Domain Testing and Linear Fault Detection

Steven J. Zeil

COINS Technical Report 86-38

August 1986

Software Development Laboratory
Computer and Information Science Department
University of Massachusetts
Amherst, Massachusetts 01003

This work was supported by grants DCR-8404217 and DCR-8318776 from the National Science Foundation, 84M103 from Control Data Corporation, and RADC grant F30602-86-C-0006.

Abstract

Domain testing is an approach to software testing that attempts to find errors in the flow of control through a program. Three forms of domain testing have been proposed, all dealing with the detection of linear errors in linear functions. This paper examines the competing forms of domain testing using perturbation analysis, a technique for measuring the error detection capacity of a set of test data. The different forms of domain testing are shown to be closer in error detection ability than had been supposed and may all be considered effective for finding linear errors in linear predicate functions. A simple extension is proposed which allows them to detect linear errors in non-linear predicate functions, at little additional cost. A new algorithm is proposed that shifts the emphasis from linear errors to linear faults, thereby allowing domain testing to take into account the interactions among tests for different paths.

1. Introduction

Because computer software can be applied to such a wide range of applications, the penalties for failing to detect software faults range across a broad spectrum from negligible to disastrous. Consequently, the degree of confidence required from testing procedures will vary widely. While some projects may justifiably proceed with informal, ad hoc testing procedures, others will require far more rigorous testing with the eventual goal of making strong statements about confidence gained in the the program code. Of course, nothing short of exhaustive testing of all possible inputs can provide completely reliable testing for all programs. Consequently, techniques that can be shown to capture useful classes of errors in common classes of programs would be quite valuable.

This paper is concerned with a set of testing strategies, the *domain testing* strategies, intended to catch faults affecting the flow of control through a program – faults that cause the "wrong branch" to be taken during execution. This paper examines the arguments that have been used to justify these strategies and shows why those arguments were invalid. The domain testing strategies are then examined using the perturbation testing criteria [21,22,23] and are shown to satisfy the original claims made regarding their effectiveness. The perturbational analysis also permits the extension of the domain techniques to a wider range of software and permits the construction of an integrated testing strategy capable of detecting most errors in program computations as well as errors in control flow.

Section II provides definitions for the terminology that will be employed throughout this paper. Section III reviews the domain testing methods and Section IV examines the arguments that have been advanced in favor of these methods' use and shows why those arguments may be invalid. Section V presents an error-based model with which the use of domain testing can be justified and from which extensions of domain testing to non-linear computations can be obtained. Finally, Section VI shifts the emphasis from errors to faults, introducing a new algorithm that takes advantage of tests across multiple paths to reduce the total number of tests while achieving comparable levels of error detection.

2. Definitions

2.1 Program Concepts

In most programs, there are many possible sequences of statements that may be traversed from the start of the program to its end. Each such sequence is called a *path* through the program. A subsequence of a path will be called a *subpath*. When a subpath begins at the start of the program, it will be called an *initial subpath*. (A path may be considered to be a special case of an initial subpath.)

We can associate with each initial subpath a set of inputs that cause execution of the program to follow that subpath. This set of inputs will be called the *subpath domain*. We can also associate with each subpath (not necessarily initial) a transformation representing the effect upon the program variables of executing that subpath. This transformation, a mapping from the set of inputs and the old set of values for the program variables to a new set of values for the program variables, will

```
input X, Y;
if X >= 0.  then
    Z := Y - X;
    D := 0.;
    while Z > D loop
        D := D + 1.;
    end loop;
    if X <= D then
        if Y <= 1.001*D then
        ⋮
```

Figure 1: Sample Program

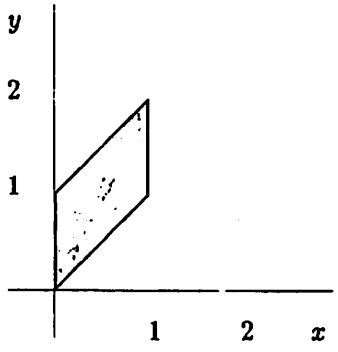

Figure 2: Subpath Domain

be called the *subpath computation*. The *path computation* corresponding to a complete execution of the program is somewhat more limited, consisting of the mapping from the program's inputs to the program's outputs.

As an example of the determination of subpath domains and computations, consider the program fragment in Figure 1. If we consider the subpath that enters the "then" portion of the outermost `if` statement, exits the `while` loop after one pass, and enters the "then" portion of the final `if`, then the subpath domain would be the shaded area shown in Figure 2, defined by the conjunction of the conditions $x \geq 0$, $y - x > 0$, $y - x \leq 1$, and $x \leq 1$. where x and y denote the first two numbers in the program's input stream. The subpath computation, which maps $x \times y \times X \times Y \times Z \times D$ onto $X \times Y \times Z \times D$, would be represented by the function $f(x, y, X, Y, Z, D) = (x, y, y - x, 1.)$. Subpath domains and computations can often be determined automatically by symbolic execution of the subpath [4,6]. We will return to this path domain as an example several times during this paper.

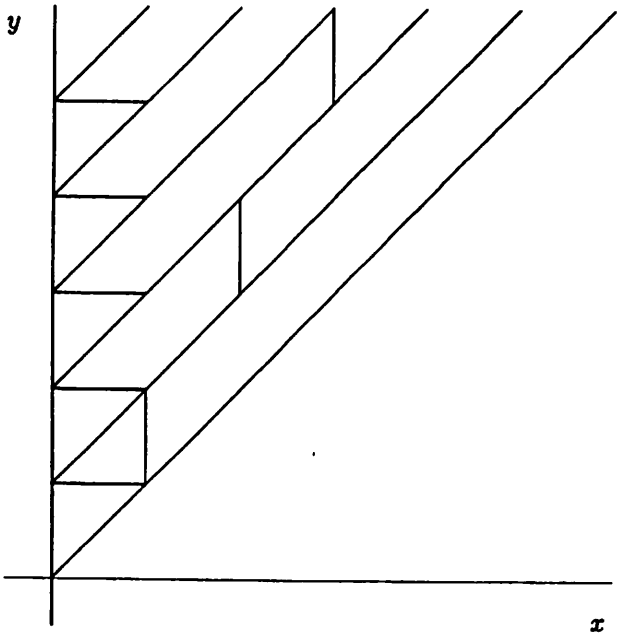

Figure 3: Input Space Partition

Because, in the absence of concurrency, there is a unique path traversed by any executions using the same input values, the intersection of the domains of any two distinct paths is empty. Similarly, the intersection of the subpath domains for any two initial subpaths, neither of which is a subsequence of the other, is also empty. Thus the set of subpath domains for the paths in a program form a partition of the program's input space. Figure 3 shows the structure of this partition for the sample program. Each of the diagonal lines represents a constraint imposed by the `while` statement, each of the vertical lines except the one at $x = 0$ represents a constraint imposed by the `if X <= D` statement, and the horizontal lines represent constraints imposed by the final `if` statement. Additional conditional statements appearing later in the source code may impose further divisions of the input space. Additional input statements would add new coordinate axes.

2.2 Faults and Errors

It is useful to separate the ideas of defects in the source code from the resulting defects in the program function. Defects in the source code will be called *faults*. An *error* is a defect in the function computed by the program. Faults may cause errors, but are not themselves considered to be errors. In fact, it may be said that the goal of testing is to choose data so that any faults present in the program result in errors.

A classification of errors with strong intuitive appeal is the division into domain and computation errors first proposed by Howden in [12]. A program is said to exhibit a *domain error* when incorrect output is generated due to executing the wrong path through a program. A *computation error* occurs when the correct path through the program is taken, but the output is incorrect because of faults in the computations along that path. This definition differs from the original formal definition in [12] only in that the computation performed during a domain error is not here required to correspond to a correct computation for any other inputs, a relaxation that is consistent with the ways in which domain errors have been treated in [6,17,18] and that also appears to be more realistic for practical programming languages.

Identifying a particular error as belonging one of these two error classes depends upon the ability to determine whether or not the "correct path" was executed. This in turn implies the existence of a correct version of the program against which a comparison can be made. There are, of course, an infinite number of correct programs for any computable function. If a correct program is chosen that bears little resemblance to the one being tested, then the definitions of domain and computation errors become essentially meaningless. If we assume, however, that the program being tested bears a reasonable resemblance to some correct version, then it is likely that we will have little trouble with the domain error/computation error classification. If an error can be associated with the execution of the wrong path through some correct program lying within a neighborhood of programs similar to the one being tested, then it is a domain error.[1] If it can be associated with the proper path through such a correct program, then it is a computation error. The two classes of errors may not be disjoint, but that simply means that some errors may be detectable using either computation- or domain-oriented testing methods.

A domain error may arise from a *predicate fault*, where the fault exists in a conditional statement, or from an *assignment fault*, where the fault exists in a computation that affects a later conditional statement.

Domain errors can be further broken down into two classes, *path selection errors* and *missing path errors*. These are distinguished by whether a path through the program exists that, had it been taken, would have produced correct output. Where such a path exists, the error is considered to be a path selection error. Where the conditional statement and computations associated with part of the input data domain are missing entirely, it is called a missing path error [12,18]. It is, in general, undecidable whether a program is missing a path. In practice, this appears to be most difficult when the missing conditional statement is an equality.[2] Test data which captures path selection errors should catch most missing path errors where the missing condition is an inequality, since these may be viewed as limiting cases of path selection errors in the model presented in the succeeding sections. This paper is therefore primarily concerned with path selection errors.

---

[1]The idea that testing can be viewed in terms of neighborhoods of similar programs was introduced by Budd and Angluin [2]. The "competent programmer hypothesis" of mutation testing [1,3,8] is one example of this approach. A formal basis for this "neighborhood paradigm" was established by Gourlay in [10], and its relation to Howden's definitions in [12] is noted there.

[2]In this paper, the operators "$=$" and "$\neq$" are both referred to as equalities in order to distinguish them from the inequalities "$<$", "$\leq$", "$>$", and "$\geq$".

3. Domain Testing

Proposed testing procedures for catching path selection errors have ranged from simple, intuitively motivated strategies as in [9,13] to strategies that offer provably high reliability but whose application is limited to special classes of programs. Chief among the latter are the *domain testing strategies*, which are characterized by the selection of test points according to geometric characteristics of a path domain. This section will review the domain testing method as proposed by Cohen and White [7,18] and as modified by Clarke, Hassell, and Richardson [5].

Domain testing is intended to choose numerical data that will reliably test a previously selected path for path selection errors. The domain testing method does not specify how the test paths will be chosen, but a number of path selection strategies have appeared in the literature [11,14,15,16]. Attempting to test every path through the program would usually be impractical, since even very simple programs can have a prohibitively large number of paths.

Domain testing has been largely limited to programs in which the borders of the path domains are linear functions of the program inputs, which may be integers or real numbers. Cohen and White do speculate on possible extensions to test polynomial borders [7,18], but the possibility is not examined in much detail, nor is a satisfactory procedure ever derived.

3.1 The Nx1 Strategy

As noted earlier, each path from the start of a program may be associated with a (possibly empty) path domain. Each conditional branch encountered during execution imposes some restriction on the path domain. The conjunction of all these restrictions is called the *path condition*. The path domain is therefore the set of all inputs for which the path condition is true.

In programs where all branching and looping depends solely on numerical variables, the path condition takes the form of a system of simultaneous inequalities (and equalities) representing the necessary and sufficient condition for following that path. If, as we shall henceforth assume without loss of generality, the predicates encountered in the conditional statements are simple relational expressions (no AND's, OR's or other boolean operators), the path domain will be a convex polyhedron in N-space, where N is the number of inputs to the program.[3] Referring back to Figures 1 and 2, for example, because the input space has only two variables, the "N-dimensional polyhedron" representing this path domain becomes a planar polygon and the "faces" of the polyhedron become line segments. Each border segment (face) of the polyhedron is generated by the execution of some conditional statement. If that conditional statement occurs within a loop, it may be responsible for several border segments, each corresponding to a different execution of the same statement.

