Invited Paper, Proceedings of the IFIP Sixth International Workshop on Protocol Test Systems, Pau, France, 1993.

Conformance relations and test derivation

Alexandre Petrenko*, Gregor v. Bochmann, and Rachida Dssouli

Université de Montréal, Canada

Abstract

It seems that finite state machines (FSM's) and (finite) labeled transition systems (LTS's) are competing descriptive models for system specifications in various areas, including communication protocols. Much work on the derivation of tests from a given system specification has been done separately for these two models. In this paper, we attempt to make a comparative study of existing approaches of test derivation for these two similar, but still distinct models, and we show that borrowing ideas from one type of model for the realm of the other might be useful.

In order to formally compare the two models of FSM's and LTS's, we use a formal framework for conformance testing based on conformance relations and a set of possible observations. We show how the conformance relations can be combined with explicit fault models for the tested implementations. It is shown that, within the FSM framework, a hierarchy of conformance relations used for the comparison between the specification and the tested implementation, reflects the hierarchy of classes of models (completely specified and deterministic, deterministic, and non-deterministic). In particular, the reduction relation between non-deterministic FSM's can be used to derive other useful relations for the other classes of models. A review of test derivation methods for FSM models shows that guaranteed fault coverage within a predefined fault domain may be obtained.

The second part of the paper shows that the methods for test suite derivation from FSM specifications can be applied for testing various conformance relations in respect to LTS specifications. The idea is to define, for a given LTS specification and conformance relation, a corresponding FSM specification such that the application of the FSM test suite (based on trace conformance) is equivalent to the verification of the given conformance relation in respect to the LTS specification. The feasibility of this approach is presented for the so-called failure trace equivalence between the LTS specification and the implementation. A corresponding canonical FSM tester can also be defined.

Keyword Codes: C.2.2; D.2; D.2.5
Keywords: Network Protocols; Software Engineering; Testing and Debugging

1. INTRODUCTION

* on leave from the Institute of Electronics and Computer Science, Riga, Latvia.

It seems that finite state machines (FSM's) and (finite) labeled transitions systems (LTS's) are competing descriptive models for system specifications in various areas, including communication protocols. Much work on the derivation of tests from a given system specification has been done separately for these two models. Basic concepts in the area of the FSM models have been defined in early work, such as [Moor56], [Gill62], [Henn64], [Vasi73], [Chow78]. Much work on the derivation of tests from FSM specifications has been done more recently in the context of protocol engineering where conformance testing of implementations is a major issue [Fuji91], [Petr91], [Petr92], [Vuon89], [Yann91], [Yao93]. While the FSM model usually does not cover the whole aspects of a protocol specification, it often captures the essential features of the protocol behavior related to the temporal ordering of interactions and is used in ESTELLE and SDL languages. Most work on test derivation has been limited to completely specified, deterministic specifications [Ural91], [Sidh89].

Labeled transitions systems (LTS's) are in some sense a more general specification model than FSM's, since interactions of a specified subsystem with its environment are usually considered as rendezvous interactions making no distinction between input and output. LTS's are usually not completely specified; the non-specified interactions are not possible (blocking). The LOTOS language has been developed for the specification of communication protocols and distributed systems and uses the semantics of LTS's. In recent years, much work on the derivation of conformance tests from a given LOTOS, or corresponding LTS specification, has been done in the protocol engineering community [Alde90], [Brin87], [Burg92], [Drir93], [FuBo91], [Lang89], [Ledu91], [Pitt90], [Weze90]. Most of this work has addressed the difficult problem of dealing with non-deterministic specifications. Also a formalization of the conformance testing framework and the definition of different conformance relations has first been done within this community [Brin89], [Tret91], [Phal92].

It seems that the theories of test derivation from FSM and LTS specifications have been developed almost independently. We also note that research for appropriate conformance relations and test derivation methods have concentrated, until recently, almost exclusively on deterministic systems in the realm of FSM's, and in the realm of LTS's on non-deterministic systems. In this paper, we attempt to make a comparative study of existing approaches of test derivation for these two similar, but still distinct models, and to show how borrowing ideas from one type of model for the realm of the other might be useful.

The paper is organized as follows. In Section 2 we discuss several interpretations of different types of FSM models useful for conformance testing, including partially specified and non-deterministic FSM's.

In Section 3 we introduce a formal, conceptual framework for conformance testing which is based on the approach of [Brin89] enriched with the concepts of a fault model. The notion of a complete test suite for a given (FSM or LTS) specification in respect to a given conformance relation and a fault model is introduced, based on this framework. We analyze also the existing fault models which lead to finite executable tests.

Section 4 gives a review of the existing FSM-based methods that can generate complete test suites, i.e., can provide full fault coverage within a predefined finite fault domain. It is shown that, within the FSM framework, a hierarchy of conformance relations used for the comparison between the specification and the tested implementation, reflects the hierarchy of classes of models (completely specified and partially specified, deterministic, and non-

deterministic machines). In particular, the reduction relation between non-deterministic FSM's can be used to derive other useful relations for the other classes of the FSM model.

In Section 5 we consider the hierarchy of existing semantics and corresponding relations for LTS's that are considered as candidates for conformance relations. We raise the problem of relating conformance relations for the FSM and LTS models and show for the example of the failure trace semantics, that the FSM-based test derivation methodology can be applied to check the LTS conformance relations. We also show how a "canonical" tester for a given FSM can be constructed; on the same line, based on this tester, a canonical tester of a corresponding LTS for a chosen conformance relation can be obtained in a transparent way. Section 6 contains our conclusions.

2. FINITE STATE MACHINES

2.1. Definitions and categorization of FSM's

A = (S,X,Y,h) is called a *finite state machine* (FSM) if S, X and Y are finite non-empty sets of states, inputs and outputs, respectively; and h is a function h: $D_A \varnothing P(SxY)$, where D_A is a subset of SxX and P(SxY) is the set of all non-empty subsets of SxY.

This model of FSM is a slightly more general that the one given, for example, in [Stark72] and [Kain72] in that it can model a partially specified behavior.

The function h, called the *behavior* function, defines transitions between states and it is specified over the *specification domain* D_A. If D_A=SxX then the FSM A is called completely specified (or a *complete* FSM, for short), otherwise it is partially specified or a *partial* FSM.

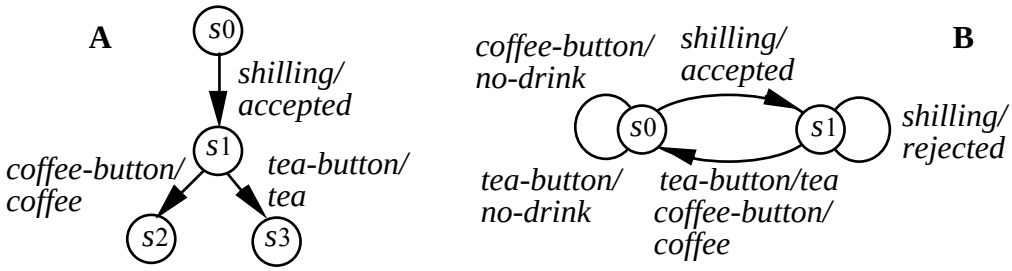

Figure 1: Two FSM's A and B for a vending machine

Figure 1 shows the examples of FSM's which model two slightly different versions of a vending machine, supplying coffee and tea. Inputs are *shilling, coffee-button, tea-button;* and outputs are *accepted, rejected, coffee, tea, no-drink.* The FSM A is partial, since its behavior is not specified for all pairs of (s,x), and the FSM B is complete.

The FSM's evolve relative to some universal discrete time, i.e., the behavior of an FSM A = (S, X, Y, h) is considered on a discrete time scale in a countable infinite number of steps t = 1,2,3, ..., according to the following rule: if s_t is the state of A in step t, then in step t+1 A enters the state s_{t+1} if the FSM chooses to execute the transition $(s_t\text{-}x_t/y_t\text{->}s_{t+1})$, $s_{t+1} \in h^1(s_t,x_t)$, $y_t \in h^2(s_t\, x_t)$, where h^1 is the first and h^2 is the second projection of h, i.e.,

$$h^1(s,x) = \{ s' \mid \exists\, y \in Y\ [(s',y) \in h(s,x)] \},\ h^2(s,x) = \{ y \mid \exists\, s' \in S\ [(s',y) \in h(s,x)] \}.$$

The function h^1 is called a *next-state* or *transition* function and will be also denoted δ; the function h^2 is an *output* function of an FSM and will be also denoted λ. An FSM is *trivial* or

combinatorial if it has one state only, below we consider mainly FSM's with more than one state. An FSM becomes an *accepting machine* or automaton if we neglect its outputs.

An FSM A is called *weakly initialized* if a non-empty subset $S_0 \subseteq S$ is fixed as the set of possible states in step 1 (*initial* states). If $S_0=\{s_0\}$, then A is an *initialized* FSM with the initial state s_0.

We define for a state $s \in S$ an *acceptable* sequence of events from X as a sequence $\beta = x_1x_2...x_k$ such that $(s_i,x_i) \in D_A$ and $s_{i+1} \in h^1(s_i,x_i)$ for all i=1,2,...,k with $s=s_1$.

