). A completed test run is successful if the derived $t^{'}$ of $P' \| t'$ satisfies $\forall \mu \in L^{'} : t^{'} + \mu \rightarrow$ and deadlock if not successful. The set {successful, deadlock} is the set of observationsΣ. The set of all possible observations of $P \| t$ is denoted as $O(P \| t)$, which is a subset of Σ .

We also use the concept of a verdict for a test t, which is a mapping $\nu_t : \text{Power}(\Sigma) \rightarrow \{\text{pass, fail}\}$ with the meaning that an implementation P passes a test t, if

$\nu_t(O) =$ pass, where O is the set of all observations obtained through successive test runs with the test process t. For the case of failure preorder testing, as discussed below, the following verdict can be used: $\nu_t(O)$ = pass if $O \subseteq O(S \parallel t)$ and = fail otherwise. Finally, we use the concept of a test suite, which is a set of tests, also called "test cases". We say that an implementation I passes a test suite if it passes all test cases in the suite.
It is important to note that for a given system P which is tested, there may be several possible observations that may be obtained for a given test process. This is due to the non-determinism of the tested system. In order to obtain all possible observations for a given system, it is therefore necessary, in general, to repeat several test runs for each test process. However, if the obtained set of observations for a given test case is not the complete set Σ (in our case {T,F}), one can never be sure that all possible observations have been obtained, because of the non-deterministic nature of the tested system. In the following, we assume that the set of all possible observations for a given test case can be obtained by repeating test runs a certain, limited number of times. We assume a certain fairness between the different behaviors allowed by the non-determinism of the tested system.

3.2 Failure preorder as implementation relation

If we want to test an implementation, we need to define an implementation relation which answers such questions as:

- What is the meaning of a valid implementation?
- What is the condition for valid implementation?

In the case of deterministic specifications, the relation is usually equal behavior, that is, the implementation should provide the same traces of interaction sequences. For non-deterministic specifications, however, several implementation relations have been proposed in the literature, such as conformance [Brin 88], failure preorder [Deni 84], reduction [Brin 87], extension [Brin 87], failure trace preorder and generalized failure preorder [Lang 89] and conformance based on acceptance [Hern 85, Boch 89f]. In this paper, we use the failure preorder relation (written $\leq_F$) as an implementation relation. It is noted that failure preorder can be constructed from trace preorder $(\leq_T)$ and the conformance relation as follows [Tret 89]: $I \leq_F S$ iff $(I \leq_T S) \wedge (I \textbf{ conf } S)$.

In the following, we usually assume a LTS consisting of two disconnected parts, representing a specification S and an implementation I. We use S and I also to represent the initial state of the specification and implementation, respectively. S_i and I_k are states which are reachable from S and I respectively. The <u>failure preorder</u> relation between two states I and S holds, written $I \leq_F S$, iff $\forall \sigma \in L^*: \forall A \subset L:$

if $\exists I_k : \forall a \in A : I = \sigma \Rightarrow I_k \neq a \Rightarrow$

then $\exists S_i : \forall a \in A : S = \sigma \Rightarrow S_i \neq a \Rightarrow$

The <u>testing equivalence</u> relation between two states of I and S holds, written $I \approx_F S$, iff $(I \leq_F S)$ and $(S \leq_F I)$

By using the notion of testing described above, we can derive the theorem below which gives us the means to check by testing whether $I \leq_F S$. This theorem can be interpreted as follows. If $O(S \parallel t) = \{T\}$, then $O(I \parallel t)$ should be $\{T\}$. If $O(S \parallel t) = \{F\}$, then $O(I \parallel t)$

should be $\{F\}$. That is, if a process t is always successful with a specification S, then t must also be always successful with the implementation. And the same should hold for deadlock.

Theorem 3.A (check failure preorder by testing)

$I \leq_F S$ iff $\forall t \in \Omega : O(I \parallel t) \subseteq O(S \parallel t)$

3.3. Comparing observations on multi-states

In Section 2.3, the notion of multi-states was introduced to model the fact that a non-deterministic machine, after a given sequence of actions, may be in any one of a set of states. Since we are often interested in testing a system after a given sequence of observed actions, we can use the following concepts which are extensions of those above.