Domain testing is concerned with the possibility that one or more border segments of the path domain may have shifted away from the correct position, due to an error in the program predicate

[3]Cohen and White assume that an implicit bound exists on all inputs, by virtue of their being represented in a finite number of bits. They include this restriction on the input values as an explicit part of the path condition, thus guaranteeing that all path domains will be closed. Clarke et al., however, allow the inputs to range over $\pm\infty$.

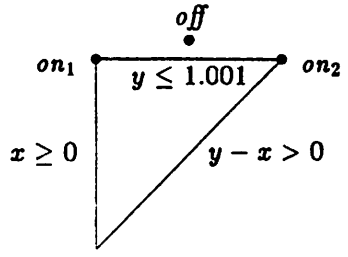

Figure 4: Nx1 Domain Test

or in the computations affecting that predicate. Domain testing focuses, one at a time, upon each of the border segments for a path being tested. That border segment is referred to as the *given border*, to distinguish it from the *correct border* whose position is initially unknown and which may or may not be identical to the given border, depending on whether or not a fault exists. Thus there is a critical region of inputs that satisfy one, but not both, of the constraints represented by the given and the correct borders. Because taking the wrong path is presumed to be a sufficient condition for detection of the error,[4] each point in that region represents an input for which a domain error occurs. Intuitively then, a strategy for detecting domain errors must choose points close to the borders so that even a small shift in a border's position would cause at least one test point to fall within the critical region.

For a given test path requiring N inputs per execution and therefore having N numeric inputs per input point, White and Cohen propose choosing N points on each border segment at or near the vertices and one point just slightly off that segment, on the open side of the border. These two sets of points are referred to as the *on* and *off* points, respectively. By choosing the *off* point very close to the border, they argue that all but the smallest shifts in the predicate function will be detected.

As an example of this strategy, henceforth referred to as the $N \times 1$ strategy, let us assume that we wish to perform domain testing on the border formed by the final statement in Figure 1 using a path reaching this statement after a single iteration of the loop and passing on to the "then" portion of this statement. If there are no subsequent conditional statements reached along this path, then the path domain is as shown in Figure 4. The Figure shows the *on* points, in this case located at $(0, 1.001)$ and $(1, 1.001)$, and the *off* point at $(0.5, 1.001 + \epsilon)$ where ϵ is some small positive number.

Successful execution of these tests clearly places severe restrictions on the possible positions in which the correct border could occur, if it is not actually identical to the given border. The correct border cannot drop below the given border at any point within the subpath domain without crossing one of the *on* points, causing execution of that test to proceed along an incorrect path. If

[4] White and Cohen argue that chances of a given input producing identical output when adjacent domains compute different functions is quite small [18]. Clarke et al. make the additional argument that symbolic execution of the correct and incorrect paths can often be used to detect such occurrences [5].

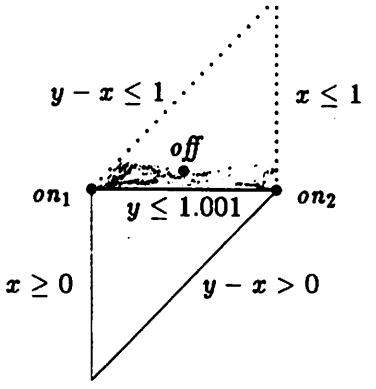

Figure 5: Border Shift Region for Nx1

the correct border lies entirely above the given one, the distance between the two must be small enough that the correct border does not cross the *off* point, or else execution of that point would follow an incorrect path. Hence there is a relatively small region of the input space in which the correct border could lie, if the given one is indeed incorrect.

This region is quantified in [5,19] as follows: Consider the set of constraints adjacent to the given border segment — i.e., those constraints that intersect the given border segment at one or more of its vertices. Define the *border shift region* as the intersection of the set of points satisfying all these adjacent constraints with the set of points through which the given border could be shifted without crossing any of the *on* or *off* points. Define the *Border Shift Error* (BSE) as the volume of this border shift region. In Figure 4, the constraints adjacent to the given border are $y - x \leq 1$ and $x \leq 1$; they are not visible in the Figure because the border segments they contribute are extremely short. The border shift region for this choice of test points is shown as the shaded region in Figure 5. The dotted lines indicate the adjacent constraints that are not border segments for the path domain.

3.2 The NxN and VxV Strategies

Clarke et al. show that the $N \times 1$ strategy allows the BSE to be infinite [5]. To make this less likely, they recommend that the N test points chosen on the border enclose the centroid of the border segment. They also define two new domain testing strategies – the $N \times N$ strategy in which N points each are chosen on and off the border, each point at or near a vertex, and the $V \times V$ strategy in which a test point is chosen exactly on and just off each of the vertices of the border segment. These strategies and their border shift region are illustrated in Figure 6.[5] Note that the

[5]For $N = 2$ the $N \times N$ and $V \times V$ strategies are identical. For $N > 2$, the number of vertices formed by the intersection of the border segments may be greater than N and hence the $V \times V$ strategy could require more test points than would the $N \times N$ strategy. In a three-dimensional input space, for example, the border segments would be planar polygons with arbitrary numbers of vertices.

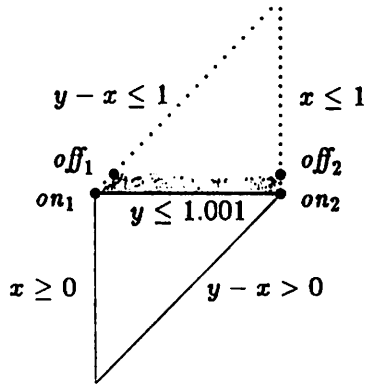

Figure 6: NxN Domain Test

same *on* points are used by the $N \times 1$ and $N \times N$ strategies. It is only in the choice of *off* points that they differ.

4. The Border Shift Error

Although the BSE has been the major focus of previous authors' arguments for the power of domain testing, the choice of the BSE itself has been largely unmotivated. The desirability of minimizing the BSE is taken as intuitively obvious. One can conceivably advance two arguments in favor of using the BSE. The first argument is that removing one or more points from the border shift region implies detecting any errors that would cause the correct border to pass through those points. Hence reducing the BSE means reducing the number of possible errors and so the increasing the confidence in the correctness of the given border. By this argument the BSE is a measure of correctness for the given border. The second argument says that each point in the border shift region represents an input for which an incorrect path through the program will be taken, probably resulting in an error. Reducing the number of such points reduces the probability that an arbitrary input will result in such an error. By this argument the BSE is a measure of program reliability.

This section will investigate each of these possible claims further. It will be shown that, in fact, neither argument is correct. The BSE fails as a measure of correctness because it does not really correlate with the number of possible errors in the given border. It fails as a measure of reliability because it fails to consider all relevant constraints on the path domain and because it fails to take into account the interactions that occur when more than one path is tested passing through the same sets of statements (and hence passing through the same faults in the program code).

4.1 The BSE as a measure of correctness

Clearly the BSE does have some bearing on the number of possible errors in the given border. If we begin with a given set of test points, determine the BSE for those points and then, by addition of

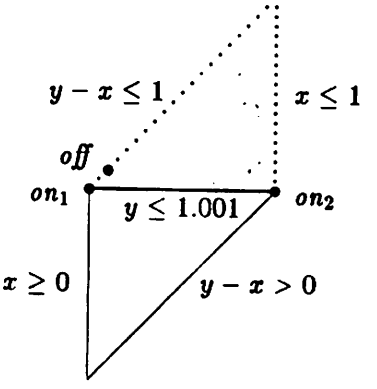

Figure 7: BSE versus Possible Errors

one more test point, reduce that BSE, it is clear that we have eliminated some errors from possible existence. In this situation, however, the border shift region after adding the final test is a subset of the border shift region prior to adding that test. When comparing more diverse collections of test points, such as the $N \times 1$ and $N \times N$ test sets, such subset relations may not always hold.

When comparing two sets of test points whose border shift regions are not related by subsetting, it is possible for the border shift region with the larger volume (BSE) to permit the existence of fewer possible errors. This can occur because there may be, on average, fewer possible correct borders passing through any given point in the larger border shift region than through any given point in the smaller one. To illustrate this possibility, consider Figure 7. The border shift region for the choice of test points in this Figure is the entire region enclosed by the given border and the adjacent constraints, except for the points on the $y - x \leq 1$ border itself. The BSE for this choice of test points is therefore clearly larger than the BSE for Figure 5. Furthermore, note that sliding the *off* point in Figure 7 closer to on_1 leaves the border shift region, and hence the BSE, unchanged.

Consider now the probability that an arbitrary error, and hence an arbitrary position for the correct border, would go undetected by the two sets of test points in Figures 5 and 7. In Figure 7, a correct border position different from the given one will go undetected only if it intersects the $y - x \leq 1$ border between the on_1 and *off* points. By sliding the *off* point closer to on_1, this probability can be made arbitrarily small. In particular, it can be made smaller than the probability of an arbitrary error going undetected in Figure 5. Since the BSE for Figure 7 is larger than that in Figure 5, it is clear that the BSE does not correlate with the number (or fraction) of possible errors detected by a set of tests. Section 5. will develop this idea more fully by directly utilizing the number of possible errors that go undetected as a measure of test quality.

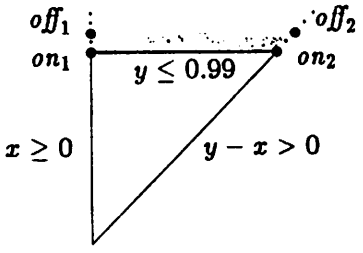

Figure 8: New NxN Test

4.2 The BSE as a measure of reliability

The argument that the BSE can be viewed as a measure of reliability turns upon the idea that the points in the border shift region represent the set of inputs for which an incorrect path will be taken. This idea is, in fact, incorrect. Suppose that the final `if` statement in the program in Figure 1 were replaced by "`if Y <= 0.99*D then ...`". The change between this predicate and the predicate "`Y <= 1.001*D`" employed in the earlier examples is too small to be visible in Figures 4 through 7. What is changed is that the constraints adjacent to the given border in those Figures will now be $x \geq 0$ and $y - x > 0$ rather than $x \leq 1$ and $y - x \leq 1$. Figure 8 shows the $N \times N$ test points and their border shift region for this new case. (As before, the dotted lines show part of the adjacent constraints that are used to compute the *off* point positions and the border shift region, although not part of the actual border segment for this path subdomain.)

The rather small border shift region shown in this Figure is deceptive. Since the given border is only reached by tests that satisfy the conditions $y - x \leq 1$ and $x \leq 1$ as well as $x \geq 0$ and $y - x > 0$, it is clear that some of the points shown in the border shift region will never cause this border to be tested. They may reach the same conditional statement, but they will do so only along another path, on which the predicate will have a completely different interpretation and thus will form a different border segment.

Of particular interest is the fact that neither of the *off* points reach the given border. It follows that any position of the correct border that satisfies the two *on* points will be accepted by this set of tests. The true set of inputs through which the given border might be shifted without detection by this test set, a region which we shall call call the ***untested input region***, is therefore the shaded area shown in Figure 9. Clearly the BSE based upon the border shift region drastically underestimates the size of the untested input region in this example.