The set of all such sequences for state $s_i \in S$ is denoted as X_i^*. The set of acceptable sequences X_A^* for an initialized FSM is the one for its initial state. For example, the initialized FSM A in Figure 1 has three acceptable sequences: *shilling*, *shilling.coffee-button* and *shilling.tea-button,* where "." stands for the concatenation.

We extend the behavior function to the set X_A^* of all acceptable words containing the empty word ε, i.e., h: $S \infty X_A^* \cup P(S \infty Y^*)$. Assume $h(s,\varepsilon) = (s,\varepsilon)$ for all $s \in S$, and suppose that $h(s,\beta)$ is already specified. Then

$$h(s,\beta x) = \{ (s',\gamma y) \mid \exists\, s'' \in S\ [(s'',\gamma) \in h(s,\beta) \ \&\ (s',y) \in h(s'',x)] \}.$$

Given a weakly initialized FSM $A = (S, X, Y, h, S_0)$, A is said to be *initially connected* if $\forall\, s \in S\ \exists\, s_0 \in S_0\ \exists\, \alpha \in X_A^*\ \ (s \in h^1(s_0,\alpha))$.

A is *strongly connected* if $\forall\, s_i,s \in S\ \exists\, \alpha \in X_i^*\ \ (s \in h^1(s_i,\alpha))$.

An FSM $A = (S,X,Y,h,s_0)$ is *deterministic* (DFSM), if $|h(s, x)| =1$ for all $(s,x) \in D_A$; otherwise it is *non-deterministic* (NFSM). An FSM $A = (S,X,Y,h,s_0)$ is *observable* (ONFSM), if $|\{s' \mid (s',y) \in h(s,x)\}|=1$ for all $(s,x) \in D_A$ and all $y \in h^2(s,x)$. This means, in observable machines, a state and an I/O sequence can uniquely determine at most one next state [Star72]. Non-deterministic FSM's can be transformed into observable forms, however, at the price of an exponential growth in the number of states in the worst case [Star72], [Hopc79].

We refer to the model given above as the Mealy model of FSM, the definition of a Moore FSM can be found, e.g., in [Kain72].

2.2. Interpretations of an FSM for conformance testing

2.2.1. Black box interface

In the context of conformance testing, the FSM model is applied to represent the behavior of a system under test considered as a black box interacting with its environment, e.g., tester or observer. We may assume (similar as in [Glab90]) the existence of an interface to the black box controlled by the FSM, as shown in Figure 2.

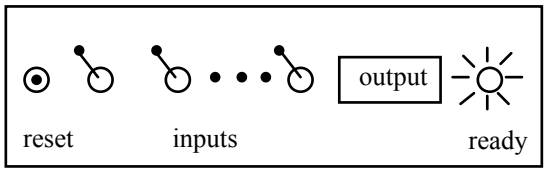

Figure 2: The black box interface of an FSM

The interface has as many switches as there are inputs in the input set X, a reset button, a display, and a "ready" lamp. At most one input may be switched on at any given time. The FSM reads the input, produces an output whose name is shown on the display and executes a transition (not visible at the interface), after that it lights the lamp signaling its readiness to accept the next input. The reset button is used to bring the FSM back to its initial state. It is especially needed for experiments involving an FSM that is not strongly connected. In the initial state, when all inputs are switched off, the "ready" lamp is lit, and display is empty, showing that the system idles.

The fundamental difference between inputs and outputs reflects the cause/effect dependency between events in the modeled system. When an input event happens it is the tester which takes the initiative; in reverse, output events are driven by the FSM. The FSM model implies that each transition of a modeled system is executed in an uninterruptable manner, in particular, the system does not accept a next input until it has completed its reaction to the previous input event. This assumption is needed to maintain cause/effect dependency between events in the modeled system and it is called "the fundamental mode of operation" in the classical textbooks on sequential switching circuits.

An alternative interpretation of input events can be obtained by allowing the tester to use simultaneously several switches. An abstract input for the FSM is related in this case to a combination of elementary inputs determined by the position of switches, i.e., the set X of input events is the set P(E) of all non-empty subsets of another set E of elementary events corresponding to the switches on the interface.

Similar to inputs, an abstract output event can actually represent a sequence of elementary output events produced during a single transition in response to an input in an uninterruptable manner. The "ready" lamp signals the end of such a sequence.

In certain applications of the FSM model, including protocols, the notion of a *null* output is useful. This particular output is assumed to be observed if no name of other outputs has been shown on the display by the FSM. Performing an experiment on a physical system, the tester measures a time elapsed after it has offered an input and if nothing happens during an interval which exceeds an internal delay of the system, it concludes that the null output is produced.

2.2.2. Partial specifications

So far we have assumed by default that the behavior of the FSM is completely specified. The semantics of partial machines needs, however, further clarification. A specific feature of a partial FSM is that it has "undefined" transitions called also "non-core" or "don't care". In the LTS realm, there is no notion of "undefined", since "blocking by default" [Boch93] is assumed for this model. For input/output machines, there are different conventions that may be used to give a formal meaning to "undefined":

1) "implicitly defined" transitions [Boch93];
2) "undefined by default";
3) "forbidden" transitions [Petr91].

In the sequel, we shortly discuss these conventions.

"Implicitly defined" transitions. A partial FSM represents a completely specified FSM by adopting some *"completeness assumption"*. Such an assumption is based on the fact that all implementations can be represented by complete machines which never refuse any input. It states that all "don't care" transitions in the partial machine are substituted by looping transitions with the null outputs (convention 1a), see, e.g., Figure 1, or with transitions to an error (terminal) state with an "error" output (convention 1b). In SDL specifications, for

instance, unexpected or inopportune events are ignored, so the convention 1a is applied. A convention used during the process of implementation is assumed to be known for testing purposes.

As we mentioned above, the LTS model assumes deadlocking for undefined situations. The requirement of neglecting an inopportune event can also be represented explicitly in an LTS by a looping transition. This implies that the behavior of such an LTS becomes infinite and most existing methods for test derivation cannot be applied. Checking non-core transitions in both models is often called a *robustness* testing; however, the difference is that the conformance test derivation techniques based on FSM's tend to consider it as a part of the general process of conformance testing, whereas in LTS's it is usually excluded from that process.

"Undefined by default" transitions. This convention means that a partial FSM A = (S,X,Y,h,s_0) is interpreted as a set of complete FSM's taking into account that the result of a "don't care" transition might be any of the elements of the set $P(S\infty Y)$. This interpretation is suitable for defining relations between different specifications of the same system (some are more defined than the others) and for conformance testing of the given implementation of a partial FSM. In the last case, the set $P(S\infty Y)$ represents all possible options left to an implementor of the partial machine and a conforming implementation is, in fact, a complete FSM that implements one of these options [Petr91]. This interpretation is formalized in the conformance relation between machines called quasi-equivalence, which is given in Section 4.

"Forbidden" transitions. In some cases, the behavior of a system is not completely specified because its environment will never execute certain input sequences. Invalid entries in state tables used in the ISO standards for protocols exemplify such a case. The set of acceptable input sequences X_A^* of an FSM A represents all "admissible" sequences, the remaining ones are "forbidden", they define "forbidden" transitions in the given partial FSM. This convention is similar to the one of "undefined by default" transitions, the difference is that in this case, a tester cannot submit certain inputs to certain states of the given partial FSM. We illustrate this convention by the following example.

Consider a vending machine that cannot be directly accessed by a tester, there is a butler who operates the machine. We wish to derive an abstract test suite for the vending machine and finally execute them through the butler. It is natural to assume that the test interface (the butler) is error-free. Suppose such a butler that takes an order for tea or coffee only having got shilling from the tester by neglecting other requests. If we assume that the reference behavior of the machine is the one of FSM A (Figure 1) then we do not have any problem in testing the IUT with the given test interface. Suppose now that the specification FSM B (Figure 1) is the reference machine. In this case, any attempt to execute non-core looping transitions with outputs *no-drink, rejected* is abandoned by the test interface, and the actual behavior of the vending machine remains unknown.

Yet another illustrations are a lower service provider that cannot deliver any PDU to certain states of a tested protocol entity and an upper protocol entity that cannot deliver an unexpected service primitive to any state of the IUT. In all these situations we have to treat "don't care" transitions as "forbidden" and to avoid them in test derivation. The essential distinction between the "undefined by default" and "forbidden" conventions is an "undefined" transition can be tested only in the first case.

The conclusion is that to derive an abstract test suite from even a completed specification of an embedded IUT we might have to come back to a partially specified machine. Therefore, even though some specification languages based on the FSM model impose complete specifications by a build-in mechanism of implicitly defined transitions, there is a need for deriving conformance tests directly from partial machines.

2.2.3. Non-deterministic specifications

We note that a non-deterministic FSM model may be used to represent different situations, such as the following:
- a protocol entity with inherent non-determinism;
- a set of deterministic protocol entities considered as options of a given protocol [Petr91];
- a deterministic IUT embedded in a given SUT in such a way that a tester cannot directly observe and control it (see [Petr93] for more detail).

The variety of interpretations of non-deterministic machines for conformance testing does not imply, however, a variety of conformance relations, as shown in Section 4.3.

Additional remarks on the conventions of "undefined" in the case of partial non-deterministic FSM's that are not observable, are needed. We note that not all the conventions of "undefined" are relevant for each machine in this class. Consider, as an example, the specification FSM given in Figure 3.

