11 Evaluating Coverage Based Testing

Christophe Gaston[1] and Dirk Seifert[2]

[1] Commissariat a l'energie atomique
Logiciels pour la Sûreté des Procédés
christophe.gaston@cea.fr

[2] Technische Universität Berlin
Software Engineering Research Group
seifert@cs.tu-berlin.de

Abstract. In the previous chapters, various formal testing theories have been discussed. The correctness of an implementation with respect to a model is denoted by a so-called conformance relation. Conformance relations are relations between mathematical abstractions of implementations and models. Based on these conformance relations, different testing strategies have been defined. In this chapter, we concentrate on formal objects used to select test suites. These formal objects are so-called coverage criteria. A coverage criterion is a property that a selected test suite has to satisfy. We explore to which purposes these coverage criteria can be used for. Then we concentrate on the fault detection ability of a test suite satisfying a given coverage criterion.

11.1 Introduction

All testing methodologies introduced in this book follow the same generic test process. Test cases are generated according to a given model of the implementation. The model results from a requirements analysis and has to be (if testing is done automatically) a formal description of the requirements. Test cases are sequences of input/output pairs and a finite set of test cases is called test suite. For each test case of a test suite, the input specified in the first pair of the sequence is refined with concrete data called test data. Test data are submitted to the implementation through its environment. The implementation generates a result which is captured through its environment. The result is compared (with respect to a conformance relation) to the output specified in the pair. If the conformance relation is not contradicted, the process goes on with the following pair. If generated outputs all correspond to the intended outputs, the test case is executed successfully. If all the test cases of the test suite are executed successfully, a success verdict is assigned to the test process, since no test case of the test suite allows to show that the implementation does not conform to the specification. Figure 11.1 shows this testing framework.

The number of test cases required to obtain confidence in the system under test is usually infinitely large for real life applications. Consequently, a so called *domain expert* is involved in the test process, as he is able to extract interesting test suites due to his knowledge. For automated test case generation, the problem remains unsolved. So, for current testing practices, one of the open questions is: Which test suite should be extracted from a possibly infinite set of test cases?

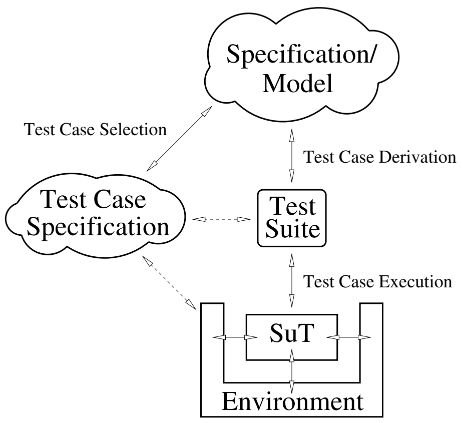

Fig. 11.1. A Usual Test Framework

Information provided by the model to extract test cases can have two different natures. On the one hand, functional aspects can be used. These aspects of the model describe the intended functionality of the implementation. The goal of test purpose based techniques is to generate test suites to validate such kind of properties [FJJV96, BGM91]. On the other hand, structural aspects of the model can be used. These structural aspects can be the *state space* description or the *dynamics* description of the implementation. For example, in *Z*, *VDM*, or *B* specifications, the state space is given by a set of typed variables and predicates describing the invariant. The dynamics description is constituted of operations which map input data and the state before applying the operation to the output data and the state after applying the operation. In Extended Finite State Machines, the state space is expressed by variables and by guards over these variables, while the dynamics description is given by assignments on variables, transition functions, and functions which define the output generated depending on an input received and the current state. In coverage based testing techniques, test suites are selected to cover some structural aspects of the model with respect to given coverage criteria. Coverage criteria can be seen as predicates defined on triples (P, M, T), where P is a program whose associated model is M, and T is a test suite. The meaning of such a criterion can be understood in two ways:

- As an *adequacy criterion*, which is a set of rules used to determine whether or not testing is complete for a given program, specification and criterion.
- As a *selection criterion*, which is a set of rules used to select test cases for a given program and specification.

A selection criterion helps to select a test suite in order to fulfill a goal, whereas an adequacy criterion helps to check that a previously selected test suite satisfies a goal. The notion of coverage criteria has been originally defined for white-box testing techniques. In these techniques, structural aspects to cover are related to programs: for example, a coverage criterion may be to cover all statement sequences in a program. In this book, we focus on coverage criteria related to models: for example, a coverage criterion may be to cover all the states of a model.

This chapter provides an overview of different research activities dealing with the coverage based testing techniques and their evaluation. In Section 11.2, we discuss how coverage criteria can help to select meaningful test suites. In Section 11.3, we present common coverage criteria. In Section 11.4 we concentrate on providing quantitative elements to evaluate the ability to detect faults of coverage based testing techniques. Finally, in Section 11.5, we summarize the main results and present open problems.

11.2 Coverage Criteria

To address the question *Which test cases should be extracted from a possibly infinite set of test cases?* it is possible to use advanced specifications describing which test cases to choose. Such specifications are usually called test case specifications (cf. Figure 11.1), and are, strictly speaking, selection criteria.

The main reasons for an infinitely large test suite are that a specification can represent an infinite number of traces (e.g., caused by variables ranging over infinite domains) and that a specification can contain infinite traces (e.g., caused by loops). For the first reason the problem is to select traces for the second reason the problem is to chop traces. Basically, the specification (ideally) describes all possible test cases while the test case specification describes which of these test cases are likely to find errors and, consequently, increase the confidence in the correctness of the implementation. Most of the criteria used for selecting test cases can be coarsely classified according to the following three aspects:

(1) The test case specification is a description of a structural criterion which should be covered.
(2) The test case specification is a description of functional aspects, also called scenarios, which should be covered.
(3) The test case specification contains stochastic information about different aspects of the implementation which is be used to concentrate on particular details.

The third variant is also used to restrict a test case specification if this represents an infinite model (for example, if path coverage is required which is usually infeasible to achieve as explained in the following section).

11.2.1 Structural Criteria

Basically, coverage criteria are used to measure the *quality* or, more precisely, the *adequacy* of test suites: A test suite is adequate according to a criterion if a designated percentage of this coverage criterion is reached. Depending on this, testing is continued or stopped. A structural criterion is an assertion about the structure of the specification. For example, in model based specifications states or transitions can be used. In class diagrams, for example, it could be required to use at least one instance of each class in the diagram.

For test case generation this means that test cases are generated according to a structural criterion, and that a test case is selected if this test case increases the coverage of this criterion. In the case of several test cases one can choose the test case with the major contribution.

For example, **transition coverage** requires choosing test cases in such a way that all transitions of the specification are covered. The **boundary interior** test requires that every loop is repeated zero, one and at most k times. The strongest coverage criterion is the **path coverage** criterion. This criterion is satisfied by a test suite if and only if for any possible path in the model, the test suite contains at least one test case which enforces an execution of this path in the implementation. Path coverage is in general impossible to achieve and impractical for real life testing. Reasons for that are loops in the model or infinite data spaces.

11.2.2 Functional Criteria

Another method to select test cases is using a model of the environment. Usually, such a model is called *scenario model*, *use case* or *user profile* and allows to describe scenarios which involve user enabled functionalities. The idea is that the test case specification is used to determine the inputs used to test the implementation and the model is used to estimate the expected outputs of the implementation.

For example, if the system under test is a banking application, usual functionality is the deposit of money into an account, the withdrawal of money from an account, or checking the balance. All these functionalities can be described as scenarios. Another example are the so called attack traces in security engineering. Here, possible ways to attack the system under test by an *unfriendly* user are modeled in scenarios and used to control the test selection process. So, functional criteria restrict the test case generation to particular scenarios and thus restrict the number of test cases.

As mentioned before, further reduction is required in case that the test case specification represents a possibly infinite set of scenarios (for example, if the test case specification is modeled as a state machine containing loops).

11.2.3 Stochastic Criteria

Usually, stochastic criteria[1] result from analysis of the expected user behavior or system usage, respectively. The simplest case is that all parts of the implementation, or all its functionalities have equal probability of execution. In this case, test case selection is done randomly. In contrast, if some functions are frequently used or represent important functionalities, test cases connected to these functionalities are preferred.

[1] Note that stochastic criteria can also be referred to as *statistical criteria*.

In the following section we review some usual coverage criteria that are used to select test cases. The aim of the section is to give the reader an intuition of usual criteria. A more exhaustive presentation of coverage criteria can be found elsewhere [FW88, RW85, ZHM97].

11.3 Coverage Based Testing

The problem of tractable coverage criteria that are easier to satisfy than the path coverage criterion, has been studied for a long time in the context of white-box testing. A lot of criteria have been defined [Mey79]. In the following, we present different coverage criteria used in the context of model based testing approaches which are adapted from white-box testing. Most of these coverage criteria can be classified into two main classes: *control flow oriented* coverage criteria and *data flow oriented* coverage criteria.

Control flow oriented coverage criteria are based on logical expressions introduced in the specification which determine the branch and loop structure of the implementation. Data flow oriented coverage criteria focus on the data flow part of the implementation. More precisely, they focus on the way values are associated to their variables and how these assignments affect the execution of the implementation [VB01].

In the following coverage criteria are introduced and explained, keeping in mind the analogy between models and abstract programs. Our discussions fits also for coverage criteria associated to programs with respect to white-box testing (i.e., coverage of code). In this context coverage criteria are usually described as flow graphs. However, we do not need this notion to present control flow criteria. Thus, a flow graph definition is only introduced in Section 11.3.2.

11.3.1 Control Flow Oriented Coverage Criteria

Basically, control flow oriented coverage criteria rely on the notions of *decision* and *condition* [VB01]. A condition is an *elementary* boolean expression which cannot be divided into further boolean expressions. A decision can be seen as a control point in the specification at which the control flow can follow various paths. In programming languages, this is a boolean expression consisting of several conditions combined by logical connectives. An instantiation of the common `IF-THEN-ELSE` construct in programming languages is an example for a decision. The most basic control flow criterion is the decision coverage criterion.