The difference between the border shift region and the untested input region is important, not only because it illustrates problems with the BSE, but also because it illustrates a problem with the domain testing strategies in [5,18]. The *off* points in the above example were useless as detectors of border shifts. The requirement that these points satisfy the constraints adjacent to the given border is simply not strong enough. Other constraints must also be satisfied, even though they are not adjacent to the given border and might not contribute border segments to the domain of the

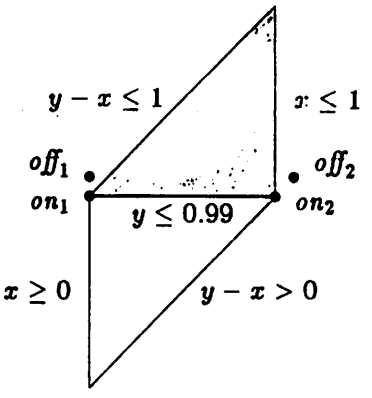

Figure 9: Untested Input Region

path being tested.

The basic problem is the formulation of domain testing as a means for testing paths rather than as a means for testing initial subpaths. Returning to Figure 3, for example, try selecting an arbitrary path domain and then selecting any one of the border segments for that domain to serve as a given border for domain testing. Now, which of the constraints denoted by the other line segments in that Figure must be satisfied by any test points for your chosen border?

In fact, the question can no longer be answered once our view of these borders has been reduced to individual path domains. The key to the answer is to consider the the subpath domain for the initial subpath leading up to, but not yet including, the given border to be tested. The given border is then not so much a border of a path domain (Figures 7 and 8) as it is a division of the current subpath domain (Figure 9). The set of constraints that must be satisfied by any and all tests of the given border is then simply the set of constraints defining the domain of the initial subpath just prior to imposing the given border. Any constraints that occur after the imposition of the given border clearly cannot affect the position of that border. If the given border is erroneous, an input point either does or does not take an incorrect path due to that border shift — any later constraints simply determine which correct or which incorrect path is taken, but not whether the path taken is correct or incorrect.

This can lead to a significant savings in the number of test points used by domain testing over a number of paths. In Figure 3, for example, the constraint $x \geq 0$ is split into many small segments by the other borders. Domain testing as defined in [5,18] would test each of these segments separately, choosing 3 or 4 tests per segment. In the initial subpath view, however, since this is the first division of the entire input domain, the entire border would be tested as a single unit at no loss in the ability to detect shifts of this border.

If domain testing is redefined using this initial subpath view, and if the BSE is redefined to be the volume of the untested input region rather than of the border shift region, would the BSE then be a reasonable measure of the program's reliability? The answer is still no, because the BSE would still fail to take into account the interactions among different paths through the same statements. A given border corresponds to some conditional statement in the code and to some series of computations that affected the evaluation of that condition on the chosen initial subpath. Since most statements in a typical program can be reached along more than one path,

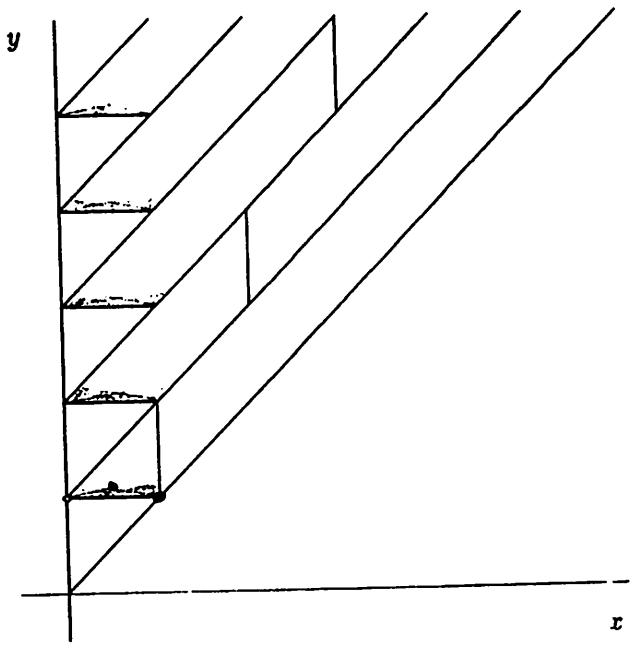

Figure 10: Interactions Among Paths

tests conducted within one subpath domain affect our confidence in the borders appearing in other path domains. Thus the BSE could only be regarded as a valid measure of the program's reliability if we could somehow integrate it across all domains. For example, suppose that we are interested only in possible predicate faults in the statement "`if Y <= 0.99 then`" and have carried out the tests shown in Figure 5. Then we might expect a global view of the untested input regions for this program similar to that shown in Figure 10, where the horizontal border segments are all imposed by executions of that statement on different paths and the shaded regions indicate untested input regions after the tests in Figure 5. It should be clear from this global picture that any subsequent domain tests of the horizontal borders in other path domains would be of questionable worth, although the BSE evaluated separately for each domain would argue for using the same pattern of test points in each path domain. In more complicated cases where the untested input regions for the different paths were less closely related, we would have the additional problem that a set of test points that produced a small untested input region on the path being tested might leave large untested regions in other paths, while a set of tests that did not do as well on the path being tested might yield smaller untested input regions in the other paths. The BSE is simply not designed to take such interactions among paths into account.

In this section we have seen three problems with the BSE measure as a means of justifying domain testing. Although the BSE is not the only measure that has been used to evaluate domain testing, the others, the Domain Error Magnitude [18] and the Maximum Angular Displacement [19]

may be viewed as approximations to the BSE and are subject to the same problems. In the next Section we will address the first problem, the lack of correlation between the BSE and the number of possible errors left untested by developing a new measure of test effectiveness based directly upon the idea that one set of tests is better than another if it detects a larger number of possible errors. A solution to second problem, the failure to consider all relevant constraints, has already been provided by switching domain testing to an initial subpath view. The third problem, failure to consider interactions among paths, will be addressed in Section 6. by changing the emphasis from the elimination of possible errors to the elimination of possible faults.

5. Domain Testing and Error Detection

This Section examines the ability of the domain testing strategies to detect errors. The criterion for test data selection to be used here can be summed up by the idea that one set of test data is better than another if it detects more possible errors. A model will be developed that permits one to "count" the possible errors that would have been detected or left undetected by an arbitrary set of test data. This model is based upon the perturbation analysis described in [21,23].

Initially, the set of errors to be considered will be all those errors that cause a correct border that is linear in the program inputs to be transformed into an incorrect given border that is also linear in the program inputs. There are two forms of such errors. The first is the border shift as discussed in the previous Section. The second is the substitution of one relational operator for another. Of course, combinations of these two forms are also possible.

Domain testing has been concerned with both of these forms of error, even though the discussion in the previous sections of this paper and most of the discussion in [5,18] has focused entirely on border shifts. For example, when the given border is an equality, all forms of domain testing require an extra set of *off* points so that *off* points are chosen on both sides of the border, effectively giving us $N \times (1+1)$, $N \times (N+N)$, and $V \times (V+V)$ strategies. These extra *off* points would not be required to detect border shifts. for the N *on* points uniquely define the hyperplane containing the border segment and so would suffice to detect any border shifts not accompanied by a change of relational operator.

When the given border is an inequality, most changes of relational operator can be represented as border shifts. For example, a substitution of "$<$" for "$\leq$" can be considered to be a shift of the border by a very small distance. The one change of relational operator that cannot be treated as a border shift would be the replacement of an inequality by an equality. Interestingly enough, this substitution is not detected by any of the domain testing strategies. For example, if the given border is imposed by the constraint $exp_1 \leq exp_2$, the various domain testing strategies will choose *on* points for which $exp_1 = exp_2$ and *off* points for which $exp_1 > exp_2$, but will not choose points for which $exp_1 < exp_2$. The resulting set of tests will thus not be able to tell if the correct border should have been $exp_1 = exp_2$.

This problem is easily remedied by requiring at least one of the *on* points to be moved slightly off of the border to the accepted side (i.e. opposite the side of the border on which the *off* points are chosen). We will assume in the remainder of this paper that this requirement is imposed, although

we shall not usually mention it explicitly.[6] Thus the remainder of the analysis will concentrate upon border shifts.

5.1 An Error–Space Model For Domain Errors

Consider now an initial subpath leading up to a predicate. Let the set of inputs for that path be denoted by $\bar{x}$ with the individual elements of $\bar{x}$ denoted by x_i. Let $T(\bar{x})$ *relop* 0 be the constraint imposed by the correct border expected from that predicate and let $T'(\bar{x})$ *relop* 0 be the constraint imposed by the given border actually formed by that predicate, where *relop* is one of $\{<, \leq, >, \geq, =, \neq\}$. Since we are assuming that both T and T' are linear, the border shift can be represented by the linear function formed by their difference:

$$e = T' - T. \tag{1}$$

Because e is linear in the program inputs, it can be written as

$$e(\bar{x}) = \alpha_0 + \sum_{i=1}^{N} \alpha_i x_i. \tag{2}$$

The vector $\bar{\alpha}$ can then be regarded as the coordinates of a point in an *error space* whose coordinate axes are the functions 1 and $x_i, i = 1 \ldots N$. Each point in this space denotes a distinct error that could be added to some correct border to yield the given border. The origin of this space represents the special case where the given and correct borders are identical, so that no actual error exists.

It is often convenient to think of e as a vector in the error space, having both a size and a direction:

$$e(\bar{x}) = \alpha \hat{e}(\bar{x}) \tag{3}$$

where $\hat{e}$ is a unit-length vector in the error space and is considered to be the direction of e, and α is then the size of e. The reason for dividing e in this manner is that the reasons why a border shift does or does not result in a domain error for a given test can be divided into size-independent and size-dependent cases.

For the size-independent case, if the direction of the border shift is such that $\hat{e}(\bar{x}_0) = 0$ for some test $\bar{x}_0$, then that border shift will not result in a domain error, since $T(\bar{x}) = T'(\bar{x})$. If $\hat{e}(\bar{x}) = 0$ for all $\bar{x}$ in the subpath domain, then the path is said to be *blind* to the error direction $\hat{e}$ [21]. This can occur only because the subpath domain has previously been restricted by one or more conditional statements containing linear equality predicates $f_j(\bar{x}) = 0$ such that $\hat{e}$ can be formed from a linear combination of the f_j. Since such error directions can never result in a domain error, they can be safely ignored.

The size-dependent case is more interesting. Suppose, for example, that the given border constraint is $x_1 - x_2 \geq 0$, but that the correct border is $x_1 \geq 0$. The border shift, then, is of the

[6] The justification for ignoring this requirement in the analysis that follows will actually be provided in Section 5.3.

form αx_2 with $\alpha = -1$. If we test the given border with $(x_1, x_2) = (1.0, 0.5)$, no domain error results because the border shift simply wasn't big enough on that test. Had the magnitude of α been larger, a domain error could have occurred. Thus a test can be said to constrain the size of possible errors, such that errors of sufficient size get detected while smaller errors go undetected.

The constraint imposed by a test is sensitive to the sign of α. In the example above, any errors of the form αx_2 would be detected if $\alpha < -2$; positive values of α remain unconstrained. For any error direction $\hat{e}$, we will say that $\hat{e}$ is *unbounded* after a set of tests if all errors of the form $\alpha\hat{e}$ would be undetected by those tests; we will say that $\hat{e}$ is *doubly bounded* if there exist constants $c_1 \leq 0, c_2 \geq 0$ such that errors of the form $\alpha\hat{e}$ are undetected only when $c_1 \leq \alpha \leq c_2$; we will say that $\hat{e}$ is *singly bounded* if $\hat{e}$ is not doubly bounded and there exists a constant $c_1 \geq 0$ such that errors of the form $\alpha\hat{e}$ are undetected only when $\alpha \leq c_1$ or only when $\alpha \geq -c_1$.