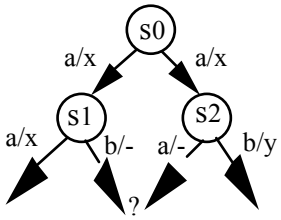

Figure 3: Partially specified non-deterministic behavior

Here, the behavior is left undefined in state s1 for input *b* and in state s2 for input *a*. It is a usual situation for LTS's: since "blocking by default" is the only convention for LTS's, the corresponding LTS non-deterministically refuses event *a* or *b* after the trace *a.x*. However, under the "undefined by default" convention, the requirements of the specification FSM are not consistent, viz. on the one hand, the specification says that the machine, after the input/output event *a/x,* is required to respond to *a* with *x* (state s1), and on the other hand, it allows an arbitrary (even non-deterministic) behavior after the same event in state s2. Under the "forbidden" convention we cannot determine after *a/x* which further behavior is actually testable. All conventions can be applied only for those partial NFSM's which possess "harmonized" traces and do not have such configurations, as explained in [Petr93], these non-deterministic FSM's can be transformed into observable forms.

3. A FRAMEWORK FOR CONFORMANCE TESTING

In this section we consider a conceptual framework for conformance testing which is based on the approach of [Brin89] enriched with the concepts of a fault model.

Let S be a specification written in a given formalism. In the context of conformance testing, an implementation under test I is considered as a black box, its behavior is assumed to be represented with the same formalism as the specification S. Let Υ be a process domain,

that is the set of all FSM's or LTS's, respectively, S,I□ϒ. The goal of testing is to verify if a certain conformance relation "**conf**" holds between I and S: I **conf** S, where **conf** ⁄ ϒxS. We write I not**conf** S to state that the implementation I is not in the relation "**conf**" with the specification S.

The given specification and the conformance relation partition the domain ϒ into the subsets of conforming implementations C(S)={I | I□ϒ, I **conf** S} and non-conforming implementations, called a *fault universe* M(S) for the given specification S, i.e., a set of all *mutants* of S: M(S)={I | I□ϒ, I not**conf** S}.

Since the structure of I is unknow, we have to check the conformance by black-box testing solely based on the observation of behaviors of S and I placed in a common environment. An *observation* of the behavior of I is a set of developments of its possible interactions at the interface with the environment. Generally speaking, these interactions can be of different kinds: observable actions, deadlocks, etc. In case of an FSM, input/output sequences are observed; and in case of an LTS, possible observation are defined by a certain semantics (see Section 5).

For non-deterministic implementations, a given test may give rise to different observations. It is therefore not enough to execute a test once, the same test should be repeatedly executed until all possible observations are obtained. If the non-determinism within the IUT can not be influenced from the testing environment, it may be difficult, in general, to determine after how many executions of the given test all possible observations have been obtained. In order to determine the verdict for a given test, it is therefore necessary to assume the so-called *complete testing assumption* [Luo93] which supposes that all possible observations have been obtained (see also the "all weather conditions" assumption for LTS's in [Miln80]).

To operationalize a conformance relation we define a testing framework as a triple (T, Σ, **exec**), where T is a set of all possible **tests**, defined in a suitable process domain and represented in a given formalism whose alphabets comprise alphabets of S and I (a test is an FSM if S is the FSM*, and it is an LTS if S is the LTS); Σ is a set of observations that may be obtained by the executions of a given test on a given process which are formalized by a mapping **exec**: TxϒP(Σ). In particular, O = **exec**(t, S) is a set of observations obtained by the test t from the specification S.

Let **ob-conf** be a relation between sets of observations, i.e., **ob-conf** ⁄ P(Σ)xP(Σ) such that

$$\forall\, t \in T \quad (\ \mathbf{exec}(t, I)\ \ \mathbf{ob\text{-}conf}\ \ \mathbf{exec}(t, S)\ ) \quad \text{iff} \quad I\ \mathbf{conf}\ S.$$

For each test t and a set of observation O a **verdict** v_t is defined as follows:

$$v_t(O) = \mathbf{pass} \ \text{ if } O\ \mathbf{ob\text{-}conf}\ \mathbf{exec}(t, S)$$
$$= \mathbf{fail} \ \text{ otherwise.}$$

Then one has for an arbitrary implementation I:

$$I\ \mathbf{conf}\ S \ \text{ iff } \ \forall\, t \in T \ \ v_t(\mathbf{exec}(t, I) = \mathbf{pass},$$

and

$$I\ \text{not}\mathbf{conf}\ S \ \text{ iff } \ \exists\ t \in T \ \ v_t(\mathbf{exec}(t, I) = \mathbf{fail}.$$

Fault models have been introduced to avoid exhaustive testing [Boch91]. A fault model characterizes a subset of mutants of the specification S in behavioral terms: distortions, degradations [Drir93] or in structural terms: output faults, transition faults, single and multiple [Boch91], faults defined for certain transitions [Petr92], superfluously accepted events in a certain state, "forgotten" transitions [Brin91] and etc.

---

* we discuss in detail this case in Section 5.4.

A *fault model* f for the domain Υ is a mapping f: $\Upsilon \varnothing P(\Upsilon)$. It defines how the given specification can transformed into an implementation to represent a fault that fits to the given fault model. The set f(S) may contain non-conforming as well as conforming implementations. The subset of (non-conforming) mutants F(S)=f(S)(M(S) is called a *fault domain* for the given specification S and fault model f. Fault models that determine finite fault domains lead to finite testing.

Given a specification S, a conformance relation, a testing framework (T, Σ, **exec**), and a fault model f, *a test suite for S complete w.r.t.* f is a set of tests $T \subset T$ such that $I \square F(S)$ implies $\exists t \square T$ (v_t (**exec**(t, I) = **fail**). Such a test suite provides full fault coverage in a sense that it is capable of finding out all mutants defined by the given fault model f and conformance relation. Clearly, the set T is complete for S in the fault universe M(S), it has, however, infinite number of tests which are not necessarily deterministic. When in the following we consider a test suite, we assume that all the tests that are included in the suite have a deterministic behavior. We say that a test suite is *executable* if it consists of a finite number of finite deterministic test behaviors.

To exemplify the above definition, consider the case of a completely specified deterministic FSM S. A complete test suite for S w.r.t. f and the relation **conf** has for every implementation FSM $I \square F(S)$ a test that can produce an output sequence in the FSM I different from the output sequence that is produced by the FSM A when this test is applied. The relation **conf** becomes the equivalence relation defined for completely specified deterministic FSM's, and the fault model f might be, for example, the output type of faults [Nait81]. In case of FSM's, a test may be often defined as an input sequence since the corresponding output sequences are easily obtained from the specification FSM.

The possibilities of constructing executable complete test suites for the given specification are limited as follows from an observation that exhaustive testing is impossible in practice and specifically, from the general result on identification of finite state machines given in [Gill62].

Proposition 3.1. [Gill62]: A finite state machine is not identifiable unless the entire input alphabet and the maximum number of states in its minimal form are known in advance.

Similarly, in the case of an LTS, we can never determine by black-box testing neither, for example, all traces nor all refusal sets of the implementation LTS unless we know in advance all the actions and the maximal number of states implemented in it.

As follows from this proposition, it is feasible to construct from the given specification S a test suite which guarantees the given conformance relation only in a *predefined finite* fault domain F(S).

In the realm of FSM's, there are the following approaches for obtaining a finite fault domain based on different fault models:

1) explicit enumeration of possible mutants (a mutation technique) [Poag64];
2) output faults only [Nait81];
3) grouped faults specified by a so-called fault function [Petr92];
4) limiting the number of states in implementations [Gill62].

The last approach is the most frequently used one, a fault domain is restricted in this case within the universe of all FSM's which have the input alphabet of the specification FSM, and the number of states not exceeding a certain integer. In other words, suppose we know that the number of states in all potential implementations of the given n-state specification FSM S is

m or fewer ($m \geq n$). Let Υ_m be a universe of all these implementations, $F(S) \wp \Upsilon_m$. An actual fault coverage of a complete test suite in Υ_m, is illustrated by Figure 4.

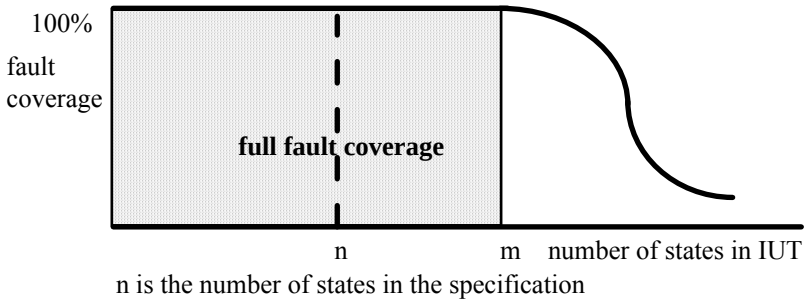

Figure 4: Fault coverage of a complete test suite vs. the number of states

Guessing the bound m is an intuitive process based on the knowledge of a specification, the class of implementations and their interior structure which have to be tested for conformance. The value of m can be also limited due to economic considerations. For instance, for the vending machine represented by the FSM B in Figure 1, one could restrict testing by a certain amount of drinks and derive a corresponding value for m. The larger the bound is chosen, the larger the complete test suite is, as we illustrate by the existing upper bounds for the complexity of tests in Section 4. The complexity of a corresponding complete test suite seems to be the price for assuming such a broad fault model. A restricted fault model requires, however, a justification for each particular application.

