Verdict Functions in Testing with a Fault Domain or Test Hypotheses

ROBERT M. HIERONS
Brunel University

In state based testing it is common to include verdicts within test cases, the result of the test case being the verdict reached by the test run. In addition, approaches that reason about test effectiveness or produce tests that are guaranteed to find certain classes of faults are often based on either a fault domain or a set of test hypotheses. This paper considers how the presence of a fault domain or test hypotheses affects our notion of a test verdict. The analysis reveals the need for new verdicts that provide more information than the current verdicts and for verdict functions that return a verdict based on a set of test runs rather than a single test run. The concepts are illustrated in the contexts of testing from a non-deterministic finite state machine and the testing of a datatype specified using an algebraic specification language but are potentially relevant whenever fault domains or test hypotheses are used.

Categories and Subject Descriptors: D2.4 [**Software Engineering**]: Software/Program Verification; D2.5 [**Software Engineering**]: Testing and Debugging

1. INTRODUCTION

In state based testing a test case, in the form of an adaptive process, can be augmented by verdicts (see, for example, [Pickin et al. 2007]). A verdict is associated with each possible behaviour of a *system under test (SUT)* in response to the test case and traditionally there have been three possible values for this verdict: *pass*, *fail*, and *inconclusive*. The verdict *fail* indicates that the observed behaviour is a failure and is not allowed by the specification. The verdict *pass* is used when the observed behaviour is consistent with the specification and the test objective has been achieved. The verdict *inconclusive* represents the situation in which no failures have been observed but the test objective has not been achieved. For example, if a test case aims to establish a connection over an unreliable medium and then test some associated operation *op* and it does not establish a connection then this does not indicate a failure but also does not test *op*. Verdicts are used within the standardised ETSI test description language TTCN-3 and the UML 2.0 Test Profile (see, for example [ETSI ES 201 873-1 V3.1.1 2005; Zander et al. 2005]).

In testing we aim to draw conclusions regarding the system under test (SUT) on the basis of observed behaviour and verdicts are one way of describing these

R. M. Hierons: School of Information Systems, Computing and Mathematics, Brunel University Uxbridge, Middlesex, United Kingdom, UB8 3PH.

conclusions. If we observe one or more failures then we can conclude that the SUT is faulty but even where this is not the case we may be able to deduce properties of the SUT (see, for example, [Gaudel 1995; Moore 1956]). One approach to deducing properties of the SUT based on observations is to use a fault domain. Given a specification M a fault domain Φ is a set of models such that it is believed that the SUT is functionally equivalent to an unknown element of Φ. For example, when testing from a finite state machine M with n states the fault domain might be the set of finite state machines that have the same input and output alphabets as M and at most m states for some predefined $m \geq n$. Fault domains can be used to reason about test effectiveness and drive test data generation: we aim to produce test cases that distinguish between M and the elements of Φ that do not conform to M (see, for example, [Chow 1978; Hennie 1964; Hierons and Ural 2006; Inan and Ural 1999; Luo et al. 1994; Luo et al. 1994; Petrenko et al. 1994; Petrenko et al. 1996; Rezaki and Ural 1995; Ural et al. 1997; Yevtushenko et al. 1991]). The concept of a fault domain is similar to the notion of using assumptions about the SUT, called test hypotheses, which came from the area of testing from an algebraic specification (see, for example, [Bouge et al. 1986; Gaudel 1995]). Given a set of test hypotheses there is a corresponding fault domain: the set of models that satisfy the test hypotheses. Similarly, the use of a fault domain Φ can be represented by a test hypothesis: that the SUT is functionally equivalent to a model from Φ. In this paper we use the term fault domain both for an explicit fault domain and for an implicit fault domain defined by a set of test hypotheses and all results and discussions regarding fault domains are also relevant to test hypotheses.

Fault domains and verdicts have been separately studied in state based testing. This paper considers the situation in which there is a fault domain and verdicts are being used. The analysis in Section 3 suggests that there is a need for both new verdicts and verdict functions. New verdicts are required since the presence of a fault domain makes it possible to make stronger statements regarding the SUT on the basis of test runs. For example, it is sometimes possible to deduce that the SUT must conform to the specification. Interesting, we show that sometimes it is also possible to deduce that the SUT must be faulty even if we have not observed any failures. This occurs if the set of observations made in testing is inconsistent with all of the elements of the fault domain that conform to the specification. A verdict function returns a verdict given a *set* $\mathcal{O}$ of observations, in contrast to current verdicts that are included within individual test cases and so correspond to individual observations. We show that verdict functions allow us to return verdicts based on a set $\mathcal{O}$ of observations that cannot be returned on the basis of any single element of $\mathcal{O}$. For example, it may be possible to deduce that the SUT is faulty on the basis of $\mathcal{O}$ but not from any single observation in $\mathcal{O}$. The main contributions of this paper are thus bringing together fault domains and verdicts, providing new test verdicts, the concept of verdict functions, and an analysis of some of the desirable properties of these functions. We also consider the notion of refining a verdict function for a given fault domain and how properties of verdict functions change as a fault domain is refined.

In this paper we illustrate our ideas in two contexts. The first is testing from an algebraic specification of a datatype in the presence of test hypotheses. The

second, and main, context is testing a deterministic SUT against a (possibly non-deterministic) finite state machine (FSM). In this situation the usual fault domain is the set Φ_M^m of deterministic finite state machines (DFSMs) that have at most m states for some predefined m. A test suite is a finite set of test cases, where each test case is either an input sequence or an adaptive process. When a test case is applied to the SUT we observe an input/output sequence called a *trace*. Then test suite $\mathcal{X}$ is a *checking experiment* if for all $\pi \in \Phi_M^m$, if π does not conform to M then π produces a failure on at least one test case from $\mathcal{X}$. There has been much interest in the automated generation of a checking experiment from an FSM (see, for example, [Chow 1978; Hennie 1964; Inan and Ural 1999; Luo et al. 1994; Luo et al. 1994; Petrenko et al. 1994; Petrenko et al. 1996; Petrenko and Yevtushenko 2005; Rezaki and Ural 1995; Ural et al. 1997; Yevtushenko et al. 1991]). While the work is illustrated by examples from two areas, it is potentially applicable to any area in which fault domains or test hypotheses are used.

The material contained in this paper relates to work on generating checking experiments and on testing in the presence of test hypotheses. However, these previous lines of research have focussed on the problem of generating a test suite that is guaranteed to determine the correctness of *any* SUT for the given specification and fault domain or test hypotheses. While this is undoubtedly useful, the intention is that verdict functions state what one can conclude about the *current* SUT on the basis of the observations that have been made in testing. It is possible that the test suite applied is not guaranteed to determine correctness, and so is not a checking experiment, but that the observations allow us to determine whether the current SUT is correct. For example, for any test suite we can deduce that the SUT is faulty if we observe one or more failures. Verdict functions are also more general than the verdicts currently used in languages such as TTCN-3 and the UML 2.0 Test Profile since they consider a set of observations and not just a single trace. Interestingly, a logic has recently been defined for deciding whether a set of observations allows one to deduce that the SUT is correct when testing from an FSM M using the standard fault domain Φ_M^m [Rodríguez et al. 2006]. This work can be seen as providing the basis for a partial verdict function.

The paper is structured as follows. Section 2 defines the basic concepts and notation used in this paper. Section 3 then uses examples from the areas of testing from an FSM and testing from an algebraic specification in order to show that verdict functions, which take a set of observations, can provide benefits that are not possible using verdicts associated with individual observations. It also defines new verdicts for use when testing in the presence of a fault domain. We define verdict functions in Section 4 and establish some desirable properties of these functions. Section 5 then explores the notion of refining a verdict function and Section 6 considers what it means to refine a fault domain or set of test hypotheses. Section 7 describes possible verdict functions for finite state machines and finally Section 8 draws conclusions.

2. PRELIMINARIES

2.1 Basic notation

In general, the application of a test case leads to an observation. If we are applying an input sequence then the observation is usually a trace but this need not always be the case. For example, a test case might involve creating several objects, applying sequences of operations to these, and then checking that the resultant outputs satisfy some property.