Given a multi-state **S,** the observation set of **S** is defined as:

$O(\mathbf{S} \parallel t) = \cup_{Si \in \mathbf{S}} O(S_i \parallel t)$ if **S** is not empty, and empty otherwise. The failure preorder relation between multi-states is written as $\mathbf{S}_i \leq_F \mathbf{S}_j$ and is true iff $\forall t \in \Omega : O(\mathbf{S}_i \parallel t) \subseteq O(\mathbf{S}_j \parallel t)$. We also have the following theorem which links the failure preorder relations between single states (or LTS's) and multi-states. According to this theorem, $I \leq_F S$ implies that after observing a certain trace of actions, the corresponding multi-states in S and I also satisfy the failure preorder relation.

Theorem 3.B (relation between states and multi-states concerning failure preorder)

$I \leq_F S$ iff $\forall \sigma \in L^* : (I \text{ after } \sigma) \leq_F (S \text{ after } \sigma)$

3.4. An approach to testing failure preorder

The following theorem states that an implementation I satisfies the failure preorder to a specification S exactly if a certain type of mapping exists from the multi-states of I to sets of multi-states of S. The existence of such a mapping can be checked by testing, as explained in Section 4. This theorem is therefore the basis for the test suite development described in this paper.

Theorem 3.C (mapping for $I \leq_F S$)

$I \leq_F S$ iff there exists a mapping f: $\hat{\underline{I}} \rightarrow \text{Power}(\hat{\underline{S}})$ which satisfies the following three conditions:

(0) $\{S0\} \in f(\{I0\})$

(1) $\forall \mathbf{S}_i \in f(\mathbf{I}_k) : \quad \mathbf{I}_k < \mathbf{S}_i$

(2) If $\mathbf{I}_k = a \Rightarrow \mathbf{I}_l$, then $\forall \mathbf{S}_i \in f(\mathbf{I}_k) : \exists \mathbf{S}_j \in f(\mathbf{I}_l)$ such that $\mathbf{S}_i = a \Rightarrow \mathbf{S}_j$

where < is a relation, which we call <u>local action preorder</u>, which holds between **I** and **S**, if the following two conditions hold:

(i) $\forall I_k \in \mathbf{I} \quad \exists S_i \in \mathbf{S}$ such that $\text{out}(S_i) \subseteq \text{out}(I_k)$

(ii) $\text{out}(\mathbf{I}) \subseteq \text{out}(\mathbf{S})$.

Here the so-called outset is defined by : $\text{out}(\mathbf{S}) = \cup_{Si \in \mathbf{S}} \text{out}(S_i)$, where $\text{out}(S_i)$ is the set of possible next actions in state S_i.

3.5. Algorithm for checking failure preorder

The theorem above is the basis for the algorithm described below, which can be used for checking the failure preorder relation between two labelled transitions systems. This

algorithm can be used for comparing an implementation I with its specification S if we assume that the description of the implementation, in terms of a labelled transition system, is given. In Section 4, we will show how we can use the same algorithm for deriving a test suite for a given specification, without using the knowledge about the internal structure of the implementation.

We will explain the following algorithm with an example using the specification S of Figure 1 and the implementation I-1 of Figure 3. The testing tree for I-1 is shown in Figure 4. It has 4 distinct multi-states {I0}, {I1}, {I0,I1}, and {I2}. We start by checking condition (0) of Theorem 3.C which can be satisfied by posing f({I0}) = { {S0} }. Then we check that condition (1) is satisfied for this pair {I0} and {S0}, which is clearly true. Then we consider condition (2) and construct the mapping accordingly. In this case, the only action to be considered is the action a, which leads to the multi-state {I1} for the implementation I-1 and to {S1,S2} for the specification. We therefore pose f({I1}) = { {S1,S2} }. Since this is a new element of the mapping, we have to consider conditions (1) and (2) for this element, and the checking continues recursively. In some instances, it may be necessary to add another multi-state to an element of the mapping, for instance, the mapping for {I0} must be extended to f({I0}) = { {S0}, {S0,S2} } (see below). In general, the following algorithm can be used.

Algorithm 3.A (checking failure preorder)

This algorithm checks whether the relation $\mathbf{I}0 \leq_F \mathbf{S}0$ holds. The mapping f: $\hat{\underline{\mathbf{I}}} \rightarrow$ Power ($\hat{\underline{\mathbf{S}}}$) is represented by a set G of pairs of multi-states (**S,I**), where G contains exactly those pairs (**S,I**) for which **S** is included in f(**I**). The algorithm constructs the set G in several steps, starting with the initial set G0 = ({S0}, {I0}).