4. CONFORMANCE TESTS FOR FSM'S

4.1. Complete deterministic FSM's

4.1.1. Conformance relation

Complete deterministic FSM's (CDFSM) constitute a narrow subclass of FSM's. Conformance test derivation from a specification CDFSM is based on the equivalence relation defined for their states (see, e.g., [Koha70], [Gill62]).

The *equivalence* relation between two states s and s' in the CDFSM $A=(S,X,Y,\delta,\lambda,s_0)$, written $s - s'$, holds if $\forall \alpha \in X^* \ (\lambda(s,\alpha) = \lambda(s',\alpha))$.
If states are not equivalent, they are *nonequivalent* or *distinguishable* states ($s \not\equiv s'$).

Two CDFSM's A and B are *equivalent*, if their initial states are equivalent, otherwise they are *distinguishable*. If A is a specification CDFSM and B is an implementation CDFSM equivalent to A, then this implementation is conforming to the specification and "–" is the finest conformance relation we could have for the model of complete deterministic FSM's. Every CDFSM that is distinguishable from the specification CDFSM, represents a non-conforming implementation.

4.1.2. Upper bounds for complete test suites

Complete test suites for a given specification CDFSM are surprisingly easy to derive if one does not care about their optimality. In particular, as well known, two minimal CDFSM with m and n states can be distinguished (if at all) by an input sequence of the length $l \leq (m+n-1)$

[Gill62], therefore the set X^{m+n-1} of all input sequences of the length (m+n-1) is a complete test suite for every n-state CDFSM with the input alphabet X w.r.t. Υ_m. This set is a universal test suite in that it does not take into account any peculiarity of a specification CDFSM except for the input alphabet and the number of states. It can be reduced to the set $VX^{m-n+k+1}$, where V is a *state cover* [Vasi73], [Chow78], [Fuji91], as was shown in [Yevt89]. Here k is the parameter, called a *distinguishability degree* of the given CDFSM A. This parameter gives a maximal length k for input sequences that tell any two distinguishable states of A apart. Note that $1 \le k \le n-1$ for any minimal CDFSM [Gill62]. A CDFSM is called *minimal (reduced)* if none of its states are equivalent. An unreduced CDFSM can be easily transformed into an equivalent minimal CDFSM, because state merging does not change traces (I/O sequences) in this kind of FSM's.

The maximal length of a sequence from the state cover V never exceeds n-1, therefore, in the worst case, the total length of a complete test suite is not more than mn^2p^{m-n+1}, where $p = |X|$, as was first shown in [Vasi73]. The presented upper bounds for complete test suite assume the reliable reset function in the IUT's, see [Yann91] for other bounds.

4.2. Partial deterministic FSM's

4.2.1. Conformance relation

Given a partial deterministic FSM (PDFSM) $A= (S,X,Y,\delta,\lambda,s_0)$ with the specification domain D_A and an arbitrary PDFSM $B=(T,X,Y',\Delta,\Lambda,t_{0B})$, B is said to be *quasi-equivalent to* A over the specification domain D_A, written $B \cong_D A$, if for all acceptable in A input sequences B and A produce the same output sequence, i.e., $\forall \alpha \in X_A^*$ ($\Delta(t_0,\alpha) = \lambda(s_0,\alpha)$).
B is said to *distinguishable from* A, written $B \text{—}_D A$, if it is not quasi-equivalent to A.

If A and B are completely specified then the relation "$\cong_D$" reduces to "$\cong$". A notion of quasi-equivalence was introduced first for state reduction of PDFSM's [Gill62], [Star72] and then adopted for test derivation [Petr91], [Yevt89], [Yevt90a]. Constructing a complete test suite for a machine from this class is complicated by the fact that minimality of a specification PDFSM cannot anymore be taken for granted, unlike in the case of CDFSM's.

4.2.2. Reduced and unreduced PDFSM's

Two states s_i and s_j in the PDFSM $A = (S,X,Y,\delta,\lambda,s_0)$ are said to be *distinguishable* if there is an input sequence acceptable to both states that tells them apart, i.e.,

$$\exists \alpha \in X_i^* \; (\; X_j^* \quad (\lambda(s_i, \alpha) \neq \lambda(s_j,\alpha)),$$

otherwise they are *compatible* states [Koha78].

A is *reduced* if all its states are pairwise distinguishable, otherwise it is *unreduced*. As an example, the PDFSM A in Figure 1 is unreduced. All the states of this machine are compatible and could be merged into one state. Merging compatible states may result in new I/O traces which are not presented in the initial specification PDFSM. However, the resulting machine would supply free drinks once inputs *coffee-button* and *tea-button* become acceptable in the initial state too.

The conclusion is that partial deterministic as well as non-deterministic FSM's should not be transformed into the reduced forms to avoid the situations when a conforming implementation could not pass a test. Clearly, during testing, compatible states cannot be

distinguished, however, testing may discover that they have been implemented as distinguishable states.

4.2.3. Upper bounds for complete test suites

Unlike the complete machines, PDFSM's may require more lengthy test sequences to obtain a complete test suite. Specifically, the maximal length of such a sequence can reach the value of mn, where n is the number of states in a given PDFSM and m is defined in Υ_m. As shown in [Yevt89], the set $X^{mn}(X^*_A$ is a complete test suite for any n-state PDFSM with the set X^*_A of acceptable input sequences.

For the given PDFSM A, it is usually possible to reduce it to the set $VX^{mz-n+k+1}(X^*_A$, where V is a state cover, k is a distinguishability degree, and z is a *fuzziness degree* of A [Yevt89]. In contrast to the complete FSM's, k may reach n(n-1)/2. A fuzziness degree z of the given machine characterizes the variety of reduced forms that can be produced from the given machine; more formally, it is the number of blocks in the minimal partition of the set of states into the subsets of pairwise distinguishable states (for detail, see [Yevt89] and also [Luo93]). Note, that $1 \le z \le n$, where z=1 corresponds to reduced, and z=n to "fully unreduced" machines (all the states are pairwise compatible, as in case of the PDFSM A in Figure 1). Shorter complete test suites can be produced by the methods which are discussed in Section 4.4.

4.3. Non-deterministic FSM's

Given the NFSM $A=(S,X,Y,h,s_0)$ and NFSM $B=(T,X,Y',H,t_0)$, B is said to be a *reduction* of A [Boch93], [Petr93], written $B \le A$, if

$$X^*_A / X^*_B, \text{ and } \forall \alpha \in X^*_A \ (H^2(t_0,\alpha) / h^2(s_0,\alpha)).$$

This conformance relation is based on the following requirement. All output sequences that are produced by the implementation in response to all acceptable input sequences shall be described by the specification [Boch93], [Fmct92]. By adopting this informal interpretation of a conforming implementation we allow the implementations to be equally or less non-deterministic than their corresponding specification NFSM. For non-deterministic implementations the relation similar to the quasi-equivalence relation for the partial deterministic FSM's seems also to be of practical interest.

An NFSM B is said to be *quasi-equivalent to* A over the specification domain D_A, written $B \cong_D A$, if X^*_A / X^*_B, and $H^2(t_0,\alpha)=h^2(s_0,\alpha)$ for all acceptable input sequences $\alpha \in X^*_A$ [Luo93]. B is said to be *distinguishable from* A, written $B —_D A$, if it is not quasi-equivalent to A. The intuition captured by this special type of a conformance relation is the following. All output sequences that are described by the specification and only they shall be produced by the implementation in response to all acceptable input sequences.

If A, B are complete NFSM's then $B \le A \ \& \ A \le B \Rightarrow B \cong A$, where "≅" is the equivalence relation as in the deterministic case.

The quasi-equivalence relation is a special case of the reduction relation, in that it requires that B is equally non-deterministic as A. If machines are deterministic then both relations reduce to the quasi-equivalence relation for partially specified machines or to the equivalence relation in the case of completely specified finite state machines.

As shown in [Star72], to distinguish two states that are not equivalent, in a complete non-deterministic FSM with n states, one has to apply an input sequence that has the length at most 2^n-2; and to distinguish two initialized non-deterministic FSM's with n and m states, respectively, one needs a sequence of the length 2^{n+m}-2. Only for observable complete FSM's the upper bounds look more optimistic: n-1 and n+m-1, respectively.

Note that testing non-deterministic implementations requires repeated execution of tests, the complexity of testing is, therefore, mostly affected by the maximal number of repetitions needed to insure that all features of the implementation are exhibited during the test campaign. This bound seems to be implementation-dependent.

4.4. Test derivation methods for FSM's

4.4.1. Test hypotheses

As discussed in Section 3, conformance test derivation from the FSM model is based on the fact that fault detection in a black-box implementation is in general undecidable unless it is dealt within a framework with certain assumptions. Specifically, a minimal test hypothesis about the class of IUT's states that every tested implementation can be represented by a single FSM; this FSM is complete, reduced, connected; it has the input alphabet of the specification FSM and the number of its states does not exceed a certain bound. If IUT's are known to be non-deterministic then the complete testing assumption is also assumed.