Throughout this paper, $\mathcal{O}$ will denote the set of possible observations and given a specification S, $\mathcal{M}(S)$ will denote the set of observations allowed by S. Given a specification S and SUT N we will write $N \leq S$ to mean that N conforms to S and assume that this requires that $\mathcal{M}(N) \subseteq \mathcal{M}(S)$. We will normally use the symbol S to denote a specification, the exception being the use of the symbol M when specifically considering testing from an FSM.

In this paper sequences are represented by listing their elements preceded by the symbol $\langle$, followed by the symbol $\rangle$, and separated by commas. For example, $\langle 0, 1\rangle$ denotes the sequence that contains two values, 0 followed by 1. Where a variable represents a sequence its name will have a bar above it, an example being $\bar{x}$.

2.2 Nondeterministic finite state machines

A (completely specified) *finite state machine (FSM)* M is defined by a tuple (S, s_0, X, Y, h) in which S is a finite set of states, $s_0 \in S$ is the initial state, X is the finite input alphabet, Y is the finite output alphabet, and h is the transition relation of type $S \times X \leftrightarrow S \times Y$. In this paper we will only consider completely specified FSMs. Given $s \in S$ and $x \in X$, $(s', y) \in h(s, x)$ should be interpreted as meaning that if M receives input x when in state s then it can move to state s' and output y and this defines a *transition* $(s, s', x/y)$. Consider, for example, the FSM M_0 shown in Figure 1. Here we have that $h(s_1, b) = \{(s_0, 0), (s_3, 1)\}$ and so $(s_1, s_0, b/0)$ and $(s_1, s_3, b/1)$ are transitions of M_0.

We will use two projections of h, h_1 and h_2, which represent the state transitions and outputs respectively. Thus, $h_1(s, x) = \{s' \in S \mid \exists y \in Y.(s', y) \in h(s, x)\}$ and $h_2(s, x) = \{y \in Y \mid \exists s' \in S.(s', y) \in h(s, x)\}$. In M_0 we therefore have that $h_1(s_1, b) = \{s_0, s_3\}$ and $h_2(s_1, b) = \{0, 1\}$. The relations h, h_1 and h_2 can be extended to input sequences in the usual way giving relations of type $S \times X^* \leftrightarrow S \times Y^*$, $S \times X^* \leftrightarrow S$ and $S \times X^* \leftrightarrow Y^*$ respectively. For example, in M_0 we have that $h(s_0, \langle a, b\rangle) = \{(s_0, \langle 0, 0\rangle), (s_3, \langle 0, 1\rangle)\}$.

A state s of M is *deterministically reachable* (*d-reachable*) if there exists an input sequence $\bar{x}$ such that s is the only state reached from s_0 by $\bar{x}$: $h_1(s_0, \bar{x}) = \{s\}$. For example, in M_0 the state s_1 is d-reachable since $h_1(s_0, a) = \{s_1\}$. For states s_1 and s_2, an input $\bar{x}$ *distinguishes* s_1 and s_2 if there is no common response to $\bar{x}$: $h_2(s_1, \bar{x}) \cap h_2(s_1, \bar{x}) = \emptyset$. For example, in M_0 we have that $\langle a\rangle$ distinguishes states s_0 and s_1 since $h_2(s_0, a) = \{0\}$ and $h_2(s_1, a) = \{1\}$. The (completely specified) FSM M is a *deterministic finite state machine (DFSM)* if for all $s \in S$ and $x \in X$ we have that $|h(s, x)| = 1$.

When testing from an FSM we apply a sequence of inputs and observe the corresponding sequence of inputs and outputs. If $\bar{x} = \langle x_1, \ldots, x_k\rangle$ is an input sequence and $\bar{y} = \langle y_1, \ldots, y_k\rangle$ is an output sequence then $\langle x_1/y_1, \ldots, x_k/y_k\rangle$ is an

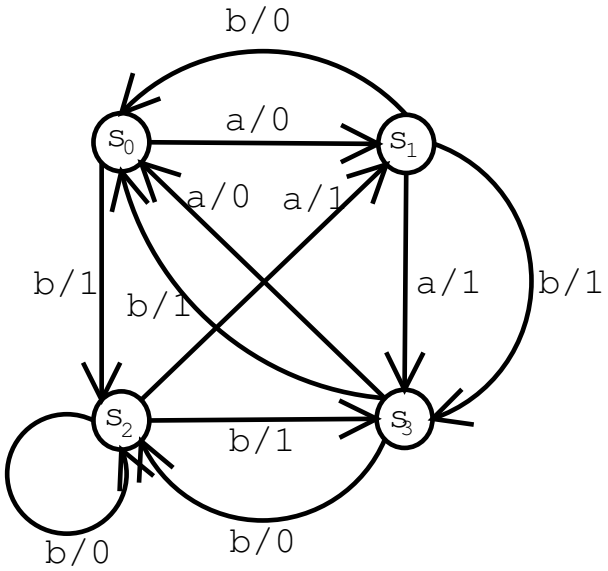

Fig. 1. The FSM M_0

input/output sequence or *trace* and this can be denoted $\bar{x}/\bar{y}$. Thus, when testing from an FSM observations are just traces. In order to simplify the exposition, when discussing testing from an FSM a *test case* will simply be an input sequence. However, the concepts and results extend to more general test cases when testing from an FSM since our observations are still traces.

Given an FSM M, a sequence of consecutive transitions $\langle(s_1, s_2, x_1/y_1), \ldots, (s_k, s_{k+1}, x_k/y_k)\rangle$ defines a trace $\langle x_1/y_1, \ldots, x_k/y_k\rangle$ from state s_1 and the set of such traces for state s_1 is denoted $L_M(s_1)$. For a state s, the set $L_M(s)$ is a regular language and $L(M)$ denotes $L_M(s_0)$: the set of traces that can be observed from the initial state of M. The language $L(M)$ defines the semantics of M and so two FSMs M_1 and M_2 are said to be *equivalent* if $L(M_1) = L(M_2)$. FSM M is *minimal* if no FSM with fewer states that M is equivalent to M.

There are two standard notions of what it means for one FSM N to conform to another FSM M. One definition says that N conforms to M if N and M are equivalent and this is appropriate when M defines the set of behaviours that should be implemented. More often, however, nondeterminism in the specification denotes alternative behaviours and where this is the case we say that N conforms to M if every behaviour of N is also a behaviour of M and this is the notion of conformance used in this paper. More formally, N conforms to M, written $N \leq M$, if N and M have the same input alphabets and $L(N) \subseteq L(M)$. Similarly, given state s of FSM M and state t of FSM N, where M and N have the same input alphabets, we write $t \leq s$ if $L_N(t) \subseteq L_M(s)$.

If we test the SUT against FSM M by applying an input sequence $\bar{x}$ and observe output sequence $\bar{y}$ then there has been a *failure* if $\bar{x}/\bar{y} \notin L(M)$. Recall that Φ_M^m is the set of FSMs with the same input and output alphabets as M and no more

than m states and $\mathcal{M}(M)$ is the set of observations allowed by M. Since for an FSM M, $L(M)$ is the set of possible observations, we have that $\mathcal{M}(M) = L(M)$. If $M' \in \Phi_M^m$ then the trace $\bar{x}/\bar{y}$ *kills* M' if $\bar{x}/\bar{y} \notin L(M')$; this observed behaviour shows that the SUT cannot be equivalent to M'.

2.3 Algebraic Specifications

Algebraic specification languages use axioms in order to specify required properties. An algebraic specification thus consists of a set of sorts, a list of operations, and a set of axioms. Examples of algebraic specification languages include OBJ [Goguen and Tardo 1979; Goguen and Malcolm 2000] and the common algebraic specification language (CASL) [Bidoit and Mosses 2003; Mosses 2004]. An algebraic specification of a variant on the datatype of sets of natural numbers, `sett`, might have operations:

—`empty` to create a new empty `sett`;

—`isempty` to decide whether a `sett` is empty;

—`in` to decide whether an element is in a `sett`;

—`add` to add an element to a `sett`;

—`delete` to remove an element from a `sett`;

—`retrieve` to return some element of a `sett`.