In each step (say k+1), we derive a new set of pairs (**S**, **I**), written as G_{k+1}, from the set G_k. G is the union of all these G_k. In the step (k+1), we choose a pair (**S**, **I**)ŒG_k that is not included in $G_0 \cup G_1 \cup ... \cup G_{k-1}$. For each a $\in$ out (**I**) we consider the new pair (**S', I'**) where $\mathbf{I} = a \Rightarrow \mathbf{I}'$ and $\mathbf{S} = a \Rightarrow \mathbf{S}'$. If $\mathbf{S} \neq a \Rightarrow$ then $\mathbf{I} \leq_F \mathbf{S}$ is not satisfied. If the new pair (**S', I'**) is not in the set G_k then we check that $\mathbf{I}' < \mathbf{S}'$. If this is satisfied the pair (**S', I'**) is included in G_{k+1}, otherwise $\mathbf{I} \leq_F \mathbf{S}$ is not satisfied. If G_{k+1} remains empty (i.e. all new pair have already been encountered) then the algorithm terminates and $\mathbf{I} \leq_F \mathbf{S}$ holds.

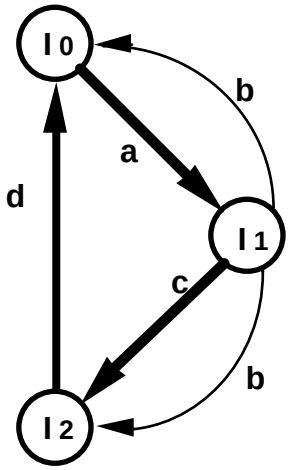

Figure 3: Valid implementation I-1

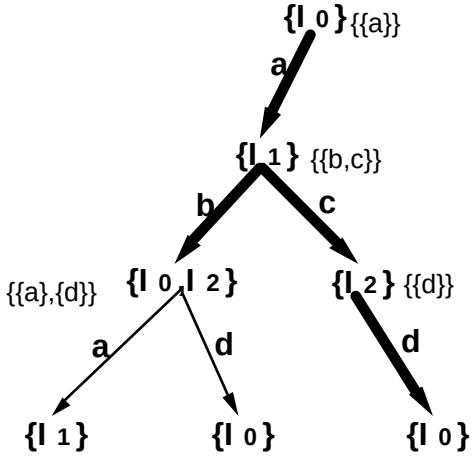

Figure 4: Testing tree for implementation I-1

As an example, Figure 5 shows the application of this algorithm to the checking of $I\text{-}1 \leq_F S$. We start with the pair $G_0 = (\{S0\}, \{I0\})$. In step 1, since $\{I0\} = a \Rightarrow \{I1\}$, we derive the next pair $(\{S1, S2\}, \{I0\})$. In step 2, since $\{I1\} = b \Rightarrow \{I0, I2\}$ and $\{I1\} = c \Rightarrow \{I2\}$, we derive the next pairs $(\{S0, S3\}, \{I0, I2\})$ and $(\{S3\}, \{I2\})$. We continue this procedure until we find $G_4 = \varnothing$. A second example showing a faulty implementation is given in [Fuji 91].