To guarantee full fault coverage within the predefined fault domain certain test derivation methods require even stronger assumptions that the reliable reset is available in every IUT, and the maximal number of its states is not greater than that of the specification FSM, or implementation errors are restricted to a certain known type.

Each FSM-based test derivation method also requires for its application additional properties from a specification FSM which define its applicability, as we discuss in the next subsection.

4.4.2. Taxonomy of test derivation methods

As it follows from our discussions on complete test suite in Section 3, if a method for deriving tests from an FSM is guided by a conformance relation defined for a given class of FSM's then it guarantees full fault coverage within a fixed number of states in the IUT's. The hierarchy of the methods for complete test suite derivation together with the appropriate classes of FSM's and the corresponding conformance relations is presented in Figure 5.

The methods developed especially for CDFSM's are all based on transition checking approach [Henn64]; each method, however, requires a distinct type of a characterization set [Vasi73] (a W-set) used for state identification. The UIOv-method [Vuon89], for instance, constructs the W-set only from so-called unique input/output sequences. The methods [Yevt90a], [Petr91], [Petr92] are based on "harmonized" state identifiers eliminating the verification phase that is necessary for the W-like methods [Chow78], [Fuji91]. The FF-method [Petr92] tunes a test suite to a user-defined fault domain described by a given fault function. This group of methods can be divided into two subgroups. The methods in [Vasi73], [Chow78], [Fuji91], [Yevt90a] allow the IUT's to have more states than in the specification (i.e., $m \varepsilon n$); and the others [Vuon89], [Petr91], [Petr92] do not (i.e., m=n). Which of these methods can produce a shorter complete test suite for a given CDFSM and m=n, is still an open question.

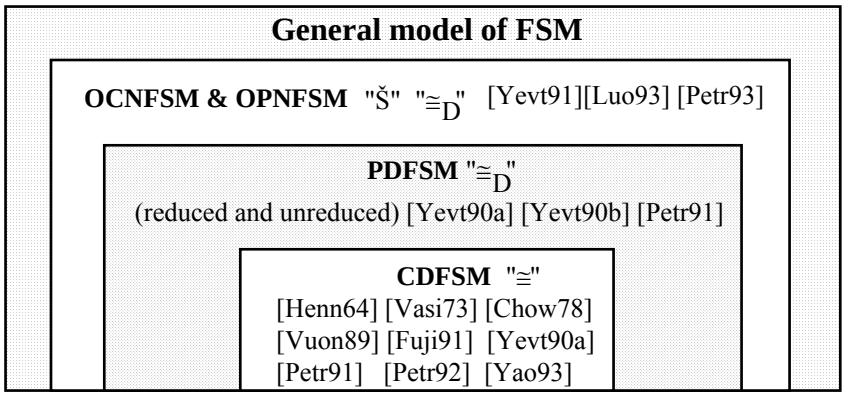

Figure 5: Taxonomy of test derivation methods and conformance relations

The methods [Yevt90a], [Yevt90b], [Petr91] that are applicable not only to CDFSM's but also to reduced and unreduced PDFSM's, are based on the idea of so-called "harmonized" state identifiers, which was further developed to cover observable complete and partial non-deterministic machines (OCNFSM and OPNFSM). In these methods, the number of states in implementations may exceed that of the specification FSM. Specifically, the method [Yevt91] deals with non-deterministic specifications and deterministic implementations based on the reduction relation. For non-deterministic implementations the HSI-method [Luo93] generates complete tests w.r.t. the quasi-equivalence relation and the SC-method [Petr93] does it w.r.t. the reduction relation.

All the mentioned methods assume that a reliable reset is available in the IUT's; note that there is another group of methods which guarantee full fault coverage without this assumption (see, e.g., [Henn64], and the UIOG-method in [Yao93]). They can be applied for reduced, deterministic, strongly connected FSM's. These methods yield, however, much longer complete test sequences than the former. Earlier reviews of the test derivation methods especially of those that do not guarantee full fault coverage, can be found elsewhere (see, e.g., [Ural91], [Sidh89]).

5. TESTING LTS CONFORMANCE RELATIONS USING FSM METHODS

5.1. Labeled transition systems

A *labeled transition system* (LTS) is a 4-tuple <S,Act,T,s_0>, where S is a non-empty set of states, $s_0 \in S$ is the initial state, Act is a set of observable actions, and $T \subseteq S \times (Act \cup \{\tau\}) \times S$ is the transition relation, τ represent an unobservable, internal action. A *finite* LTS consists of only finite sets S and Act, this class of LTS's is the subject of the remaining of this paper. The LTS's serve as a semantic model for a number of specification languages which abstract from distinguishing inputs and outputs.

An element $(s,a,s') \in T$ is denoted: $s \xrightarrow{a} s'$. The set of actions $\{a \in Act \mid \exists s' \in S\ (s \xrightarrow{a} s')\}$ is called the *acceptance set* of state s and is denoted as A(s).

A state s is *stable* if $\tau \notin A(s)$; an LTS is *stable* (called also concrete) if all its states are stable. A state s (and LTS) is *unstable* if $\tau \in A(s)$.

An LTS is *non-deterministic* if it is unstable or there is a state s and an action a such that $s \xrightarrow{a} s'$, $s \xrightarrow{a} s''$ and s'□s", in a *deterministic* LTS, the outgoing transitions of any state are uniquely labeled.

Given state s, if A(s)=□ then s is *inacting* state; an *acting* state has A(s)□□. Inacting states are usually present in an LTS with a *finite behavior* when all sequences of executed actions (traces) are finite. Unstable LTS's with infinite behavior may have *divergences* that are infinite chains of internal actions.

Similar to the case of FSM's, the transition relation is extended to sequences of actions: $\sigma \in (Act''\{\tau\})^*$, $s \xrightarrow{\sigma} s'$. An LTS is (initially) *connected* if $\forall s \in S\ \exists \sigma \in (Act''\{\tau\})^*\ (s_0 \xrightarrow{\sigma} s)$. Further we consider only connected LTS's. An LTS is *strongly connected* if $\forall s,s' \in S\ \exists \sigma \in (Act''\{\tau\})^*\ (s \xrightarrow{\sigma} s')$.

5.2. Conformance relations for LTS's

There are different criteria for determining whether an implementation LTS conforms to its specification LTS. Such a criterion is usually based on a certain aspect of a system and constitutes the *semantics* of the relations in a process theory [Glab90]. As pointed out in that work, which aspects of the behavior of a system are of importance to a certain user depends on the environment in which the system will be running, and on the interests of the particular user.

We follow the approach based on experiments on processes [Miln80], [Miln81], [Glab90] to introduce semantics for different types of conformance relations for LTS's. Depending on the properties of a tester and an interface between the tester and the process, different experiments on processes can be defined, giving rise to an abundance of semantics for the relations between processes. We reproduce in Figure 6 the spectrum of semantics for the domain of processes given in [Glab90] (related hierarchies can be found also in [Taub89], [Lang89]).

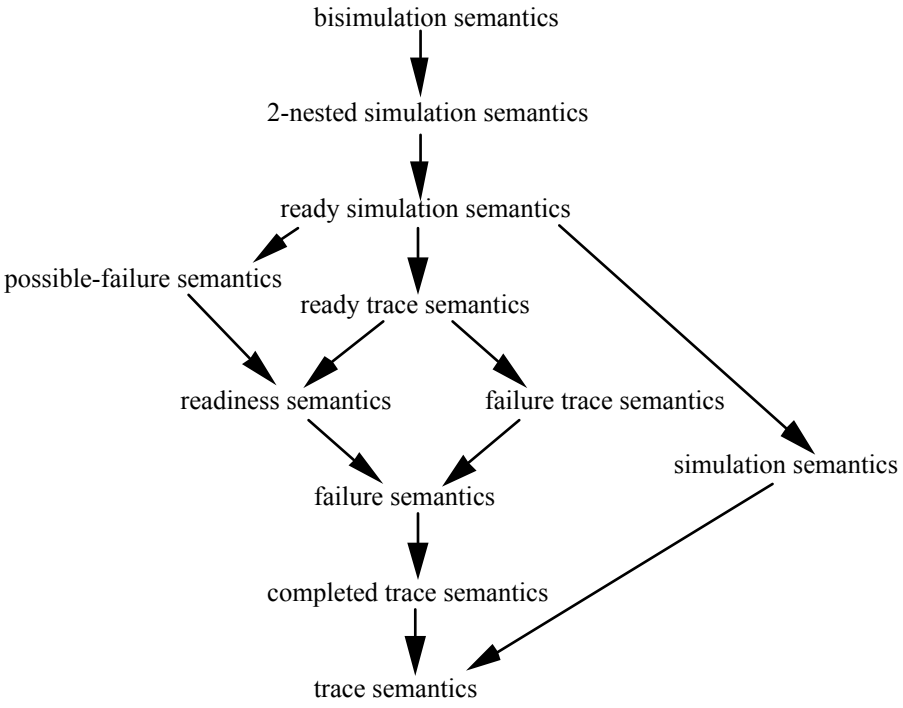

Figure 6: An hierarchy of semantics

They range between the finest bisimulation and the coarsest trace semantics. Each semantics implies certain test hypotheses, in the trace semantics, for example, only executed actions are assumed to be observed. In the failure trace semantics, a tester can not only observe deadlocks but also continue testing after them, which is not allowed in the failure semantics. In several simulation-related semantics, the tester is, at any time during a run of the implementation under test, capable of making arbitrary many copies of the IUT in its current state and observe them independently. The latter assumption is, in fact, much stronger than, say, a reliable reset assumption in the realm of FSM's. It seems to us that test hypotheses deeply influence the choice of a conformance relation among this hierarchy of the preorder relations and the associated equivalence relations.

