Comparing test sets and criteria in the presence of test hypotheses and fault domains

R. M. Hierons
Deparment of Information Systems and Computing,
Brunel University, UK

A number of authors have considered the problem of comparing test sets and criteria. Ideally test sets are compared using a preorder with the property that test set T_1 is at least as strong as T_2 if whenever T_2 determines that an implementation p is faulty, T_1 will also determine that p is faulty. This notion can be extended to test criteria. However, it has been noted that very few test sets and criteria are comparable under such an ordering; instead orderings are based on weaker properties such as subsumes. This paper explores an alternative approach, in which comparisons are made in the presence of a test hypothesis or fault domain. This approach allows strong statements about fault detecting ability to be made and yet for a number of test sets and criteria to be comparable. It may also drive incremental test generation.

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

1. INTRODUCTION

Testing forms an important, but expensive, part of the software verification process. In testing, the tester executes the *implementation under test (IUT)* on certain input and observes the output produced. The output observed is then compared with that expected. Testing thus involves exploring the behaviour of the IUT in order to determine whether it conforms to the specification.

In general, it is not feasible to use the entire input space of the IUT during testing; instead the tester chooses a subset of this. Each value chosen is a *test input* and the subset used is called a *test set*. The choice of test set is often based on some *test criterion* that states what it means for a test set to be sufficient. While there are many test generation techniques, based around alternative criteria, there are relatively few results that allow the tester to compare test sets and techniques. This makes it difficult for the tester to choose between different test sets and criteria.

It has been argued that test generation can be made more systematic through the use of a *test hypothesis* that represents properties the tester believes the IUT has. For example, the uniformity hypothesis is used in partition analysis: here it

R. M. Hierons, Department of Information Systems and Computing, Brunel University, Uxbridge, Middlesex, UB8 3PH, United Kingdom.
Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.
© 20YY ACM 0000-0000/20YY/0000-0001 $5.00

is assumed that for a given subdomain D_i of the input domain, either all values in D_i lead to a failure or no value from D_i leads to a failure. If we were testing an implementation p, that is intended to apply the absolute value function to integers, we might choose one subdomain D_1 of positive integers. The uniformity hypothesis with D_1 would state that either p is correct on all positive integers or it fails on all positive integers. The test hypothesis might reflect expert knowledge or be based on information derived using program analysis. Further, it may be possible to prove that the test hypothesis holds. Given test hypothesis H, it might be possible to devise a finite test set T with the property that, as long as the IUT I satisfies H, T determines the correctness of I [Bernot et al. 91; Gaudel 1995]. Thus the existence of test hypotheses allows stronger statements to be made about test effectiveness. Of course, the existence of a test hypothesis H will not always lead to a feasible test set that determines correctness under H and it may not always be possible to prove that the IUT satisfies the hypothesis H.

A fault domain is a set F of (functional) behaviours with the property that the tester believes that the functional behaviour of the IUT is equivalent to some (unknown) element of F [ITU-T 1997]. Where there is some such fault domain F it may be possible to produce a test that determines correctness under the assumption that if the IUT is faulty then it behaves like some element of F. Clearly, fault domains and test hypotheses are related concepts.

Naturally there are a number of practical issues related to the use of test hypotheses or fault domains. These issues include the problem of choosing the assumptions to be made and the problem of generating tests on the basis of such assumptions. These issues have been considered in the areas of protocol conformance testing [ITU-T 1997], hardware testing, and testing from algebraic specifications [Bernot et al. 91; Gaudel 1995].

This paper explores the problem of comparing the effectiveness of alternative test sets and criteria when testing a deterministic implementation. We make comparisons in the context of test hypotheses and fault domains, assuming that there is some test hypothesis H or fault domain F that represents properties that the tester believes the IUT has. We show that under such assumptions it is possible to make much stronger statements about the relative fault detecting ability of test sets and criteria. This is the case even when the tests considered do not determine correctness under the assumptions made. It transpires that, given a test hypothesis or fault domain and test set T, it may be possible to extend T without increasing its fault detecting ability. In fact, a large test set may have the same fault detecting ability as the empty set. This illustrates the importance of considering the available information, regarding the IUT, during test generation.

It is argued that the relation introduced may be used to drive incremental test generation. Specifically, a test set T should only be extended to test set T' if T' is strictly stronger than T under the assumptions made.

The paper is structured as follows. Section 2 describes test criteria, test hypotheses and related work. The relation used to compare test sets and criteria is defined and explored in Section 3. Section 4 then shows how this relation may drive incremental test development. A number of types of test sets and criteria are considered in Section 5. Section 6 discusses possible future work and finally, in

Section 7 conclusions are drawn.

2. PRELIMINARIES

2.1 Testing and test criteria

Throughout this paper testing will refer to the process of executing the IUT with input and observing the output produced. $\mathcal{X}$ will denote the input domain of the specifications and programs considered, $\mathcal{Y}$ will denote the output domain, $\mathcal{S}$ will denote the set of specifications, and $\mathcal{P}$ will denote the set of implementations. $P(\mathcal{X})$ will denote the set of test sets, which is the power set of $\mathcal{X}$. For all $p \in \mathcal{P}$, $s \in \mathcal{S}$, $p \preceq s$ will denote p conforming to s. It will be assumed that $\mathcal{P}$ contains representatives of the set of computable functions from $\mathcal{X}$ to $\mathcal{Y}$. It will also be assumed that every specification $s \in \mathcal{S}$ is feasible: there is some program that conforms to s. When reasoning about testing, the following definitions (which are based on those in [Gourlay 1983]) will be of use.

DEFINITION 1. *Given $s \in \mathcal{S}$, $p \in \mathcal{P}$, and $T \in P(\mathcal{X})$,* p conforms to s on T *if and only if for all $t \in T$ the behaviour of p on t is consistent with s. This is written $p \preceq_T s$. Similarly,* p fails on T *if for some $t \in T$, the behaviour of p on t is not consistent with s. This is written $p \not\preceq_T s$.*

From these definitions it is possible to make the following observation.

PROPOSITION 1. *For all $p \in \mathcal{P}$ and $s \in \mathcal{S}$, $p \preceq s$ if and only if $p \preceq_{\mathcal{X}} s$.*

It is important to have some way of determining when to stop testing. While this might be based on limits on budget or time, ideally it is based on some notion of what it means for a test set to be sufficient. In turn, such notions are usually described in terms of *test criteria*. A test criterion is some property, that might depend upon the code and the specification, that states whether testing is sufficient. For example, one (relatively weak) test criterion is that, in testing, every reachable statement of the IUT is executed.

A test criterion C is a predicate that takes a specification, a program and a test set and returns a boolean. Under C, test set T is sufficient for $p \in \mathcal{P}$ and $s \in \mathcal{S}$ if and only if $C(s, p, T)$ evaluates to true. Given specification s and program p, test criterion C may be partially evaluated to give $C(s, p)$ defined by: $C(s, p)(T) \Leftrightarrow C(s, p, T)$. Naturally, there are a number of properties that test criteria should possess [Weyuker 1986; 1988; Zhu and Hall 1993]. Test criteria that depend upon the program p usually consider the structure of p, insisting that certain constructs of p are executed in testing [Ntafos 1988].

Given a test criterion C, specification s, and program p, test generation can be seen as a process of producing some test set T that satisfies $C(s, p)$. Thus, rather than compare the effectiveness of test generation techniques it is normal to compare test criteria.

2.2 Previous approaches to comparing test sets and criteria

This section provides a brief overview of some approaches to comparing test sets and criteria. For a more detailed survey see [Weyuker 2002].