We might define the `sett` type in the following way, in which `Nat` denotes an imported type for the natural numbers.

```
spec Sett =
     Nat
then
     sort sett
     preds
          isempty:  sett;
          in:  Nat * sett;
     ops
          empty:  sett
          add :  Nat * sett -> sett;
          delete :  Nat * sett -> sett;
          retrieve :  sett -> Nat;
     vars
          n, n':  Nat; s:  sett
     axioms
          isempty(empty);
          ¬ isempty(add(n,s));
          ¬ in(n,empty);
          ¬ empty(s) ⇒ in(retrieve(s),s);
          in(n,add(n',s)) ⇔ n = n' ∨ in(n,s);
          in(n,delete(n',s)) ⇔ n ≠ n' ∧ in(n,s);
end
```

The first two axioms say that a `sett` is empty if and only if it is `empty` while the third axiom says that no element is in `empty`. The fourth axiom says that

the element retrieved from a non-empty `sett` is in that `sett`. The remaining two axioms say that the elements in `add(n',s)` are `n'` and the elements in `s` while the elements in `delete(n',s)` are those that are in `s` and are not `n'`.

When testing against `sett` we will check that the axioms hold by instantiating the elements in them. For example, we could instantiate the last axiom with `n` being `1`, `n'` being `2` and `s` being `empty`, in which case we would check that `in(1,delete(2,empty))` is the same as `in(1,empty)`. Similarly, we could instantiate the fourth axiom with `s` being `add(1,empty)`. Note that in this case, in testing we will observe the value returned by `retrieve` as well as the outcome of `in(retrieve(add(1,empty)),add(1,empty))` and so an observation will be a boolean that represents whether the axiom held for that instance and also some values returned during this process.

There are two main types of test hypotheses in the literature on testing: uniformity hypotheses and regularity hypotheses (see, for example, [Gaudel 1995]). Uniformity hypotheses state that certain values can be treated as equivalent for the purpose of testing. For example, in `sett` if we are generating tests from the third axiom then we might say that all of the values for `n` are equivalent and so it is sufficient to produce one test case from this axiom, for example `in(1,empty)`. A regularity hypothesis states that if all tests using structures with at most a given size or complexity pass then all tests will pass. For `sett` we might choose an integer m and say that it is sufficient to test with all elements of `sett` that can be constructed by adding at most m elements to `empty`.

While the two main classes of test hypotheses discussed in the literature are the uniformity and regularity hypotheses, it is possible to use other types of test hypotheses. For example, we might believe that the implementation of `sett` is deterministic and that the element returned by `retrieve` is always the element of the `sett` that was most recently added. Later we will see that this test hypothesis can lead to some interesting situations in testing.

3. THE NEED FOR NEW VERDICTS

Let us suppose that we are testing deterministic SUT N against FSM M using test suite $\mathcal{X}$, we have observed the set $\mathcal{T}$ of traces and this contains no failures. Without the use of a fault domain all we can conclude is that the SUT conforms to M on these test cases. Previous work has shown how the existence of a fault domain Φ can sometimes allow us to conclude that the SUT is correct on the basis of a set of observed traces (see, for example, [Hennie 1964]). We can make this conclusion if $\mathcal{T}$ kills all of the elements of Φ that do not conform to M: $\forall \pi \in \Phi . \pi \not\leq M \Rightarrow \mathcal{T} \not\subseteq L(\pi)$. It seems natural to use a verdict to represent this situation but this is not the intended use of the standard verdict *pass*. We thus introduce a new verdict *correct* to denote the situation in which the behaviours observed in testing show that if the SUT is equivalent to a member of Φ then it must be correct. More generally, *correct* denotes the situation in which we can conclude that the SUT must be correct on the basis of the set $\mathcal{O}$ of observations made in testing and the fault domain or set of test hypotheses used.

Relatively little attention has been paid to the case in which all of the elements

of Φ that *conform* to M have been killed[1]: $\forall \pi \in \Phi.\pi \leq M \Rightarrow \mathcal{T} \not\subseteq L(\pi)$. Here we can conclude that if the SUT really is functionally equivalent to an (unknown) element of Φ then the SUT must be faulty. This suggests that it may be possible to conclude that the SUT is faulty even though we have not observed any failures.

Consider the FSM M_0 shown in Figure 1 and let us suppose that we are using the fault domain $\Phi^4_{M_0}$ in which the SUT N_0 must be deterministic and have no more states than M_0. Here we may observe that all of the states are deterministically reachable and pairwise distinguishable (using input sequence $\langle a, a \rangle$) and thus for an SUT to conform to M_0 it must contain separate states $t_0, \ldots, t_3$ such that $t_i \leq s_i$ (for all $0 \leq i \leq 3$). Now let us suppose that we test an SUT N_0 using test case $\langle b, b, b \rangle$ and we observe the trace $\langle b/1, b/0, b/1 \rangle$. This trace is contained in $L(M_0)$ and thus N_0 conforms to M_0 on the test case $\langle b, b, b \rangle$. However, the traces $\langle b/1 \rangle$ and $\langle b/1, b/0 \rangle$ reach the same state s_2 of M_0 and the corresponding states of N_0 are distinguished by the input of b since we observe $b/0$ after the trace $\langle b/1 \rangle$ and we observe $b/1$ after the trace $\langle b/1, b/0 \rangle$. Thus, if N_0 conforms to M_0 then it must have at least two states that conform to s_2 and so must have at least five states in total. However, our fault domain contains FSMs with no more than 4 states. We can thus conclude that if we observe the trace $\langle b/1, b/0, b/1 \rangle$ from the initial state of an SUT N_0 from $\Phi^4_{M_0}$ then this SUT must be faulty even though we have not observed a failure.

A similar situation can occur when testing against `sett` using the test hypothesis that the element retrieved from a `sett` s is always the element of s that was most recently added. We might use the test case `in(retrieve(add(1,add(2,empty))), add(1,add(2,empty)))` and get 2 returned by `retrieve(add(1,add(2,empty)))`. This result is consistent with the specification but no implementation that conforms to `sett` and satisfies the test hypothesis can produce this observation. We use *incorrect* to represent the situation in which we can conclude that the SUT must be faulty on the basis of the set $\mathcal{O}$ of observations made in testing and the fault domain used.

Again consider M_0 and assume that we are using the fault domain $\Phi^4_{M_0}$. Let us suppose that we observe the trace $\langle b/1, b/0, b/0, b/0, b/0, b/0, b/1 \rangle$. The states of the SUT reached by the prefixes of lengths one to six must be different since from this trace we can conclude that these states respond differently to $\langle b, b, b, b, b \rangle$. Thus, if we observe this trace then the SUT cannot behave like an element of the fault domain: we cannot determine whether the SUT is correct or faulty but we can deduce that the assumption that the SUT behaves like an element of $\Phi^4_{M_0}$ was incorrect. This situation is not covered by the standard verdicts or by our new verdicts *correct* and *incorrect* and so we need an additional verdict that we call *inconsistent*: this represents the situation in which no SUT that satisfies the set of test hypotheses or is in the fault domain allows the set $\mathcal{O}$ of observations made in testing. Using `sett` and the test hypothesis described earlier, we get verdict *inconsistent* if we test with `in(retrieve(add(1,add(2,empty))),add(1,add(2,empty)))` twice and different values are produced by the two uses of `retrieve(add(1,add(2,empty)))`. As usual we require a verdict for situations in which none of these verdicts apply

[1] In [Hierons 1998] it has, however, been observed that where there is a fault domain it is possible to distinguish states of an FSM that cannot be distinguished without the fault domain.