To our knowledge, current research in testing LTS's (see, e.g., [Alde90], [Brin89], [Burg92], [Drir93], [FuBo91], [Lang89], [Weze90]) has been concentrated on conformance relations at most as fine as the failure trace semantics.

Unlike the LTS's, the FSM-based conformance relations are mainly determined by the type of the given specification FSM: complete or partial, deterministic or not-deterministic. It is, therefore, an interesting problem to find a correspondence between the conformance relations used in the FSM model and those of the LTS model. If such a correspondence can be found (at least for certain semantics of LTS's) then the test derivation methodology developed for the input/output model of FSM can be applied to check certain conformance relations expressed in the LTS formalism.

In order to apply the FSM-based methodology for conformance testing of a system specified as an LTS, one may construct an FSM from the given LTS in at least two possible ways:

- identify what actions are, in fact, driven only by the environment, and are, therefore, the inputs of an FSM; and what actions are produced by the system itself in response to an input, which are the outputs of the FSM;
- obey the rules of the experiments that can be performed on the given system within the fixed semantics and construct an FSM that matches them.

The first approach does not seem to provide a universal solution for every possible environment since the given LTS treats its actions uniformly, i.e., it does not differentiate its actions.

The next subsections will report on some preliminary results in elaborating the second approach based on a semantics-driven transformation of a LTS into an FSM. We will illustrate the approach with the failure trace semantics due to the above mentioned observation that current research in testing LTS's has been concentrated on conformance relations at most as fine as this semantics.

Once such a transformation is found the full range of the FSM-based testing theory becomes applicable to the realm of LTS's, e.g., complete test suites w.r.t. to a predefined bound on the number of states in LTS implementations can be constructed for a given conformance relation, the bounds for complexity of test suites follow from the ones for FSM's. We will show, in particular, how a "canonical tester" for a given FSM can be constructed; based on this tester, a canonical tester of a corresponding LTS for a chosen conformance relation can be obtained in a transparent way.

5.3. FSM for failure trace semantics

In the failure trace semantics, the implementation under test (or process), is modeled as a black box which has an interface to the environment consisting of a display on which the name of the action is shown that is carried out by the process in the current state; switches, one per action; a reset button and a "ready" lamp (Figure 7).

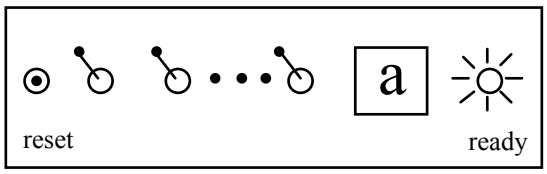

Figure 7: The black box interface of an LTS

The lamp on the interface enables a tester to distinguish two consecutive executions of the same action: when an LTS has completed the execution of a transition, it lits the "ready" lamp.

This interface bears strong similarities to the interface of the FSM model (Figure 2). The main difference is that by means of the switches the environment, i.e., a tester, now determines which actions are *free* and which are *blocked*. The process autonomously chooses an execution path labeled by a free action that belongs to the acceptance set of its current state. The reset button is needed to bring the process back to its initial state, like in the case of FSM's. If the process reaches a state where all acceptable actions are blocked, the display becomes empty. However, it does not stagnate permanently, instead it idles (the "ready" lamp is lit) and can execute further actions if the tester changes the position of switches that cause the deadlock. The tester can record the sequences of actions interleaved by the observed refusal sets.

Let A= <S,Act,T,s_0> be a stable LTS that describes the given system with the subset of acting states S_{ac}. Let $a_1a_2...a_k$, kε0 be a trace of the LTS A. A sequence $\sigma = A_0a_1A_1a_2...A_ka_k$ □ (Act"P(Act))* is a *failure trace* of A, iff there are states s_1, ..., s_k such that $s_0 \xrightarrow{a1} s_1 \xrightarrow{a2} ... \xrightarrow{ak} s_k$ and $A_i \not\ni Act\backslash A(s_i)$ $(0 \le i \le k)$. Let FT(A) denote the set of all failures traces of A. Two LTS's A and B are *failure trace equivalent,* written as $A \cong_{ft} B$, iff FT(A)=FT(B). A and B are *failure trace distinguishable,* written as $A —_{ft} B$, iff FT(A)□FT(B). If FT(A)⊅FT(B) then $A \le_{ft} B$.

Note that we consider here only stable LTS's (see e.g., [Phil87], [Lang89] for other definitions related to the unstable case).

We use a counterexample of [Glab90] to illustrate the hierarchy of different characterizations of LTS's. Figure 8 shows two LTS's A and B that are trace, complete trace, and failure equivalent, but failure trace distinguishable.

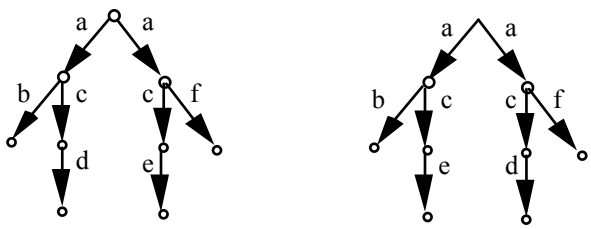

Figure 8: Failure trace distinguishable LTS's A and B

Now we wish to define an FSM which adheres to the interface in the failure trace semantics. This FSM should substitute the given LTS in a black box, such that no external observer (i.e., tester) can deduce which of them actually controls the interface from inside the black box. To this end, the FSM should have as many abstract inputs as there are combinations of switches. Under the assumption that the tester always sets at least one free action, we have $2^{|Act|}$-1 inputs which represent sets of free actions. The FSM has exactly |Act|+1 outputs, since the process is sequential, the display can show each action name and it can be empty in case of deadlock. We introduce one state in the FSM for every state in the given LTS except for inacting states in the LTS which we represent by a single terminal state of the FSM. The terminal state of the FSM ignores all the inputs, i.e., they result in the empty display (which we model in the FSM by a special output Θ) and do not change the terminal state. The output function gives those actions (or an action) that can be chosen among the free actions by the process for execution. The next-state function is defined by the transitions between the states in the given LTS, taking into account that all inacting states of the LTS are represented by a single terminal state in the FSM. Each deadlock is represented by a looping transition in the corresponding state with the input corresponding to the refusal subset and with the Θ output. The resulting FSM is called the *Failure Trace FSM* FTFSM(A) of the given LTS A.

Figure 9 shows the FTFSM(A) and FTFSM(B) constructed in this manner from the LTS's A and B (Figure 8). In these figures, the transitions are labeled by input/output pairs, for instance, *abcdef/a,* where *abcdef* represents the set of free actions (the input), and *a* is the executed action shown on display (the output). To keep the picture perceptible, only maximal sets of actions for inputs are depicted. For instance, a transition labeled by *abcdef/a* is also labeled by input/output pairs, where an input corresponds to a subset of {abcdef} which includes "a" (switch "a" is on), and the output is the action *a,* that is, *a/a, ab/a, ac/a, ..., abc/a, ..., abcdef/a.* Since a subset of a refusal set is also a refusal set, the transitions labeled, for instance, by *adef/Θ* are also labeled by *a/Θ, d/Θ, ad/Θ, ...,adef/Θ.*

Note that the constructed machines are completely specified and non-deterministic. It is easy to see that each non-deterministic LTS has a Failure Trace FSM that is not observable. If we transform such a machine into an observable form then the FSM methods discussed in Section 4 (e.g., the HSI-method) can be applied to derive a complete test suite.

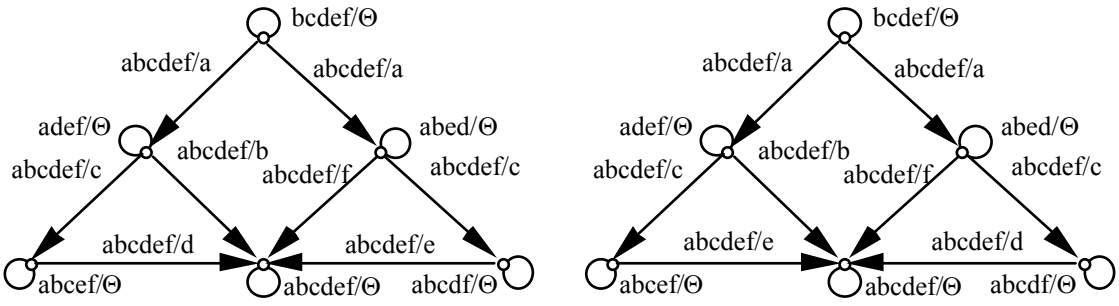

Figure 9: FTFSM's for LTS's in Figure 8

The above construction can be formalized in the following way. Given the LTS A=<S,Act,T,s_0> with the subset of acting states S_{ac}, an FSM (Q,X,Y,h,q_0) with the terminal state q_{term} is called a *Failure Trace FSM* FTFSM(A) for A iff:

(a) X = P(Act);
(b) Y = Act "{Θ};
(c) |Q| = |S_{ac}|+1;
(d) there is a function μ:S ∅ Q such that
(i) μ(s_i)=q_i for all s_i□S_{ac}, μ(s_i)=q_{term} for all s_i{S_{ac} and
(ii) for all s and x, h(μ(s),x) = { (μ(q), a) | s $\xrightarrow{a}$ q & a□x } if x(A(s)□□
and h(μ(s),x) = { (μ(s),Θ) } otherwise;
(e) h(q_{term},x) = { (q_{term},Θ) } for all x□X.