The literature contains many test criteria and there has thus been much interest in comparing the effectiveness of the corresponding tests. A natural way of comparing

two criteria C_1 and C_2 is to say that C_1 is at least as strong as C_2 if and only if whenever the use of C_2 is capable of determining that the IUT p is faulty, the use of C_1 is guaranteed to determine that p is faulty. This corresponds to Hamlet's test comparison relation [Hamlet 1989], which may be defined in the following way.

DEFINITION 2. *$C_2 \leq C_1$ if and only if for every $s \in \mathcal{S}, p \in \mathcal{P}$, if there exists $T_2 \in P(\mathcal{X})$ such that $C_2(s, p, T_2)$ is true and $p \not\preceq_{T_2} s$ then for every $T_1 \in P(\mathcal{X})$, $C_1(s, p, T_1) \Rightarrow p \not\preceq_{T_1} s$.*

This notion relates directly to the ability of the criterion to determine correctness and thus of the ability of the criterion to lead to a faulty implementation being identified. Unfortunately, while this might be a property of interest to the tester, very few real test criteria are comparable under $\leq$. This is because there is usually a wide range of test sets that satisfy a test criterion, some of which are capable of finding a particular fault while others miss this fault. In particular, assuming each test criterion C is monotonic ($\forall T, T' \in P(\mathcal{X}).C(s, p, T) \wedge T \subseteq T' \Rightarrow C(s, p, T')$), we have the following property: given test criterion C, $s \in \mathcal{S}$, and $p \in \mathcal{P}$, if p is faulty then there is some test set $T \in P(\mathcal{X})$ such that $C(s, p, T)$ and $p \not\preceq_T s$. This leads to the following observation made by Hamlet [Hamlet 1989].

PROPOSITION 2. *No two effective, monotonic test methods are comparable under $\leq$.*

The relation $\leq$ may be extended to test sets in the following way.

DEFINITION 3. *$T_2 \leq T_1$ if and only if for all $s \in \mathcal{S}$ and $p \in \mathcal{P}$ if $p \not\preceq_{T_2} s$ then $p \not\preceq_{T_1} s$.*

However, assuming $\mathcal{P}$ contains representatives of every computable function from $\mathcal{X}$ to $\mathcal{Y}$, then given $s \in \mathcal{S}$ and $x \in \mathcal{X}$ there is some $p_x \in \mathcal{P}$ such that $p_x \not\preceq_{\{x\}} s$ and $p_x \preceq_{\mathcal{X} \setminus \{x\}} s$. This leads to the following observation.

PROPOSITION 3. *Assuming $\mathcal{P}$ contains representatives of every computable function from $\mathcal{X}$ to $\mathcal{Y}$, $T_2 \leq T_1$ if and only if $T_2 \subseteq T_1$.*

An alternative way of comparing test criteria is to determine whether one criterion *subsumes* another, where the subsumes relation is defined as follows (see, for example, [Zhu 1996]).

DEFINITION 4. *Criterion C_1* subsumes *C_2 if and only if for all $s \in \mathcal{S}, p \in \mathcal{P}$, whenever a test set $T \in P(\mathcal{X})$ satisfies $C_1(s, p)$ it also satisfies $C_2(s, p)$.*

Thus, if C_1 subsumes C_2 and a test set T satisfies $C_1(s, p)$, T is guaranteed to satisfy $C_2(s, p)$. This suggests that the tester need not consider C_2 if C_1 is being used.

Many test criteria are comparable under the subsumes relation [Ntafos 1988]. However, C_1 subsuming C_2 does not guarantee that faulty implementations that can be identified using C_2 will be identified when using C_1. It is also possible that while C_1 subsumes C_2 there are corresponding test generation techniques TG_1 and TG_2 such that TG_2 is more likely to lead to fault detection than TG_1. See [Hamlet 1989] for a critique of the subsumes relation.

Alternative approaches to comparing test criteria are based upon simulations [Chen and Yu 1996; Duran and Ntafos 1984; Hamlet and Taylor 1990; Hierons and Wiper 1997; Weyuker et al. 1991]. Probabilistic arguments have also been successfully applied: these consider the probability of finding at least one failure or the expected number of failures found [Frankl and Weyuker 1993a; 1993b]. These studies provide extremely useful information about the general effectiveness of test techniques and criteria. However, one criterion C_1 being stronger than another criterion C_2, under these relations, does not mean that an implementation that can be found to be faulty under C_2 is guaranteed to be found to be faulty under C_1. The issue, of how test hypotheses and fault domains might affect probabilistic comparisons, is a topic for future work.

This paper describes a new approach to comparing the effectiveness of test sets and test criteria. This approach uses test hypotheses to produce a relation that has the advantages associated with $\leq$ but is applicable to more test sets and criteria than $\leq$.

2.3 Test hypotheses and Fault domains

A *test hypothesis* H is some property the tester believes the IUT has. Given a test hypothesis H and specification s there may be some finite test set T such that T determines correctness under H [Bernot et al. 91; Gaudel 1995]. This is defined more formally by the following.

DEFINITION 5. *Test set* $T \in P(\mathcal{X})$ determines conformance to $s \in \mathcal{S}$ under test hypothesis H *if and only if for all* $p \in \mathcal{P}.H(p) \wedge p \preceq_T s \Rightarrow p \preceq s$.

A classic example of a test hypothesis is the uniformity hypothesis. Here there is some subset $\mathcal{X}_1$ of the input domain $\mathcal{X}$ and it is assumed that all test input in $\mathcal{X}_1$ are equivalent: if the input of some value in $\mathcal{X}_1$ leads to a failure then all values in $\mathcal{X}_1$ lead to failures.

Given specification s, test generation can be seen as a process of refining the test hypothesis until there is some hypothesis H for which there is a feasible test set T such that T determines conformance to s under H. This process starts with some minimal hypothesis H_{min} that usually simply states the input and output domains of the IUT. Here the minimal hypothesis will also restrict the implementation to being deterministic.

Naturally, the tester might not always have enough information to produce a test hypothesis H in which they can have confidence and which leads to a feasible test set that determines correctness under H. However, we will see that even when this is the case, test hypotheses can play a role in allowing test sets and criteria to be compared and in driving incremental test generation.

While a test hypothesis represents properties the IUT is believed to have, a *fault domain* represents a possible set of functional behaviours for the implementation. Thus, a fault domain is a set $\mathcal{B}$ of behaviours with the property that it is believed that the IUT behaves like some (unknown) element of $\mathcal{B}$. Naturally, associated with $\mathcal{B}$ is a (generally uncomputable) set $\mathcal{P}_\mathcal{F} \subseteq \mathcal{P}$: $\mathcal{P}_\mathcal{F}$ is the set of elements of $\mathcal{P}$ that are functionally equivalent to elements of $\mathcal{B}$.

Fault domains are widely used in a number of areas, including protocol conformance testing [ITU-T 1997] and hardware testing. Given a fault domain $\mathcal{B}$ there

may be some test that determines correctness under the assumption that the IUT behaves like some unknown element of $\mathcal{B}$. Naturally, fault domains and test hypotheses are related.

(1) Given fault domain $\mathcal{B}$ there is the corresponding test hypothesis that the IUT p is functionally equivalent to some (unknown) element of $\mathcal{B}$.
(2) Given test hypothesis H there is the corresponding fault domain consisting of the behaviours of the elements of $\mathcal{P}$ that satisfy the test hypothesis.

Interestingly, the argument regarding the equivalence of test hypotheses and fault domains only holds for black-box techniques. It will transpire, in Section 5, that fault domains are less applicable when considering a particular white-box test technique: mutation testing.

While the results in this paper are phrased in terms of test hypotheses, they also apply to fault domains.