The **decision coverage** criterion [Mey79], also known as **branch coverage**, requires that each possible outcome (i.e., *true* or *false*) of every decision in the specification is produced at least once. For example, the specification contains a decision D: `IF (A∧B) THEN S`, where `A` and `B` are conditions. It is required that at least one test case makes `(A∧B)` evaluate to *true* and one makes `(A∧B)` evaluate to *false*. A test suite which contains two test cases, one such that `A` is *true* and `B` is *true* and the other one such that `A` is *false* and `B` is *true*, is sufficient to test decision D. The example clearly demonstrates that decision coverage does not ensure test suites which cover all different outcomes of a condition involved in a given decision. For example, the fact that `B` is failure causing could remain undetected with this criterion. To overcome this weakness, three refined criteria have been introduced.

The **condition coverage** criterion requires that each possible outcome of every condition in each decision is produced at least once. To give an example, we consider again decision D. The condition coverage criterion requires that `A` and `B` have taken all possible outcomes. Thus, a test suite which contains two test cases, one such that `A` is *true* and `B` is *false* and the other one such that `A` is *false* and `B` is *true* is sufficient to test decision D.

Even though the condition coverage criterion captures all conditions, it is not powerful enough to capture coverage of decisions. The test suite described above for condition D illustrates this fact: It consists of two test cases which both make $(\mathtt{A} \wedge \mathtt{B})$ evaluate to false. To overcome this weakness one must combine condition coverage and decision coverage. This is done in decision condition coverage.

The **decision condition coverage** criterion requires that each possible outcome of every condition in each decision is produced at least once and that each possible outcome of every decision in the specification is produced at least once. For decision D, a test suite which only contains two test cases, one such that `A` is *true* and `B` is *true* and the other one such that `A` is *false* and `B` is *false*, is sufficient to test decision D with regard to decision condition coverage. Decision condition coverage is strictly stronger than both decision coverage and condition coverage in the sense that each test suite which satisfies decision condition coverage satisfies both decision coverage and condition coverage.

The **multiple condition coverage** criterion requires that each possible combination of conditions outcomes in each decision is produced at least once. Again, we consider decision D, a test suite containing four test cases (`A` is *true* and `B` is *true*, `A` is *true* and `B` is *false*, `A` is *false* and `B` is *true* and `A` is *false* and `B` is *false*) is necessary to test D with regard to multiple condition coverage.

Note that multiple condition coverage requires full search of various combinations of condition values [VB01]. If the number of conditions in a decision is equal to n, the number of test cases to satisfy multiple condition coverage grows up to 2^n. This becomes unmanageable even for relatively moderate values of n. Decision coverage, condition coverage and decision condition coverage criteria require less test cases. For example, condition coverage requires two test cases per condition. If a decision contains n conditions, the criterion requires at maximum $2n$ test cases. However, a test suite which satisfies one of these three weaker criteria will not cover all combinations of conditions outcomes. The modified condition decision coverage criterion provides an intermediate position.

The **modified condition decision coverage** criterion requires that each possible outcome of every condition in a decision is produced at least once, each possible outcome of every decision is produced at least once and that each condition in a decision has been shown to affect the decision's outcome independently. A condition is shown to affect a decision's outcome independently by varying that condition while all other possible conditions are fixed.

Modified condition decision coverage includes in its definition both decision coverage and condition coverage. Furthermore, decision coverage can be deduced from condition coverage in combination with the independently affect property. Again, we consider decision D. A test suite with three test cases such that `A` is *true* and `B` is *true*, `A` is *true* and `B` is *false*, and `A` is *false* and `B` is *true* satisfies the modified condition decision coverage. Obviously, such a test suite satisfies the condition decision coverage and also the independently affect property. From these two facts, it is easy to see that decision coverage is satisfied. The number of required test cases ranges between $n + 1$ and $2n$ which is manageable even for large values of n. However, there are situations in which it is impossible to vary one condition value while keeping the others unchanged. This is the case if `A` is *true* implies that `B` is *true.* To overcome this problem, Vilkomir et. al. provide an improved formal definition of the modified condition decision coverage criterion [VB02]. There it is sufficient to choose any combination that varies both condition and decision even-though other conditions may also vary.

At last, the **full predicate coverage** criterion requires that each possible outcome of every condition in a decision is produced at least once, where the value of a decision is directly correlated with the value of a condition. Intuitively, multiple condition decision coverage is relaxed in the sense that it is not required that conditions in a decision independently affect the decision.

11.3.2 Data Flow Oriented Coverage Criteria

Data flow oriented criteria are based on data flow analysis with respect to compiler optimization activities. They require test cases that follow instruction sequences from points where values are assigned to variables to points where those variables are used. To introduce different criteria, we define *flow graphs* associated to a model. Strictly speaking, we discuss code coverage as used in white-box testing approaches. The relationship between models and code is that *behavioral* models can be compiled into code and that described coverage criteria can be applied to this code. However, there are many possibilities for this relationship; the way criteria are applied depends on the concrete approach. For example, using modified condition decision coverage at level of assembly code does not make sense.

A **flow graph** associated to a model is a directed graph that consists of a set of nodes and a set of edges connecting these nodes. Nodes contain linear sequences of computations (i.e., access to external values, variable assignments, data changes, etc). Edges represent transfer of control between nodes specified in the specification. Additionally, each edge is associated with a boolean expression that is the condition of the corresponding control transfer. A flow graph contains an initial node, which denotes the beginning of an abstract[2] computation, and a set of terminal nodes which denote exit points. Depending on the type of model, initial and terminal nodes have to be chosen or added (for example, extended finite state machine formalisms do not use notions of *beginning* and *ending* of

[2] Abstract in the sense that models are abstractions of programs.

computations). In some cases, a data flow graph can be seen as an annotated control flow graph.

In the following, a flow graph is a representation of all statement sequences of the model, and a test case is a possible (instantiated) instance of a path in the flow graph (i.e., inputs which execute the instruction sequence denoted by the path) [ZHM97]. Moreover each occurrence of a variable in a node is classified either as definition occurrence or as use occurrence. The latter can be further divided into computational use or, if it is used within a predicate, into predicate use. In the following necessary definitions are given:

- A *definition clear path* with respect to a variable x in a flow graph is a path where for all nodes in the path there is no definition occurrence of x.
- A definition occurrence of variable x within a node u *reaches a computational use occurrence* of x within a node v, if and only if there is a path $p = (u, w_1, \ldots, w_n, v)$, such that $(w_1, \ldots, w_n)$ is definition clear with respect to x.
- A definition occurrence of variable x in u *reaches a predicate use occurrence* of x on the edge (w_n, v), if and only if there is a path $p = (u, w_1, \ldots, w_n, v)$, such that $(w_1, \ldots, w_n)$ is definition clear with respect to x and there exist a predicate occurrence of x associated to the edge from w_n to v.
- For both predicate and computational use occurrence, a definition occurrence of x in u *feasibly reaches* the use occurrence of x, if and only if there is a path p, such that there exists inputs which enforce the execution of p.

The simplest data flow criteria rely on paths that start with the definition of a variable and end with the use of the same variable. The following criteria are adapted from the work of Frankl and Weyuker [FW88].

A test suite T satisfies the **all definitions coverage** criterion, if and only if for all definition occurrences of a variable x, such that there is a use occurrence of x which is feasibly reachable from the definition, there is at least one element in T which is a numerical instance of a path p, that contains a sub path through which the definition of x reaches some use occurrence of x. Thus, the all definitions coverage criterion ensures that all defined variables will be tested at least once by one of their uses in the model. However, this is insufficient as tester require to test all uses of all variable definitions. It is ensured by the all uses criterion.

A test suite T satisfies the **all uses coverage** criterion, if and only if for all definition occurrences of a variable x and for all use occurrences of x which are feasibly reachable from the definition, there is at least one element in T which is a numerical instance of a path p that contains a sub path through which the definition of x reaches the use occurrence.

The previously described criteria have been specialized to take into account that a use occurrence can be a computational use or a predicate use [RW85]. However, the all uses coverage criterion does not ensure that all possible ways to reach a use occurrence have been tested. As there may be several sub paths which allow a definition occurrence of a variable to reach a use occurrence of this variable. Note that some of these paths may be infinite due to cycles and cannot be covered. A possible solution is to restrain to cycle free sub paths. The only cycles allowed are those that begin and end at the same node. The all definitions uses paths criterion requires each of such cycle free sub paths to be covered by at least one test case.

A test suite T satisfies the **all definitions uses paths coverage** criterion, if and only if for all definition occurrences of a variable x and for all paths q through which a use occurrence of x is reached, there is at least one element in T which is a numerical instance of a path p that contains q as a sub-path. Moreover, q is required either to be cycle free or to be such that the first node is also the last node. This criterion may never be satisfied since such cycle free paths are infeasible to generate. But, more complex criteria (involving several definition and use occurrences) are definable.

Ntafos introduces a family of coverage criteria (required k-tuples) and their definition relies on the notion of k-dr interactions [Nta88]. For $k > 1$ a ***k*-dr interaction** is a sequence $K = [d_1(x_1), u_1(x_1), d_2(x_2), u_2(x_2), \ldots, d_k(x_k), u_k(x_k)]$ and for all $i < k$:

- A definition occurrence of x_i is $d_i(x_i)$.
- A definition use of x_i id $u_i(x_i)$.
- The use occurrence $u_i(x_i)$ and the definition occurrence $d_{i+1}(x_{i+1})$ are associated with the same node n_{i+1} in a path $p = (n_1) \cdot p_1 \cdot (n_2) \cdot \ldots \cdot (n_{k-1}) \cdot p_{k-1} \cdot (n_k)$ such that the definition occurrence $d_1(x_1)$ is associated to n_1.
- The i^{th} definition occurrence $d_i(x_i)$ reaches the i^{th} use occurrence $u_i(x_i)$ through p_i.

Where p is an interaction path for the k-dr interaction K. The aim of the required k-tuples criteria is to achieve test suites which allow to test j-dr interactions for $j \leq k$.

A test suite T satisfies the **required *k*-tuples** criterion, if and only if for all j-dr interactions L with $1 < j \leq k$, there is at least one test case in T which is a numerical instance of a path p such that p includes a sub path that is an interaction path for L.