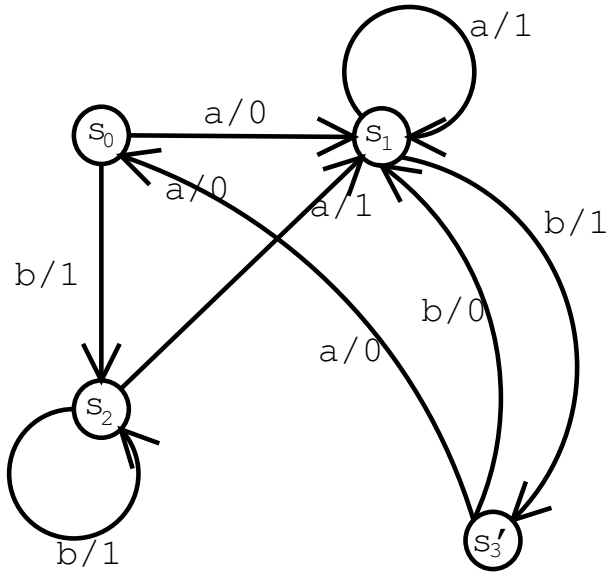

Fig. 2. The DFSM N_0

and use *uncertain*.

We have seen that there are situations in which the result of testing cannot be captured by the three standard verdicts. Now suppose that the SUT is intended to conform to M_0 but actually behaves like the DFSM N_0 shown in Figure 2. Let us also suppose that we separately apply input sequences $\langle b, b, b\rangle$ and $\langle a, b, b\rangle$ from the initial state of N_0 and so we observe the traces $\langle b/1, b/1, b/1\rangle$ and $\langle a/0, b/1, b/0\rangle$. Each trace is contained in $L(M_0)$ and is contained in the languages of both correct and incorrect elements of the fault domain $\Phi^4_{M_0}$. So, if we assign a verdict to either trace on its own then we obtain the verdict *uncertain*. However, we know that if the SUT conforms to M_0 then the traces $\langle b/1, b/1\rangle$ and $\langle a/0, b/1\rangle$ must both reach states of the SUT that conform to s_3. However, the SUT responds differently to b after these two traces and so a conforming SUT that contains these traces must have at least two states that conform to s_3. Such an SUT must have at least five states in total and so we can conclude that the SUT cannot be an element of $\Phi^4_{M_0}$ that conforms to M_0.

In this example we obtain verdict *uncertain* if we consider the observations separately but we can obtain the verdict *incorrect* if we consider the observations together. As a result of this we propose the use of *verdict functions* that take a set of observations and return a verdict. The benefit is that, as shown above, verdict functions can allow us to obtain more information than can be gained through the usual practice of associating verdicts with outcomes of a single test case.

4. VERDICT FUNCTIONS

Let us suppose that we have tested the SUT N against specification S with fault domain Φ and we have made the set $\mathcal{O}$ of observations in testing. The intention is

that the verdict tells us what we may conclude, regarding whether N conforms to S, on the basis of $\mathcal{O}$ and Φ. A verdict function $\mathcal{V}$ is thus a function from sets of observations to verdicts, where for each set of observations $\mathcal{O}$ the verdict $\mathcal{V}(\mathcal{O})$ is one of the following:

—the value *correct* — this is intended to represent the situation in which we can conclude that N conforms to S.

—the value *incorrect* — this is intended to represent the situation in which we can conclude that N does not conform to S.

—the value *uncertain* — this is intended to represent the situation in which we cannot conclude that N conforms to S, we cannot conclude that N does not conform to S, but N might be equivalent to an element of Φ.

—the value *inconsistent* — this is intended to represent the situation in which no element of Φ can produce the set $\mathcal{O}$ of observations and so the assumption that $N \in \Phi$ cannot hold.

A verdict function is always with respect to a fault domain. In most cases the fault domain is clear but if not, for a verdict function $\mathcal{V}$ for fault domain Φ we say that $\mathcal{V}$ *is a verdict function with respect to* Φ.

The verdicts represent our ability to kill elements of Φ on the basis of $\mathcal{O}$. The verdicts are thus related: for example, we obtain verdict *inconsistent* if we can kill all elements of Φ while we obtain verdict *correct* if we can kill all elements of Φ that do not conform to the specification but we cannot kill some elements of Φ that conform to S. There is then a natural partial ordering on verdicts: one verdict is 'above' another if the first corresponds to killing more elements of Φ than the second.

***Definition* 4.1** The partial order $\preceq$ on verdicts is defined by the transitive reflexive closure of the following:

(1) $uncertain \preceq correct$

(2) $uncertain \preceq incorrect$

(3) $correct \preceq inconsistent$

(4) $incorrect \preceq inconsistent$

The set of verdicts, with the partial order $\preceq$, forms a lattice since for any pair of verdicts it is possible to find a unique least upper bound and a unique greatest lower bound. Given verdicts v_1 and v_2 we will let $v_1 \vee v_2$ denote the least upper bound of v_1 and v_2. For example, $correct \vee uncertain = correct$, $correct \vee inconsistent = inconsistent$, and $correct \vee incorrect = inconsistent$. Essentially, if we have verdicts v_1 and v_2 and these correspond to knowing that the SUT is not equivalent to any element in $\Phi_1 \subseteq \Phi$ and $\Phi_2 \subseteq \Phi$ respectively then $v_1 \vee v_2$ is the verdict that corresponds to knowing that the SUT is not equivalent to any element in $\Phi_1 \cup \Phi_2$.

Traditionally, testing has returned a verdict $fail$ if and only if a failure has been observed and this corresponds to the following verdict function.

***Definition* 4.2** The basic verdict function $\mathcal{V}_\emptyset$ is defined by:

$$\mathcal{V}_\emptyset(\mathcal{O}) = \begin{cases} incorrect & \text{if } \mathcal{O} \not\subseteq \mathcal{M}(S) \\ uncertain & \text{otherwise} \end{cases}$$

This is equivalent to not having a fault domain or using the minimal test hypothesis [Gaudel 1995]. In the work on test hypotheses there are two extremes: the minimal hypothesis in which we make no assumptions beyond the input and output sets and the hypothesis in which we assume that the SUT is correct. The latter corresponds to the following verdict function.

***Definition* 4.3** The *maximal* verdict function $\mathcal{V}_T$ is defined by: for all $\mathcal{O}$ we have that $\mathcal{V}_T(\mathcal{O}) = correct$.

However, this does not fully correspond to our understanding of testing since it allows us to declare an SUT as being correct even if we have observed failures. Thus, we want our verdict functions to satisfy the following condition.

***Definition* 4.4** The function $\mathcal{V}$ is a *valid* verdict function if whenever $\mathcal{O} \not\subseteq \mathcal{M}(S)$ we have that $\mathcal{V}(\mathcal{O})$ returns either *incorrect* or *inconsistent*.

This leads to an alternative maximal verdict function.

***Definition* 4.5** The *maximal valid* verdict function $\mathcal{V}_T^v$ is defined by: for all $\mathcal{O}$ we have that

$$\mathcal{V}_T^v(\mathcal{O}) = \begin{cases} inconsistent & \text{if } \mathcal{O} \not\subseteq \mathcal{M}(S) \\ correct & \text{otherwise} \end{cases}$$

We will use the following notation. Given a set $\mathcal{O}$ of observations and fault domain Φ we let $C(\mathcal{O}, \Phi)$ denote the set $\{\pi \in \Phi | \mathcal{O} \subseteq \mathcal{M}(\pi)\}$ of elements of Φ that are consistent with $\mathcal{O}$. Given specification S we let $conf(S)$ denote the set of SUT that conform to S and let $\overline{conf(S)}$ denote the set of SUT that do not conform to S.

We can see the process of determining a verdict as involving eliminating elements of the fault domain: the verdict returned depends on properties of the models that are not killed. We should only eliminate a model π from Φ if $\mathcal{O}$ demonstrates that the SUT cannot be equivalent to π. This observation leads to the following additional desirable property that says that the verdict function should not be able to return verdicts that do not follow from the fault domain and the set of observations made.