3. COMPARING TESTS: A NEW APPROACH

This section introduces a new way of comparing test sets and criteria. This is based on the existence of a test hypothesis: given test hypothesis H, two test sets or test criteria are related by a preorder $\leq_H$ that is defined in terms of H. Properties of $\leq_H$ are then explored. Section 4 describes how $\leq_H$ may be used to drive incremental test generation. Throughout this section it will be assumed that a specification $s \in \mathcal{S}$ has been given and thus that H may refer to s. For example, H might state that there are structural similarities between the specification and the implementation. Naturally, the results and discussion contained in this section transfer immediately to fault domains.

The essential observation is that the presence of a test hypothesis H represents information about the class of faults that might occur. This might lead to relationships between test sets or criteria that do not hold in general. Section 5 will show how $\leq_H$ may be applied to some existing test criteria and corresponding test sets.

3.1 Comparing test sets

In this section we will say what it means for one test set to be at least as strong as another in the presence of a test hypothesis H and then explore a number of properties of this relation. Essentially, one test set T_1 will be said to be at least as strong as another test set T_2 in the presence of H if and only if for every erroneous implementation $p \in \mathcal{P}$ that satisfies H, if p fails T_2 then p fails T_1. This is defined by the following.

DEFINITION 6. *Given $T_1, T_2 \in P(\mathcal{X})$, $T_2 \leq_H T_1$ if and only if for all $p \in \mathcal{P}$, $H(p) \wedge p \not\preceq_{T_2} s \Rightarrow p \not\preceq_{T_1} s$. Where $T_2 \leq_H T_1$ we say that T_1 is* at least as strong as *T_2 under H. Where this is not the case we write $T_2 \not\leq_H T_1$.*

Note that this definition relates to the ability of a test set to detect one or more failures. Naturally, the failures found by different test sets may refer to different faults.

PROPOSITION 4. *$\leq_H$ is a preorder on the elements of $P(\mathcal{X})$.*

PROOF. By definition, $\leq_H$ is a preorder if and only if the following hold.

(1) For every test set $T \in P(\mathcal{X})$, $T \leq_H T$.
(2) Given $T_1, T_2, T_3 \in P(\mathcal{X})$, if $T_3 \leq_H T_2$ and $T_2 \leq_H T_1$ then $T_3 \leq_H T_1$.

The first property holds immediately. Now consider the second property and assume that test sets T_1, T_2, and T_3 are given such that $T_3 \leq_H T_2$ and $T_2 \leq_H T_1$. Suppose also that $p \in \mathcal{P}$, $H(p)$ is true and $p \not\simeq_{T_3} s$. It is sufficient to prove that $p \not\simeq_{T_1} s$. By the definition of $\leq_H$ we know that, since $p \not\simeq_{T_3} s$ and $T_3 \leq_H T_2$, $p \not\simeq_{T_2} s$. By the definition of $\leq_H$, since $T_2 \leq_H T_1$, $p \not\simeq_{T_1} s$. Thus $T_3 \leq_H T_1$ as required. □

It is worth noting that Bernot et al. [Bernot et al. 91] describe a more general notion of refinement in which they refine a triple consisting of a test hypothesis; a test set; and a test oracle. This is described within the context of testing from algebraic specifications.

The following result, which places a lower bound on the relation $\leq_H$, follows immediately from the definition of $\leq_H$.

PROPOSITION 5. *Given test hypothesis H and $T_1, T_2 \in P(\mathcal{X})$, $T_2 \subseteq T_1 \Rightarrow T_2 \leq_H T_1$.*

It is also possible to say something about when this lower bound may be met. Under the minimal hypothesis H_{min}, since we know nothing about the faults, two test sets are related if and only if one is contained within the other. This corresponds to Proposition 3 regarding the extension of $\leq$ to test sets.

PROPOSITION 6. *For all $T_1, T_2 \in P(\mathcal{X})$, $T_2 \leq_{H_{min}} T_1 \Leftrightarrow T_2 \subseteq T_1$.*

While $T_2 \subseteq T_1 \Rightarrow T_2 \leq_H T_1$, we will see that there may be test sets that are comparable under $\leq_H$ but not under $\subseteq$.

The following observations may be made.

PROPOSITION 7. *For every test set $T \in P(\mathcal{X})$ and hypothesis H*

(1) $T \leq_H \mathcal{X}$
(2) $\emptyset \leq_H T$.

The first of these states that given hypothesis H and test set T, the entire input domain $\mathcal{X}$ is always at least as strong as T. The second observation states that given hypothesis H and test set T, T is at least as strong as the empty set. Naturally both hold for $\leq$ as well as $\leq_H$. The following says that if (under H) a test set is as least as strong as all other test sets then this test set determines correctness under H.

PROPOSITION 8. *A test set $T \in P(\mathcal{X})$ determines correctness under H if and only if $\forall T_2 \in P(\mathcal{X}).T_2 \leq_H T$.*

PROOF. We will initially prove the $\Rightarrow$ part. Suppose T determines correctness under H and thus, if T is applied to a faulty implementation p from $\mathcal{P}$ that satisfies H, p must fail on T. It follows immediately from the definition of $\leq_H$ that for all $T_2 \in P(\mathcal{X}).T_2 \leq_H T$.

We will now consider the $\Leftarrow$ direction. Proof by contradiction will be used: suppose T does not determine correctness under H. Then there exists $p \in \mathcal{P}$ such

that p satisfies H, $p \preceq_T s$ but p does not conform to s. Since p does not conform to s there is some test set T_2 that can find this fault (otherwise the fault cannot be exhibited, in which case it is not a fault). But $T_2 \leq_H T$ and thus, by definition, p does not conform to s on T. This provides a contradiction as required. □

It is natural to say that two test sets are equivalent under a test hypothesis H if each is related to the other under $\leq_H$.

DEFINITION 7. *$T_1 \equiv_H T_2$ if and only if $T_1 \leq_H T_2$ and $T_2 \leq_H T_1$.*

It is clear that $\equiv_H$ is an equivalence relation. Let H_{corr} denote the test hypothesis that states that the IUT is correct. All test sets are equivalent under this.

PROPOSITION 9. *For all $T_1, T_2 \in P(\mathcal{X})$, $T_1 \equiv_{H_{corr}} T_2$.*

Interestingly a non-empty test set may be equivalent, under some test hypothesis H, to the empty set.

PROPOSITION 10. *There exist test hypothesis H and non-empty test set $T \in P(\mathcal{X})$ such that $T \equiv_H \emptyset$.*

PROOF. This follows immediately from Proposition 9. □

This result holds for a wider range of hypotheses than H_{corr}.

PROPOSITION 11. *There exist a test hypothesis $H \neq H_{corr}$ and non-empty test set $T \in P(\mathcal{X})$ such that $T \equiv_H \emptyset$.*

PROOF. We will construct such H and T. Initially we choose some subset $X \subset \mathcal{X}$ and use the test hypothesis H that states that the program is correct on all values in $\mathcal{X} \setminus X$. Then, under H any test set that does not contain a value from X is equivalent to the empty set. □

These results demonstrate why it is important to consider properties of the IUT in test generation: otherwise the tester might produce a test set that is not capable of finding a fault. Further, the tester might extend a test set T with input that does not increase the fault detecting ability of T. Naturally, there are practical issues regarding the generation of test input that strengthens a test set. In some cases, such as where the uniformity hypothesis is applied based on the specification, it may be relatively straightforward to find such tests. In other cases it may be extremely difficult.

It is possible to define what it means for one test set to be stronger than another under H.

DEFINITION 8. *Test set T_1 is* stronger than *test set T_2 if and only if $T_2 \leq_H T_1$ and $T_1 \not\leq_H T_2$. This is denoted $T_2 <_H T_1$.*

In Section 4 we will see that this notion is useful in test generation: it only makes sense to extend a test set T_2 to a test set T_1 if $T_2 <_H T_1$.