The presented criteria are basic coverage criteria. Numerous criteria have been defined elsewhere (for example, criteria that take the number of loogs into account). For further study, the paper of Zhu et. al. provides a presentation of a large number of coverage criteria [ZHM97].

11.4 Coverage Based Testing and Fault Detection Ability

In the following, we concentrate on systems which are non-reactive: that is, they can be seen as functions (taking an input as argument and yielding a result). For these systems, test cases are pairs of inputs and intended outputs. We present contributions which aim at providing quantitative elements to evaluate the ability to detect faults of structural coverage based testing techniques.

We assume that structural coverage based testing techniques can be seen as partition based testing techniques. Partition based testing consists in splitting the whole input domain of the implementation into several subdomains. For example, domain $D = \{0, 1, 2\}$ is separated into two subdomains $D_1 = \{0, 1\}$ and $D_2 = \{2\}$. D_1 and D_2 define together a *partition* (in the mathematical sense) of D. Usually, the terminology of partition based testing is associated to any technique involving a division of the input domain into several subdomains even if these subdomains overlap. For example, if the domain $D = \{0, 1, 2\}$ is divided into the following two subdomains: $D'_1 = \{0, 1\}$ and $D'_2 = \{1, 2\}$.

Let us consider any structural selection criterion applied on a given model and program. Selecting a test suite implies dividing the whole input domain of the implementation. For example, a model described in a formalism containing the "if-then-else" statement with x ranging over D: *if* $(x \leq 1)$ *then* $inst_1$ *else* $inst_2$. By using decision coverage, the selected test suite contains at least two test cases: one for which the selected test data is such that $x \leq 1$ and one for which the test data is such that $x > 1$. This example clearly demonstrates that testing techniques in which test case selection processes are based on structural criteria are in fact partition based testing techniques.

The first part of the following section is to compare abilities to detect faults of structural coverage based testing techniques and of random based testing. Random based testing consists in selecting a certain number of test data randomly out of the input domain and evaluating outputs caused by test data with regard to intended results expressed in the model. Following the discussion above, we discuss contributions which compare random based testing and partition based testing techniques. Section 11.4.1 provides a structured presentation of several significant contributions to this aspect.

The second part of the following section is to compare techniques based on different structural criteria on the basis of abilities to detect faults. One of the most well known ways to compare criteria is by the subsume relation. It is a way to compare the severity of testing methods (in terms of adequacy of test suites). In Section 11.4.2 we present several relations derived from the subsume relation. Then, it is studied whether or not these relations impact the fault detection ability. That is, the following question is addressed: If two criteria are involved in one of these relations, what can we say about their respective abilities to detect faults?

11.4.1 Partition Testing Versus Random Testing

Here we focus on contributions which address the problem of comparing respective abilities to detect faults of random based and partition based testing [DN84, HT90, Nta98, Gut99]. All these contributions are based on a common mathematical framework. This framework is called the failure rate model and is now described.

We suppose that an implementation is used for a long period of time with various samples of randomly selected test data. Furthermore we suppose that we are able to observe the number of detected faults at any time. The number of faults will converge towards a constant. We denote θ the ratio between this constant and the total number of possible inputs. The ratio θ is called the failure rate associated to the domain D of the implementation. The probability to randomly select one test data which does not reveal a fault is $1-\theta$ and the probability to randomly select n test data, none of which reveals a fault, is $(1-\theta)^n$. The probability to reveal at least one fault for n randomly selected test data can be expressed as follows: $P_r = 1-(1-\theta)^n$.

Now let us consider that a partition based testing technique partitions the domain D into k subdomains $D_1 \dots D_k$. For each D_i, $i \in \{1, \dots, k\}$, θ_i denotes the failure rate associated to D_i. Now suppose that the testing technique states that n_i test data must be selected in D_i. A total of n test data is selected, therefore $n = \sum_{i=1}^{k} n_i$. The probability to select n_i test data in D_i, none of which reveals a fault, is $(1-\theta_i)^{n_i}$ and the probability to detect at least one fault when selecting n_i test data in each D_i is $P_p = 1 - \prod_{i=1}^{k} (1-\theta_i)^{n_i}$.

For each D_i, $i \in \{1, \dots, k\}$, p_i is the probability that a randomly chosen test data is in D_i, so that $\theta = \sum_{i=1}^{k} p_i\theta_i$. Thus, P_r and P_p can be expressed as follows:

$$P_p = 1 - \prod_{i=1}^{k} (1-\theta_i)^{n_i} \qquad \text{(for partition based testing), and}$$
$$P_r = 1 - (1 - \sum_{i=1}^{k} p_i\theta_i)^n \qquad \text{(for random based testing).}$$

In the following, contributions introduced can be classified into two different types. In the first type of contributions the results are based on simulation experiments. The idea is to perform comparisons between P_r and P_p with different valuations of their variable parts. In the second type of contributions (the fundamental approaches) the results are based on mathematical proofs. Under particular assumptions, fundamental results are proved.

Simulation Experiments

Duran and Ntafos [DN84] follow the framework described above to address the problem whether or not one of the two testing methods is more efficient at detecting faults than the other. That is, they compare P_r and P_p through various simulation experiments. Moreover the authors compare the two testing methods through another criterion: the expected number of errors that a set of test data will discover. Using an ideal partition scheme in which each subdomain contains at most one fault, the expected number of errors discovered with partition based testing is given by the formula $E_p(k) = \sum_{i=1}^{k} \theta_i$. Here, one test data is randomly chosen out of each subdomain D_i. The expected number of errors found by n random test data $E_r(k,n)$ is given by the formula $E_r(k,n) = k - \sum_{i=1}^{k} (1-p_i\theta_i)^n$.

The simulation experiments consist of different variations: the number k of subdomains, the number n_i of test data in each subdomain (and thus the overall number n of test data), the failure rate θ_i in each subdomain and the probability p_i that a randomly chosen test data is in the subdomain D_i (and thus the overall failure rate θ). For each variation Duran and Ntafos study the ratio $\frac{P_r}{P_p}$ and $\frac{E_r}{E_p}$. The experiments reported are based on two different assumptions on failure rates. On the one hand, a usual belief about partition based testing is that it allows to obtain homogeneous subdomains. That is, if an input of a subdomain is failure causing, then all inputs of the subdomain have a high probability to be failure causing and conversely. Under this assumption, failure rates should be either close to 0 or close to 1. On the other hand, there are examples of program paths that compute correct values for some, but not all, of their input data. Under this assumption, the failure rate distribution should be more uniform than suggested above.

- In the first experiment, the authors suppose that the partition based testing technique divides the domain into 25 subdomains. It is supposed that the partition based technique requires the selection of one test data per subdomain. To provide a fair comparison the random based testing method requires to select 25 test data randomly. Several values for θ_i are selected. The θ_i's are chosen from a distribution such that 2 percent of the time $\theta_i \geq 0.98$ and 98 percent of the time $\theta_i \leq 0.049$. These assignments reflect a situation in which subdomains are homogeneous. The p_i are chosen from a uniform distribution. It appears that on a total of 50 trials 14 trials are such that $P_r \geq P_p$. However the mean value of $\frac{P_r}{P_p}$ is 0.932. Under the same hypothesis on failure rates, the experiment is repeated for $k = n = 50$ and the results are even more surprising. Indeed, one could think that increasing the number of subdomains should favor partition based testing. However this experiment does not corroborate this intuition: the mean value of $\frac{P_r}{P_p}$ was 0.949. The mean value of the ratio $\frac{E_r}{E_p}$ is for 25 subdomains and 50 trials it is equal to 0.89, and for 50 subdomains and 50 trials it is equal to 0.836.
- In the second experiment, the assumption on the θ_i distribution is that θ_i's are allowed to vary uniformly from 0 to a given value $\theta_{max} \leq 1$. Several possible values are assigned to θ_{max}. Experiments are performed for $k = n = 25$ and $k = n = 50$. As θ_{max} increases P_r and P_p tend to 1. Random based testing performs better for the lower failure rates and also when the size of the partition is 25 instead of 50. Similar studies are carried out for E_r and E_p. In these studies, the number of randomly selected test data is allowed to be greater than the number of test data selected for partition based testing (100 for random based testing versus 50 for partition based testing): this is consistent with the fact that carrying out some partition based testing scheme is much more expensive than performing an equivalent number of random test data. Under these assumptions, random based testing performed better than partition based testing most of the time ($E_r > E_p$).

All these experiments deeply question the value of partition based testing with regard to random based testing. However, these are simulation results. Therefore, the authors concentrate on actual evaluations of random based testing. Duran and Ntafos propose to evaluate random based testing on three programs containing known bugs. The first program contains three errors. The first error is detected 11 out of 50 times, the second error 24 times and the third error 45 out of 50 times. The simple error in the second program is detected by 21 out of 24 times. For the third program 50 test cases were generated. The simple error was detected 18 of 50 times. More programs were tested with similar results.

One of the features of coverage criteria is that they can be used to measure coverage of test suites generated by other methods. The authors evaluate some test suites generated by random based testing, with program based coverage criteria. Test suites are generated for the programs previously used to evaluate random based testing. The idea is to simply generate a test suite and then to use a given criterion to see if the test suite satisfies the requirements stated by the criterion. Several criteria are then considered to measure random based testing adequacy. The number of test data generated ranges between 20 and 120 and five programs from the previous experiments were used. The over-all result is, that for a moderate number of random test data random based testing allows to cover these criteria for coverage percentages ranging from 57 percent up to 94 percent depending on the criterion.

The experiments presented in the paper indicates that it is reasonable to assume that random based testing can find more errors per unit cost than partition based testing, since carrying out some partition based testing scheme is much more expensive than performing an equivalent number of random test data. This holds for homogeneous subdomains and for values of θ_i uniformly distributed. Assumptions on failure rates may be unrealistic but actual evaluations show that random based testing seems to discover some relatively subtle errors without great efforts. Moreover, random based testing seems to ensure a high level of coverage for some usual coverage criteria.