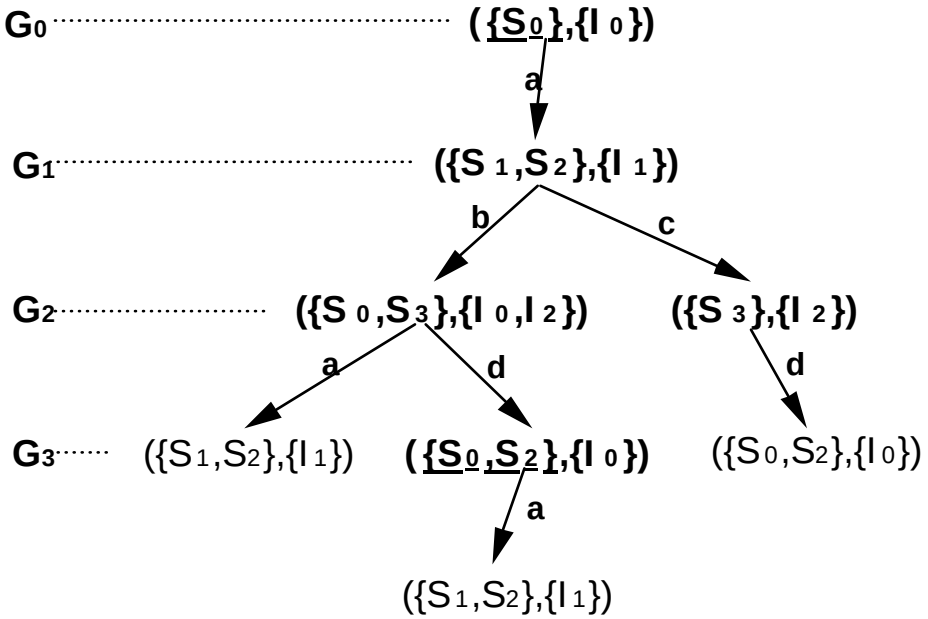

Figure 5: Test procedure for implementation I-1

4. Testing for failure preorder

In this section we consider the implementation under test as a black box. For a given specification, we want to develop a test suite which verifies whether the implementation satisfies the failure preorder relation in respect to the specification. In order to use Algorithm 3.A above, we need some means for identifying the current multi-state of the implementation and a method for verifying whether the current multi-state $\mathbf{I}_X$ satisfies $\mathbf{I}_X < \mathbf{S}_X$ for the corresponding multi-state $\mathbf{S}_X$ of the specification.

For the first problem, we propose the use of a distinguishing set Z, which is a generalization of the distinguishing set used for deterministic machines. In fact, it can be shown [Fuji 91] that the concept of characterization set **W** as introduced in the context of deterministic FSM's [Chow 78] and the concept of minimality, can be adapted to the context of a set of multi-states, corresponding to a non-deterministic FSM. A distinguishing set $Z = (\{\varepsilon\} \cup L \cup L^2 \cup ... \cup L^{m-k}).V$ can be defined which can distinguish m multi-states if the set of test processes V can distinguish k different multi-states in the implementation.

For the second problem, we use so-called mandatory test processes, as explained in Section 4.1, which are inspired from the COOP test method [Weze 89]. In Section 4.2, we describe our test selection method and give an algorithm for selecting a test suite. An example is considered in Section 4.3.

4.1. Mandatory test processes

For checking the local action preorder relation, we introduce a set of so-called mandatory test processes, similar to those defined in [Weze 89], which can be used for this purpose.

Definition 4.E (Compulsory set of a multi-state)
Let **S** be a multi-state. Then the compulsory set for **S**, written CO(**S**), is defined as
$CO(\mathbf{S}) = \{out(S_i) \mid S_i \in \mathbf{S}\}$.

Definition 4.F (mandatory processes)
Let **S** be a multi-state. The set of mandatory processes of **S**, written M(**S**), is the following set of processes: $\mathbf{M}(\mathbf{S}) = \mathbf{M_T}(\mathbf{S}) \cup \{\mathbf{M_F}(\mathbf{S})\}$ where:

$$\mathbf{M_T}(\mathbf{S}) = \{\mathbf{CHOICE}(\mathbf{V}) \mid \mathbf{V} \in \mathbf{orth}(\mathbf{CO}(\mathbf{S}))\}$$
$$\mathbf{M_F}(\mathbf{S}) = \mathbf{CHOICE}(\mathbf{L} - \mathbf{out}(\mathbf{S}))$$

where CHOICE(V) is a test process which has a transition out of its initial state for each action $a \in V$ which all lead to a deadlock state with no further transitions. The function orth(C) provides as result the set of all sets that can be formed by choosing precisely one member from each element of the set C.

The following theorem gives us the means for checking the local action preorder relation by testing. It states that the relation **I** < **S** can be tested by using all mandatory processes M(**S**).

Theorem 4.C (testing for local action preorder)
Let **S** and **I** be two multi-states.
I < **S** iff **I** passes M(**S**)

4.2 Algorithm for test suite development

In this section, we describe our test procedure by using the means introduced in the previous sections. We adopt a two-phase approach where Phase 1 derives a suitable set Z for multi-state identification in I, Phase 2 checks the existence of a mapping f. In the following, we describe the procedure of each phase in more details.

The purpose of Phase 1 is to determine a set of test processes, called Z, which would be able to distinguish m different multi-sets in the implementation (if there are that many). In order to find such a set, we begin with a set of test processes, in the following called U, which we believe to identify many multi-states. We also need a set of test sequences T

which leads the implementation from the initial state I0 to the different multi-states. These sequences are called transfer sequences. Since we do not know the structure of the implementation, we select these sets U and T based on the known specification S.

We propose to use for U the characterization set W of S, and for T the set of transfer sequences which lead the specification S from its initial state to its different multi-sets. This set T can be easily obtained from S by deriving its testing tree, as shown in Figure 2. Note that any other choice of U and T would also lead to correct test results, however, the resulting set Z may be less optimal.