3.2 Comparing test criteria

This section extends the definitions given in Section 3.1 to allow test criteria to be compared. One approach might be to say that one criterion C_1 is at least as strong as another C_2 (denoted $C_2 \leq_H^1 C_1$) if and only if we have that for every $p \in \mathcal{P}$

that satisfies H, if some test set that satisfies C_2 determines that p is faulty, all test sets that satisfy C_1 determine that p is faulty. There is, however, an issue to consider: most test criteria will allow exhaustive testing and thus can allow any fault to be detected. This means that, under the definition of $\leq_H^1$ given above, for this relationship to hold either C_1 must be guaranteed to determine correctness under H or C_2 must preclude certain test cases being used.

A solution to this problem is, when comparing criteria C_1 and C_2, to compare *non-reducible* test sets that satisfy these criteria. A test set $T \in P(\mathcal{X})$ is *non-reducible* for criterion C_1, program p and specification s if $C(s, p, T)$ is true and for every proper subset $T_2 \subset T$, $C(s, p, T_2)$ is false. Naturally, non-reducible test sets are not unique and a non-reducible test set need not be the smallest test set that satisfies the criterion. Non-reducible tests sets are considered because testers often aim to produce non-reducible test sets and thus this comparison relates to the likely use of test criteria. Note that this paper is concerned with test effectiveness rather than test efficiency and thus, when making comparisons, there is no need to normalise the sizes of the test sets.

The notion, of one criterion being at least as strong as another in the presence of a test hypothesis H, may be represented by a relation $\leq_H$ which is defined by the following.

DEFINITION 9. *$C_2 \leq_H C_1$ if and only if for every $p \in \mathcal{P}$ such that $H(p)$ is true, if there is some non-reducible test set $T_2 \in P(\mathcal{X})$ that satisfies $C_2(s, p)$ such that $p \not\preceq_{T_2} s$ then for every test set T_1 that satisfies $C_1(s, p)$, $p \not\preceq_{T_1} s$.*

It is vital that the test sets satisfying C_2 are non-reducible: otherwise, assuming that C_2 is monotonic, we need to consider the test set $\mathcal{X}$ and this is guaranteed to find any failure. Thus, without the restriction to non-reducible test sets, monotonic test criteria could not be comparable under $\leq_H$.

PROPOSITION 12. *$\leq_H$ is a preorder on the set of test criteria.*

PROPOSITION 13. *Given test criterion C, the following are equivalent:*

(1) for every test criterion C_2, $C_2 \leq_H C$

(2) C determines correctness under H.

PROOF. The first case: 1) $\Rightarrow$ 2). Proof by contradiction: assume that C does not determine correctness under H. There exists $p \in \mathcal{P}$ such that $H(p)$, p does not conform to s, and there is a non-reducible test set $T_1 \in P(\mathcal{X})$ that satisfies $C(s, p)$ such that $p \preceq_{T_1} s$. Since p does not conform to s there is some input $t \in \mathcal{X}$ such that $p \not\preceq_{\{t\}} s$. Choose the test criterion C' such that $C'(s, p, T)$ is true if and only if $t \in T$. Any non-reducible test satisfying C' determines that p does not conform to s. Thus, since $C' \leq_H C$, any test that satisfies $C(s, p)$ must also determine that p does not conform to s, providing a contradiction as required.

The second case, 2) $\Rightarrow$ 1), follows immediately. □

We also have the notion of equivalence as before.

DEFINITION 10. *$C_1 \equiv_H C_2$ if and only if $C_1 \leq_H C_2$ and $C_2 \leq_H C_1$.*

All test criteria are equivalent under H_{corr}.

PROPOSITION 14. *For all test criteria C_1, C_2, $C_1 \equiv_{H_{corr}} C_2$.*

Let $C_\emptyset$ denote the test criterion that allows any test set, even one that is empty. Clearly there is only one non-reducible test set under $C_\emptyset$: the empty set. A test criterion, that does not lead to empty test sets, may be equivalent, under a test hypothesis H, to $C_\emptyset$.

PROPOSITION 15. *There exist test hypothesis H and criterion C, with the property that $\forall s \in \mathcal{S}$, $p \in \mathcal{P}.\neg C(s, p, \emptyset)$, such that $C \equiv_H C_\emptyset$.*

PROOF. By choosing $H = H_{corr}$, this follows directly from Proposition 14. □

It is clear that $\leq_H$ and $\leq$ are related: they become equivalent when all elements of $\mathcal{P}$ satisfy H. Assuming an appropriate choice of $\mathcal{P}$, this is equivalent to H being the minimal hypothesis. This is stated more formally in the following.

PROPOSITION 16. *Assuming each element in $\mathcal{P}$ satisfies the minimal hypothesis H_{min}, $C_2 \leq C_1$ if and only if $C_2 \leq_{H_{min}} C_1$.*

4. INCREMENTAL TEST DEVELOPMENT

This section explores the role test hypotheses may play in the development of test sets. Again it will be assumed that the specification $s \in \mathcal{S}$ is given. We will show that even when it is not feasible to produce a test set that determines conformance to s under the test hypothesis H being used, the existence of H can assist test generation. It is worth noting that the notion of refinement used in this section is similar to that discussed in Bernot et al. [Bernot et al. 91] within the context of testing from algebraic specifications. However, Bernot et al. do not consider test criteria and do not consider incremental test development.

The presence of hypothesis H can assist incremental test generation in the following way: under H it is only worth considering incrementing test set $T \in P(\mathcal{X})$ to $T' \in P(\mathcal{X})$ $(T \subset T')$ if $T <_H T'$. This is because, it is known that $T \leq_H T'$ (since $T \subset T'$) and thus if $T \not<_H T'$ we must have that $T \equiv_H T'$. From this we may conclude that if $T \not<_H T'$ then any faulty implementation that T' identifies is also identified by T.

The following shows that whenever a test set does not determine correctness under H, it is possible to improve its fault detecting ability by extending it by a single value.

PROPOSITION 17. *If $T \in P(\mathcal{X})$ does not determine conformance to s under H there is some $t \in \mathcal{X}$ such that $T <_H T \cup \{t\}$.*

PROOF. Since T does not determine correctness under H, there exist $p \in \mathcal{P}$ such that p does not conform to s but $p \preceq_T s$. Now choose some $t \in \mathcal{X}$ such that $p \not\preceq_{\{t\}} s$ (since p does not conform to s there must be some such value). Clearly $T \leq_H T \cup \{t\}$. Since p passes T but fails $T \cup \{t\}$, $T \cup \{t\} \not\leq_H T$. Thus $T <_H T \cup \{t\}$ as required. □

The following shows that every finite non-reducible test set may be generated though a process of adding single tests that increase the fault detecting ability.

PROPOSITION 18. *If $T = \{t_1, \ldots, t_n\}$ is a non-reducible test set under H then for all $1 \leq m \leq n.\{t_1, \ldots, t_{m-1}\} <_H \{t_1, \ldots, t_m\}$.*

PROOF. Let T_i denote $\{t_1, \ldots, t_i\}$. A proof by contradiction will be produced, initially assuming that there exists $1 \leq m \leq n$ such that $\{t_1, \ldots, t_{m-1}\} \equiv_H \{t_1, \ldots, t_m\}$. From this we may conclude that for all $p \in \mathcal{P}$, $p \not\preceq_{\{t_m\}} s \Rightarrow p \not\preceq_{T_{m-1}} s$. Thus $p \not\preceq_T s \Rightarrow p \not\preceq_{T \setminus \{t_m\}} s$ and so $T \equiv_H T \setminus \{t_m\}$, contradicting T being non-reducible. The result thus follows. □

This suggests that, given test hypothesis H, a finite non-reducible test set may be generated by a sequence of refinements, at each stage adding a new test case that increases the fault detecting power.