Hamlet and Taylor [HT90] explore the results of Duran and Ntafos more deeply. They perform experiments based on statistical assumptions very similar to those made by Duran and Ntafos.

They compare partition based testing and random based testing with respect to the conventional failure rate model used by Duran and Ntafos. They are compared by different numerical valuations of their respective probabilities to detect faults for the same number of selected test data ($\sum_{i=1}^{k} n_i = n$). Different relationships between θ and θ_i are proposed:

- The first relationship is based on the assumption that if a test data is randomly selected, the probability that this test data is an element of any subdomain is $1/k$. Thus θ is the average of the sum of all θ_i: $\theta = \frac{1}{k}\sum_{i=1}^{k} \theta_i$. The difference between random based and partition based testing in terms of the probability of finding at least one failure will be maximal when the variance of the θ_i has a maximum. If only one test data per subdomain is selected, this occurs if only one subdomain, the j^{th} one, is failure causing ($\theta_j = k\theta$ and $\theta_i = 0$ for $i \neq j$). This situation is studied for different failure rates and different sizes of partitions. To give a significant advantage to partition based testing, the number k of subdomains has to be of the same order of magnitude as the inverse of θ: in the frame of this investigation, the most favorable case is that random based testing is about 0.63 as effective as partition based testing. This result is clearly better than the results obtained by Duran and Ntafos [DN84]. But Hamlet and Taylor also observe that the assumption $p_i = 1/k$ is not realistic.

- The second relationship is introduced by Duran and Ntafos [DN84]. This relationship is based on the assumption that when a test data is randomly selected the probability that this test data is an element of subdomain D_i is an arbitrary number p_i. The influence of the number of subdomains, the distribution of θ_i and of lower and upper bounds for θ_i is investigated by different experiments. This deeper investigation does not contradict previous results given by Duran and Ntafos [DN84] which indicate that there are little differences between partition based and random based testing with regard to their probabilities of revealing failures. Even-though partition based testing is sometimes better, slight advantage for partition based testing can be reduced by using a higher number of random test data.
- The third relationship explores modifications of the relationship described above. The aim is to gain information on the importance of the way the subdomains are chosen and, the impact of homogeneity on the effectiveness of partition based testing. To obtain information on the importance of subdomain selection, one needs a correlation between the probability of a random test data in a given subdomain (p_i) and its failure rate (θ_i). The correlation is denoted by a weight associated to each θ_i. This weight is used to calculate p_i. The higher the weight is, the more subdomains with high failure rates have a low probability that a random test data would fall into them. The model intuitively favors partition based testing if the weight associated to a failure causing subdomain is high. Experiments are consistent with this intuition but the effectiveness of random based testing is not dramatically affected: in the worst case, random based testing is 0.77 as effective as partition based testing. Some other experiments in which failure rates are controlled were conducted. Some subdomains (hidden subdomains) have small probability of being used by random test data while other subdomains (exposed subdomains) have a high probability of being used. Failure rates of subdomains are then varied. When failure rates of hidden subdomains are higher than the overall failure rate, partition based testing is favored. When failure rates of hidden subdomains are lower than the overall failure rate, random based testing is favored. The only result is that the advantage of partition based testing arises from increased sampling in regions where failures occur. Other experiments are performed to obtain information on the importance of the impact of homogeneity on the effectiveness of partition based testing. In these experiments failure rates of hidden subdomains are permitted to vary uniformly from 1 to 0.2 (low homogeneity) and results are compared to the case where they varied from 1 to 0.9 (high homogeneity). The largest impact of low homogeneity is found to be only a 22 percent decrease in the effectiveness of partition based testing. Most of the time experiments do not

show that homogeneity is an important factor which impact partition based testing effectiveness.

Besides conventional failure rate model used by Duran and Ntafos [DN84], Hamlet and Taylor also investigate a comparison between partition based and random based testing by the so-called *Valiant's Model* [Val84]. The motivation of this study is that faults are uniformly distributed over the state space of the program code, not over its input space. Valid partitions are therefore those that result from reflecting uniform coverage of program states into the input domain where testing is done. Valiant's Model does not allow to calculate such partition but it allows to relate the number of test data to the probability of missing a failure. Thus, for a given probability of missing a failure, it is possible to compare the number of test test data for both random based and partition based testing. Experimental results indicate that random based testing outperforms partition based testing many times.

Experiments performed in the contribution of Hamlet and Taylor confirm conclusion of Duran and Ntafos: partition based and random based testing are of almost equal value with respect to their ability to detect faults. Hamlet and Taylor explore the impact of homogeneity of subdomains on the ability to detect faults of partition based testing. They are not able to show that homogeneity is an important factor.

Ntafos [Nta98] presents further comparisons between random based and partition based testing. Additionally, the expected cost of failures is taken into account as a way to evaluate the effectiveness of testing strategies. A comparison is made between random based testing and proportional partition based testing. The latter is a partition based testing method where the number of allocated test data for each subdomain depends on the probability that a chosen test data falls into this subdomain. Shortly, the ratio between the number of selected test data for two arbitrary subdomains is equal to the ratio between probabilities that a test data falls into these subdomains.

First of all the power of proportional partition based testing is investigated. A problem here is that occurrences of rare special conditions (subdomains with low probability that randomly chosen test data fall into them) require a large number of test data. Suppose that an input domain is divided into two subdomains and one of them corresponds to a rare special condition which occurs once in a million runs. Then proportional partition based testing would require a total of $1,000,001$ test data to test a program that consist of a single *IF* statement. It is also argued that if the number of required test data grows, proportional partition based testing allocates test data which are the same as randomly selected test data. Thus, even though some experiments show that proportional partition based testing performs at least as well as random based testing, the difference between the respective performances tends to zero while the number of test data grows. Simulation experiments in which P_r and P_p are compared are presented. The allocation of test data in each subdomain is parameterized by the probability that a randomly chosen test data fall into this subdomain. Different failure rates, number of subdomains and test data are used. None of the experiments allows to conclude that one method is better than the other.

Comparisons between proportional partition based, partition based, and random based approaches with regard to the cost of missing a failure are also provided. The measure used is given by the expression $\Sigma c_i(1-\theta_i)^{n_i}$, where for each subdomain D_i c_i is the cost of a failure for test data in D_i, θ_i is the failure rate for D_i, and n_i is the number of test data out of D_i. For various values of k and n, sample simulation results are given that compare proportional partition based, partition based, and random based testing. Random probabilities are assigned to each subdomain. The only interesting result is that uniform partition based testing performs better than the other two strategies.

Fundamental Approaches

Gutjahr [Gut99] proposes a probabilistic approach: in contrast to the previously introduced papers, the contribution is based on mathematical proofs. The mathematical framework is obtained by slightly modifying the one used by Duran and Ntafos [DN84]. These modifications are motivated as follows: from a pragmatic point of view, neither the domain of failure nor the failure rate are known. Therefore the deterministic variables θ and θ_i are considered to be random variables associated to the probability distributions. These probability distributions are supposed to be deduced from knowledge of experts of the domain of interest. This knowledge includes the type of program, its size, the programming language used, etc. Thus θ_i and θ are replaced by $\overline{\theta}_i = E(\theta_i)$ and $\overline{\Theta} = E(\theta)$, where E is the mathematical expectation for the distribution. In this context, the probability of selecting at least one test data which reveals a fault is expressed as follows:

$$\overline{P}_p = E(1 - \prod_{i=1}^{k} (1-\theta_i)) \qquad \text{(for partition based testing), and}$$
$$\overline{P}_r = E(1 - (1-\theta)^k) \qquad \text{(for random based testing).}$$

The probabilities depend on a class of programs and models of a given domain and no longer on the program itself. Different results led the authors to draw the following conclusions:

- If no particularly error prone subdomain is identified before testing and if finding out the failure rate of one subdomain does not change estimations of failure rates in other subdomains, then partition based testing techniques for such a partition have a higher probability to detect errors than random based testing techniques. If the failure rate in each subdomain is close to the overall failure rate, fault detection probabilities for both testing techniques are nearly equivalent.
- Under the same assumptions than described above, if the input domain is partitioned in k subdomains and the same number of test data is selected out of each domain, then the fault detection probability of partition based

testing can be up to k times higher than that of random based testing. This is the case whenever:

(a) There are many small subdomains and only one (or a few) large subdomain(s).

(b) Homogeneous subdomains contain either mostly inputs that are correctly processed or essentially inputs that are failure causing.

All results presented in this paper are based on strong assumptions on failure rates and distribution of probabilities: One can not deduce a fundamental superiority of partition based testing over random based testing. However, the author claims that there are arguments for the conjecture that, in some practical applications, both conditions (a) and (b) are at least approximately satisfied. The first argument is that most of structural partition based testing techniques define partitions on the basis of predicates used in the program. These introduce extremely unbalanced subdomain sizes. As a result, condition (a) is lucky enough to be almost true. Concerning condition (b), it is argued that reasonable subdivision techniques bundle up inputs to subdomains that are processed by the program in a similar way. In such context, if one input of a subdomain is recognized as failure causing, this increases the probability that the other inputs are also failure causing. Conversely, this probability is decreased if an input is recognized as correctly processed.

Notes All contributions show that we know very little about the comparison between random based and partition based testing with regard to their respective ability to detect faults. Independently from the technical background (simulation, theoretical approaches), the presented results and conclusion are based on strong assumptions on failure rates. It is difficult to judge the relevance of these assumptions with regard to real failure rates. Random based testing seems to be the most valuable technique to test reliability of software. This is due to the fact that random selection of test data makes no assumption on the inputs. In contrast, partition based selection constrains relations between inputs. Thus, selected test suites have great chances to be non-representatives for usual uses of the software. If one wants to constrain test suites while addressing reliability, the constraints should be based on operational profiles rather than on structure of the model. This increases chances to run test data which are representatives of real use cases. Nevertheless, partition based testing techniques have great value. In particular, it is known that in practice they are the only ones that tackle efficiently the problem of specific fault detection. For example logical faults or boundary faults can be efficiently analyzed by these kinds of approaches. Unfortunately, the failure rate model does not allow to capture the notion of specific fault (common mistakes made by programmers). Thus, this model can not be used to ground theoretically this fact. Contributions allowing to define models that could take into account this notion of specific faults would be of great value. This would allow to compare partition based testing and random based testing with regard to their abilities to detect these specific faults. Concerning the nature of systems under test, all contributions presented here deal with non-reactive systems. Contributions allowing to relate partition based testing and random based testing for reactive systems would be an interesting prospect. However the failure rate model should be adapted to take into account infinite runs.