The testing Phase 1 consists of applying each transfer sequence in T followed with each test process in U. For each pair, the transfer sequence is first applied, and if the test run is successful, the test process from U will be applied subsequently. Because of the possible non-determinism, each of these pairs must be tested repeatedly, as explained at the end of Section 3.1. These tests will give rise to the identification of a certain number of multi-states in the implementation. Let us assume that their number is k. Then we choose the distinguishing set $Z = (\{\varepsilon\} \cup L \cup L^2 \cup ... \cup L^{m-k}) . U$ which is known to distinguish up to m different multi-states in the implementation (see [Fuji 91]).

The testing procedure in Phase 2 follows largely Algorithm 3.A, however, instead of relying on the knowledge about the structure of the implementation, the testing procedure explores the multi-states and the transitions of the implementation through testing. Instead of a pair (**S**, **I**), we consider now a pair (**S**, O_Z(I0 after σ)), where the notation O_X(I) means the list of observation sets obtained for the list of test processes in X, that is, for a set of test processes X = {t1, t2,., tn} we have $O_X(I)$ is defined as: $O_X(I) = (O(I \| t1), O(I \| t2), ..., O(I \| tn))$; and σ is a sequence leading from the initial state I0 to a multi-state **I** for which we have observed O_Z. (**I**). The experimentally obtained list of observation sets O_Z(I0 after σ) identifies the multi-set reached after the application of σ from the initial state of the implementation.

The procedure of Phase 2 proceeds in several steps. During the first step, we apply Z to the initial state I0 and observe O_Z(I0). We form the set of pairs G0 consisting solely of the pair ({S0}, O_Z(I0)). In the (k+1)-th step, we do the following for each of the pairs $(\mathbf{S}, O_Z(I0 \text{ after } \sigma)) \in G_k$ that are not included in $G_0 \cup G_1 \cup ... \cup G_{k-1}$.

For each action **a** in out(**S**), that is, each action that may follow according to the specification, we determine the next multi-state **S'** (according to the specification) to be such that $\{S0\} = \sigma . a \Rightarrow \mathbf{S}'$. We test that $(I0 \text{ after } \sigma . a)$ **passes** M(**S'**) by applying each test process of M(**S'**) after the sequence σ.a. If $(I0 \text{ after } \sigma . a)$ fails with a certain test process of M(**S'**), then we conclude that **not** $(I \leq_F S)$. We then apply each test process of Z to (I0 after σ.a). Through these test we observe O_Z(I0 after σ.a), and the pair (**S'**, O_Z(I0 after σ.a)) is added to the set of pairs Gk+1.

If the set G_{k+1} is included in $G_{k+1} \subseteq G_0 \cup G_1 \cup ... \cup G_k$ then the testing procedure terminates successfully, otherwise another step must be executed.

It is shown in [Fuji 91] that the above algorithm terminates after a finite number of steps and that it terminates successfully if and only if $I \leq_F S$. It is also noted that the number of steps in the algorithm may be reduced by considering the fact that $\mathbf{S}_i \subseteq \mathbf{S}_j$ implies $\mathbf{S}_i \leq_F \mathbf{S}_j$ (see [Fuji 91]).

4.3 An example

In order to demonstrate the above algorithm, we use again the specification S of Figure 1. We consider the valid implementation I-3 shown in Figure 6. Figure 7 shows the testing tree of I-3. The implementation I-3 has 5 multi-states which are {I0}, {I1}, {I2,I4}, {I3}, and {I4}. The number of multi-states in the specification is 5; we choose m = 5, that is, we assume that the number of multi-states of the implementation is less or equal to 5.