Test hypothesis H' is a *refinement* of test hypothesis H if and only if $H' \Rightarrow H$. It has been noted that a test hypothesis can be developed through a process of refinement [Gaudel 1995]. Since test generation, in the presence of a test hypothesis, can be seen as a process of refinement, this suggests that the two forms of refinement might proceed together. Interestingly, relationships under $\leq_H$ are preserved if H is refined.

PROPOSITION 19. *Suppose $C_2 \leq_H C_1$ and H' is a refinement of H. Then $C_2 \leq_{H'} C_1$.*

PROOF. Proof by contradiction: assume there exist $p \in \mathcal{P}$ such that the following hold.

(1) $H'(p)$ is true.
(2) There is some non-reducible test set $T_2 \in P(\mathcal{X})$ that satisfies $C_2(s, p)$ such that $p \not\preceq_{T_2} s$.
(3) There is a non-reducible test set T_1 that satisfies $C_1(s, p)$ with $p \preceq_{T_1} s$.

Now it is sufficient to note that since $H'(p)$ is true, $H(p)$ is true. Thus $C_2 \not\leq_H C_1$, providing a contradiction as required. □

This is an important property since it means that decisions made, with regard to the choice of test set or criterion, are not invalidated by refinements of the test hypothesis. Further, it suggests that test development might proceed through an iterative process in which both test hypotheses and test sets are refined. However, the converse of the above result does not hold.

PROPOSITION 20. *There exist C_2, C_1, H, and H' such that $C_2 \leq_{H'} C_1$, H' is a refinement of H but $C_2 \not\leq_H C_1$.*

PROOF. Here it is sufficient to choose $H = H_{min}$ and $H' = H_{corr}$. All test criteria are comparable under H'. Consider now the following two criteria that are based on any two values $t_1, t_2 \in \mathcal{X}$ $(t_1 \neq t_2)$.

(1) $C_1(s, p, T)$ is true if and only if $t_1 \in T$.
(2) $C_2(s, p, T)$ is true if and only if $t_2 \in T$.

Clearly $C_1 \leq_{H'} C_2$ but $C_1 \not\leq_H C_2$ as required. □

While the refinement of the test hypothesis cannot remove relations from $\leq_H$, it can have an impact on efficiency

PROPOSITION 21. *Suppose $T \in P(\mathcal{X})$ is a non-reducible test under H and H' is a refinement of H. Then T may contain redundancy under H'.*

PROOF. Here it is sufficient to choose $H = H_{min}$, $H' = H_{corr}$, and T to be any non-empty test set. Then all test sets are non-reducible under H_{min}, and thus T is non-reducible under H. Further, only the empty set is non-reducible under H_{corr} so T contains redundancy under H'. □

These results suggest that test set refinement and test hypothesis refinement can proceed together, potentially allowing tests to be executed before hypothesis refinement has completed. This might reduce the time taken to complete testing. Information derived in test execution and refinement might also feed into hypothesis refinement. However, the results show that there may be a cost associated with this process: the introduction of redundancy in the test used and thus a reduction in test efficiency.

5. EXAMPLES OF CURRENT TEST TECHNIQUES

The notion of test hypotheses has traditionally been used in black-box testing. Thus, the approach of using the existence of a test hypothesis when comparing tests currently makes most sense when considering specification-based testing. However, this might be seen as a challenge to develop sensible test hypotheses that are useful in white-box testing.

This section will consider three types of black-box testing, one type of white-box testing, and associated test hypotheses. In each case it is shown that properties of the test criteria may be expressed in terms of $\leq_H$ for some hypothesis H. In each case the key benefit is that it is possible to state when guarantees, regarding the ability of tests to find faults, may be made.

5.1 Testing Boolean specification

A number of authors have considered the problem of generating tests from a Boolean specification, in which all variables are Booleans, based on fault classes [Richardson and Thompson 1988; 1993; Weyuker et al. 1994]. Recently, Kuhn [Kuhn 1999] has shown that certain standard techniques are related. Specifically, the following fault classes were considered for expressions [Kuhn 1999]:

—Variable Reference Fault (VRF) - a variable is replaced by some other variable.

—Variable Negation Fault (VNF) - some variable is replaced by its negation.

—Expression Negation Fault (ENF) - some expression E that forms a complete clause in S (i.e. S is equivalent to $E \vee E'$ for some E') is replaced by its negation.

Kuhn showed how, given a specification, for each fault in one of these classes it is possible to define a condition such that: a test input satisfies this condition if and only if the test input finds the fault. If s denotes the specification and s' denotes the specification with a fault added, the condition for detecting the fault in s' is $\neg(s \Leftrightarrow s')$.

Assuming that the variable or expression has been identified, S_{VRF} denotes the condition under which a Variable Reference Fault is found, S_{VNF} denotes the condition under which a Variable Negation Fault is found, and S_{ENF} denotes the condition under which a Expression Negation Fault is found. Kuhn proved the following results [Kuhn 1999].

PROPOSITION 22. *If the variable replaced in VRF is the same variable negated in VNF then* $S_{VRF} \Rightarrow S_{VNF}$.

PROPOSITION 23. *If expressions containing the variable negated in VNF are negated in ENF then* $S_{VNF} \Rightarrow S_{ENF}$.

Interestingly, Kuhn also introduces results such as the following.

COROLLARY 1. *Any test that detects a variable reference fault for a variable* x *will also detect a variable negation fault for* x.

Implicit in this result is an assumption: that only the single fault exists. If we do not make such an assumption, it is straightforward to produce a counterexample in which another fault masks these faults for certain ranges of values. For example, suppose we consider the simple specification $S \equiv x$. Then a test that detects the variable reference fault in which x is replaced by y need not find a variable negation fault combined with an additional fault that transforms S to $\neg x \vee (x \wedge p)$: here the fault goes undetected if x and p are both true. An example of such a test is one in which $x = true$, $y = false$, and $p = true$. This detects the fault in which $S \equiv x$ is replaced by y since the values of x and y differ. However, $\neg x \vee (x \wedge p)$ is $\neg true \vee (true \wedge true)$ which simplifies to $true$. Thus, the test does not detect x being replaced by $\neg x \vee (x \wedge p)$.

We will now give a test hypothesis H_B and rephrase the above results in terms of $\leq_{H_B}$.

DEFINITION 11. *The test hypothesis* H_B *states that the implementation is equivalent to some behaviour that may be formed by introducing one instance of one of the following types of fault into the specification:*

—*Variable Reference Fault.*

—*Variable Negation Fault.*

—*Expression Negation Fault.*

We may now define the following test criteria.

DEFINITION 12. *Given a specification* s

—*a test set* T *satisfies* C_{VRF} *if for every* s' *that may be formed from* s *by introducing a single VRF, some test in* T *distinguishes* s' *from* s.

—*a test set* T *satisfies* C_{VNF} *if for every* s' *that may be formed from* s *by introducing a single VNF, some test in* T *distinguishes* s' *from* s.

—*a test set* T *satisfies* C_{ENF} *if for every* s' *that may be formed from* s *by introducing a single ENF, some test in* T *distinguishes* s' *from* s.

PROPOSITION 24. *The following relations hold:*

—$C_{VNF} \leq_{H_B} C_{VRF}$

—$C_{ENF} \leq_{H_B} C_{VNF}$

PROOF. These results follow from Propositions 22 and 23 respectively. □

Thus it can be argued that Kuhn in effect proved results about the relation $\leq_{H_B}$. The process of rephrasing these results in terms of $\leq_{H_B}$ has a number of additional benefits. First, it makes explicit the conditions under which the criteria are related in terms of fault detecting ability. Where these conditions might not hold, the tester is made aware of the limitations in these results. Secondly, it introduces the possibility of adding additional hypotheses, that represent known system properties, that might induce further relations between test techniques.