11.4.2 Structural Criteria and Ability to Detect Faults

In this section, we compare the ability to detect faults for different testing methods involving structural coverage criteria to select test suites. We present a contribution by Frankl and Weyuker [FW93]. They propose to define relations between criteria and to study, for each of these relations, what knowing that a criterion C_1 is in relation with a criterion C_2 tells us about their respective ability to detect faults. One of the most well known way to compare two coverage criteria is the **subsume relation**. A criterion C_1 subsumes a criterion C_2 if and only if for any program and associated model, C_1 is satisfied by a test suite T implies C_2 is satisfied by T. The subsume relation compares constraints imposed by criteria to select test suites. In contrast, relations proposed by Frankl and Weyuker only compares partition induced by criteria. This allows to compare fault detection abilities of criteria by different assumptions on the test data selection process. These assumptions are made explicit in the way fault detection ability is measured. Frankl and Weyuker propose three different measures. We note $\mathcal{SD}_C(P, M) = \{D_1, D_2, \dots, D_k\}$ the partition induced by a given criterion C for a given program P and associated model M. For $i \in \{1, \dots, k\}$, we denote $d_i =\mid D_i \mid$ and m_i the number of failure causing inputs in D_i. The measures proposed by Frankl and Weyuker are:

- $M_1(C, P, M) = \underset{1 \leq i \leq k}{max} (\frac{m_i}{d_i})$ measures to what extent failure causing inputs are concentrated at subdomains. The only assumption made on the test data selection process is that at least one test data is selected in each subdomain. With this assumption, $M1(C, P, M)$ is a lower bound of the probability that a test suite will expose at least one fault.
- $M_2(C, P, M) = 1 - \prod_{i=1}^{k} (1 - \frac{m_i}{d_i})$ measures the exact probability that an adequate test suite exposes at least one fault, assuming that the test data selection process requires exactly one selection per subdomain.
- $M_3(C, P, M, n) = 1 - \prod_{i=1}^{k} (1 - \frac{m_i}{d_i})^n$ measures the exact probability that an adequate test suite exposes at least one fault, provided that the test data selection process requires n selections per subdomain.

For each relation R defined between criteria, for every program P and every model M, the authors investigate the following questions:

(A) Does $R(C_1, C_2)$ imply $M_1(C_1, P, M) \geq M_1(C_2, P, M)$?
(B) Does $R(C_1, C_2)$ imply $M_2(C_1, P, M) \geq M_2(C_2, P, M)$?
(C) Does $R(C_1, C_2)$ imply $M_3(C_1, P, M, 1) \geq M_3(C_2, P, M, n)$ where $n = \frac{|\mathcal{SD}_{C_1}(P,M)|}{|\mathcal{SD}_{C_2}(P,M)|}$?

Let us comment the last question. One problem with using M_2 as a measure is that one criterion C_1 may divide the domain into k_1 subdomains while another criterion C_2 divides the domain into k_2 subdomains where $k_1 > k_2$. Then, M_2 gives C_1 an unfair advantage since C_1 will require k_1 test data while C_2 will only require k_2 test data. M_3 allows to overcome this problem by comparing $M_3(C_1, P, M, 1)$ and $M_3(C_2, P, M, \frac{k_1}{k_2})$.

We now introduce five relations defined by Frankl and Weyuker.

The Narrows Relation (1)

C_1 **narrows** C_2 **for** (P, M) if for every subdomain $D \in \mathcal{SD}_{C_2}(P, M)$ there is a subdomain $D' \in \mathcal{SD}_{C_1}(P, S)$ such that $D' \subseteq D$. If for every (P, S) C_1 narrows C_2, one says that C_1 **universally narrows** C_2.

Example: We consider a program P whose input domain is the set of integers between $-N$ and $+N$, with $N > 1$. C_1 is a criterion that requires the selection of at least one test data that is 0 and at least one test data that is different of 0. C_2 is a criterion that requires the selection of at least one test data that is greater than or equal to 0 and at least one test data that is less or equal to 0. Therefore C_1 uses two subdomains: $D_1 = \{0\}$ and $D_2 = \{x \mid -N \leq x \leq N \wedge x \neq 0\}$. C_2 uses two subdomains: $D_3 = \{x \mid 0 \leq x \leq N\}$ and $D_4 = \{x \mid -N \leq x \leq 0\}$. Since D_3 and D_4 both contain D_1, C_1 narrows C_2.

Relation to the subsume relation: Consider that for any (P, M), C_1 and C_2 give rise to the same set of subdomains, but C_2 requires selection of two test data out of each subdomain whereas C_1 only requires selection of one test data out of each subdomain. Trivially, C_1 universally narrows C_2. However, C_1 does not subsume C_2, since a test suite consisting of one element out of each subdomain is C_1-adequate but not C_2-adequate. However, we have the following theorem:

Theorem 11.1. *Let C_1 and C_2 be two criteria which explicitly require the selection of at least one test data out of each subdomain, then C_1 subsumes C_2 if and only if C_1 universally narrows C_2.*

Proof. Assume C_1 universally narrows C_2. Let T be a test suite that is C_1-adequate for some program P and model M. T requires the selection of at least one test data out of each subdomain of $\mathcal{SD}_{C_1}(P, M)$. Thus, since each subdomain in $\mathcal{SD}_{C_2}(P, M)$ is a superset of some subdomains belonging to $\mathcal{SD}_{C_1}(P, M)$, T is a test suite which requires the selection of at least one test data out of each subdomain of $\mathcal{SD}_{C_2}(P, M)$. We conclude that C_1 subsumes C_2.

Conversely, assume C_1 does not universally narrow C_2. There exists a program P and a model M such that some subdomain $D \in \mathcal{SD}_{C_2}(P, M)$ is not a superset of any subdomain of $\mathcal{SD}_{C_1}(P, M)$. Thus for each $D' \in \mathcal{SD}_{C_1}(P, M)$, $D' - D \neq \emptyset$. Let T be a test suite which requires the selection of exactly one test data out of $D' - D$ for each $D' \in \mathcal{SD}_{C_1}(P, M)$. T is C_1-adequate but not C_2-adequate. So C_1 does not subsume C_2.

Relation to the three measures: We consider questions (A), (B) and (C) introduced in the introduction of this section. In order to answer these questions we consider the following example. Domain D of a program P is $\{0,1,2\}$. M is the model associated to P. We suppose that $\mathcal{SD}_{C_1}(P,M) = \{\{0,1\},\{0,2\}\}$ and $\mathcal{SD}_{C_2}(P,M) = \{\{0,1\},\{0,1,2\}\}$. Since $\{0,1\} \subseteq \{0,1\}$ and $\{0,2\} \subseteq \{0,1,2\}$, C_1 narrows C_2.

(A)Does C_1 narrow C_2 imply $M_1(C_1,P,M) \geq M_1(C_2,P,M)$?

We answer in the negative. Suppose that only 1 and 2 are failure causing: $M_1(C_1,P,M) = \frac{1}{2}$ while $M_1(C_2,P,M) = \frac{2}{3}$ and thus $M_1(C_1,P,M) < M_1(C_2,P,M)$.

(B) Does C_1 narrow C_2 imply $M_2(C_1,P,M) \geq M_2(C_2,P,M)$?

We answer in the negative. Suppose that only 1 and 2 are failure causing: $M_2(C_1,P,M) = 1-(1-\frac{1}{2})(1-\frac{1}{2}) = \frac{3}{4}$, $M_2(C_2,P,M) = 1-(1-\frac{1}{2})(1-\frac{2}{3}) = \frac{5}{6}$ and thus $M_2(C_1,P,M) < M_2(C_2,P,M)$.

(C) Does C_1 narrow C_2 imply $M_3(C_1,P,M,1) \geq M_3(C_2,P,M,n)$ where $n = \frac{|\mathcal{SD}_{C_1}(P,M)|}{|\mathcal{SD}_{C_2}(P,M)|}$?

We answer in the negative. Since in our example $n = 1$, question (C) is equivalent to Question (B).

As stated above, the narrows relation does not necessarily induce a better fault detection ability for each of the three measures considered. Thus from Theorem 11.1, it is naturally deduced that the subsume relation does not necessarily induce a better fault detection ability for each of the three measures considered.

The Covers Relation (2) The narrows relation can be strengthened to impose that each subdomain of the partition induced by C_2 can be expressed as a union of some subdomains of the partition induced by C_1. This gives rises to the following definition:

C_1 **covers** C_2 **for** (P,M) if for every subdomain $D \in \mathcal{SD}_{C_2}(P,M)$ if there is a non-empty collection of subdomains $\{D_1,\ldots,D_n\}$ belonging to $\mathcal{SD}_{C_1}(P,M)$ such that $D_1 \cup \cdots \cup D_n = D$. If for every (P,M) C_1 covers C_2, one says that C_1 **universally covers** C_2.

Example: We consider criteria C_1 and C_2 and Program P used to illustrate the narrow relation. Since D_3 and D_4 both contain D_1, C_1 narrows C_2. However, since $D_3 \neq D_1$, $D_3 \neq D_2$ and $D_3 \neq D_1 \cup D_2$, C_1 does not cover C_2.

In contrast, we consider a program P' whose input domain are the integers between $-N$ and N $(N > 0)$. Suppose that criterion C'_1 induces a partition into two subdomains: $D'_1 = \{x \mid -N+1 \leq x \leq N\}$ and $D'_2 = \{x \mid -N \leq x \leq N-1\}$ and that criterion C'_2 induces a partition into one subdomain: $D'_3 = \{x \mid -N \leq x \leq N\}$. Since $D'_3 = D'_1 \cup D'_2$, C'_1 covers C'_2.