We are now in a position to specify a criterion for test data selection that will favor those test sets that leave the smallest number of possible errors undetected. Since tests can be viewed as constraining the set of errors that go undetected, we can anticipate computing the portion of the error space consisting of errors that would have escaped detection with a given set if tests. Call this set of errors the *untested error region*. Since each point in the error space corresponds to a different error, the simplest measure of the "number of possible errors" that go undetected would be the volume of the untested error region. This measure is too crude, however, for two reasons. First, if any error directions are not doubly bounded, then the volume is infinite. In fact, we shall see that, for any path, there is always at least one direction that cannot be doubly bounded, so some means of distinguishing among these various infinite volumes is necessary. Second, if that problem did not exist, then if any error direction $\hat{e}$ were doubly bounded such that $\alpha\hat{e}$ went undetected only when $0 \leq \alpha \leq 0$, or simply $\alpha = 0$, then all volumes would be zero no matter how well or how poorly the other error directions were bounded.

To resolve the problem of infinite volumes, we shall say that *one set of test points is considered better than another if there exists an r_0 such that for all $r > r_0$, the portion of the untested error region for the first test set lying within a radius r of the origin has a smaller volume than the portion of the untested error region for the other test set lying within that radius of the origin.* To resolve the problem of zero volumes, we shall stipulate that, for the purpose of comparing two test sets, any error directions that are doubly bounded with $\alpha = 0$ will be treated as if they were bounded only by $-\epsilon \leq \alpha \leq \epsilon$, where ϵ is an arbitrarily small positive number.

The following theorem, a special case of the one presented in [22,24], forms the basis for relating this criterion for comparing test sets to the earlier discussion of how a set of tests can be viewed as imposing constraints upon the untested error region.

Theorem 1 *Given a border $T'(\bar{x})$ op_1 0 that has been tested with inputs $\bar{x}^0$, an error $\bar{e}$ in T' fails to cause a domain error iff*

$$\bar{e}(\bar{x}^0) = 0 \tag{4}$$

or

$$(\bar{e} \cdot \hat{e}_{x''})\ op_2\ (T'(\bar{x}^0)/\hat{e}_{x''}(\bar{x}^0)) \tag{5}$$

op_1	$T'(\bar{x}^0)\ op_1\ 0$	op_2
$>$	true	$<$
$>$	false	$\geq$
$\geq$	true	$\leq$
$\geq$	false	$>$
$<$	true	$>$
$<$	false	$\leq$
$\leq$	true	$\geq$
$\leq$	false	$<$
$=$	true	$=$
$=$	false	$\neq$
$\neq$	true	$\neq$
$\neq$	false	$=$

Table 1: Operators for Theorem 1

where op_1 and op_2 are given in Table 1 and where $\hat{e}_{\bar{x}^0}$ is the unique unit-length vector such that $\hat{e}_{\bar{x}^0}(\bar{x}^0) > 0$ and, for all $\hat{e}$ orthogonal to $\hat{e}_{\bar{x}^0}$, $\hat{e}(\bar{x}^0) = 0$.

Proof: If equation 4 is true, then clearly no domain error can result from this error in the border because no border shift occurs at that test point. If, however, the error in the border does result in some border shift, then a domain error results iff the shift is large enough that the given and correct borders take on different truth values on this test.

Since the given border is $T'(\bar{x})\ op_1\ 0$, a domain error results iff $T'(\bar{x}^0)\ op_1\ 0$ and $T(\bar{x}^0)\ op_1\ 0$ are different. We may therefore state that a domain error fails to occur exactly when

$$(T'(\bar{x}^0)\ op_1\ 0) \Leftrightarrow (T(\bar{x}^0)\ op_1\ 0).$$

The correct border T is unknown, but from the definition of $\bar{e}$ in equation 1 we have that a domain error occurs when

$$(T'(\bar{x}^0)\ op_1\ 0) \Leftrightarrow (T'(\bar{x}^0) - \bar{e}(\bar{x}^0)\ op_1\ 0).$$

or, equivalently, when

$$\bar{e}(\bar{x}^0)\ op_2\ T'(\bar{x}^0) \tag{6}$$

where op_2 is given by Table 1 (depending upon the value of $T'(\bar{x}^0)\ op_1\ 0$). It is worth noting that, even if $\bar{e}$ and T' are not linear functions of $\bar{x}$, equation 6 is nonetheless a linear constraint on the error space containing $\bar{e}$ as long as $\bar{e}$ can be written as a linear combination of linearly independent functions (as, for example, in equation 2).

Now consider the error direction $\hat{e}_{x^0}$, whose existence is guaranteed by the fact that $\bar{e}(\bar{x}^0) \neq 0$ and which can be shown to be given by

$$\hat{e}_{x^0}(\bar{x}) = \sum_i \hat{e}_i(\bar{x}^0)\hat{e}_i(\bar{x})/c \tag{7}$$

where the $\hat{e}_i$ are the linearly independent functions chosen as the coordinate axes ot the error space and where the normalization constant c is given by

$$c = \sqrt{\sum_i \hat{e}_i^2(\bar{x}^0)}.$$

The error $\bar{e}$ can be written as a linear combination of $\hat{e}_{x^0}$ and some direction $\hat{e}$ orthogonal to $\hat{e}_{x^0}$:

$$\bar{e} = (\bar{e} \cdot \hat{e}_{x^0})\hat{e}_{x^0} + (\bar{e} \cdot \hat{e})\hat{e}.$$

Since $\hat{e}$ is orthogonal to $\hat{e}_{x^0}$, we have that $\hat{e}(\bar{x}^0) = 0$ and therefore

$$\bar{e}(\bar{x}^0) = (\bar{e} \cdot \hat{e}_{x^0})\hat{e}_{x^0}(\bar{x}^0).$$

Substituting into equation 6, we have that

$$(\bar{e} \cdot \hat{e}_{x^0})\hat{e}_{x^0}(\bar{x}^0)\ op_2\ T'(\bar{x}^0)$$

and, since $\hat{e}_{x^0}(\bar{x}^0) > 0$, we get

$$(\bar{e} \cdot \hat{e}_{x^0})\ op_2\ (T'(\bar{x}^0)/\hat{e}_{x^0}(\bar{x}^0)),$$

proving the theorem.

The first point of interest provided by Theorem 1 is that the constraint on the error space imposed by any single test is a simple linear constraint. An error lies in the untested error region if and only if it satisfies these constraints for all tests performed so far. The system of linear constraints imposed by repeated use of equation 5 for each test can therefore be used to determine the geometry of the untested error region.

Another point of interest in this theorem is that equation 5 can be interpreted as stating that $\bar{e}$ goes undetected when its component along the direction $\pm\hat{e}_{x^0}$ is no larger than $|T'(\bar{x}^0)/\hat{e}_{x^0}(\bar{x}^0)|$. The sign of $\hat{e}_{x^0}$ depends upon the relational operators involved. Another way of stating this is to say that a test $\bar{x}^0$ imposes a linear constraint upon the untested error region consisting of a hyperplane orthogonal to $\pm\hat{e}_{x^0}$ and at a distance $|T'(\bar{x}^0)/\hat{e}_{x^0}(\bar{x}^0)|$ from the origin. Note that the direction $\hat{e}_{x^0}$ can be viewed as the error direction to which the test $\bar{x}^0$ is most sensitive. This interpretation of equation 5 will be particularly important to the analysis that follows.

Finally, note that the linearity of the constraints means that any linear combination of two or more unbounded directions is unbounded, and any linear combination of two or more singly bounded directions is at best singly bounded and may be unbounded. The number of linearly

independent error directions that have been left unbounded or only singly bounded is therefore crucial to the error detection capabilities of a set of test data. Define the *unbounded dimension* of the untested error region for a set of tests as the minimum number of additional constraints needed to make the untested error region a completely closed region. For a completely unconstrained error region of dimension n, the unbounded dimension is $n + 1$. For example, in 2-space, the simplest closed figure is the three-sided triangle, in 3-space it is the 4-sided tetrahedron, etc. It then follows from the earlier criterion for comparing test sets that:

Corollary 1 *Given two sets of tests whose untested error regions have different unbounded dimensions, the set with the smaller unbounded dimension is better.*

By this corollary, we can divide the goal of choosing tests to leave the smallest possible untested error region into two subgoals:

- Bound as many mutually orthogonal error directions as possible.
- Make the constraints on the bounded directions as tight as possible.

In most instances an incremental improvement in progress toward the first subgoal may be regarded as far more important than an incremental improvement in progress toward the second goal, since the corresponding effect on the number of errors that could escape detection would be far more larger in the first case.

5.2 Error Detection By the Three Strategies

We now examine the degree to which each of the three domain testing strategies satisfy each of these two subgoals for reducing the untested error regions. The domain testing strategies choose the test points on the basis of their geometric properties in the input space. Our two subgoals, however, depend upon geometric properties in the error space. The next Theorem shows how these two spaces are related.

Theorem 2 *When testing for linear errors in domain borders with an error space of dimension n, any set of k test points, $k \leq n$, chosen in general position*[7] *leaves an untested error region with unbounded dimension $n - k + 1$.*

Proof: Consider a set of test points $\{\bar{x}^i\}$, $i = 1 \ldots k$, in general position. The Theorem holds for $k = 1$ since, by Theorem 1, the first test imposes a constraint on the direction $\hat{e}_{\bar{x}^1}$ and on all directions having a non-zero component along $\hat{e}_{\bar{x}^1}$.

Now suppose that the Theorem holds for $\{\bar{x}^i\}$, $i = 1 \ldots k-1$, and that a new test point $\bar{x}^k$ is added that maintains the general positioning of the entire test set. Consider the error

[7] A set $\{\bar{z}_i\}$ containing n points is in *general position* if the $n-1$ vectors $\bar{z}_i - \bar{z}_1$, $i = 2 \ldots n$ are linearly independent. Intuitively, this means that the points cannot be contained in a space of dimension less than $n - 1$.

directions $\hat{e}_{\bar{x}^i}$ associated with these points. Since the error space is of the form given in equation 2, the error directions $\hat{e}_{\bar{x}^i}$ given in equation 7 simplify to

$$\hat{e}_{\bar{x}^i}(\bar{x}) = (1 + \sum_j x_j^i x_j)/c. \tag{8}$$

Because the various $\bar{x}^i$ are in general position, the corresponding error directions $\bar{e}_{\bar{x}^i}$ are linearly independent.

Now, since $\bar{e}_{\bar{x}^k}$ is linearly independent of the remaining $\bar{e}_{\bar{x}^i}$, there is some error direction orthogonal to the remaining $\bar{e}_{\bar{x}^i}$ but having a non-zero component along $\bar{e}_{\bar{x}^k}$. By Theorem 1, that direction would be left unbounded by the previous $\bar{x}^i$, $i = 1 \ldots k-1$, but would be singly bounded by $\bar{x}^k$. Thus the addition of $\bar{x}^k$ reduces the unbounded dimension by 1, satisfying the Theorem for $i = 1 \ldots k$. The Theorem therefore holds by induction for $k = 1 \ldots n$. (For $k > n$, it is no longer possible to obtain a linearly independent $\bar{e}_{\bar{x}^i}$, or for the kth test point to still be in general position.)

The importance of this Theorem lies in the fact that the various domain testing strategies, by choosing their *on* points at the vertices of borders that are guaranteed to form convex polyhedra in N-space, are thereby guaranteed to choose the *on* points in general position. Furthermore, since all of the *on* points lie in the hyperplane of the given border, and none of the *off* points lie in that hyperplane, the set of *on* points plus any one *off* point are in general position. These $N + 1$ test points must therefore reduce the unbounded dimension by $N + 1$. Since the dimension of the error space for domain testing, given in equation 2, is $N + 1$, the various domain testing strategies all leave an untested error region with an unbounded dimension of 1 — that is, they leave one error direction only singly bounded but all directions orthogonal to that one are doubly bounded.