It can be seen from the Failure Trace FSM's in Figure 9 that no input sequence that corresponds to a failure pair, distinguishes these FTFSM's, and the sequence, for instance, (abcdef).(adef).(abcdef).(abcdef) can produce the output sequence *aΘcd* in the FTFSM(A) and *aΘce,* in the FTFSM(B).

Proposition 5.1: Let A and B be two stable LTS's and the FTFSM(A) and FTFSM(B) be their Failure Trace FSM's, respectively.

$$A \mathrel{-_{ft}} B \Leftrightarrow \text{FTFSM(A)} - \text{FTFSM(B)}.$$
$$A \leq_{ft} B \Leftrightarrow \text{FTFSM(A)} \leq \text{FTFSM(B)}.$$

Proof follows from the corresponding definitions.

Now we see that Proposition 3.1 comes into play for the LTS formalism. Specifically, a complete test suite for the LTS can be derived as a complete test suite for its Failure Trace FSM w.r.t. the assumed upper bound on the number of states in the implementations. Note also that there is a need for state minimization procedures for LTS's in respect to the chosen conformance relation, similar to the FSM case, because state reduction usually results in test suite reduction.

Based on Proposition 5.1, the results presented in Section 4.3 and the definition of the failure trace equivalence, we immediately get the upper bounds for a complete test suite for the given stable LTS w.r.t. the failure trace equivalence and the maximal number of states in implementations.

Proposition 5.2: Given a stable LTS A=<S,Act,T,s_0> with the subset of acting states S_{ac}, given the upper bound m on the number of the acting states in all possible implementations that have the same set of actions as the specification LTS, the set of sequences

$$(P(Act))2^{|Sac|+m+2}-2$$

is a complete test suite for the LTS A w.r.t. the failure trace equivalence and given bound m.

Corollary: If no implementation error could increase the number of acting states then there is a complete test suite for the given stable LTS with |Act| actions and n states w.r.t. the failure trace equivalence such that the number of test sequences does not exceed $(2^{|Act|}-1)^{2^{2n}-2}$ and the length of any test sequence does not exceed $2^{2n}-2$.

We conjecture that this bound can be improved not only for the class of stable LTS's but also in the general class of unstable LTS's, based on the closure property of the acceptance sets in the given LTS.

We finally note that yet another type of complete test suites can be obtained for a strongly connected LTS from its FTFSM since a complete test suite that does not necessarily rely on the reset in the implementations [Yao93].

5.4. Canonical tester for an FSM

Research in conformance test generation for LTS's has traditionally been aimed at the design of canonical testers for a chosen conformance relation (see e.g. [Brin87], [Alde90], [Brin89], Lang89], [Ledu91], [Pitt90], [Weze90]). An attractive property of a canonical tester is that all tests are, in fact, its reductions, so it serves as a basis for test selection [Brin91], [Tret91]. Once an FSM can be constructed from a given LTS and a conformance relation, it is interesting to see if a similar tester exists for FSM's as well.

Consider a closed system where an FSM is directly communicating with its environment, i.e., tester (Figure 10). The inputs of the FSM are outputs of the tester, and the inputs of the tester are the outputs of the FSM. Starting from the given Mealy FSM, it is possible to construct an FSM which represents the "canonical" tester in this closed system. This machine has to be a Moore FSM whose output function does not explicitly depends on inputs, in order to maintain cause/effect dependancy between events in the closed system. We illustrate this by two examples of vending machines.

Consider the FSM A (Figure 1). It exhibits only finite behavior, so should the tester. We construct the latter as a Moore machine with two terminal states, viz. "pass" and "fail", by changing inputs into outputs and vice versa. We assume that a global reset is used to take both machines into their initial states.

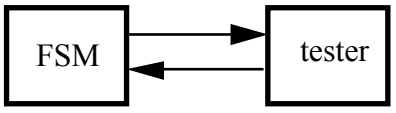

Figure 10: A closed system

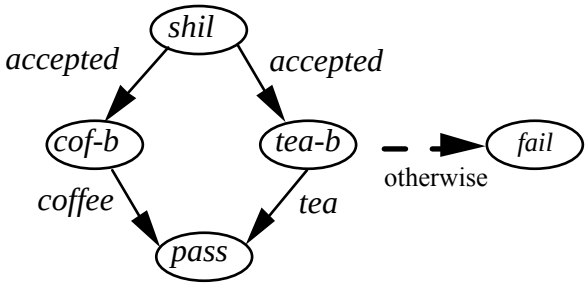

Figure 11: A canonical tester for the FSM A in Figure 1

Figure 11 shows the FSM whose outputs are assigned to the states, according to the standard convention for Moore machines. This non-deterministic machine represents, in fact, two test cases. Unspecified receptions of inputs correspond to the errors observed on the output of the IUT. Our tester will execute in these cases a transition from a non-terminal state to the terminal state "fail" for every unspecified reception of inputs represented by a global "otherwise" transition in Figure 11. This tester has full fault coverage for A w.r.t. to the quasi-equivalence relation. It is easy to convert this machine into an LTS that is the canonical tester for the LTS derived from the FSM A. It also possesses a mirror property, i.e., the canonical tester of the canonical tester is the original specification.

Even for an FSM with infinite behavior such a tester can be derived based on the standard techniques for constructing an FSM from regular expressions available in automata theory (the McNaughton-Yamada algorithm, see e.g., [Boot68]). We illustrate the procedure through

an example of the FSM B in Figure 1. The behavior of this machine can be expressed by the following regular expression

{cb/nd ≈ tb/nd ≈ sh/ac. {sh/re}*. (tb/te ≈ cb/co)}*.
1 2 3 4 5 6

The inputs - **cb** (*coffee-button*), **tb** (*tea-button*) and **sh** (*shilling*) are the outputs of the corresponding Moore machine; the outputs - **nd** (*no-drink*), **ac** (*accepted*), **te** (*tea*) and **co** (*coffee*) serve as inputs for the latter. A state is introduced each time when there is an output, as indicated by numbers below the regular expression.

Figure 12 shows the resulting Moore FSM which has 6 non-terminal states. This machine is non-deterministic and weakly initialized; the set of its possible initial states is {1,2,3}. However, there is no way (except for a demon) to introduce for this machine any transition leading to the other terminal state "pass".

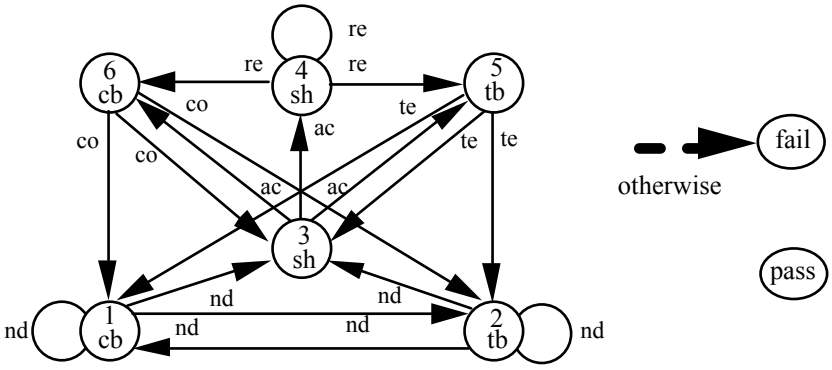

Figure 12: A canonical tester for the FSM B in Figure 1

Such a "canonical" tester guarantees full fault coverage, with respect to the equivalence relation, but it can never produce a final positive verdict. The reason is that the specification has infinite traces and the tester must execute all of them in order to successfully terminate exhaustive testing. A finite tester which successfully terminates can be obtained from the canonical tester by adopting the idea of n-testers [Pitt90], but its fault coverage would be unknown. At the same time, if we restrict the length of all input sequences which are applied to each state by a given bound, then it is possible to determine a fault domain covered by these tests (see Section 4.1.2).

Now it becomes clear that test derivation for a given LTS in respect to the failure trace equivalence, for example, can be carried out in the following two steps:
(1) construct a corresponding FTFSM and
(2) derive a test suite from the FTFSM.

The second step can be performed following different approaches:
(a) construct a canonical tester for the FTFSM and select tests based on economical and intuitive considerations; or
(b) apply a traditional FSM method.

6. CONCLUSIONS

We have shown in this paper that the same formal framework for conformance testing can be used in the context of test derivation from specifications written in the form of FSM's or LTS's. We have shown how the conformance relations can be combined with explicit fault models for the tested implementations. In the case of FSM specification, due to the input/output nature of the interactions, the conformance relation between the specification and the tested implementation will normally be based on the observed traces of input and output interactions. It is shown that a hierarchy of conformance relations used for the comparison between the specification and the tested implementation reflects the hierarchy of classes of models (completely specified and deterministic, deterministic, and non-deterministic FSM's). In particular, the reduction relation between non-deterministic FSM's can be used to derive other useful relations for the other classes of models. A review of the test derivation methods for FSM models shows that guaranteed fault coverage within a predefined fault domain may be obtained.