5.2 Mutation testing

In mutation testing [Budd 1981; King and Offutt 1991; Offutt 1992; Woodward 1993] a set $\mathcal{M}$ of mutants is produced by mutating, or changing, the program p. Changes to p are produced using mutation operators. Typically, each mutant is produced by making one application of a mutation operator to p. Such mutants are called *first order mutants*.

A test input t is said to *kill* mutant $p_m \in \mathcal{M}$ if and only if p and p_m produce different output when given input t. A mutant is an *equivalent mutant* if no test input kills it.

Given a set $\mathcal{M}$ of mutants, the quality of a test set is judged by determining which of the non-equivalent mutants are killed by it. Often the percentage of mutants killed is recorded, the target being to kill all of the non-equivalent mutants. The argument is that if a test set T is good at distinguishing p from its mutants then, assuming p is faulty, T it is also likely to be good at distinguishing p from a correct program.

Ultimately, the effectiveness of a test set is judged by determining which mutants it kills. We will now explore how this may be formalised in terms of $\leq_H$ for some appropriate test hypothesis.

The first step is to define the test hypothesis.

DEFINITION 13. *The test hypothesis, H_m, holds if either p is correct or there exists some first order mutant of p that is correct.*

It is straightforward to demonstrate that $\leq_{H_m}$ satisfies the following property that seems to capture the essence of mutation testing: that there is a link between the strength of the test set and the set of mutants killed by the test set.

PROPOSITION 25. *Given test sets T_1 and T_2, $T_2 \leq_{H_m} T_1$ if and only if every first order mutant killed by T_2 is also killed by T_1.*

An interesting question is: are there (relevant) weaker hypotheses under which this result holds? The existence of such hypotheses might make mutation testing more systematic.

Interestingly, the test hypothesis H_m is difficult to represent using a fault domain. This is because typically a fault domain contains variants on the specification that are intended to bound the possible behaviour of the IUT. In mutation testing, instead we fix the IUT and produce a set of possible behaviours that relate to the behaviour of the IUT, not the specification.

5.3 Partition analysis

Suppose the input domain has been partitioned to form a set $\Pi = \{\pi_1, \pi_2, \ldots\}$ of subdomains. For $s \in \mathcal{S}$, $p \in \mathcal{P}$, and Π the *uniformity hypothesis* states that for every $\pi_i \in \Pi$, if there exists some $x \in \pi_i$ such that $p \preceq_{\{x\}} s$ then for every $x' \in \pi_i$ we have that $p \preceq_{\{x'\}} s$. Naturally, the instance of the uniformity hypothesis used might be based on a number of factors including: the structure of the specification; expert knowledge; and information derived from program analysis.

Let H^s_Π denote the uniformity hypothesis for partition Π and specification s. A program p satisfies H^s_Π if and only if for every $\pi_i \in \Pi$, one of the following holds:

(1) $\forall x \in \pi_i . p \preceq_{\{x\}} s$.
(2) $\forall x \in \pi_i . p \not\preceq_{\{x\}} s$.

The uniformity hypothesis formalises the intuition behind *partition analysis*, in which the tester partitions the input domain to form a set Π of subdomains $\{\pi_1, \pi_2, \ldots\}$. The partition Π represents the belief that the behaviour of the IUT is uniform on each π_i. Ideally Π is finite. While the notion of uniformity on subdomain π_i is not formally defined, the intuition behind uniformity is clear. This is that faults in the behaviour of the IUT p on π_i are likely to be exhibited throughout π_i. This suggests that, assuming the behaviour of p is uniform on π_i, only a few test cases need be taken within π_i. Often these are supplemented by tests around the boundaries [Clarke et al. 1982; White and Cohen 1980]. Boundary tests may be represented by adding corresponding subdomains. A formal treatment of the uniformity hypothesis may be found in [Gaudel 1995].

Let $\Pi_\mathcal{X}$ denote the partition $\{\{x\} | x \in \mathcal{X}\}$: the partition in which each subdomain contains a single element. The minimal hypothesis might be seen to be an instance of the uniformity hypothesis using partition $\Pi_\mathcal{X}$.

PROPOSITION 26. *The hypothesis H_{min} is equivalent to $H^s_{\Pi_\mathcal{X}}$.*

The following gives sufficient and necessary conditions for a test set to determine correctness under an instance of the uniformity hypothesis [Howden 1976].

PROPOSITION 27. *A test $T \in P(\mathcal{X})$ determines correctness under H^s_Π if and only if $\forall \pi_i \in \Pi . \pi_i \cap T \neq \emptyset$.*

Of course, in practice it is normal to generate several tests from each subdomain. This is because of a lack of confidence in the uniformity hypothesis holding.

It is worth noting that, since Π need not be finite, there may be no practical test set that satisfies this condition. Where Π is finite, it may be sufficiently large to make it infeasible to satisfy this condition. However, it is still possible to compare the effectiveness of test sets in the presence of H_Π.

Given partition Π, it is straightforward to say when one test set is at least as strong as another under the hypothesis H_Π.

PROPOSITION 28. *Given $s \in \mathcal{S}$, $T_1, T_2 \in P(\mathcal{X})$ and partition Π, $T_2 \leq_{H^s_\Pi} T_1$ if and only if for all $\pi_i \in \Pi$, $T_2 \cap \pi_i \neq \emptyset \Rightarrow T_1 \cap \pi_i \neq \emptyset$.*

PROOF. Case 1: $\Rightarrow$. Proof by contradiction: suppose $T_2 \leq_{H^s_\Pi} T_1$ and there is some $\pi_i \in \Pi$ such that $T_2 \cap \pi_i \neq \emptyset$ and $T_1 \cap \pi_i = \emptyset$. Now consider a program $p_1 \in \mathcal{P}$

such that $p_1 \preceq_{\mathcal{X}\setminus\pi_i} s$ and $\forall x \in \pi_i . p \not\preceq_{\{x\}} s$. Then p satisfies H^s_Π. Since $T_1 \cap \pi_i = \emptyset$, $p_1 \preceq_{T_1} s$ and since $T_2 \cap \pi_i \neq \emptyset$, $p_1 \preceq_{T_2} s$. This contradicts $T_2 \leq_{H^s_\Pi} T_1$ as required.

Case 2: $\Leftarrow$. Proof by contradiction: assume for all $\pi_i \in \Pi$, $T_2 \cap \pi_i \neq \emptyset \Rightarrow T_1 \cap \pi_i \neq \emptyset$ but $T_2 \not\leq_{H^s_\Pi} T_1$. Since $T_2 \not\leq_{H^s_\Pi} T_1$ there is some program $p \in \mathcal{P}$ that satisfies H^s_Π, $p \preceq_{T_1} s$ and $p \not\preceq_{T_2} s$. Consider now some $x \in T_2$ such that $p \not\preceq_{\{x\}} s$ with $x \in \pi_i$ say. Since $T_2 \cap \pi_i \neq \emptyset \Rightarrow T_1 \cap \pi_i \neq \emptyset$, there is some $x' \in T_1$ with $x' \in \pi_i$. But since p satisfies H^s_Π and p fails on some value in π_i, p must fail on all values from π_i and, in particular p must fail on x'. Thus $p \not\preceq_{T_1} s$, providing a contradiction as required. □

This result shows how, by ignoring the hypothesis H^s_Π, the tester might add to a test set without increasing its fault detecting ability. In order to see this, suppose test set T is to be extended. Let Π_T denote $\{\pi \in \Pi | T \cap \pi \neq \emptyset\}$. If tests are added to T to form T', by Proposition 28 we know that $T \equiv_{H^s_\Pi} T'$ if $\Pi_T = \Pi_{T'}$. Where some of the subdomains in Π_T are infinite, this allows a (possibly small) test set T to be extended to an infinite test set (such as $\bigcup_{\pi_i \in \Pi_T} \pi_i$) that has the same fault detecting ability as T.