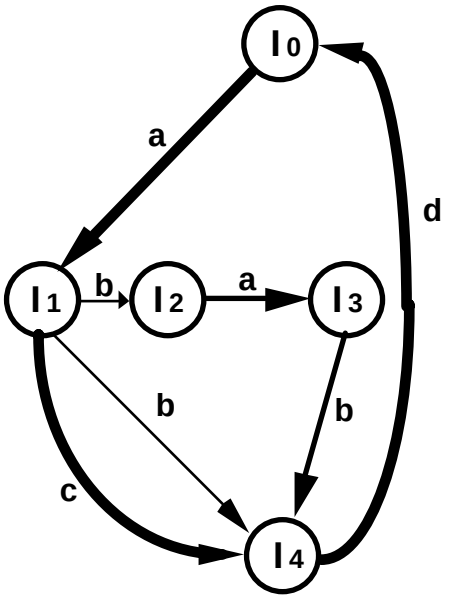

Figure 6. Valid implementation I-3

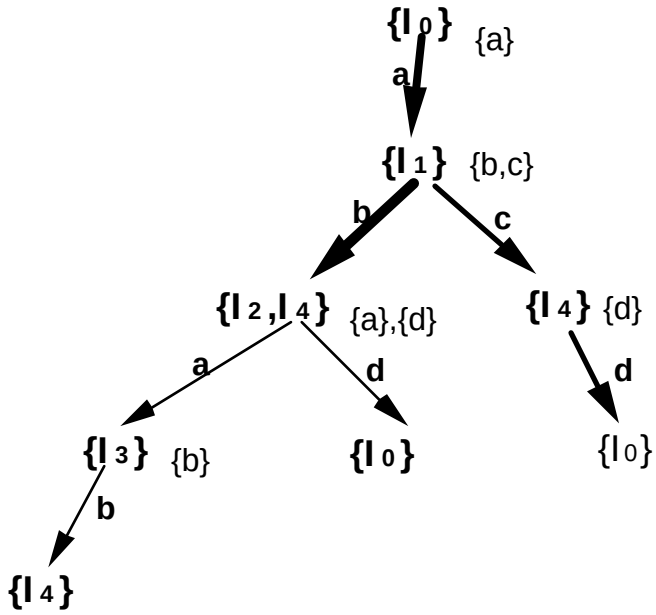

Figure 7. Testing tree for implementation I-3

We use the characterization set W={a; **stop**, b; **stop**} of S as U, and the transfer sequence set T={r, r.a, r.a.b, r.a.c, r.a.b.d}. The results that will be observed in Phases 1 and 2 are given in Tables 4.A and 4.B, respectively.

Table 4.A: Observation set for I-3 (Phase 1)

transfer sequence	**S**	CO(**S**)	a	b	**I**	CO(**I**)	a	b
r.	{S0}	{a}	{T}	{F}	{I0}	{a}	{T}	{F}
r.a	{S1,S2}	{b},{b,c}	{F}	{T}	{I1}	{b,c}	{F}	{T}
r.a.b	{S0,S3}	{a},{d}	{T,F}	{F}	{I2,I4}	{a},{d}	{T,F}	{F}
r.a.c	{S3}	{d}	{F}	{F}	{I4}	{d}	{F}	{F}
r.a.b.d	{S0,S2}	{a},{b,c}	{T,F}	{T,F}	{I3}	{b}	{F}	{T}

Table 4.B: Observation set for I-3 (Phase 2)

σ	CO(S)	CO(I)	a	b	a.a	a.b	b.a	b.b	c.a	c.b	d.a	d.b	Obs
r	{a}	{a}	T	F	F	T	F	F	F	F	F	F	O_A
r.a	{b},{b,c}	{b,c}	F	T	F	F	T,F	F	F	F	F	F	O_B
r.a.b	{a},{d}	{a},{d}	T,F	F	F	T,F	F	F	F	F	T,F	F	O_C
r.a.c	{d}	{d}	F	F	F	F	F	F	F	F	T	F	O_D
r.a.b.a	{b},{b,c}	{b}	F	T	F	F	F	F	F	F	F	F	O_E
r.a.b.d	{a},{b,c}	{a}	T	F	F	T	F	F	F	F	F	F	O_A
r.a.c.d	{a},{b,c}	{a}	T	F	F	T	F	F	F	F	F	F	O_A
r.a.b.a.b	{a},{d}	{d}	F	F	F	F	F	F	F	F	T	F	O_D

The results in the right part of Table 4.A show that during Phase 1 only 4 multi-states are identified, since {I1} and {I3} produce the same results. Therefore we use $Z = (\{\varepsilon\} \cup L)$. $U = \{a, b, a.a, a.b, b.a, b.b, c.a, c.b, d.a, d.b\}$.

Figure 8 shows how our test procedure will work during Phase 2 with the implementation I-3. In step 3, we get the pair $(\{S0, S2\}, O_A)$. Since we have $(\{S0\}, O_A)$ in step 0, we can stop further development considering that $S_i \subseteq S_j$ implies $S_i \leq_F S_j$ (see [Fuji 91]). For the same reason, we do not have to develop the pair $(\{S0, S3\}, O_D)$ of G_4. Finally we have $G_4 \subseteq G_0 \cup G_1 \cup ... \cup G_3$ and I passes all mandatory test processes. Therefore we can conclude that $(I\text{-}3) \leq_F S$.