***Definition* 4.6** The function $\mathcal{V}$ is a *sound* verdict function with respect to Φ if the following hold:

(1) $C(\mathcal{O}, \Phi) \cap conf(S) \neq \emptyset \Rightarrow \mathcal{V}(\mathcal{O}) \preceq correct$

(2) $C(\mathcal{O}, \Phi) \cap \overline{conf(S)} \neq \emptyset \Rightarrow \mathcal{V}(\mathcal{O}) \preceq incorrect$

The first of these says that if there exist elements of the fault domain Φ that are consistent with $\mathcal{O}$ and that conform to the specification then the verdict should either be *correct* or *uncertain*. The second says that if there exist elements of the fault domain Φ that are consistent with $\mathcal{O}$ and that do not conform to the specification then the verdict should either be *incorrect* or *uncertain*.

If we have sets $\mathcal{O}_1$ and $\mathcal{O}_2$ of observations and $\mathcal{O}_1 \subseteq \mathcal{O}_2$ then we should be able to deduce at least as much about the SUT from $\mathcal{O}_2$ as from $\mathcal{O}_1$. This observation is captured by the notion of a verdict function being monotonic.

***Definition* 4.7** Verdict function $\mathcal{V}$ is *monotonic* if for all sets $\mathcal{O}_1$ and $\mathcal{O}_2$ of observations such that $\mathcal{O}_1 \subseteq \mathcal{O}_2$ we have that $\mathcal{V}(\mathcal{O}_1) \preceq \mathcal{V}(\mathcal{O}_2)$.

It is crucial that a verdict function is sound — the verdict function should not suggest that elements of the fault domain can be eliminated if they cannot. Further, we expect verdict functions to be monotonic and valid. The following is the ideal case.

***Definition* 4.8** The function $\mathcal{V}$ is a *precise* verdict function if the value returned satisfies the following properties.

(1) $\mathcal{V}(\mathcal{O}) = correct \Leftrightarrow \emptyset \neq C(\mathcal{O}, \Phi) \subseteq conf(S)$.
(2) $\mathcal{V}(\mathcal{O}) = incorrect \Leftrightarrow \emptyset \neq C(\mathcal{O}, \Phi) \subseteq \overline{conf(S)}$.
(3) $\mathcal{V}(\mathcal{O}) = uncertain \Leftrightarrow C(\mathcal{O}, \Phi) \cap conf(S) \neq \emptyset \wedge C(\mathcal{O}, \Phi) \cap \overline{conf(S)} \neq \emptyset$.
(4) $\mathcal{V}(\mathcal{O}) = inconsistent \Leftrightarrow C(\mathcal{O}, \Phi) = \emptyset$.

Proposition 4.9 *If $\mathcal{V}$ is a precise verdict function then $\mathcal{V}$ is valid, sound, and monotonic.*

We can combine verdict functions.

***Definition* 4.10** If $\mathcal{V}_1$ and $\mathcal{V}_2$ are verdict functions then the verdict function $\mathcal{V}_1 \uplus \mathcal{V}_2$ is defined by, given set $\mathcal{O}$ of observations we have that $(\mathcal{V}_1 \uplus \mathcal{V}_2)(\mathcal{O})$ is the least upper bound $\mathcal{V}_1(\mathcal{O}) \vee \mathcal{V}_2(\mathcal{O})$ of $\mathcal{V}_1(\mathcal{O})$ and $\mathcal{V}_2(\mathcal{O})$.

Here, we have two verdict functions $\mathcal{V}_1(\mathcal{O})$ and $\mathcal{V}_2(\mathcal{O})$ that return information about what we can deduce regarding the SUT given a set of observations. The verdict function $\mathcal{V}_1 \uplus \mathcal{V}_2$ combines this information. We now explore some properties of $\mathcal{V}_1 \uplus \mathcal{V}_2$.

Proposition 4.11 *If $\mathcal{V}_1$ and $\mathcal{V}_2$ are sound verdict functions then the verdict function $\mathcal{V}_1 \uplus \mathcal{V}_2$ is sound.*

Proof. First, assume that $C(\mathcal{O}, \Phi) \cap conf(S) \neq \emptyset$. Since $\mathcal{V}_1$ and $\mathcal{V}_2$ are sound we must have that $\mathcal{V}_1(\mathcal{O}) \preceq correct$ and $\mathcal{V}_2(\mathcal{O}) \preceq correct$. Thus, $(\mathcal{V}_1 \uplus \mathcal{V}_2)(\mathcal{O}) \preceq correct$ as required.

Now, assume that $C(\mathcal{O}, \Phi) \cap \overline{conf(S)} \neq \emptyset$. Since $\mathcal{V}_1$ and $\mathcal{V}_2$ are sound we must have that $\mathcal{V}_1(\mathcal{O}) \preceq incorrect$ and $\mathcal{V}_2(\mathcal{O}) \preceq incorrect$. Thus, $(\mathcal{V}_1 \uplus \mathcal{V}_2)(\mathcal{O}) \preceq incorrect$ as required. □

Proposition 4.12 *If $\mathcal{V}_1$ and $\mathcal{V}_2$ are monotonic verdict functions then the verdict function $\mathcal{V}_1 \uplus \mathcal{V}_2$ is monotonic.*

PROOF. We require to prove that whenever we have sets $\mathcal{O}_1$ and $\mathcal{O}_2$ of observations and $\mathcal{O}_1 \subseteq \mathcal{O}_2$ then $(\mathcal{V}_1 \uplus \mathcal{V}_2)(\mathcal{O}_1) = \mathcal{V}_1(\mathcal{O}_1) \vee \mathcal{V}_2(\mathcal{O}_1) \preceq (\mathcal{V}_1 \uplus \mathcal{V}_2)(\mathcal{O}_2) = \mathcal{V}_1(\mathcal{O}_2) \vee \mathcal{V}_2(\mathcal{O}_2)$. Since $\mathcal{V}_1$ and $\mathcal{V}_2$ are monotonic, $\mathcal{V}_1(\mathcal{O}_1) \preceq \mathcal{V}_1(\mathcal{O}_2)$ and $\mathcal{V}_2(\mathcal{O}_1) \preceq \mathcal{V}_2(\mathcal{O}_2)$ and so every upper bound on both $\mathcal{V}_1(\mathcal{O}_2)$ and $\mathcal{V}_2(\mathcal{O}_2)$ is also an upper bound on both $\mathcal{V}_1(\mathcal{O}_1)$ and $\mathcal{V}_2(\mathcal{O}_1)$. The result thus follows. □

Note that if one of $\mathcal{V}_1$ and $\mathcal{V}_2$ is monotonic but the other is not then the verdict function $\mathcal{V}_1 \uplus \mathcal{V}_2$ need not be monotonic. To see this assume that the specification is some FSM and let $\mathcal{V}_1$ be the basic verdict function $\mathcal{V}_\emptyset$. If $\mathcal{V}_2$ is a verdict function that returns *inconsistent* for every set of observations except for one specific set $\mathcal{O}'$ that contains only one trace (that is an element of $L(M)$), which is mapped to *uncertain*, then $\mathcal{V}_1 \uplus \mathcal{V}_2 = \mathcal{V}_2$ and clearly is not monotonic.

Proposition 4.13 *If at least one of the verdict functions $\mathcal{V}_1$ and $\mathcal{V}_2$ is valid then $\mathcal{V}_1 \uplus \mathcal{V}_2$ is valid.*

PROOF. Without loss of generality, assume that $\mathcal{V}_1$ is valid. Then whenever $\mathcal{O} \not\subseteq \mathcal{M}(S)$ we have that $incorrect \preceq \mathcal{V}_1(\mathcal{O})$. But, given $v = \mathcal{V}_2(\mathcal{O})$ we know that $v \vee inconsistent = inconsistent$ and $incorrect \preceq v \vee incorrect$. Thus, if $\mathcal{O} \not\subseteq \mathcal{M}(S)$ then we have that $(\mathcal{V}_1 \uplus \mathcal{V}_2)(\mathcal{O})$ is either *incorrect* or *inconsistent* as required. □

Proposition 4.14 *If at least one of the verdict functions $\mathcal{V}_1$ and $\mathcal{V}_2$ is precise and the other is sound then $\mathcal{V}_1 \uplus \mathcal{V}_2$ is precise.*

PROOF. Without loss of generality, assume that $\mathcal{V}_1$ is precise and $\mathcal{V}_2$ is sound. It is now sufficient to consider the four cases.