Relation to the subsume relation: The following theorem is obvious:

Theorem 11.2. *Let C_1 and C_2 be two criteria. C_1 universally covers C_2 implies C_1 universally narrows C_2.*

From Theorem 11.1, we immediately have the following theorem:

Theorem 11.3. *Let C_1 and C_2 be two criteria which explicitly require the selection of at least one test data out of each subdomain, then C_1 universally covers C_2 implies C_1 subsumes C_2.*

Relation to the three measures: In order to answer questions (A), (B) and (C), we consider the following example. Domain D of a program P is $\{0,1,2,3\}$. M is the model associated to P. We suppose that $\mathcal{SD}_{C_1}(P,M) = \{\{0,1\},\{1,2\},\{3\}\}$ and $\mathcal{SD}_{C_2}(P,M) = \{\{0,1,2\},\{1,2,3\}\}$. Since $\{0,1,2\} = \{0,1\} \cup \{1,2\}$ and $\{1,2,3\} = \{1,2\} \cup \{3\}$, C_1 covers C_2.

(A) Does C_1 cover C_2 imply $M_1(C_1,P,M) \geq M_1(C_2,P,M)$?

We answer in the negative. Suppose that only 0 and 2 are failure causing: $M_1(C_1,P,M) = \frac{1}{2}$ while $M_1(C_2,P,M) = \frac{2}{3}$ and thus $M_1(C_1,P,M) < M_1(C_2,P,M)$.

(B) Does C_1 cover C_2 imply $M_2(C_1,P,M) \geq M_2(C_2,P,M)$?

We answer in the negative. Suppose that only 2 is failure causing. $M_2(C_1,P,M) = 1-(1-\frac{1}{2}) = \frac{1}{2}$ while $M_2(C_2,P,M) = 1-(1-\frac{1}{3})(1-\frac{1}{3}) = \frac{5}{9}$ and thus $M_2(C_1,P,M) < M_2(C_2,P,M)$.

(C) Does C_1 cover C_2 imply $M_3(C_1,P,M,1) \geq M_3(C_2,P,M,n)$ where $n = \frac{|\mathcal{SD}_{C_1}(P,M)|}{|\mathcal{SD}_{C_2}(P,M)|}$?

We answer in the negative. It is obvious that for $n \geq 1$, $M_3(C_2,P,M,n) \geq M_2(C_2,P,M)$. Now $M_3(C_1,P,M,1) = M_2(C_1,P,M)$. Since we have proven that there exists P and M such that $M_2(C_1,P,M) < M_2(C_2,P,M)$, we deduce that there exists P and M such that $M_3(C_1,P,M,1) < M_3(C_2,P,M,n)$.

As for the narrows relation, the covers relation does not necessarily induce a better fault detection ability for each of the three measures considered. Thus, ensuring that each subdomain of the partition induced by C_2 can be expressed as a union of some subdomains of the partition induced by C_1 is not sufficient to gain a superiority of C_1 over C_2 (at least with respect to the three measures considered).

The Partitions Relation (3) The cover relation is then strengthened to ensure that for each subdomain of the partition induced by C_2, a partition consisting of pairwise disjoint subdomains induced by C_1 may be defined.

C_1 **partitions** C_2 **for** (P, M) if for every subdomain $D \in \mathcal{SD}_{C_2}(P, M)$ there is a non-empty collection of pairwise disjoint subdomains $\{D_1, \ldots, D_n\}$ belonging to $\mathcal{SD}_{C_1}(P, M)$ such that $D_1 \cup \cdots \cup D_n = D$. If for every (P, M) C_1 partitions C_2, one says that C_1 **universally partitions** C_2.

Example: Let us consider criteria C_1' and C_2' and the program P' used to illustrate the covers relation. Since $D_3' = D_1' \cup D_2'$ C_1' covers C_2' and, since $D_1' \cap D_2' \neq \emptyset$, C_1' does not partition C_2'.

In contrast we consider a program P'' whose input domain are the integers between $-N$ and N $(N > 0)$. Suppose that a criterion C_1'' induces a partition into two subdomains: $D_1'' = \{x \mid 0 \leq x \leq N\}$ and $D_2'' = \{x \mid -N \leq x < 0\}$ and that criterion C_2'' induces a partition into one subdomain: $D_3'' = \{x \mid -N \leq x \leq N\}$. Since $D_3'' = D_1'' \cup D_2''$ and $D_1'' \cap D_2'' = \emptyset$, C_1'' partitions C_2''.

Relation to the subsume relation: The following theorem is obvious:

Theorem 11.4. *Let C_1 and C_2 be two criteria. C_1 universally partitions C_2 implies C_1 universally covers C_2.*

From Theorem 11.2, we have the following theorem:

Theorem 11.5. *Let C_1 and C_2 be two criteria. C_1 universally partitions C_2 implies C_1 universally narrows C_2.*

From Theorem 11.1 we have the following theorem:

Theorem 11.6. *Let C_1 and C_2 be two criteria which explicitly require the selection of at least one test data out of each subdomain, then C_1 universally partitions C_2 implies C_1 subsumes C_2.*

Relation to the three measures:

(A) Does C_1 partition C_2 imply $M_1(C_1, P, M) \geq M_1(C_2, P, M)$?

The answer is positive, as stated in the following theorem:

Theorem 11.7. *If C_1 partitions C_2 for a program P and a model M then $M_1(C_1, P, M) \geq M_1(C_2, P, M)$.*

Proof. Let $D_0 \in \mathcal{SD}_{C_2}(P, M)$. Let $D_1, \ldots, D_n$ be disjoint subdomains belonging to $\mathcal{SD}_{C_1}(P, M)$ such that $D_0 = D_1 \cup \cdots \cup D_n$. Then $m_0 = m_1 + \cdots + m_n$ and $d_0 = d_1 + \cdots + d_n$. Thus $max_{i=1}^{n}(\frac{m_i}{d_i})$ is minimized when each $\frac{m_i}{d_i} = \frac{m_0}{d_0}$. So $max_{i=1}^{n}(\frac{m_i}{d_i}) \geq \frac{m_0}{d_0}$ and therefore $M_1(C_1, P, M) \geq M_1(C_2, P, M)$.

(B) Does C_1 partitions C_2 imply $M_2(C_1, P, M) \geq M_2(C_2, P, M)$?

We answer in the negative. Domain D of a program P is $\{0, 1, 2, 3\}$. M is the model associated to P. We suppose that $\mathcal{SD}_{C_1}(P, M) = \{\{0\}, \{1, 2\}, \{3\}\}$

and $\mathcal{SD}_{C_2}(P,M) = \{\{0,1,2\},\{1,2,3\}\}$. Since $\{0,1,2\} = \{0\} \cup \{1,2\}$, $\{0\} \cap \{1,2\} = \emptyset$, $\{1,2,3\} = \{1,2\} \cup \{3\}$, and $\{1,2\} \cap \{3\} = \emptyset$, C_1 partitions C_2. Suppose that only 2 is failure causing. $M_2(C_1,P,M) = 1 - (1 - \frac{1}{2}) = \frac{1}{2}$ while $M_2(C_2,P,M) = 1 - (1 - \frac{1}{3})(1 - \frac{1}{3}) = \frac{5}{9}$ and thus $M_2(C_1,P,M) < M_2(C_2,P,M)$.

(C) Does C_1 partitions C_2 imply $M_3(C_1,P,M,1) \geq M_3(C_2,P,M,n)$ where $n = \frac{|\mathcal{SD}_{C_1}(P,M)|}{|\mathcal{SD}_{C_2}(P,M)|}$?

We answer in the negative. For $n \geq 1$, $M_3(C_2,P,M,n) \geq M_2(C_2,P,M)$. Now $M_3(C_1,P,M,1) = M_2(C_1,P,M)$. Since we have proven that there exists P and M such that $M_2(C_1,P,M) < M_2(C_2,P,M)$, we deduce that there exists P and M such that $M_3(C_1,P,M,1) < M_3(C_2,P,M,n)$.

The partitions relation ensures a better fault detection ability for measure M_1 (if C_1 partitions C_2 then C_1 is better at detecting faults than C_2 with regard to M_1) but not necessarily for the two others. Recall that measure M_1 is a lower bound of the probability that a test suite will expose at least one fault. Thus Theorem 11.7 only ensures that if C_1 partitions C_2, this lower bound of the probability that a test suite will expose at least one fault is greater for C_1 than for C_2. Another intuitive way to understand this result is that a partition induced by C_1 concentrates more failure causing inputs in one specific subdomain than a partition induced by C_2 does.

The Properly Covers Relation (4) In order to obtain a better fault detection ability regarding measure M_2, the cover relation is specialized so that each subdomain D of the partition $\mathcal{SD}_{C_1}(P,S)$ is used only once to define a partition of a subdomain of $\mathcal{SD}_{C_2}(P,S)$.

Let us note $\mathcal{SD}_{C_1}(P,S) = \{D^1_1,\ldots,D^1_m\}$ and $\mathcal{SD}_{C_2}(P,S) = \{D^2_1,\ldots,D^2_n\}$.

C_1 **properly covers** C_2 **for** (P,M) if there is a multi-set

$$\mathcal{M} = \{D^1_{1,1},\ldots D^1_{1,k_1},\ldots,D^1_{n,1},\ldots D^1_{n,k_n}\}$$

such that $\mathcal{M} \subseteq \mathcal{SD}_{C_1}(P,M)$ and $D^2_i = D^1_{i,1} \cup \cdots \cup D^1_{i,k_i}$ for $i \in \{1,\ldots,n\}$. If for every (P,M) C_1 properly covers C_2, one says that C_1 **universally properly covers** C_2.

Example: Consider a program P with integer input domain $\{x \mid 0 \leq x \leq 3\}$, and criteria C_1 and C_2 such that $\mathcal{SD}_{C_1} = \{D_a,D_b,D_c\}$ and $\mathcal{SD}_{C_2} = \{D_d,D_e\}$, where $D_a = \{0\}$, $D_b = \{1,2\}$, $D_c = \{3\}$, $D_d = \{0,1,2\}$, and $D_e = \{1,2,3\}$. Then $D_d = D_a \cup D_b$ and $D_e = D_c \cup D_b$, so C_1 covers (and also partitions) C_2. However, C_1 does not properly cover C_2 because subdomain D_b is needed in to cover both D_d and D_e, but only occurs once in the multi-set $\mathcal{SD}_{C_1}$.