This is, in fact, the best possible unbounded dimension when testing for linear errors in linear borders, because there exists one error direction that can never be doubly bounded. This direction is the one parallel to the given border, denoted by $\hat{e}_{T'}$ where $T' = \alpha\hat{e}_{T'}$. Suppose, for example, that the given border is $x_1 + x_2 > 0$. Then we can add errors of the form $\alpha(x_1 + x_2)$, $\alpha > 0$ to this border without detecting the change since such changes can never really result in a domain error. Hence the error direction $\alpha(x_1 + x_2)$ cannot be doubly bounded. It can be singly bounded, however, because subtracting $x_1 + x_2$ from the given border does result in a detectable border shift.

Since the $N \times 1$, $N \times N$, and the $V \times V$ strategies all reduce the unbounded dimension to its minimum possible value, we have the following corollary:

Corollary 2 *The $N \times 1$, $N \times N$, and $V \times V$ strategies all satisfy the first subgoal for reducing the untested error region.*

If there is to be any difference in error detection ability among these three strategies, it must lie in the degree to which they satisfy the second subgoal, although this is probably not as important as the fact that they do all satisfy the first subgoal and may all therefore be considered effective means for detecting linear errors in linear borders. The next Theorem establishes that, with appropriate modification, the $N \times N$ strategy may be considered better than the $N \times 1$ strategy because it imposes tighter bounds on errors nearly parallel to the direction $\hat{e}_{T'}$.

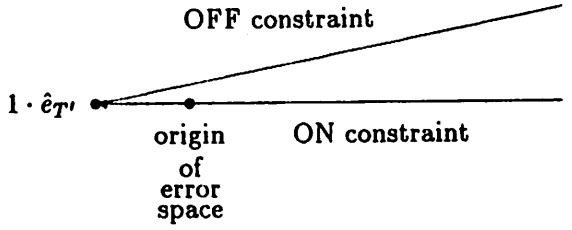

Figure 11: Two-Dimensional Slice of the Error Space for Nx1

Theorem 3 *The $N \times N$ strategy chooses better tests for detecting linear errors in linear borders than does the $N \times 1$ strategy, if the N* off *points are chosen near the N* on *points.*

Proof: The key to this Theorem is the fact that, since the error direction $\hat{e}_{T'}$ is only singly bounded by the domain testing strategies, there must be various error directions nearly parallel to $\hat{e}_{T'}$ that are also only singly bounded. As we increase the radius r from the origin, the volume of undetected errors associated with these directions that lie within r of the origin will eventually dominate any other contributions to the volume of the untested error region.

Now consider the shape of the untested error region after only the *on* points have been tested. Since $T'(\bar{x}) = 0$ for the *on* points, equation 5 reduces to $(\bar{e} \cdot \hat{e}_{x''})\ op_2\ 0$. Thus each of the error space constraints imposed by the *on* points intersect the origin of the error space. In fact, since none of the *on* points can bound $\hat{e}_{T'}$, which by definition passes through the origin, the line along which each of the error space constraints intersect is $\pm\hat{e}_{T'}$.

Now consider the addition of a single *off* point to satisfy the $N \times 1$ strategy. This point imposes a constraint intersecting $\hat{e}_{T'}$ at a distance of 1 from the origin, since subtracting T' from the given border T' results in a domain error detectable by any *off* point. It can be shown that, if we are only going to have one *off* point, that the best choice is to choose it near the centroid of the *on* points, as recommended in [19]. Then, if we were to view a two-dimensional slice of the error space containing the direction T', we would see something like Figure 11.

```
input X, Y, Z;
if X**2 + Y**2 =< 1.0 then
   if Z > 5.  then
```

Figure 12: Non-Linear Adjacent Borders

Clearly, the more narrow angle between the *on* and *off* constraints, the slower the increase in the size of the untested error region as we move away from the origin. The cosine of the angle between the *on* and *off* point constraints is given by $\hat{e}_{on} \cdot \hat{e}_{off}$, so the more nearly parallel the directions $\hat{e}_{on}$ and $\hat{e}_{off}$, the narrower the angle in Figure 11 would be. To make this angle as narrow as possible, therefore, we would like $\hat{e}_{on}$ and $\hat{e}_{off}$ to be nearly identical (except possibly for a change of sign). By equation 8, this means that the *on* and *off* points must be nearly identical. The $N \times N$ strategy can approaches this goal when the N *off* points are chosen near the same vertices as are the *on* points. (In [5] the *off* points were required merely to be near vertices of the given border but not necessarily near the same vertices as the *on* points.) Then, for any two-dimensional slice of the error space chosen orthogonal to an *on* point constraint and containing the origin and the direction $\hat{e}_{T'}$, there will be an *off* point constraint forming a more acute angle with the given *on* point constraint than would be obtained with the *off* point from the $N \times 1$ strategy. Consequently, the size of the untested error region should increase with distance from the origin more slowly for the $N \times N$ strategy than for the $N \times 1$ strategy and the Theorem is proven.

Thus we see that the $N \times N$ strategy does offer a slight edge over the $N \times 1$ strategy when testing for linear errors in linear borders, provided that the N *off* points are chosen near the N *on* points. It should be clear from the proof of Theorem 3, however, that the $V \times V$ strategy is not guaranteed to detect more errors than is $N \times N$ although it cannot do worse than $N \times N$ since by definition the test points chosen by the $V \times V$ strategy are a superset of those chosen by the $N \times N$strategy.

5.3 Non-Linear Domains

Little attention has been paid to the problem of applying domain testing where the program's borders are non-linear. This Section will examine this problem and will demonstrate that domain testing can be extended to non-linear domains with only a small increase in the number of test points, although the calculations involved in deriving those test points become more complex.

We begin with the simple step of allowing the borders of the path domain intersecting the given border to be non-linear. The main conceptual problem this creates is that we must abandon the previous emphasis upon the use of vertices of the path domain. For example, if we were testing the border imposed by the second IF statement in Figure 12, we would be faced with the problem that the given border consists of a circle and so has no vertices. The key to dealing with such borders is provided by Theorem 2, where it is established that the most important property of the

on points is that they be in general position, not that they occur at vertices. (This observation also allows us to drop the various assumptions made in [5,18] to ensure convex domains in the linear case.) For numerical reasons, and to help reduce the size of the factor $T'(\bar{x}^0)/\hat{e}_{\bar{x}^0}(\bar{x}^0)$ in Theorem 1, it is desirable that the points be scattered as far away from one another as possible, but this is secondary to having them in general position. Note also that, for borders such as this one, the $V \times V$ strategy loses all practical meaning, so that we will henceforth be concerned only with the $N \times 1$ and $N \times N$ strategies and their non-linear analogues.

Now consider the problem of formulating domain tests when the given border is non-linear. Theorem 1 can still serve to guide the selection of tests, since it does not depend upon linearity. Instead, Theorem 1 requires that the set of possible errors be a finitely dimensioned vector space, closed under the operations of addition and of multiplication by scalars. Examples of such spaces include not only linear functions but also polynomial and multinomial functions of fixed maximum degree. Other functions can often be treated by using a multinomial error space of sufficiently high degree to contain reasonable approximations of the true functions. It is therefore possible to formulate testing strategies simply by directly attempting to reduce the volume of the untested error region defined by the constraints of equation 5. Such strategies are discussed in [22,24], but they are extremely expensive, both in terms of the number of tests required and in terms of the complexity of the calculations required to derive each test.

In this Section, we will consider a less ambitious goal that is also more in keeping with the spirit of previous domain testing strategies. We will consider the problem of detecting linear errors in non-linear borders. Not only should this suffice to detect most domain errors in non-linear borders, but it is a prerequisite for detecting more general border shifts such as polynomial errors in polynomial borders. We can then show that, with the addition of a single test point each, the $N \times 1$ and $N \times N$ strategies can be made to satisfy the first goal for reducing the untested error region.

Theorem 4 *When testing for linear errors in non-linear borders, the $N \times 1$ and $N \times N$ strategies result in an untested error region with unbounded dimension zero provided that*

- *The* on *points are chosen in general position.*
- *Each* off *point is in general position with respect to the* on *points (although not necessarily with respect to any other* off *points).*
- *An additional test point is chosen that, compared to any* off *point, lies on the opposite side of the given border or on the opposite side of the hyperplane containing the* on *points, but not both.*

Proof: As explained earlier, the natural extension of the domain testing procedure for choosing *on* points involves selecting them in general position. Theorem 2 depended only upon the linearity of the errors, not upon the linearity of the borders, so after executing the *on* points we should have an unbounded error dimension of 2.

The point of divergence from the linear border analysis of Corollary 2 occurs when we attempt to determine the direction left unbounded. In the case of linear borders, this

unbounded direction was $\hat{e}_{T'} = \alpha T'$. In this case, however, T' is non-linear and so does not lie within the error space. Since, however, there are exactly N *on* points, there must exist a hyperplane in the N-dimensional input space containing those points. Let the equation of this plane be denoted by

$$\beta_0 + \sum_{i=1}^{N} \beta_i x_i = 0 \tag{9}$$

subject to

$$\sum_{i=0}^{N} \beta_i^2 = 1.$$

Note that the β_i can be obtained by the solution of the linear system of equations $\hat{\beta} \cdot \bar{x}^i = 0$ for all *on* points $\bar{x}^i$, $i = 1 \ldots N$. It follows by Theorem 1 that the error direction $\hat{\beta} \cdot \bar{x}$ is unbounded by the *on* points.[8] If T' is non-linear, it is possible for $\hat{\beta} \cdot \bar{x}$ to be doubly bounded, resulting in a completely closed untested error region, an impossibility in the purely linear case of Section 5.2.

By Theorem 2, the remaining error direction can be singly bounded by any test point that is in general position with respect to the *on* points. Consider the choice of any *off* point $\bar{x}^{off}$, chosen not only off of the border but also off of the hyperplane defined by equation 9. By Theorem 2, $\bar{x}^{off}$ singly bounds the remaining unbounded error direction. If we consider the error $\bar{e} = \alpha\hat{\beta}$, then by Theorem 1 the resulting constraint would be:

$$(\alpha\hat{\beta} \cdot \hat{e}_{\bar{x}^{off}})\ op_2\ (T'(\bar{x}^{off})/\hat{e}_{\bar{x}^{off}}(\bar{x}^{off})). \tag{10}$$

To obtain the final constraint necessary to reduce the unbounded dimension to zero, we must doubly bound the direction $\hat{\beta} \cdot \bar{x}$. Consider now the choice of an additional test point, $\bar{x}^{new}$, chosen off of the given border (to either the accepted or unaccepted side) and off of the hyperplane containing the *on* points. By Theorem 1 $\bar{x}^{new}$ imposes the constraint:

$$(\alpha\hat{\beta} \cdot \hat{e}_{\bar{x}^{new}})\ op_2\ (T'(\bar{x}^{new})/\hat{e}_{\bar{x}^{new}}(\bar{x}^{new})). \tag{11}$$

Both $\bar{x}^{off}$ and $\bar{x}^{new}$ constrain the direction $\hat{\beta} \cdot \bar{x}$. By equation 8, the expressions $\hat{e}_{\bar{x}^{off}}(\bar{x}^{off}))$ and $\hat{e}_{\bar{x}^{new}}(\bar{x}^{new}))$ are both positive. If, therefore, $T'(\bar{x}^{off})$ and $T'(\bar{x}^{new})$ have opposite signs but $\hat{\beta} \cdot \hat{e}_{\bar{x}^{off}}$ and $\hat{\beta} \cdot \hat{e}_{\bar{x}^{new}}$ have the same sign (i.e. $\bar{x}^{off}$ and $\bar{x}^{new}$ lie on opposite sides of the given border but on the same side of the *on*-point plane), then the two constraints are of the form $c_{off}\alpha\ op_2\ \pm\alpha_{off}$ and $c_{new}\alpha\ op_2\ \mp\alpha_{new}$ for some positive constants α_{off}, α_{new}, c_{off}, and c_{new}. The direction $\hat{\beta} \cdot \bar{x}$ is therefore doubly bounded. If, on the other hand, $T'(\bar{x}^{off})$

[8]Note that the effect in the purely linear case of Section 5.2 of moving one of the *on* points slightly off of the border and to the accepted side of the border constraint, as recommended at the start of Section 5., is simply to change the direction left unbounded by the *on* points from $\hat{e}_{T'}$ to this $\hat{\beta} \cdot \bar{x}$, which is nearly parallel to $\hat{e}_{T'}$. After the selection of any *off* point, both $\hat{\beta} \cdot \bar{x}$ and $\hat{e}_{T'}$ will be singly bounded, thus preserving the correctness of the analysis in Section 5.2.

and $T'(\bar{x}^{new})$ have the same sign but $\hat{\beta} \cdot \hat{e}_{\bar{x}^{off}}$ and $\hat{\beta} \cdot \hat{e}_{\bar{x}^{new}}$ have opposite signs (i.e. $\bar{x}^{off}$ and $\bar{x}^{new}$ lie on the same side of the given border but on opposite sides of the *on*-point plane), then the two constraints are of the form $\pm c_{off}\alpha \ op_2 \ \alpha_{off}$ and $\mp c_{new}\alpha \ op_2 \ \alpha_{new}$. Again, the direction $\hat{\beta} \cdot \bar{x}$ is doubly bounded.