(1) The value of $\mathcal{V}_1(\mathcal{O})$ is *correct* and so $\emptyset \neq C(\mathcal{O}, \Phi) \subseteq conf(S)$. Since $\mathcal{V}_2$ is sound, we have that $\mathcal{V}_2(\mathcal{O}) \preceq correct$ and so $(\mathcal{V}_1 \uplus \mathcal{V}_2)(\mathcal{O})$ is *correct* as required.
(2) The value of $\mathcal{V}_1(\mathcal{O})$ is *incorrect* and so $\emptyset \neq C(\mathcal{O}, \Phi) \subseteq \overline{conf(S)}$. Since $\mathcal{V}_2$ is sound, we have that $\mathcal{V}_2(\mathcal{O}) \preceq incorrect$ and so $(\mathcal{V}_1 \uplus \mathcal{V}_2)(\mathcal{O})$ is *incorrect* as required.
(3) The value of $\mathcal{V}_1(\mathcal{O})$ is *uncertain*. Thus, $C(\mathcal{O}, \Phi) \cap conf(S) \neq \emptyset$ and $C(\mathcal{O}, \Phi) \cap \overline{conf(S)} \neq \emptyset$. Since $\mathcal{V}_2$ is sound, we have that $\mathcal{V}_2(\mathcal{O}) = uncertain$ and so $(\mathcal{V}_1 \uplus \mathcal{V}_2)(\mathcal{O}) = uncertain$ as required.
(4) The value of $\mathcal{V}_1(\mathcal{O})$ is *inconsistent* and so $\mathcal{V}_1(\mathcal{O}) \vee \mathcal{V}_2(\mathcal{O})$ is *inconsistent* as required.

The result thus follows. □

5. REFINING VERDICT FUNCTIONS

The literature on test hypotheses discusses the idea of refining a test hypothesis [Bouge et al. 1986] and this corresponds to reducing the size of a fault domain. In this section we assume that the fault domain Φ is fixed and instead we want to increase our ability to determine that an SUT is faulty or correct on the basis of a set of observations.

***Definition* 5.1** Verdict function $\mathcal{V}_2$ is a *refinement* of verdict function $\mathcal{V}_1$, written $\mathcal{V}_2 \sqsupseteq \mathcal{V}_1$, if for all $\mathcal{O}$ we have that $\mathcal{V}_1(\mathcal{O}) \preceq \mathcal{V}_2(\mathcal{O})$.

This says that the verdict function $\mathcal{V}_2$ is at least as effective as $\mathcal{V}_1$ in terms of eliminating elements of Φ. Thus, as long as our verdict functions are sound this means that $\mathcal{V}_2$ is at least as good as $\mathcal{V}_1$ for determining properties of the SUT.

The following are clear.

Proposition 5.2 *Given sound verdict functions $\mathcal{V}_1$ and $\mathcal{V}_2$ we have that:*

(1) $\mathcal{V}_1 \sqsupseteq \mathcal{V}_\emptyset$.

(2) if $\mathcal{V}_2$ is precise then $\mathcal{V}_2 \sqsupseteq \mathcal{V}_1$.

(3) If $\mathcal{V}_2 \sqsupseteq \mathcal{V}_1$ then for every set $\mathcal{O}$ we have that $\mathcal{V}_2(\mathcal{O}) = (\mathcal{V}_1 \uplus \mathcal{V}_2)(\mathcal{O})$.

Some important properties are preserved by refining verdict functions.

Proposition 5.3 *If $\mathcal{V}_2 \sqsupseteq \mathcal{V}_1$ and $\mathcal{V}_1$ is valid then $\mathcal{V}_2$ is valid.*

PROOF. Assume that $\mathcal{O} \not\subseteq \mathcal{M}(S)$ and so, since $\mathcal{V}_1$ is a valid verdict function we have that $\mathcal{V}_1(\mathcal{O})$ is either *incorrect* or *inconsistent*. Since $\mathcal{V}_2 \sqsupseteq \mathcal{V}_1$ we have that $\mathcal{V}_1(\mathcal{O}) \preceq \mathcal{V}_2(\mathcal{O})$ and so $\mathcal{V}_2(\mathcal{O})$ is either *incorrect* or *inconsistent* as required. □

Proposition 5.4 *If $\mathcal{V}_2 \sqsupseteq \mathcal{V}_1$, $\mathcal{V}_1$ is precise and $\mathcal{V}_2$ is sound then $\mathcal{V}_2$ is precise.*

PROOF. For a set $\mathcal{O}$ there are four cases to consider.

(1) The value of $\mathcal{V}_1(\mathcal{O})$ is *correct*. Thus, $\emptyset \neq C(\mathcal{O}, \Phi) \subseteq conf(S)$. Since $\mathcal{V}_2$ is sound, we have that $\mathcal{V}_2(\mathcal{O}) \preceq$ *correct* and so, since $\mathcal{V}_1(\mathcal{O}) \preceq \mathcal{V}_2(\mathcal{O})$, $\mathcal{V}_2(\mathcal{O})$ is *correct* as required.

(2) The value of $\mathcal{V}_1(\mathcal{O})$ is *incorrect*. Thus, $\emptyset \neq C(\mathcal{O}, \Phi) \subseteq \overline{conf(S)}$. Since $\mathcal{V}_2$ is sound, we have that $\mathcal{V}_2(\mathcal{O}) \preceq$ *incorrect* and so, since $\mathcal{V}_1(\mathcal{O}) \preceq \mathcal{V}_2(\mathcal{O})$, $\mathcal{V}_2(\mathcal{O})$ is *incorrect* as required.

(3) The value of $\mathcal{V}_1(\mathcal{O})$ is *uncertain*. Thus, $C(\mathcal{O}, \Phi) \cap conf(S) \neq \emptyset$ and $C(\mathcal{O}, \Phi) \cap \overline{conf(S)} \neq \emptyset$. Since $\mathcal{V}_2$ is sound, we have that $\mathcal{V}_2(\mathcal{O})$ is *uncertain* as required.

(4) The value of $\mathcal{V}_1(\mathcal{O})$ is *inconsistent*. Since $\mathcal{V}_1(\mathcal{O}) \preceq \mathcal{V}_2(\mathcal{O})$, $\mathcal{V}_2(\mathcal{O})$ is *inconsistent* as required.

The result thus follows. □

Proposition 5.5 *Given verdict functions $\mathcal{V}_1$ and $\mathcal{V}_2$ we have that $\mathcal{V}_1 \uplus \mathcal{V}_2 \sqsupseteq \mathcal{V}_1$ and $\mathcal{V}_1 \uplus \mathcal{V}_2 \sqsupseteq \mathcal{V}_2$.*

Note that if $\mathcal{V}_2 \sqsupseteq \mathcal{V}_1$ and $\mathcal{V}_1$ is sound then it is not necessarily the case that $\mathcal{V}_2$ is sound. To see this consider any sound verdict function $\mathcal{V}_1$ and a verdict function $\mathcal{V}_2$ that maps all sets of observations to *inconsistent*. Clearly $\mathcal{V}_2 \sqsupseteq \mathcal{V}_1$ but $\mathcal{V}_2$ is not sound. Naturally, it is also possible to refine a monotonic verdict function to get a verdict function that is not monotonic.

6. REFINING FAULT DOMAINS

The work on test hypotheses has considered the notion of refining the hypothesis being used [Bernot et al. 91] and we investigate a similar notion here. A fault domain Φ_2 is a refinement of fault domain Φ_1 if $\Phi_2 \subseteq \Phi_1$. Refining a fault domain is equivalent to making stronger statements, or assumptions, about the SUT. For example, for an FSM M the fault domain $\Phi_M^{m_2}$ refines $\Phi_M^{m_1}$ if $m_2 \leq m_1$.