On the other hand consider criterion C_3 where $\mathcal{SD}_{C_3} = \{D_a,D_b,D_b,D_c\}$. C_3 does properly cover C_2. It is legitimate to use D_b twice to cover both D_d and D_e, since it occurs twice in $\mathcal{SD}_{C_3}$.

Relation to the subsume relation: The following theorem is obvious:

Theorem 11.8. *Let C_1 and C_2 be two criteria. C_1 universally properly covers C_2 implies C_1 universally covers C_2.*

From Theorem 11.2 we have the following theorem:

Theorem 11.9. *Let C_1 and C_2 be two criteria. C_1 universally properly covers C_2 implies C_1 universally narrows C_2.*

From Theorem 11.1 we have the following theorem:

Theorem 11.10. *Let C_1 and C_2 be two criteria which explicitly require the selection of at least one test data out of each subdomain, then C_1 universally properly covers C_2 implies C_1 subsumes C_2.*

Relation to the three measures:

(A) Does C_1 properly cover C_2 imply $M_1(C_1, P, M) \geq M_1(C_2, P, M)$?

We answer in the negative. Domain D of a program P is $\{0, 1, 2, 3\}$. M is the model associated to P. We suppose that $\mathcal{SD}_{C_1}(P, M) = \{\{0, 1\}, \{1, 2\}, \{3\}\}$ and $\mathcal{SD}_{C_2}(P, M) = \{\{0, 1, 2\}, \{3\}\}$. Since $\{0, 1, 2\} = \{0, 1\} \cup \{1, 2\}$ and $\{3\}$ is an element of $\mathcal{SD}_{C_1}(P, M)$, C_1 properly covers C_2. We suppose that only 0 and 2 are failure causing. Therefore, $M_1(C_1, P, M) = \frac{1}{2}$ and $M_1(C_2, P, M) = \frac{2}{3}$. We conclude $M_1(C_1, P, M) < M_1(C_2, P, M)$.

(B) Does C_1 properly cover C_2 imply $M_2(C_1, P, M) \geq M_2(C_2, P, M)$?

The answer is positive as stated in the following theorem:

Theorem 11.11. *If C_1 properly covers C_2 for program P and model M, then $M_2(C_1, P, M) \geq M_2(C_2, P, M)$.*

Proof. The proof requires some intermediate lemma.

Lemma 11.12. *Assume $d_1, d_2 > 0$, $0 \leq x \leq d_1$, $0 \leq x \leq d_2$, $0 \leq m_1 \leq d_1 - x$ and $0 \leq m_2 \leq d_2 - x$. Then we have:*

$$\tfrac{m_1+m_2}{d_1+d_2-x} \leq (1 - (1 - \tfrac{m_1}{d_1})(1 - \tfrac{m_2}{d_2})).$$

Proof. Since

$$(1 - (1 - \tfrac{m_1}{d_1})(1 - \tfrac{m_2}{d_2})) = \tfrac{m_1 d_2 + m_2 d_1 - m_1 m_2}{d_1 d_2}$$

it suffices to show that

$$\begin{aligned} 0 &\leq (m_1 d_2 + m_2 d_1 - m_1 m_2)(d_1 + d_2 - x) - (m_1 + m_2)(d_1 d_2) \\ &= m_2 d_1 (d_1 - m_1 - x) + m_1 d_2 (d_2 - m_2 - x) + m_1 m_2 x \end{aligned}$$

This follows immediately from the assumption that $(d_1 - m_1 - x)$, $(d_2 - m_2 - x)$, d_i, m_i, and x are all non negative.

Lemma 11.13. *Let $D_3 = D_1 \cup D_2$. Then $\frac{m_3}{d_3} < (1 - (1 - \frac{m_1}{d_1})(1 - \frac{m_2}{d_2}))$.*

Proof. For any set $\{D_1, \ldots, D_n\}$ of subdomains, let us note $f(D_1, \ldots, D_n) = \prod_{i=1}^{k} (1 - \frac{m_i}{d_i})$.

We want to show that: $1 - f(D_3) < 1 - f(D_1, D_2)$ or that $f(D_3) > f(D_1, D_2)$.

We start by showing that for given values d_3 and m_3, the value of $f(D_1, D_2)$ is maximized when $D_1 \cap D_2$ contains as few failure causing inputs as possible. This is clear intuitively, since points in the intersection are more likely to be selected. Thus when as many of them as possible are not failure causing, the probability to select an input which does not cause a fault is maximal (that is if $f(D_1, D_2)$ is maximal).

Formally, let $D_a = D_3 - D_2$, $D_b = D_3 - D_1$ and $D_c = D_1 \cap D_2$. Let d_a, d_b, d_c and x_a, x_b, x_c be the size and the number of inputs which does not cause a fault of D_a, D_b and D_c respectively. The following equation holds:

$$f(D_1, D_2) = (\tfrac{x_a + x_c}{d_a + d_c})(\tfrac{x_b + x_c}{d_b + d_c})$$

Suppose it is possible to swap one non failure causing input out of D_a with one failure causing input of D_c. Let us call D_1' and D_2' the subdomains obtained from D_1 and D_2 by applying this operation. Doing so leaves the values d_a, d_b, d_c, and x_b unchanged but decrements x_a and increments x_c, yielding

$$f(D_1', D_2') = (\tfrac{x_a - 1 + x_c + 1}{d_a + d_c})(\tfrac{x_b + x_c + 1}{d_b + d_c}) > f(D_1, D_2).$$

Similarly, swapping a non failure causing input of D_b with a failure causing input of D_c leads to two subdomains D_1'' and D_2'' such that $f(D_1'', D_2'') > f(D_1, D_2)$.

Thus, to prove the lemma, it suffices to consider the following two cases.

Case 1: D_c consists entirely of non failure causing inputs. In this case, letting $x = d_c = \mid D_1 \cap D_2 \mid$, the hypotheses of Lemma 11.12 are satisfied, so:

$$\tfrac{m_1 + m_2}{d_1 + d_2 - d_c} \leq (1 - (1 - \tfrac{m_1}{d_1})(1 - \tfrac{m_2}{d_2})) \text{ holds.}$$

Since $m_3 = m_1 + m_2$ and $d_3 = d_1 + d_2 - d_c$, it gives the desired result.

Case 2: D_a and D_b consist entirely of failure causing inputs. We want to show that $f(D_3) - f(D_1, D_2) \geq 0$, where

$$f(D_3) = \tfrac{x_c}{d_a + d_b + d_c} \text{ and } f(D_1, D_2) = (\tfrac{x_c}{d_a + d_c})(\tfrac{x_c}{d_b + d_c}).$$

It suffices to show

$$0 \leq x_c((d_a + d_c)(d_b + d_c) - x_c(d_a + d_b + d_c))$$

that is

$$0 \leq x_c(d_a(d_b + d_c - x_c) + d_b(d_c - x_c) + d_c(d_c - x_c))$$

But this follows immediately from the fact that $0 \leq x_c \leq d_c$.

Lemma 11.14. *Let $D = D_1 \cup \cdots \cup D_n$. Then $f(D) \geq f(D_1, \ldots, D_n)$.*

Proof. Proof by induction on n. The base case, $n = 1$ is trivial. Now assuming that

$$D = D_1 \cup \cdots \cup D_k \Rightarrow f(D) \geq f(D_1, \ldots, D_k),$$

we want to show that

$$D' = D_1 \cup \cdots \cup D_{k+1} \Rightarrow f(D) \geq f(D_1, \ldots, D_{k+1}).$$

Since $D' = D \cup D_{k+1}$, from Lemma 11.13 we deduce $f(D') \geq f(D, D_{k+1})$. Now from the definition of f, $f(D, D_{k+1}) = f(D)f(D_{k+1})$. We deduce $f(D') \geq f(D)f(D_{k+1})$. From the inductive hypothesis, we can write $f(D) \geq f(D_1, \ldots, D_k)$.

Thus we deduce $f(D') \geq f(D_1, \ldots, D_k)f(D_{k+1})$. From the definition of f we conclude $f(D') \geq f(D_1, \ldots, D_{k+1})$.

We now prove Theorem 11.11. Assume C_1 properly covers C_2 for a program P and model M. Let us denote $\mathcal{SD}_{C_1}(P, M) = \{D_1^1, \ldots, D_m^1\}$ and $\mathcal{SD}_{C_2}(P, M) = \{D_1^2, \ldots, D_n^2\}$. Let $\mathcal{M} = \{D_{1,1}^1, \ldots D_{1,k_1}^1, \ldots, D_{n,1}^1, \ldots D_{n,k_n}^1\}$ be a set such that such that $\mathcal{M} \subseteq \mathcal{SD}_{C_1}(P, M)$ and $D_i^2 = D_{i,1}^1 \cup \cdots \cup D_{i,k_i}^1$ for $i \in \{1, \ldots, n\}$.

From the definition of f we can write

$$f(D_1^2, \ldots, D_n^2) = \prod_{i \leq n} f(D_i^2).$$

From Lemma 11.14, we have

$$\prod_{i \leq n} f(D_i^2) \geq \prod_{i \leq n,} \prod_{j \leq k_i,} f(D_{i,j}^1).$$

Since for all $i \leq m$ we have $f(D_i^1) \leq 1$, we deduce:

$$\prod_{j \leq k_i,} f(D_{i,j}^1) \geq f(D_1^1, \ldots, D_m^1).$$

Thus we deduce:

$$f(D_1^2, \ldots, D_n^2) \geq f(D_1^1, \ldots, D_m^1).$$

We conclude the proof:

$$M_2(C_1, P, M) \geq M_2(C_2, P, M).$$

(C) Does C_1 properly cover C_2 imply $M_3(C_1, P, M, 1) \geq M_3(C_2, P, M, n)$ where $n = \frac{|\mathcal{SD}_{C_1}(P,M)|}{|\mathcal{SD}_{C_2}(P,M)|}$?