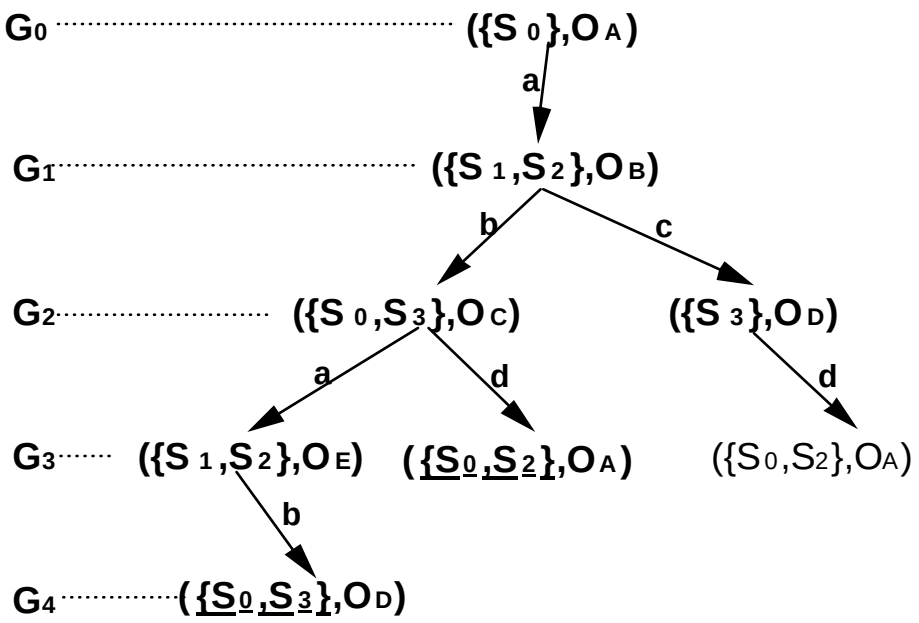

Figure 8. Test procedure for implementation I-3

5. Concluding discussion

In this paper, we have developed a method and algorithm for the development of test cases for testing an implementation in respect to a specification, which is given in the form of a state transition machine. It is assumed that both, the specification and the implementation, may have non-deterministic behavior. This makes the testing and the test case development much more difficult than in the case where specifications and implementations are deterministic. We assume that the objective of the testing is to check that the implementation satisfies a certain "implementation relation" in respect to the specification. The failure preorder relations are considered for this purpose. The underlying formalism of our testing approach is based on the observation that for non-deterministic machines, the specification (or implementation), after a given sequence of observed actions, may be in a set of different states, which we call "multi-state". For non-deterministic machines, these multi-states take the same role as states in deterministic machines.

Under the assumption that the number of multi-states of the implementation is bounded, and that a reset function is correctly implemented, our test suite development algorithm guarantees that any fault in the implementation is detectable by the derived set of test processes. This set of test processes is finite, and each of the test processes has only a finite behavior. Therefore the required testing effort is bounded.

The testing approach described here is based on many ideas [Brin 87, Brin 88, Brin 89,Tret 89, Weze 89] that have been introduced in relation with the LOTOS specification language [Loto 89] which allows for non-deterministic specifications. However, we take a different approach by insisting that a test suite should consist of a finite number of test cases, each of which having only a finite behavior. With this approach, it is impossible to have a guarantee of fault detection unless certain assumptions are made about the complexity of the tested implementation. We follow here the approach previously taken for the testing of deterministic finite state machines [Chow 78, Gone 70] where it is assumed that the number of states of the implementation is limited. While the canonical testers introduced in [Brin 88] provide for the possibility of detecting all faults, in practice they are not so useful since

they have infinite behaviors which cannot be completely explored during a finite testing session. To our knowledge, the here described algorithm for test suite development is the first of its kind by combining the guarantee of error detection (under certain assumptions) with a finite test suite in the context of non-deterministic specifications and implementations.

The following issues require further study: (1) optimization of the length of the derived test suite, (2) generalization for specifications including spontaneous t-transitions, and (3) adaptation of this testing approach to other implementation relations. We note that the testing equivalence relation is treated in [Fuji 91], and that specifications including τ-transitions can be generally transformed into equivalent specifications without such transitions [Luo 91b].

Acknowledgments

The authors would like to thank the members of the "Teleinformatique" research group at the Université de Montréal for the helpful discussion and suggestions. This work was supported by the IDACOM-NSERC-CWARC Industrial Research Chair on Communication Protocols at the Université de Montréal and the NTT Corporation, Tokyo.