Finally we note that the choice of the point $\bar{x}^{new}$ is required to reduce the unbounded dimension to zero for both strategies. The $N \times 1$ strategy simply does not choose enough test points to completely close an $N + 1$-dimensioned error space. The $N \times N$ strategy chooses enough test points, but if we continue to assume that the *off* points are chosen near the *on* points, then all the *off* points will occur on the same side of both the given border and the hyperplane containing the *on* points, since by definition that hyperplane intersects the given border at the *on* points. If the *off* points are not chosen near the *on* points, then the principal advantage of the $N \times N$ strategy, the narrowness of the angle between its constraints, is lost.

Thus we again see that the $N \times 1$ and $N \times N$ strategies are both successful at guaranteeing that *some* bound is imposed on the size of any error directions for which such bounds are possible. Unlike the purely linear case, however, here the resulting untested error region is completely closed. This fact is crucial to the interpretation of these strategies' performance with respect to the second goal. The key to the proof of Theorem 3 was the narrowness of the angle between constraints for the NxN strategy. This was decisive, however, only because the untested error region was open. In a closed region, it is entirely possible for the $N \times 1$ strategy to yield a smaller volume than would the $N \times N$ strategy and so it is not possible to categorically state that either strategy is always more effective than the other.

Should we then prefer the $N \times 1$ strategy simply because it requires little more than half the test points of the $N \times N$ strategy? The answer is not so simple. Not only is it possible for the $N \times N$ strategy to sometimes yield a smaller volume in a closed undetected error region, but there may be frequent situations where, even when testing non-linear borders, the untested error region will remain open because it is not possible to choose N *on* points in general position. This occurs when the subpath domain is contained within a hyperplane of the N-dimensional input space. Normally this results from execution of a prior equality predicate (e.g. `if A=B then ...`), but it can also be due to the conjunction of two or more inequalities (e.g. `if A >= B then ...if A =< B then ...`). In such situations, the *on* points will be limited to the hyperplane containing the subpath domain. As a result, there will be some error direction left unbounded.

The appropriate action to be taken by a testing strategy in the presence of such equality constraints is unclear. The unbounded error direction cannot result in a domain error (e.g. in the above examples, the addition of terms of the form α(A − B) to the given border cannot cause an error since such terms evaluate to zero everywhere within the subpath domain). One reaction is to therefore simply ignore such error directions and to consider the error space to be the linear subspace orthogonal to those directions. Alternatively, we can note that errors can be composed of linear combinations of these undetectable directions with other directions that can result in domain errors. The $N \times N$ strategy, by the arguments given in the proof of Theorem 3, would do

a better job of constraining those combinations and so might be considered preferable. The choice is a matter of judgment, and the author's opinion is that the second argument is not a sufficiently compelling reason to require the choice of nearly twice as many test points as would be needed for the $N \times 1$ strategy.

6. Domain Testing and Fault Detection

It would be reasonable to summarize the analysis of Section 5. by stating that domain testing represents an effective means of testing for linear border shifts on the first path employed during testing. As pointed out in Section 4.2, however, domain testing can be wasteful when testing more than one path since it fails to consider the possibility that paths may share one or more statements and that tests on one path therefore provide information about possible errors on the others.

In this Section, such interactions among paths will be dealt with by moving domain testing's past emphasis on errors to a new, fault-oriented, approach. Instead of considering linear errors in domain borders, this new approach will test for linear faults in the program predicates and assignments. First, a fault-space model is presented, similar to the error-space model of Section 5.. Next, this model will be used to develop fault-oriented versions of Theorem 1 to describe the set of possible faults that would be left undetected by a set of tests. Finally, an algorithm based upon these Theorems will be presented for choosing test points to detect linear faults in program predicates and assignment statements.

6.1 A Fault-Space Model For Domain Errors

Consider an initial subpath for the given program. Let the set of inputs for that subpath be denoted by $\bar{x}$. Let $\mathcal{V}$ be the set of variables in the given program. Then, arranging the members of $\mathcal{V}$ into some standard order, let $\bar{v}$ be a vector of values for those variables. Associated with the initial subpath P_i is a subpath computation $C_i : \bar{x} \rightarrow \bar{v}$. If P_i were the concatenation of subpaths P_1 and P_2, with P_1 initial, then we could associate subpath computations $C_1 : \bar{x} \rightarrow \bar{v}$ and $C_2 : \bar{v} \rightarrow \bar{v}$ with the two subpaths such that $C_i = C_2 \circ C_1$.

If P_i is an initial subpath ending at some conditional statement, then the predicate of that conditional statement can be viewed as a function of the values $\bar{v}^i$ of the program variables as computed by that path. Thus a predicate of the form $T(\bar{v})$ *relop* 0 would result in a border segment $T(C_i(\bar{x}))$ *relop* 0.

In Section 5. we were concerned with possible shifts in the function denoted here by $T \circ C_i$. In this Section, we will instead be concerned with possible shifts in the functions T and C_i separately.

Considering first the possibility of predicate errors, let T' be a predicate actually appearing in the program and let T be the correct form of that predicate. Then the fault in T' can be represented as

$$e = T' - T,$$

and, as before, we will be concerned with the linear case where

$$\bar{e}(\bar{v}) = \alpha_0 + \sum_{i=1}^{N} \alpha_i v_i. \tag{12}$$

$\bar{e}$ is now a vector in a *fault space*, and, as before, can be considered to have both a size and a direction.

Testing for linear faults is not the same as testing for linear errors. Consider a program having n inputs and m variables. If $m \geq n$ and if for any path leading to a predicate T' there are n variables whose values can be obtained from the inputs by an invertible linear transformation, then testing for linear faults in that predicate is at least as strong as testing for linear errors in all paths passing through that predicate. The simplest example of a program satisfying this criterion is any program in which n of the variables are used as placeholders for the inputs (e.g. a program beginning with `INPUT A, B, C;` and no subsequent redefinitions of `A`, `B`, and `C`). On the other hand, if $m < n$, then the inputs cannot usually be reconstructed from the variable values and hence there are certain linear errors that would go undetected by tests for linear faults. For example, a program that begins by looping n times through the statement `INPUT X`, collecting the sum of the input values but not saving the individual input values, would fall into this class. One can conceive of linear errors of forms such as $\alpha(x_2 - x_{12})$ that, because the individual x_i are not available from the program variables, would not be detected by a fault-oriented scheme. Of course, for such an error to be present in the program, the given version of the program would have to be so significantly different from any correct version that the chances of it performing correctly on any set of data, much less the extensive set that will be required to test for linear faults, would likely be negligible.

A similar construction can be given for faults in the right-hand side of an assignment statement. Suppose that an assignment statement whose transformation of the program variables is given by C' should have instead used the transformation C, with both transformations altering only the variable v_j (although the computation of the new value for v_j could depend on any number of other variables). Then we can define the fault in the assignment as

$$\bar{e} = v_j \circ C' - v_j \circ C \tag{13}$$

and again let $\bar{e}$ be a vector in the linear fault space described by equation 12.

For both predicate and assignment faults, our goal in choosing test points is now to reduce the size of the *untested fault region* analogous to the untested error region of the earlier discussion. It should not be surprising that tests can be shown to impose constraints on the untested fault region similar to those described in Theorem 1 for error regions.

6.2 Constraining the Untested Fault Region

We now turn to the problem of describing the sets of faults whose presence in the code would not result in a domain error on a given test. Theorem 5 shows the effect of a test upon the untested fault region for a predicate.

Theorem 5 *Given a predicate $T'(\bar{v})$ op_1 0 that has been tested with variable values $\bar{v}^0$, a fault $\bar{e}$ in T' fails to cause a domain error iff*

$$\bar{e}(\bar{v}^0) = 0$$

or

$$(\bar{e} \cdot \hat{e}_{v''})\ op_2\ (T'(\bar{v}^0)/\hat{e}_{v''}(\bar{v}^0)) \tag{14}$$

where op_1 and op_2 are given in Table 1 and where $\hat{e}_{v''}$ is the unique unit-length vector such that $\hat{e}_{v''}(\bar{v}^0) > 0$ and, for all $\hat{e}$ orthogonal to $\hat{e}_{v''}$, $\hat{e}(\bar{v}^0) = 0$.

Proof: The proof is identical to that of Theorem 1 with the program variables $\bar{v}$ substituted for the program inputs $\bar{x}$. In effect, Theorem 1 may be regarded as the special case of this Theorem where the only program variables are placeholders for the inputs.

The next Theorem establishes a similar result for assignment faults for test points chosen near to a domain border.

Theorem 6 *Given an assignment statement $v_j := C'(\bar{v})$ that has been tested with variable values $\bar{v}^0$ for which execution subsequently proceeded along the subpath P to some predicate $T(\bar{v})$ op_1 0 with $T(\bar{v})$ close to 0, a fault $\bar{e}$ in C' fails to cause a domain error iff*

$$\bar{e}(\bar{v}^0) = 0 \tag{15}$$

or

$$\left.\frac{\partial(T \circ C_P)}{\partial v_j}\right|_{v^{on}} = 0$$

or

$$(\bar{e} \cdot \hat{e}_{v''})\ op_2 \left(\left.\frac{\partial(T \circ C_P)}{\partial v_j}\right|_{v^{on}} \right)^{-1} \frac{T \circ C_P \circ C'_j(\bar{v}^0)}{\hat{e}_{v''}(\bar{v}^0)} \tag{16}$$

where op_1 and op_2 are given in Table 2 and where $\hat{e}_{v''}$ is the unique unit-length vector such that $\hat{e}_{v''}(\bar{v}^0) > 0$ and, for all $\hat{e}$ orthogonal to $\hat{e}_{v''}$, $\hat{e}(\bar{v}^0) = 0$.

Proof: If equation 15 is true, then clearly no domain error can result from this error in the border because no border shift occurs at that test point. If, however, the error in the border does result in some border shift, then a domain error results iff the shift is large enough that the given and correct borders take on different truth values on this test.