We answer in the negative. Assume that the domain of a program P is $\{0,1,2\}$. Let us note M the model associated to P. We suppose that $\mathcal{SD}_{C_1}(P,M) = \{\{0,1\},\{0,2\}\}$ and $\mathcal{SD}_{C_2}(P,M) = \{0,1,2\}$. Since $\{0,1,2\} = \{0,1\} \cup \{0,2\}$, C_1 properly covers C_2. Suppose that only 1 is failure causing then $M_3(C_1,P,M,1) = 1 - (1 - \frac{1}{2}) = \frac{1}{2}$. Now, $M_3(C_2,P,M,2) = 1 - (1 - \frac{1}{3})^2 = 1 - (\frac{2}{3})^2 = \frac{5}{9} > M_3(C_1,P,M,1)$.

The properly covers relation ensures a better fault detection ability for measure M_2 (if C_1 properly covers C_2 then C_1 is better at detecting faults than C_2 with regard to M_2) but not necessarily for the two others. Recall that measure M_2 measures the exact probability that an adequate test suite exposes at least one fault, assuming that the test data selection process requires exactly one selection per subdomain. However we answered to question (C) in the negative. This means that if the same number of test data is used for C_1 and for C_2, then nothing ensures that C_1 will be better at detecting faults than C_2 (for measure M_3). Note also that if C_1 properly covers C_2, then nothing ensures that a partition induced by C_1 concentrates more failure causing inputs in one specific subdomain than a partition induced by C_2 does. This is due to the fact that we answered in the negative to question (A).

The Properly Partitions Relation (5) The properly partitions relation constrains the partitions relation exactly as the properly covers relation constrains the covers relation. Let us note $\mathcal{SD}_{C_1}(P,S) = \{D^1_1,\dots,D^1_m\}$ and $\mathcal{SD}_{C_2}(P,S) = \{D^2_1,\dots,D^2_n\}$.

C_1 **properly partitions** C_2 **for** (P,S) if there is a multi-set

$$\mathcal{M} = \{D^1_{1,1},\dots D^1_{1,k_1},\dots,D^1_{n,1},\dots D^1_{n,k_n}\}$$

such that $\mathcal{M} \subseteq \mathcal{SD}_{C_1}(P,S)$ and $D^2_i = D^1_{i,1} \cup \dots \cup D^1_{i,k_i}$ for $i \in \{1,\dots,n\}$. Moreover, it is required that for each i, collection $\{D^1_{i,1},\dots,D^1_{i,k_i}\}$ is pairwise disjoint. If for every (P,S) C_1 properly covers C_2 for (P,S), one says that C_1 **universally properly partitions** C_2.

Example: Again, consider criteria C_2 and C_3 used to illustrate the properly cover relation. C_3 also properly partitions C_2 since $D_d = D_a \cup D_b$, $D_e = D_c \cup D_b$, $D_a \cap D_b = \emptyset$, $D_c \cup D_b = \emptyset$

Relation to the subsume relation: The two following theorems are obvious:

Theorem 11.15. *Let C_1 and C_2 be two criteria. C_1 universally properly partitions C_2 implies C_1 universally properly covers C_2.*

Theorem 11.16. *Let C_1 and C_2 be two criteria. C_1 universally properly partitions C_2 implies C_1 universally partitions C_2.*

Either from Theorem 11.8 or from Theorem 11.4, we have the following Theorem:

Theorem 11.17. *Let C_1 and C_2 be two criteria. C_1 universally properly partitions C_2 implies C_1 universally covers C_2.*

From Theorem 11.2 we have the following theorem:

Theorem 11.18. *Let C_1 and C_2 be two criteria. C_1 universally properly partitions C_2 implies C_1 universally narrows C_2.*

From Theorem 11.1 we have the following theorem:

Theorem 11.19. *Let C_1 and C_2 be two criteria which explicitly require the selection of at least one test data out of each subdomain, then C_1 universally properly partitions C_2 implies C_1 subsumes C_2.*

Relation to the three measures:

(A) Does C_1 properly partition C_2 imply $M_1(C_1, P, M) \geq M_1(C_2, P, M)$?

The answer is positive, as stated in the following theorem:

Theorem 11.20. *If C_1 properly partitions C_2 for a program P and a model M then $M_1(C_1, P, M) \geq M_1(C_2, P, M)$.*

Proof. From Theorem 11.16 C_1 partitions C_2. Theorem 11.7 allows us to conclude that $M_1(C_1, P, M) \geq M_1(C_2, P, M)$.

(B) Does C_1 properly partitions C_2 imply $M_2(C_1, P, M) \geq M_2(C_2, P, M)$?

The answer is positive, as stated in the following theorem:

Theorem 11.21. *If C_1 properly partitions C_2 for a program P and a model M then $M_2(C_1, P, M) \geq M_2(C_2, P, M)$.*

Proof. From Theorem 11.15 C_1 properly covers C_2. Theorem 11.11 allows us to conclude that $M_2(C_1, P, M) \geq M_2(C_2, P, M)$.

(C) Does C_1 properly partition C_2 imply $M_3(C_1, P, M, 1) \geq M_3(C_2, P, M, n)$ where $n = \frac{|\mathcal{SD}_{C_1}(P,M)|}{|\mathcal{SD}_{C_2}(P,M)|}$?

We answer in the negative. Domain D of a program P is $\{0, 1, 2, 3\}$. M is the model associated to P. We suppose that $\mathcal{SD}_{C_1}(P, M) = \{\{0\}, \{1\}, \{2, 3\}\}$ and $\mathcal{SD}_{C_2}(P, M) = \{0, 1, 2, 3\}$. Since $\{0, 1, 2, 3\} = \{0\} \cup \{1\} \cup \{2, 3\}$ and $\{0\} \cap \{1\} = \{0\} \cap \{2, 3\} = \{1\} \cap \{2, 3\} = \emptyset$, C_1 properly partition C_2. Suppose that only 2 is failure causing. $M_3(C_1, P, M, 1) = 1 - (1 - \frac{1}{2}) = \frac{1}{2}$ while $M_3(C_2, P, M, n) = 1 - (1 - \frac{1}{4})^3 = 1 - (\frac{3}{4})^3 = 1 - \frac{27}{64} = \frac{37}{64}$. Thus $M_3(C_2, P, M, 3) > M_3(C_1, P, M, 1)$

The properly partitions relation ensures a better fault detection ability for measure M_1 and M_2 (if C_1 properly partitions C_2 then C_1 is better at detecting faults than C_2 with regard to M_1 and M_2) but not necessarily for M_3.

Notes Since for most criteria of interest, the universally narrows relation is equivalent to the subsumes relation, one can interpret the presented results by concluding that the subsume relation is a poor basis for comparing criteria. However, it is important to note that the results here are worst case results in the sense that it is only considered whether or not the fact that one criterion subsumes another guarantees improved fault-detecting ability. The question of what C_1 subsuming, or narrowing, or covering, or partitioning C_2 tells us about their relative ability to detect faults in "typical" programs remains open. Moreover, note that the most convincing measure studied is measure M_3, since this measure takes into account the number of test data used. Thus it is possible to compare two criteria for the same number of test data. However none of the relations presented here induces a better fault detection ability for measure M_3.

11.4.3 Remarks

Questions addressed in Section 11.4.1 and Section 11.4.2 can be compared. The way random based testing is modeled in the contributions presented in Section 11.4.1 results in a "partition oriented" view of random based testing. Indeed random based testing can be seen as a partition based technique which partitions the input domain in a single subdomain (the input domain itself). Even-more, random based testing can be seen as a coverage criterion which divides the input domain of a program in one subdomain: the input domain itself. Let us call random criterion this criterion. Consider now any structural criterion. It is easy to see that such a criterion either properly partitions or properly covers the random criterion (depending on the fact that the criterion of interest induces overlapping or non overlapping subdomains). Contributions presented in Section 11.4.1 essentially make the assumption that the same number of test data is used both for random based an partition based testing. Thus comparing random based testing and partition based testing with regard to their ability to detect faults (as expressed in Section presented in Section 11.4.1) is equivalent to associate a criterion C to the partition based testing technique considered, and to compare C with the random criterion with regard to Measure M_3 (as defined in Section 11.4.2). Results introduced in Section 11.4.1 indicates that partition based testing is not better at detecting faults than random based testing. This result is thus totally consistent with the fact that both properly covers and properly partitions relations do not induce a better fault detection ability, with regard to measure M_3.

11.5 Summary

The application of coverage techniques at the model level seems a promising approach. These techniques allow rather easy test selection from executable models, while ensuring (at a certain degree) the coverage of targeted behaviors of the model (e.g. a set of test cases for which all variable definitions of the model are stimulated). Of course, criteria have to be adapted to specification formalisms: for example it makes no sense to talk about data flow criteria for models described in a specification formalism which does not handle variables. However most of the usual criteria can be easily adapted to models, since models' executable aspect makes them "look like programs". Moreover, approaches based on model coverage may be adapted to perform functional testing. This can be done through property coverage by model checking approaches or user profile usages for example. The main point is that this kind of functional testing is still based on coverage considerations, which is very valuable since generated test suites are supposed to cover in a measurable manner behaviors of the model which reflect an abstract scenario (or property).

All these properties make model-coverage-based-testing methods good candidates to detect specific faults at the earliest design level. The strength of coverage approaches relies mainly on their ability to explore in a systematic manner "missing logic" faults: bad treatment of bounds for example. These approaches are the only one to tackle the problem of detecting catastrophic failure causing inputs. However, one must keep in mind that these properties are not sufficient to ensure reliability. In particular, there is no scientific evidence that coverage based testing is better than random testing to reach this purpose. To gain more insight, a further analysis of what is a "typical program under test" is needed, since the usual failure rate model seems unsuitable to provide such evidence. The same problem occurs when one tries to classify criteria with respect to their respective ability to detect faults.

Common belief however seems to be that random testing should systematically complement coverage based approaches. Coverage based approaches should be used to detect specific faults while random approaches aim at providing confidence about programs reliability.