References

[Boch 89f] G.v. Bochmann, "Inheritance for objects with concurrency", publication #687 of D.I.R.O., Montreal University, 1989.

[Boch 89m] G.v. Bochmann, R. Dssouli and J. R. Zhao, "Trace analysis for conformance and arbitration testing", IEEE Trans. on S. E., pp. 1347-1356, Nov. 1989.

[Bolo 87] T. Bolognesi, E. Brinksma, "Introduction to the ISO Specification Language LOTOS", Computer Networks and ISDN Systems 14, pp. 25-59, 1987.

[Brin 87] Ed Brinksma, G. Scollo, C. Steenbergen, "Lotos Specifications, Their Implementations and Their Tests", in B. Sarikaya and G. V. Bochmann (eds), Protocol Specification, Testing and Verification, VI, pp. 349-360, (North-holland, Amsterdam, 1987).

[Brin 88] Ed Brinksma, "A Theory for the Derivation of Tests", in S. Aggarwal (eds), Protocol Specification, Testing and Verification, VIII, (North-holland, Amsterdam, 1988).

[Brin 89] Ed Brinksma, R. Alderden, R. Langerak, "A Formal Approach to Conformance Testing", in the 2-nd International Workshop on Protocol Test Systems, Berlin, Germany, pp. 311-325, Oct. 3-6, 1989.

[Chow 78] T.S. Chow, "Testing Design Modelled by Finite-State Machines", IEEE Trans. S.E. 4, 3, 1978.

[Deni 84] R.De Nicola, M.C.B. Hennessy, *Testing Equivalences for Processes*, Th. Comp. Sci. 34, 1984.

[Fuji 90] S. Fujiwara, G.v. Bochmann, F. Khendek, M. Amalou, A. Ghedamsi, "Test Selection Based on Finite State Models", Publication #716 of D.I.R.O, Montreal University, Feb. 1990.

[Fuji 91] S. Fujiwara and G. v. Bochmann, *Testing non-deterministic finite state machines,* Tech. Report #758, submitted for publication.

[Gone 70] G. Gonenc, "A method for the design of fault detection experiments", IEEE Trans. Computer, Vol. C-19, pp. 551-558, June 1970.

[Henn 85] M. Hennessy, "Acceptance Trees", Journal ACM, 32, No. 4 (Oct. 1985), pp. 896-928.

[Hoar 85] C.A.R. Hoare, "Communicating Sequential Processes", (Prentice-Hall, 1985).

[Lang 89] R. Langerak, "A Testing theory for LOTOS using deadlock detection", in E. Brinksma, G. Scollo, C. A. Vissers (eds), Protocol Specification, Testing and Verification, IX, (North-holland, Amsterdam, 1989).

[Loto 89] ISO, Information Processing Systems, Open Systems Interconnection, LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour, IS 8807, First Edition, January 1989.

[Luo 91b] G. Luo, G. v. Bochmann, C. Wu and A. Das, *Failure-equivalent transformation to avoid internal actions*, in preparation.

[Miln 80] R. Milner, "A Calculus of Communicating Systems", LNCS 92, (Springer-Verlag, 1980).

[Nait 81] S. Naito and M. Tsunoyama, *Fault detection for sequenctial machines by transition tours*, Proc. FTCS, 1981, pp. 238-243.

[Rayn 87] D. Rayner, "Standardizing Conformance Testing for OSI", Computer Networks and ISDN Systems, Vol. 14, No. 1, pp. 79-98, 1987.

[Sabn 88] K. Sabnani and A. Dahbura, *A protocol test generation procedure*, Computer Networks, Vol.

15, 1988, pp.285-297.
[SaDa 88] K.K. Sabnani and A.T. Dahbura, "A protocol Testing Procedure", Computer Networks and ISDN Systems, Vol. 15, No. 4, pp. 285-297, 1988.
[Sari 89] B. Sarikaya, "Conformance Testing: Architecture and Test Sequences", Computer Networks and ISDN Systems 17, pp. 111-126, 1989.
[Tret 89] J. Tretmans, "Test Case Derivation from LOTOS Specifications", in the 2-nd International Conference FORTE '89, Vancouver, Canada, Dec. 5-8, 1989.
[Weze 89] C. D. Wezeman, "The CO-OP method for Compositional Derivation of Conformance Testers", in E. Brinksma, G. Scollo, C. A. Vissers (eds), Protocol Specification, Testing and Verification, IX, (North-holland, Amsterdam, 1989).