Since the given border is $T \circ C_P \circ C_j(\bar{v})$ op_1 0, a domain error results iff $T \circ C_P \circ C'_j(\bar{v}^0)$ op_1 0 and $T \circ C_P \circ C_j(\bar{v}^0)$ op_1 0 are different. We may therefore state that a domain error fails to occur exactly when

$$(T \circ C_P \circ C'_j(\bar{v}^0)\ op_1\ 0) \Leftrightarrow (T \circ C_P \circ C_j(\bar{v}^0)\ op_1\ 0).$$

op_1	$T \circ C_P \circ C_j'(v^0)\ op_1\ 0$	op_2	
		$f(v_j^0) > 0$	$f(v_j^0) < 0$
$>$	true	$<$	$>$
$>$	false	$\geq$	$\leq$
$\geq$	true	$\leq$	$\geq$
$\geq$	false	$>$	$<$
$<$	true	$>$	$<$
$<$	false	$\leq$	$\geq$
$\leq$	true	$\geq$	$\leq$
$\leq$	false	$<$	$>$
$=$	true	$=$	$=$
$=$	false	$\neq$	$\neq$
$\neq$	true	$\neq$	$\neq$
$\neq$	false	$=$	$=$

$$f(v_j) = \left.\frac{\partial(T \circ C_P)}{\partial v_j}\right|_{v^{on}}$$

Table 2: Operators for Theorem 6

The correct border $T \circ C_P \circ C_j$ is unknown, but from the definition of $\bar{e}$ in equation 13 we have that a domain error fails to occur iff

$$(T \circ C_P \circ C_j'(\bar{v}^0)\ op_1\ 0) \Leftrightarrow (T \circ C_P \circ (C_j' - \bar{e})(\bar{v}^0)\ op_1\ 0). \tag{17}$$

Let $\bar{u}_j^0(x)$ be the vector formed by replacing the jth component of $\bar{v}^0$ by x and let I_j denote the jth column of an $n \times n$ identity matrix where n is the dimension of $\bar{v}^0$. Then we can rewrite equation 17 as

$$(T \circ C_P \circ C_j'(\bar{v}^0)\ op_1\ 0) \Leftrightarrow (T \circ C_P \circ \bar{u}_j^0 \circ I_j \circ (C_j' - \bar{e})(\bar{v}^0)\ op_1\ 0). \tag{18}$$

This last step allows us to focus on the single-argument function $T \circ C_P \circ \bar{u}_j^0$, which describes the sensitivity of the subpath P to errors in the value of v_j. Since we will be choosing *on* and *off* points for which $T \circ C_P(\bar{v})$ is close to zero, we can approximate $T \circ C_P \circ \bar{u}_j^0$ by

$$T \circ C_P \circ \bar{u}_j^0(x) = T \circ C_P \circ \bar{u}_j^0(x') + \left.\frac{\partial(T \circ C_P \circ \bar{u}_j^0)}{\partial x}\right|_{x'} (x - x')$$

provided that $x - x'$ is small. If x' is an *on* point, however, $T \circ C_P \circ \bar{u}_j^0(x') = 0$, so

$$T \circ C_P \circ \bar{u}_j^0(x) = \left.\frac{\partial(T \circ C_P \circ \bar{u}_j^0)}{\partial x}\right|_{x^{on}} (x - x^{on}). \tag{19}$$

Inserting this approximation into equation 18 and defining $\bar{v}^{on} = \bar{u}_j^0(x^{on})$ yields

$$(T \circ C_P \circ C_j'(\bar{v}^0)\ op_1\ 0) \Leftrightarrow \left(\left.\frac{\partial(T \circ C_P)}{\partial v_j}\right|_{v^{on}} (v_j - v_j^{on}) \circ I_j \cdot (C_j' - \bar{e})(\bar{v}^0)\ op_1\ 0 \right) \quad (20)$$

Now if $T \circ C_P$ does not depend upon the variable v_j (i.e., $\left.\frac{\partial(T \circ C_P)}{\partial v_j}\right|_{v^{on}} = 0$), then clearly an assignment fault affecting only v_j cannot result in a domain error for the border imposed by T on this path. If $T \circ C_P$ does, however, depend upon v_j then equation 20 simplifies to

$$(T \circ C_P \circ C_j'(\bar{v}^0)\ op_1\ 0) \Leftrightarrow \left(\left.\frac{\partial(T \circ C_P)}{\partial v_j}\right|_{v^{on}} (I_j \cdot C_j'(\bar{v}^0) - v_j^{on})\ op_1\ \left.\frac{\partial(T \circ C_P)}{\partial v_j}\right|_{v^{on}} I_j \cdot \bar{e}(\bar{v}^0) \right)$$

and then to

$$\bar{e}(\bar{v}^0)\ op_2\ (I_j \cdot C_j'(\bar{v}^0)) - v_j^{on} \quad (21)$$

where op_2 is taken from Table 2.

By equation 19, however,

$$v_j^{on} = \left(\left.\frac{\partial(T \circ C_P)}{\partial v_j}\right|_{v^{on}} \right)^{-1} (I_j \cdot C_j'(\bar{v}^0) - T \circ C_P \circ C_j'(\bar{v}^0)).$$

Replacing v^{on} in equation 21 yields

$$\bar{e}(\bar{v}^0)\ op_2 \left(\left.\frac{\partial(T \circ C_P)}{\partial v_j}\right|_{v^{on}} \right)^{-1} T \circ C_P \circ C_j'(\bar{v}^0). \quad (22)$$

The remainder of the proof of this Theorem follows from equation 22 in the same fashion as Theorem 1 followed from equation 6.

The purpose served by the partial derivative in Theorem 6 is to determine whether or not the border generated on this subpath is sensitive to changes in the variable v_j, and, if so, whether the predicate function increases or decreases as v_j increases. It should be possible to obtain this information by approximating the derivative, sampling the values of $T \circ C_P$ at $\bar{v}$ and $\bar{v}^{on}$, even in circumstances (e.g. integer-valued predicates) where the derivative does not formally exist. If this does not yield a reasonable approximation, then the test point $\bar{v}$ is of dubious utility with this border since its effect on the border cannot be determined.

The similarity of these Theorems to Theorem 1 suggests that many of the other results of Section 5. for error spaces should also hold for fault spaces. For example, the choice of test sets in general position will still be important, though it is the sets of variable values that must lie in general position, not the sets of input values.

With these two Theorems, we are now in a position to describe a test data selection algorithm to detect linear faults in program predicates and assignment statements. The key idea behind

this algorithm is the use of Theorems 5 and 6 to maintain at each statement a description of the possible linear faults in that statement that would have escaped detection with all previous tests. The process of choosing a single test then consists of first choosing a fault direction to be bounded, setting up a mathematical programming problem describing the conditions for bounding that fault direction, attempting to solve the problem to obtain a test set, and checking the solution to see if it imposes an acceptable bound.

6.3 Test Data Selection for Predicate Faults

Let P be a given initial subpath ending at a predicate T' and with path computation C_P. Let $PC : \bar{x} \rightarrow \{\text{true}, \text{false}\}$ be the path condition associated with P such that P is executed on input $\bar{x}$ iff $PC(\bar{x}) = \text{true}$.

Let $\texttt{Single}_{T'}$ be the set of constraints of the form in equation 14 describing the results of any previous executions of T' that have resulted in a direction's being singly bounded. Let $\texttt{Double}_{T'}$ be the set of constraints of the form in equation 14 describing the results of any previous executions of T' that have resulted in a direction's being doubly bounded. If T' has never previously been executed, then $\texttt{Single}_{T'}$ and $\texttt{Double}_{T'}$ would be empty.

Given T', P, $\texttt{Single}_{T'}$ and $\texttt{Double}_{T'}$ a procedure for selecting test data that will result in the minimum possible unbounded dimension is:

1. Define a set of constraints `Unsuccessful` to describe the error directions that cannot be bounded by tests on P. Initially, let `Unsuccessful` be empty.

2. Choose test data for T':

   (a) Choose a target fault $\bar{e}_t$ as the fault maximizing $|\bar{e}_t|$ subject to the constraints in $\texttt{Single}_{T'}$ $\texttt{Double}_{T'}$ and `Unsuccessful`. The fault $\bar{e}_t$ is therefore the largest fault in the untested region of the fault space. Let $\hat{e}_t = \bar{e}_t/|\bar{e}_t|$.

   (b) If $|\bar{e}_t| < \delta$, for some small positive δ, then no further tests will be chosen for T' along this path.

   (c) Attempt to select an *on* point that bounds $\hat{e}_t$: If $\bar{v}$ is the set of variable values at the end of P when executed on input $\bar{x}$, then the fault direction $\hat{e}_v$ to which $\bar{x}$ is most sensitive is the normalization of $(1, v_1, v_2, \ldots, v_N)$. From Table 1, determine op_2 for *on* points. If op_2 is "$<$" or "$\leq$", attempt to find a set of input values $\bar{x}$ maximizing $\hat{e}_v \cdot \hat{e}_t$, otherwise find a set of input values minimizing $\hat{e}_v \cdot \hat{e}_t$. In either case, the optimization is subject to the constraints $PC(\bar{x})$, and $T' \circ C_P(\bar{x}) = 0$.[9] The constraint $PC(\bar{x})$ guarantees that $\bar{x}$ does cause execution of the subpath P. The constraint $T' \circ C_P(\bar{x}) = 0$ forces $\bar{x}$ to lie on the border.[10]

[9] A rough approximation to the maximum should suffice here – there would seem to be little to be gained by expending a great deal of effort to find the exact maximum once an $\bar{x}$ has been found that is in the right neighborhood.

[10] On the first path through T', this constraint should be replaced by $T' \circ C_P(\bar{x}) = \pm\epsilon$ with the sign of ϵ chosen to force $\bar{x}$ to lie slightly off of the border toward the closed side.

(d) Determine if $\bar{x}$ imposes an acceptable new bound on the fault space: The acceptability of $\bar{x}$ is determined by determining whether the direction $\hat{e}_v$ is reasonably close to the target direction $\hat{e}_t$. If $\hat{e}_v \cdot \hat{e}_t < 0$, then the bound imposed by $\bar{x}$ is in the wrong direction and so $\bar{x}$ is rejected. (In this case, it is likely that an *off* point can be found for this path that will bound $\hat{e}_t$.) If $0 \leq \hat{e}_v \cdot \hat{e}_t < \gamma$, for some small positive γ, then a bound is imposed on $\hat{e}_t$ but the bound is very loose ($\hat{e}_v$ is nearly orthogonal to the target direction) and so $\bar{x}$ is rejected.

(e) If the *on* point $\bar{x}$ is rejected, then attempt to choose an *off* point that bounds $\hat{e}_t$: From Table 1, determine op_2 for *off* points. If op_2 is "$<$" or "$\leq$", attempt to find a set of input values $\bar{x}$ maximizing $\hat{e}_v \cdot \hat{e}_t$, otherwise find a set of input values minimizing $\hat{e}_v \cdot \hat{e}_t$. In either case, the optimization is subject to the constraints $PC(\bar{x})$, and $T' \circ C_P(\bar{x}) = \pm\epsilon$. The sign of ϵ is chosen to force $\bar{x}$ to lie on the open side of the border.

(f) Determine if the *off* point $\bar{x}$ imposes an acceptable new bound on the fault space: If $\hat{e}_v \cdot \hat{e}_t > -\gamma$ for some small positive γ, then $\bar{x}$ is rejected.

(g) If an acceptable *on* or *off* point is not found, then then the constraint $\bar{e} \cdot \hat{e}_t = 0$ is added to `Unsuccessful` so that subsequent target directions will be chosen orthogonal to this $\hat{e}_t$.

(h) If $\bar{x}$ is not rejected, it becomes part of the test set for this program. The constraint $\bar{e} \cdot \hat{e}_v \; op_2 \; T'(\bar{v})/\hat{e}_v(\bar{v})$ is added either to $\texttt{Single}_{T'}$ if op_2 is not "$=$" or to $\texttt{Double}_{T'}$ if it is. This represents the effect on the fault space for T' of executing $\bar{x}$.