It is natural to ask whether we can reuse a verdict function for Φ_1 with Φ_2.

Proposition 6.1 *If fault domain Φ_2 is a refinement of Φ_1 and $\mathcal{V}$ is a sound verdict function with respect to Φ_1 then $\mathcal{V}$ is a sound verdict function with respect to Φ_2.*

PROOF. There are two cases to consider.

(1) $\emptyset \neq C(\mathcal{O}, \Phi_2) \cap conf(S)$. Since $\Phi_2 \subseteq \Phi_1$, $\emptyset \neq C(\mathcal{O}, \Phi_1) \cap conf(S)$ and so $\mathcal{V}(\mathcal{O}) \preceq$ *correct* as required.
(2) $\emptyset \neq C(\mathcal{O}, \Phi_2) \cap \overline{conf(S)}$. Since $\Phi_2 \subseteq \Phi_1$, $\emptyset \neq C(\mathcal{O}, \Phi_1) \cap \overline{conf(S)}$ and so $\mathcal{V}(\mathcal{O}) \preceq$ *incorrect* as required.

The result thus follows. □

So, we can reuse a sound verdict function for Φ_1 with Φ_2. However, the following shows that if $\mathcal{V}$ is precise for Φ_1 it need not be precise for Φ_2.

Proposition 6.2 *There exists fault domains Φ_1 and Φ_2 with $\Phi_2 \subseteq \Phi_1$ and a verdict function $\mathcal{V}$ that is precise for Φ_1 but is not precise for Φ_2*

PROOF. To see this, consider the FSM M_0, fault domains $\Phi_1 = \Phi_{M_0}^5$, $\Phi_2 = \Phi_{M_0}^4$ and a precise verdict function $\mathcal{V}$ for Φ_1. Now consider the set of traces $\mathcal{T} = \{\langle b/0, b/1, b/0\rangle\}$. It is straightforward to show that $\mathcal{V}(\mathcal{T})$ should be *uncertain* since there are conforming and faulty elements of $\Phi_{M_0}^5$ that allow this trace. However, we have seen that for fault domain $\Phi_{M_0}^4$ the verdict should be *incorrect* and so $\mathcal{V}$ is not precise for $\Phi_{M_0}^4$ as required. □

7. VERDICTS IN TESTING FROM AN FSM

In Section 3 we saw an example of an FSM M_0 and fault domain $\Phi_{M_0}^4$ where we had a trace $\langle b/1, b/0, b/1\rangle$ that is allowed by the specification M_0 but by no conforming implementation from $\Phi_{M_0}^4$. In this section we explore verdict functions for testing from an FSM M with fault domain Φ_M^m.

Given FSM M' and a finite set $\mathcal{T}$ of finite traces, we can decide whether $\mathcal{T} \subseteq L(M')$. Thus, since the fault domain Φ is finite, we can determine which elements of Φ are killed by a given set $\mathcal{T}$ of traces and so determine the verdict that should be returned. As a result, a precise computable verdict function can be defined. However, the size of the fault domain will usually make computing the verdict in this way infeasible.

Our fault domain limits the number of states of any implementation and insists that the SUT is deterministic. Let us suppose that $m_P(\mathcal{T})$ denotes the minimum number of states that a DFSM N that conforms to M can have if $\mathcal{T} \subseteq L(N)$ and that $m_F(\mathcal{T})$ denotes the minimum number of states that a DFSM N' that does not

conform to M can have if $\mathcal{T} \subseteq L(N')$. Thus, if we know that the SUT behaves like an element of Φ_M^m we get the following situations.

(1) If we have that the $m_P(\mathcal{T}) > m$ and $m_F(\mathcal{T}) \leq m$ then on the basis of $\mathcal{T}$ and Φ_M^m we can conclude that the SUT must be faulty and return the verdict *incorrect*.
(2) If we have the $m_P(\mathcal{T}) \leq m$ and $m_F(\mathcal{T}) > m$ then on the basis of $\mathcal{T}$ and Φ_M^m we can conclude that the SUT must be correct and return the verdict *correct*.
(3) If we have the $m_P(\mathcal{T}) \leq m$ and $m_F(\mathcal{T}) \leq m$ then the SUT could be correct or faulty and so we return the verdict *uncertain*.
(4) If we have the $m_P(\mathcal{T}) > m$ and $m_F(\mathcal{T}) > m$ then the SUT does not behave like an element of the fault domain Φ_M^m and so we return the verdict *inconsistent*.

We can reduce the problem of defining a verdict function to that of computing the values of m_P and m_F.

***Definition* 7.1** Let us suppose that lb_P is a function that takes the FSM M and a set $\mathcal{T}$ of traces and returns the minimum number of states a DFSM M' must have in order to both conform to M and have that $\mathcal{T} \subseteq L(M')$. Further, let us suppose that lb_F is a function that takes the FSM M and a set $\mathcal{T}$ of traces and returns the minimum number of states a DFSM M' must have in order to both fail to conform to M and have that $\mathcal{T} \subseteq L(M')$. Then the verdict function $\mathcal{V}_{lb_F}^{lb_P}$ is defined by:

$$\mathcal{V}_{lb_F}^{lb_P}(\mathcal{T}) = \begin{cases} \textit{incorrect} & \text{if } lb_F(M,\mathcal{T}) \leq m \wedge lb_P(M,\mathcal{T}) > m \\ \textit{correct} & \text{if } lb_F(M,\mathcal{T}) > m \wedge lb_P(M,\mathcal{T}) \leq m \\ \textit{uncertain} & \text{if } lb_F(M,\mathcal{T}) \leq m \wedge lb_P(M,\mathcal{T}) \leq m \\ \textit{inconsistent} & \text{otherwise} \end{cases}$$

Proposition 7.2 *The verdict function $\mathcal{V}_{lb_F}^{lb_P}$ is precise.*

Proof. From Definitions 4.8 and 7.1, $\mathcal{V}_{lb_F}^{lb_P}$ is precise. □

Thus, if we can define functions lb_P and lp_F then we can use the corresponding verdict function. Unfortunately, the problem of defining $lb_P(M,\mathcal{T})$ is NP-hard.

Proposition 7.3 *The problem of computing $lb_P(M,\mathcal{T})$ is NP-hard.*

Proof. Consider the special case in which M is the chaos machine M_C that has one state s_0 and for every input $x \in X$ and output $y \in Y$ there is a transition from s_0 to s_0 with input x and output y. Thus, every DFSM with input alphabet X and output alphabet Y conforms to M. If we can compute $lb_P(M_C,\mathcal{T})$ in polynomial time then we can also find the number of states of a smallest DFSM M' such that $\mathcal{T} \subseteq L(M')$ in polynomial time. However, this problem is known to be NP-hard [Gold 1978] and so the result follows. □

While a verdict function could be based on lb_F and lb_P, it seems likely that this approach will not scale. This suggests that we should look for additional verdict functions that can be computed efficiently. We might base an alternative verdict function on functions lb'_F and lb'_P that approximate lb_F and lb_P respectively. Let

us suppose that for some approximation lb'_P we have at least one case $(M, \mathcal{T})$ in which $lb'_P(M, \mathcal{T}) > lb_P(M, \mathcal{T})$. Using this might lead to us incorrectly eliminating elements of Φ and in doing so incorrectly suggest that an SUT is faulty; such a verdict function is not sound and so must be rejected. Thus, any approximation for lb_P should be an under approximation: the value returned is always a lower bound on the number of states of a conforming SUT but it might not be the greatest lower bound. Similarly, any approximation to lb_F could be an under approximation but should not be an approximation that is above lb_F since it must not eliminate elements of Φ that are consistent with $\mathcal{T}$.