While the test hypothesis H_{corr} does not have a corresponding uniformity hypothesis, there is a maximal uniformity hypothesis. Under this hypothesis, all non-empty test sets are equivalent.

PROPOSITION 29. *All non-empty test sets are equivalent under the test hypothesis* $H^s_{\{\mathcal{X}\}}$.

PROOF. This follows immediately from Proposition 28. □

One of the nice properties of the class of uniformity hypotheses is that it contains minimal and maximal elements. The following result, which follows directly from the definitions, represents this fact.

PROPOSITION 30. *If* Π *is a partition of* $\mathcal{X}$ *then*

(1) H^s_Π *is a refinement of* $H^s_{\Pi_\mathcal{X}}$
(2) $H^s_{\{\mathcal{X}\}}$ *is a refinement of* H^s_Π

Let C_Π denote the criterion that the test must have at least one test from each subdomain in the partition Π. Given partitions Π_1 and Π_2, Π_1 is a refinement of Π_2 if each subdomain in Π_2 is the union of a set of subdomains from Π_1. Then the following results hold.

PROPOSITION 31. *Suppose partition* Π_1 *is a refinement of partition* Π_2. *Then*

—$C_{\Pi_2} \leq_{H_{\Pi_1}} C_{\Pi_1}$
—$C_{\Pi_2} \equiv_{H_{\Pi_2}} C_{\Pi_1}$

5.4 Testing from a finite state machine

Many systems are specified using state-based languages such as Statecharts [Harel and Politi 1998] and SDL [ITU-T 1999]. Testing from specifications written in state-based languages has thus been an active research area, particular attention being paid to the conformance testing of communication protocols [ITU-T 1997].

State-based systems are often tested using finite state machine (FSM) based techniques, one of the following approaches being applied to produce an FSM from the specification:

(1) Applying an abstraction.
(2) Expanding out the data elements (usually after restricting the range of values).

A (deterministic) finite state machine M is defined by a tuple $(Q, q_0, \delta, \lambda, \Sigma, \Gamma)$ in which Q is a finite set of states, $q_0 \in Q$ is the initial state, δ is the state transition function, λ is the output function, Σ is the finite input alphabet, and Γ is the finite output alphabet. If M receives input $x \in \Sigma$ while in state $q \in Q$, it produces output $\lambda(q, x) \in \Gamma$ and moves to state $\delta(q, x) \in Q$. This defines a *transition* $(q, \delta(q, x), x/\lambda(q, x))$.

Given FSM M that describes the required behaviour of IUT p, it is normal to assume that p behaves like some (unknown) FSM M_p [ITU-T 1997]. This might be seen as the minimal hypothesis. It is then normal to make further assumptions that limit the classes of faults that may occur. The following are (alternative) typical assumptions.

(1) The only faults are due to incorrect output in transitions.
(2) The FSM M_p has at most m states (some predefined m).

In conformance testing the set of assumptions used is seen as a fault domain [ITU-T 1997]. However, as was noted earlier, a fault domain is equivalent to a test hypothesis. We will briefly investigate these two fault domains and their impact on test generation.

5.4.1 *Output faults.* It is possible to assume that the state and transition structures are correctly implemented but that transitions might produce incorrect output. Given M, this test hypothesis might be denoted H_O^M. Hypothesis H_O^M states that p behaves like some unknown FSM that has no state transfer faults.

The existence of an internal state leads to the use of test sequences: in order to test an element of the system it may be necessary to set up the state for the test. To simplify the exposition, this section will work on the assumption that a single test sequence is being developed. However, similar results follow when using a set of test sequences.

Given test sequence $\underline{x}$ let $\tau(\underline{x})$ denote the set of transitions, of M, triggered by the input of $\underline{x}$. A test sequence determines correctness under H_O^M if and only if it triggers every transition in M. Such a test sequence is called a transition tour [Sidhu and Leung 1989].

PROPOSITION 32. *A test sequence $\underline{x}$ determines correctness under H_O^M if and only if $\tau(\underline{x})$ contains every transition from M.*

The following property of $\leq_{H_O^M}$ is clear.

PROPOSITION 33. *Given test sequences $\underline{x}$ and $\underline{x}'$, $\{\underline{x}\} \leq_{H_O^M} \{\underline{x}'\}$ if and only if $\tau(\underline{x}) \subseteq \tau(\underline{x}')$.*

An interesting extension of this may occur when the FSM has been formed through a process of abstraction. Here each transition represents a set of possible

input/initial state values. Testing from the corresponding FSM effectively involves using a uniformity hypothesis for each transition: if there is a fault for one value there is a fault for all values. Instead, the input domains of the transitions may be partitioned, the tester using the uniformity hypothesis based on these partitions. This leads to a hybrid test hypothesis that may form the basis for test generation [Hierons et al. 2001]. Future work will consider such hybrid test hypotheses.

5.4.2 *Upper bound on the number of states.* Rather than assume that only output faults may occur, it is possible to place an upper bound on the number of extra states to form H_m^M. The hypothesis H_m^M states that the IUT p behaves like some (unknown) FSM M_p that has the same input and output alphabets as M and has no more than m states. There are algorithms that generate tests, called *checking experiments*, that determine correctness under this hypothesis [Chow 1978; Hennie 1964; Petrenko et al. 1994; Rezaki and Ural 1995; Ural et al. 1997].

Let Υ_m denote a test set that determines correctness under hypothesis H_m^M.

PROPOSITION 34. *If* $m_1 \geq m_2$ *then*

(1) Υ_{m_1} *determines correctness under* $H_{m_2}^M$

(2) $\Upsilon_{m_2} \leq_{H_{m_1}^M} \Upsilon_{m_1}$

(3) $\Upsilon_{m_2} \equiv_{H_{m_2}^M} \Upsilon_{m_1}$

Interestingly, it seems significantly more difficult to characterize $\leq_{H_m^M}$ than the relations given in Section 5.3. Possibly this is because the test hypothesis refers to the structure of a state-based system rather than the input/output behaviour of a state-less system. For state-based systems the test hypotheses relate to a (regular) language of sequences of input/output pairs rather than a set of input/output pairs.

6. FUTURE WORK

This paper has highlighted a number of challenges. One challenge is to develop further test hypotheses that represent properties systems are likely to have and that allow statements about the effectiveness of testing to be made. Ideally these hypotheses should either be relatively simple to verify or to derive from the IUT. Where a test hypothesis is verified, the relation $\leq_H$ may be used with confidence. Appropriate hypotheses seem to be particularly lacking in white-box testing, the main exception to this possibly being mutation testing.

It has been shown that, given some test hypothesis H, the relation $\leq_H$ may be used to drive test generation: tests may be produced in an incremental manner. Naturally, practical issues remain and thus there is a further questions regarding when is it practical to use $\leq_H$ to drive incremental testing and how this may be achieved.

The work has raised a further interesting question. Suppose the tester knows some weakest test hypothesis H such that $C_2 \leq_H C_1$. What does this tell the tester about C_1, C_2 and their relative effectiveness? Finally, future work will consider how test hypotheses and fault domains may be used when applying probabilistic comparators.

7. CONCLUSIONS

This paper has explored the problem of comparing test sets and criteria in the presence of test hypotheses. The existence of a test hypothesis H, that represents known system properties, allows the introduction of a relation $\leq_H$ defined by: for test sets T_1 and T_2, $T_2 \leq_H T_1$ if and only if whenever T_2 finds a fault in an implementation p that satisfies H, T_1 finds a fault in p. The relation is extended to test criteria by considering non-reducible test sets that satisfy the criteria.