In the second part of the paper we have shown that it is possible to transfer the problem of deriving a conformance test suite for an LTS to the realm of the input/output FSM model, where the test derivation theory has been elaborated for several decades and a number of useful results have been obtained already. The idea is to define, for a given LTS specification and given conformance relation, a corresponding FSM specification such that the application of the FSM test suite (based on trace conformance) is equivalent to the verification of the given conformance relation in respect to the LTS specification. The feasibility of this approach is presented for the so-called failure trace equivalence [Glab90] between the LTS specification and the implementation. A corresponding canonical FSM tester can also be defined.

It remains to be investigated in detail how an FSM reflecting the peculiarities of any given LTS conformance relation should be constructed. Further research will aim also to find upper bounds for the complexity of complete test suites derived for different conformance relations, and to adapt the FSM-based test derivation methods for LTS's including unstable ones. Research in these directions would hopefully lead towards a unified conformance theory and testing methodology for these two complimentary behavioral models.

In our opinion, conformance testing should deal not only with deterministic or non-deterministic completely specified systems, but also with the ones whose aspects are not explicitly defined. Since the model of LTS implies a sole "blocking" convention for undefined behavior aspects, and the model of FSM tolerates several conventions, it remains to be seen if a more general behavioral model(s) could incorporate several useful features of both models.

Acknowledgments.

Authors would like to acknowledge the helpful discussions with the members of the "Teleinformatique" research group at the Université de Motnréal. This work was supported by the IDACOM-NSERC-CWARC Industrial Research Chair on Communication Protocols.

REFERENCES

[Alde90] R. Alderden. "COOPER, the Compositional Construction of a Canonical Tester", FORTE'90.

[Boch91] G. v. Bochmann, A. Das, R. Dssouli, M. Dubuc, A. Ghedamsi, and G. Luo, "Fault Models in Testing", IWPTS'91.

[Boch93] G. v. Bochmann, R. Gotzhein, "Specialization of Object Behaviors and Requirement Specifications", Université de Montréal, No.853, 1993, 25p.

[Boot68] T. L. Booth, Sequential Machines and Automata Theory, John Wiley & Sons, Inc., 1968.

[Brin87] E. Brinksma, "On the Existence of Canonical Testers", Memorandum INF-87-5, University of Twente, 1987.

[Brin89] E. Brinksma, et al, "A Formal Approach to Conformance Testing", IWPTS'89.

[Brin91] E. Brinksma, et al, "A Framework for Test Selection", IWPTS'91.

[Burg92] S. P. van de Burgt, J. Kroon, A.M. Peeters, "Testability of Formal Specifications", ISPSTV'92.

[Chow78] T. S. Chow, "Testing Software Design Modeled by Finite-State Machines", IEEE Transactions on Software Engineering, Vol. SE-4, No.3, 1978.

[Drir93] K. Drira, P. Azema, B. Soulas, and A. M. Chemali, "Testability of a Communicating System through an Environment", LNCS, No. 668, 1993.

[Fmtc92] ISO/IEC JTC1/SC21/WG1 N1109, CCITT COM X-28-E. "Formal Methods in Conformance Testing", Working Draft, November, 1992.

[FuBo91] S. Fujiwara, G. v. Bochmann, "Testing Non-deterministic State Machines with Fault Coverage", IWPTS'91.

[Fuji91] S. Fujiwara, G. v. Bochmann, F. Khendek, M. Amalou, A. Ghedamsi, "Test Selection Based on Finite State Models", IEEE Trans., SE-17, No.6, 1991.

[Gill62] A. Gill, Introduction to the Theory of Finite-State Machines, 1962, 270p.

[Glab90] R. J. van Glabbeek, "The Linear Time-Branching Time Spectrum", LNCS No. 458, 1990.

[Henn64] F. C. Hennie, "Fault Detecting Experiments for Sequential Circuits", Proc. of the IEEE 5th Ann. Symp. on Switching Circuits Theory and Logical Design, 1964.

[Hopc79] J. E. Hopcroft, J. D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley Publishing Company, Inc., 1979, 418p.

[Kain72] R. Y. Kain, Automata Theory: Machines and Languages, McGraw-Hill, Inc., 1972.

[Koha78] Z. Kohavi, Switching and Finite Automata Theory, McGraw-Hill, 1978.

[Lang89] R. Langerak, "A Testing Theory for LOTOS Using Deadlock Detection", ISPSTV'89.

[Ledu91] G. Leduc, "Conformance Relation, Associated Equivalence, and New Canonical Tester in LOTOS", ISPSTV'91.

[Luo93] G. Luo, A. Petrenko, G. v. Bochmann, "Selecting Test Sequences for Partially - Specified Nondeterministic Finite State Machines", Université de Montréal, No. 864, 1993, 30p.

[Miln80] R. Milner, A Calculus of Communicating Systems, LNCS, No.92, 1980.

[Miln81] R. Milner, "A Modal Characterization of Observable Machine-behavior", LNCS, No. 112, 1981.

[Moor56] E. F. Moore, "Gedanken-Experiments on Sequential Machines", Automata Studies, Princeton University Press, Princeton, New Jersey, 1956.

[Nait81] S. Naito and M. Tsunoyama, "Fault Detection for Sequential Machines by Transition Tours", Proc. of Fault Tolerant Comput. Syst., 1981, pp.238-243.

[Petr91] A. Petrenko, "Checking Experiments with Protocol Machines", IWPTS'91.

[Petr92] A. Petrenko, N. Yevtushenko, "Test Suite Generation for a FSM with a Given Type of Implementation Errors", ISPSTV'92.

[Petr93] A. Petrenko, N. Yevtushenko, A. Lebedev, A. Das, "Nondeterministic State Machines in Protocol Conformance Testing", IWPTS'93.

[Phal92] M. Phalippou, "The Limited Power of Testing", IWPTS'92.

[Phil87] I. Philips, "Refusal Testing", TCS, No. 50, 1987.

[Pitt90] D. H. Pitt, D. Freestone, "The Derivation of Conformance Tests from LOTOS Specifications", IEEE Trans., Vol. SE-16, No.12, 1990, pp.1337-1343.

[Poag64] J. F. Poage and E. J. McCluskey, Jr., "Derivation of Optimal Test Sequences for Sequential Machines", Proc. of the IEEE 5th Symp. on Switching Circuits Theory and Logical Design, 1964.

[Sidh89] D. P. Sidhu and T. K. Leung, "Formal Methods for Protocol Testing: A Detailed Study", IEEE Trans., Vol SE-15, No.4, April, 1989, pp.413-426.

[Star72] P. H. Starke, Abstract Automata, North-Holland/American Elsevier, 1972.

[Taub89] D. Taubner, "Finite Representation of CCS and TCSP Programs by Automata and Petri Nets", LNCS, No. 369, 1989.

[Tret91] J. Tretmans, P. Kars, E. Brinskma, "Protocol Conformance Testing: A Formal Perspective on ISO IS-9646", IWPTS'91.

[Ural91] H. Ural, "Formal Methods for Test Sequence Generation", Computer Communications, Vol. 15, No. 5, 1992, pp. 311-325.

[Vasi73] M. P. Vasilevski, "Failure Diagnosis of Automata", Cybernetics, Plenum Publishing Corporation, New York, No.4, 1973, pp.653-665.

[Vuon89] S. T. Vuong, W. L. Chan, and M. R. Ito, "The UIOv-method for Protocol Test Sequence Generation", IWPTS'89.

[Weze90] C. D. Wezeman, "The CO-OP Method for Compositional Derivation of Conformance Testers", ISPSTV'90.

[Yann91] M. Yannakakis, "Testing Finite State Machines", Proceedings of the 23d Annual ACM Symposium on Theory of Computing, New Orleans, Louisiana, 1991.

[Yao93] M. Yao, A. Petrenko, G. v. Bochmann, "Conformance Testing of Protocol Machines without Reset", ISPSTV'93.

[Yevt89] N. Yevtushenko and A. Petrenko, "Fault-Detection Capability of Multiple Experiments", Automatic Control and Computer Sciences, Allerton Press, Inc., New York, Vol.23, No.3, 1989, pp.7-11.

[Yevt90a] N. Yevtushenko, A. Petrenko, "Synthesis of Test Experiments in Some Classes of Automata", Automatic Control and Computer Sciences, Allerton Press, Inc., New York, Vol.24, No.4, 1990, pp.50-55.

[Yevt90b] N. Yevtushenko, A. Petrenko, "Method of Constructing a Test Experiment for an Arbitrary Deterministic Automaton", Automatic Control and Computer Sciences, Allerton Press, Inc., New York, Vol.24, No.5, 1990, pp.65-68.

[Yevt91] N. Yevtushenko, A. Lebedev, A. Petrenko, "On the Checking Experiments with Nondeterministic Automata", Automatic Control and Computer Sciences, Allerton Press, Inc., New York, Vol.25, No.6, 1991, pp.81-85.