Proposition 7.4 *Let us suppose that functions lb'_F and lb'_P have the property that for all M, $\mathcal{T}$ we have that $lb_P(M, \mathcal{T}) \geq lb'_P(M, \mathcal{T})$ and $lb_F(M, \mathcal{T}) \geq lb'_F(M, \mathcal{T})$. Then the verdict function $\mathcal{V}_{lb'_F}^{lb'_P}$ is sound and we have that $\mathcal{V}_{lb_F}^{lb_P} \sqsupseteq \mathcal{V}_{lb'_F}^{lb'_P}$.*

Proof. This follows from Definitions 4.6 and 5.1. □

There is thus the challenge to produce good approximation functions lb'_F and lb'_P. Note that approaches to state counting can be seen as producing a function lb'_F that takes a test suite and returns a lower bound that holds for *all* possible implementations that conform to the specification on that test suite (see for example [Yevtushenko and Petrenko 1990; Petrenko et al. 1996; Petrenko and Yevtushenko 2005]). Adaptive state counting extends this by using the observed traces and so is based on one possible approximation lb'_F [Hierons 2004]. However, there appears to be no work on producing an approximation lb'_P.

8. CONCLUSIONS

This paper has explored the concept of test verdicts when there is a fault domain or test hypotheses and has identified the need for new test verdicts in this situation. In addition, we have shown that it is possible to deduce properties of the system under test on the basis of a set $\mathcal{O}$ of observed behaviours in situations in which it is not possible to deduce such properties from a single element of $\mathcal{O}$. This has led us to propose the use of verdict functions that take a set of observations and return a verdict.

This paper has identified some important properties that verdict functions should have and some desirable properties of verdict functions. For example, it is vital that a verdict function is sound: it cannot return a verdict that is not a consequence of the set of observed behaviours. Further, we expect verdict functions to be monotonic and ideally they are precise: whenever we can deduce a properties of the system under test that corresponds to a verdict then this verdict is returned. We have also defined what it means to refine a verdict function or a fault domain and have shown how verdict functions can be combined.

Verdict functions have been explored in the context of testing from a nondeterministic finite state machine. It transpires that it is possible to define a computable precise verdict function for the standard fault domain, which places an upper bound on the number of states of the system under test. Unfortunately, however, the problem of computing this verdict function is NP-hard and thus there

remains the problem of finding good approximations that can be computed efficiently.

ACKNOWLEDGMENTS

I would like to thank Nina Yevtushenko for interesting discussions that inspired this work. I would also like to thank the anonymous referees for their many valuable comments, which strengthened the paper.

REFERENCES

Bernot, G., Gaudel, M.-C., and Marre, B. 91. Software testing based on formal specification: a theory and a tool. *Software Engineering Journal 6*, 387–405.

Bidoit, M. and Mosses, P. 2003. *CASL User Manual: Introduction to Using the Common Algebraic Specification Language*. Lecture Notes in Computer Science, vol. 2900. Springer-Verlag.

Bouge, L., Choquet, N., Fibourg, L., and Gaudel, M.-C. 1986. Test sets generation from algebraic specifications using logic programming. *Journal of Systems and Software 6,* 4, 343–360.

Chow, T. S. 1978. Testing software design modelled by finite state machines. *IEEE Transactions on Software Engineering 4*, 178–187.

ETSI ES 201 873-1 V3.1.1. 2005. *Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language*. ETSI, Sophia, Antipolis, France.

Gaudel, M. C. 1995. Testing can be formal too. In *6th International Joint Conference CAAP/FASE Theory and Practice of Software Development (TAPSOFT'95)*. Lecture Notes in Computer Science, vol. 915. Springer, 82–96.

Goguen, J. A. and Malcolm, G. 2000. *Software Engineering with OBJ: Algebraic Specifications in Action*. Kluwer Academic Publishers.

Goguen, J. A. and Tardo, J. J. 1979. An introduction to OBJ: a language for writing and testing formal algebraic specifications. In *The IEEE Conference on Specifications of Reliable Software*. 170–189.

Gold, E. M. 1978. Complexity of Automaton Identification from Given Data. *Information and Control 37*, 302–320.

Hennie, F. C. 1964. Fault-detecting experiments for sequential circuits. In *Proceedings of 5th Annual Symposium on Switching Circuit Theory and Logical Design*. Princeton, New Jersey, 95–110.

Hierons, R. M. 1998. Adaptive testing of a deterministic implementation against a nondetermistic finite state machine. *The Computer Journal 41,* 5, 349–355.

Hierons, R. M. 2004. Testing from a non-deterministic finite state machine using adaptive state counting. *IEEE Transactions on Computers 53,* 10, 1330–1342.

Hierons, R. M. and Ural, H. 2006. Optimizing the length of checking sequences. *IEEE Transactions on Computers 55,* 5, 618–629.

Inan, K. and Ural, H. 1999. Efficient checking sequences for testing finite state machines. *Information and Software Technology 41,* 11–12, 799–812.

Luo, G., Petrenko, A., and v. Bochmann, G. 1994. Selecting test sequences for partially-specified nondeterministic finite state machines. In *The 7th IFIP Workshop on Protocol Test Systems*. Chapman and Hall, Tokyo, Japan, 95–110.

Luo, G. L., v. Bochmann, G., and Petrenko, A. 1994. Test selection based on communicating nondeterministic finite-state machines using a generalized Wp-method. *IEEE Transactions on Software Engineering 20,* 2, 149–161.

Moore, E. P. 1956. Gedanken-experiments. In *Automata Studies*, C. Shannon and J. McCarthy, Eds. Princeton University Press.

Mosses, P. D. 2004. *CASL Reference Manual: The Complete Documentation of the Common Algebraic Specification Language*. Lecture Notes in Computer Science, vol. 2960. Springer-Verlag.

Petrenko, A. and Yevtushenko, N. 2005. Testing from partial deterministic FSM specifications. *IEEE Transactions on Computers 54,* 9, 1154–1165.

Petrenko, A., Yevtushenko, N., Lebedev, A., and Das, A. 1994. Nondeterministic state machines in protocol conformance testing. In *Proceedings of Protocol Test Systems, VI (C-19)*. Elsevier Science (North-Holland), Pau, France, 363–378.

Petrenko, A., Yevtushenko, N., and v. Bochmann, G. 1996. Testing deterministic implementations from nondeterministic FSM specifications. In *9th IFIP TC6/WG6.1 International Workshop on Testing of Communicating Systems*. Chapman and Hall, Darmstadt, Germany, 125–141.

Pickin, S., Jard, C., Jeron, T., Jezequel, J.-M., and Traon, Y. L. 2007. Test synthesis from UML models of distributed software. *IEEE Transactions on Software Engineering 33,* 4, 252–268.

Rezaki, A. and Ural, H. 1995. Construction of checking sequences based on characterization sets. *Computer Communications 18,* 12, 911–920.

Rodríguez, I., Merayo, M. G., and Núñez, M. 2006. A logic for assessing sets of heterogeneous testing hypotheses. In *18th IFIP TC6/WG6.1 International Conference on Testing of Communicating Systems (TestCom 2006)*. Lecture Notes in Computer Science, vol. 3964. Springer, 39–54.

Ural, H., Wu, X., and Zhang, F. 1997. On minimizing the lengths of checking sequences. *IEEE Transactions on Computers 46,* 1, 93–99.

Yevtushenko, N. and Petrenko, A. 1990. Synthesis of test experiments in some classes of automata. *Automatic Control and Computer Sciences 4*.

Yevtushenko, N. V., Lebedev, A. V., and Petrenko, A. F. 1991. On checking experiments with nondeterministic automata. *Automatic Control and Computer Sciences 6*, 81–85.

Zander, J., Dai, Z. R., Schieferdecker, I., and Din, G. 2005. From U2TP models to executable tests with TTCN-3 - an approach to model driven testing. In *17th IFIP TC6/WG 6.1 International Conference on Testing of Communicating Systems (TestCom 2005)*. Lecture Notes in Computer Science, vol. 3502. Springer, 289–303.

Received Month Year; revised Month Year; accepted Month Year