The relation $\leq_H$ introduced represents test effectiveness: the ability of testing to determine whether an implementation is faulty. This contrasts with relations such as subsumes, that say little about fault detection. The use of test hypotheses has proved to be vital: the relation $\leq$, produced without them, is equivalent to $\subseteq$ when applied to test sets and cannot be used to compare effective test criteria.

The comparator $\leq_H$ is based on the observation that relations that do not hold generally may hold where system properties are known. Importantly, the definition of $\leq_H$ provides a framework in which formal statements, about the relative effectiveness of test sets and criteria, can be made in the presence of known system properties.

We have observed that given a test hypothesis H, a non-empty test set T may have the same fault detecting ability as the empty set. Further, a test set T might be extended to form some test set T' with the same fault detecting ability as T. Thus it is importance to consider information, in the form of hypothesis H, during test generation.

The relation $<_H$, defined in terms of $\leq_H$, may be used to drive incremental test generation: a test input t should only be added to a test set T if $T <_H T \cup \{t\}$. Test hypothesis refinement preserves $\leq_H$: if H' refines H and $T_2 \leq_H T_1$ then $T_2 \leq_{H'} T_1$. Test development may thus be represented as an incremental process of refining test hypotheses and test sets. This allows tests to be generated, and executed, before test hypothesis refinement is complete. Of course, there are issues that must be resolved in order to make such incremental test development practical.

Four types of test hypothesis have been investigated. Where Boolean specifications are concerned, the results of Kuhn [Kuhn 1999] translate into results regarding $\leq_H$ for a natural choice of test hypothesis H. These results state what the results from [Kuhn 1999] mean in terms of the fault detecting ability of tests.

In mutation testing there is an implicit test hypothesis H_m. Once this test hypothesis is expressed, the relation $\leq_{H_m}$ is as expected: given test sets T_1 and T_2, $T_2 \leq_{H_m} T_1$ if and only if every mutant killed by T_2 is also killed by T_1.

Under the uniformity hypothesis many tests sets and criteria are related by $\leq_H$ and $\leq_H$ can be easily characterised. Test hypotheses have also been considered in the area of testing from a finite state machine. Here the results are much more mixed: under the hypothesis that only output faults exists, it is relatively simple to characterise $\leq_H$. However, due to their structural nature, $\leq_H$ is more difficult to characterise for alternative hypotheses.

Acknowledgements

I would like to thank the anonymous referees for their comments: these strengthened the paper. This work was partially funded by EPSRC grant number GR/R43150.

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.

Budd, T. A. 1981. Mutation analysis: Ideas, examples, problems and prospects. In *The Summer School on Computer Program Testing*. Academic Press, Sogesta, Urbino, Italy, 1–50.

Chen, T. Y. and Yu, Y. T. 1996. On the expected number of failures detected by subdomain testing and random testing. *IEEE Transactions on Software Engineering 4*, 109–119.

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

Clarke, L. A., Hassell, J., and Richardson, D. J. 1982. A close look at domain testing. *IEEE Transactions on Software Engineering 8*, 380–390.

Duran, J. W. and Ntafos, S. C. 1984. An evaluation of random testing. *IEEE Transactions on Software Engineering 10*, 438–444.

Frankl, P. G. and Weyuker, E. J. 1993a. A formal analysis of the fault-detecting ability of testing methods. *IEEE Transactions on Software Engineering 19*, 202–21.

Frankl, P. G. and Weyuker, E. J. 1993b. Provable improvements on branch testing. *IEEE Transactions on Software Engineering 19*, 962–975.

Gaudel, M. C. 1995. Testing can be formal too. *Lecture Notes in Computer Science 915*, 82–96.

Gourlay, J. S. 1983. A mathematical framework for the investigation of testing. *IEEE Transactions on Software Engineering 9*, 686–709.

Hamlet, R. 1989. Theoretical comparison of testing methods. In *Third Symposium on Testing, Analysis and Verification*. ACM, Key West, Florida, USA, 28–37.

Hamlet, R. and Taylor, R. 1990. Partition testing does not inspire confidence. *IEEE Transactions on Software Engineering 16*, 1402–1411.

Harel, D. and Politi, M. 1998. *Modeling reactive systems with statecharts: the STATEMATE approach*. McGraw-Hill, New York.

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

Hierons, R. M., Sadeghipour, S., and Singh, H. 2001. Testing a system specified using statecharts and Z. *Information and Software Technology 43*, 137–149.

Hierons, R. M. and Wiper, M. P. 1997. Estimating failure rates by partition and random testing. *Journal of Software Testing, Verification and Reliability 7*, 153–164.

Howden, W. E. 1976. Reliability of the path analysis testing strategy. *IEEE Transactions on Software Engineering 2*, 208–215.

ITU-T. 1997. *Recommendation Z.500 Framework on formal methods in conformance testing*. International Telecommunications Union, Geneva, Switzerland.

ITU-T. 1999. *Recommendation Z.100 Specification and description language (SDL)*. International Telecommunications Union, Geneva, Switzerland.

King, K. N. and Offutt, A. J. 1991. A FORTRAN language system for mutation-based software testing. *Software Practice and Experience 21*, 686–718.

Kuhn, D. R. 1999. Fault classes and error detection capability of specification-based testing. *ACM Transactions on Software Engineering Methodology 8*, 411–424.

Ntafos, S. C. 1988. A comparison of some structural testing strategies. *IEEE Transactions on Software Engineering 14*, 868–874.

Offutt, A. J. 1992. Investigations of the software testing coupling effect. *ACM Transactions on Software Engineering and Methodology 1*, 3–18.

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.

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

Richardson, D. J. and Thompson, M. C. 1988. The relay model of error detection and its application. In *The 2nd workshop on Software Testing, Analysis and Verification*. ACM Press, Banff, Canada, 223–230.

Richardson, D. J. and Thompson, M. C. 1993. an analysis of test data selection criteria using the RELAY model of fault detection. *IEEE Transactions on Software Engineering 19*, 533–553.

Sidhu, D. P. and Leung, T.-K. 1989. Formal methods for protocol testing: A detailed study. *IEEE Transactions on Software Engineering 15*, 413–426.

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

Weyuker, E., Goradia, T., and Singh, A. 1994. Automatically generating test data from a boolean specification. *IEEE Transactions on Software Engineering 20*, 353–363.

Weyuker, E. J. 1986. Axiomatizing software test data adequacy. *IEEE Transactions on Software Engineering 12*, 236–246.

Weyuker, E. J. 1988. The evaluation of program-based software test data adequacy criteria. *Communications of the ACM 31*, 668–675.

Weyuker, E. J. 2002. Thinking formally about testing without a formal specification. In *Formal Approaches to Testing of Software (FATES)*. INRIA Press, Brno, Czech Republic, 1–10.

Weyuker, E. J., Weiss, S. N., and Hamlet, D. 1991. Comparison of program testing strategies. In *1991 Symposium on Software Testing, Analysis, and Verification (TAV4)*. ACM Press, Victoria, British Columbia, Canada, 1–10.

White, L. J. and Cohen, E. I. 1980. A domain strategy for computer program testing. *IEEE Transactions on Software Engineering 6*, 247–257.

Woodward, M. R. 1993. Mutation testing - its origins and evolution. *Information and Software Technology 35*, 163–169.

Zhu, H. 1996. A formal interpretation of the subsumes relation between software test adequacy criteria. *IEEE Transactions on Software Engineering 22*, 248–255.

Zhu, H. and Hall, P. A. V. 1993. Test data adequacy measurement. *Software Engineering Journal 8*, 12–30.

Received June 2002; October 2002; Accepted November 2002