(i) Go back to step 2a to choose another *on* point.

3. Clean up the $\texttt{Single}_{T'}$ and $\texttt{Double}_{T'}$ sets of constraints:

(This step is optional and might best be performed only when the number of constraints in $\texttt{Single}_{T'}$ grows large.)

(a) Let `DoublyBounded` be a largest possible set of directions orthogonal to all directions $\hat{e}_t$ having constraints $\bar{e} \cdot \hat{e}_t = 0$ in `Unsuccessful`.

(b) Redefine $\texttt{Double}_{T'}$ to the set of all constraints $\bar{e} \cdot \hat{e}'_t = 0$ such that $\hat{e}'$ is in `DoublyBounded`. The point is to reduce the number of constraints to be considered on subsequent paths. Furthermore, the set $\texttt{Double}_{T'}$ can be used to help guide the choice of additional test paths, as the solution set to the system of linear equations in $\texttt{Double}_{T'}$ is a superset of the *blindness expressions* described in [20,21].

(c) Remove from $\texttt{Single}_{T'}$ any constraint $\bar{e} \cdot \hat{e}' \; op \; \alpha$ such that $\hat{e}'$ is contained within the span of the vectors comprising `DoublyBounded`. Such constraints are now subsumed by $\texttt{Double}_{T'}$.

6.4 Test Data Selection for Assignment Faults

Let P be formed from the concatenation of three subpaths, P_1, P_j, and P_2, where P_1 is an initial subpath, P_j consists of a single assignment to the variable v_j, and P_2 is a subpath from that

assignment to some predicate T. Let the computations for these subpaths be denoted by C_1, C_j', and C_2, respectively. Let $PC : \bar{x} \rightarrow \{\text{true}, \text{false}\}$ be the path condition associated with P such that P is executed on input $\bar{x}$ iff $PC(\bar{x}) = \text{true}$.

As in the algorithm for predicate faults, we will make use of the sets of constraints $\texttt{Single}_s$ and $\texttt{Double}_s$, where s denotes the assignment statement being tested. The major complication added to this algorithm is the need to verify that $T \circ C_2$ depends upon v_j, and to determine the direction in which $T \circ C_2$ changes as v_j is increased.

Given T, P, $\texttt{Single}_s$ and $\texttt{Double}_s$ a procedure for selecting test data that will result in the minimum possible unbounded dimension is:

1. If $T \circ C_2$ contains no reference to v_j, then no test data will be selected.

2. Define a set of constraints `Unsuccessful` to describe the error directions that cannot be bounded by tests on P. Initially, let `Unsuccessful` be empty.

3. Choose test data for s:

   (a) Choose a target fault $\bar{e}_t$ as the fault maximizing $|\bar{e}_t|$ subject to the constraints in $\texttt{Single}_s$ $\texttt{Double}_s$ and `Unsuccessful`. Let $\hat{e}_t = \bar{e}_t / |\bar{e}_t|$.

   (b) If $|\bar{e}_t| < \delta$, for some small positive δ, then no further tests will be chosen for s along this path.

   (c) Attempt to select an *on* point that bounds $\hat{e}_t$: Let $\bar{v}$ be the variables at the end of P_j when P is executed on input $\bar{x}$. From Table 2, determine op_2 for *on* points. If op_2 is "$<$" or "$\leq$", attempt to find a set of input values $\bar{x}$ maximizing $\hat{e}_v \cdot \hat{e}_t$, otherwise find a set of input values minimizing $\hat{e}_v \cdot \hat{e}_t$. In either case, the optimization is subject to the constraints $PC(\bar{x})$ and $T \circ C_P(\bar{x}) = 0$.

   (d) Let $\bar{v}'$ be a set of variables obtained from $\bar{v}$ by adding a small increment Δv_j to v_j. Estimate $\left.\frac{\partial(T \circ C_P)}{\partial v_j}\right|_{v^{on}}$ by $\frac{T \circ C_2(\bar{v}') - T \circ C_2(\bar{v})}{\Delta v_j}$.

   (e) Determine if $\bar{x}$ imposes an acceptable new bound on the fault space:
   If $\left.\frac{\partial(T \circ C_P)}{\partial v_j}\right|_{v^{on}}$ is nearly zero, reject $\bar{x}$.
   Reject $\bar{x}$ if $\hat{e}_v \cdot \hat{e}_t < \gamma$, for some small positive γ.

   (f) If the *on* point $\bar{x}$ is rejected, then attempt to choose an *off* point that bounds $\hat{e}_t$: From Table 2, determine op_2 for *off* points. If op_2 is "$<$" or "$\leq$", attempt to find a set of input values $\bar{x}$ maximizing $\hat{e}_v \cdot \hat{e}_t$, otherwise find a set of input values minimizing $\hat{e}_v \cdot \hat{e}_t$. In either case, the optimization is subject to the constraints $PC(\bar{x})$, and $T' \circ C_P(\bar{x}) = \pm\epsilon$. The sign of ϵ is chosen to force $\bar{x}$ to lie on the open side of the border.

   (g) Let $\bar{v}^{on}$ be a set of variables obtained from $\bar{v}$ by adding a small increment Δv_j to v_j such that $T \circ C_P(\bar{v}^{on}) = 0$.
   Estimate $\left.\frac{\partial(T \circ C_P)}{\partial v_j}\right|_{v^{on}}$ by $\frac{T \circ C_2(\bar{v}) - T \circ C_2(\bar{v}^{on})}{\Delta v_j}$.

(h) Reject $\bar{x}$ if $\frac{\partial(T \circ C_P)}{\partial v_j}\Big|_{on} = 0$ or if $\hat{e}_v \cdot \hat{e}_t > -\gamma$ for some small positive γ.

(i) If an acceptable *on* or *off* point is not found, then the constraint $\bar{e} \cdot \hat{e}_t = 0$ is added to `Unsuccessful` so that subsequent target directions will be chosen orthogonal to, or with negative components along, this $\bar{e}_t$.

(j) If $\bar{x}$ is not rejected, it becomes part of the test set for this program. The constraint described in equation 16 is added either to $\texttt{Single}_s$ if op_2 is not "=" or to $\texttt{Double}_s$ if it is.

(k) Go back to step 3a to choose another *on* point.

4. Clean up the $\texttt{Single}_s$ and $\texttt{Double}_s$ sets of constraints:

(a) Let `DoublyBounded` be a largest possible set of directions orthogonal to all directions $\hat{e}_t$ having constraints $\bar{e} \cdot \hat{e}_t$ *op* 0 in `Unsuccessful`.

(b) Redefine $\texttt{Double}_s$ to be the set of all constraints $\bar{e} \cdot \hat{e}' = 0$ such that $\hat{e}'$ is in `DoublyBounded`.

(c) Remove from $\texttt{Single}_s$ any constraint $\bar{e} \cdot \hat{e}'$ *op* α such that $\hat{e}'$ is contained within the span of the vectors comprising `DoublyBounded`. Such constraints are now subsumed by $\texttt{Double}_s$.

Although this paper has focused entirely upon domain errors, the above algorithm can also be used to detect computation errors. Suppose that T were an expression to be printed by an output statement instead of a predicate. We could treat this output as if it were a predicate of the form $T(\bar{v}) = f(\bar{x})$ where f is the correct function to be computed by this program. An incorrect output value would correspond to a shift in this predicate, so the *on* point selection rules are applicable to the detection of computation errors caused by linear faults (We would not be able to generate any *off* points. In fact, if the program were correct then there would be no *off* points for this predicate).

6.5 Sharing Test Points

The above algorithms assume that we are testing a single predicate or assignment statement on a given path. Through the use of the sets of constraints $\texttt{Single}_s$ and $\texttt{Double}_s$, the algorithms allow for the interactions among test points chosen for those same statements along different paths. In practice, however, the test points that are chosen for a given statement will also cause the execution of other statements as well. The final step in the formulation of a strategy for linear fault detection is to take these interactions into account as well when applying the predicate and assignment fault algorithms.

The following procedure describes how to apply the above algorithms to a given path P to take interactions among tests for different statements into account:

1. Initially, set $\texttt{Single}_s$ and $\texttt{Double}_s$ to the empty set for all statements s. If s is a conditional statement with a linear predicate $T(\bar{v})$ *relop* 0, then add the constraint $\bar{e} \cdot T \geq 0$ to $\texttt{Single}_s$.

2. Choose a test path P from the start to the end of the module being tested.

3. For each occurrence of a statement s in P do the following:

   (a) If s is a conditional statement, then apply the algorithm of Section 6.3 until done or until a single new test is generated.

   (b) If s is an assignment statement, then find a subsequent predicate or output statement in P and apply the algorithm of Section 6.4 until done or until a new test is generated. If no test is obtained, try another predicate or output statement in P until all such have been exhausted.

   (c) If a test has been generated, execute that test and record its effect on $\texttt{Single}_{s'}$ and $\texttt{Double}_{s'}$ for all occurrences of statement s' along P:

      i. If s' is a conditional statement then let $\bar{v}^0$ be the variable values when s' is reached. If $\hat{e}_{v^0}$ satisfies the equalities in $\texttt{Double}_{s'}$ then this test has a negligible effect on the untested fault region for s'. Otherwise, add the constraint given be equation 14 to $\texttt{Single}_{s'}$ if op_2 is not "=", to $\texttt{Double}_{s'}$ if it is.

      ii. If s' is an assignment statement, then let $\bar{v}^0$ be the variable values when s' is reached. If $\hat{e}_{v^0}$ satisfies the equalities in $\texttt{Double}_{s'}$ then this test has a negligible effect on the untested fault region for s'. Otherwise, set $c_1 = c_2 = M$ for some large M and perform the following steps for each output statement or predicate appearing in P after s' until $c_1 < \gamma$ and $c_2 < \gamma$ for some small positive γ or until no further output statements or predicates remain in P.

         A. Construct the constraint of the form given in equation 16 for s' and the chosen output or conditional statement.

         B. If the resulting constraint can be written as $\bar{e} \cdot \hat{e}_{v^0} < \alpha$ or $\bar{e} \cdot \hat{e}_{v^0} \leq \alpha$ with α non-negative and if $\alpha < c_2$, then set $c_2 = \alpha$.

         C. If the resulting constraint can be written as $\bar{e} \cdot \hat{e}_{v^0} > \alpha$ or $\bar{e} \cdot \hat{e}_{v^0} \geq \alpha$ with α non-positive and if $-\alpha < c_1$, then set $c_1 = -\alpha$.

      iii. If $c_1 < \gamma$ and $c_2 < \gamma$ for some small positive γ, then add the constraint $\bar{e} \cdot \hat{e}_{v^0} = 0$ to $\texttt{Double}_{s'}$. Otherwise, if $c_1 < M$ add $\bar{e} \cdot \hat{e}_{v^0} \geq -c_1$ to $\texttt{Single}_{s'}$ and if $c_2 < M$ add $\bar{e} \cdot \hat{e}_{v^0} \leq c_2$ to $\texttt{Single}_{s'}$.

   (d) If not finished generating tests for s on P, return to step 2a.

6.6 An Example Of Linear Fault Detection

To illustrate the algorithms presented in the previous sections, we now return to the sample program fragment that has been employed throughout this paper and show the test data that would be selected for a single test path through that program. The program fragment is shown again in Figure 13, with line numbers added to facilitate the discussion that follows. We will choose as the test path to be examined the path through statements $(1, 2, 3, 4, 5, 6, 5, 7, 8)$, the path that reaches statement 8 after exactly one iteration of the loop.

```
1   input X, Y;
2   if X >= 0.  then
3       Z := Y - X;
4       D := 0.;
5       while Z > D loop
6           D := D + 1.;
        end loop;
7       if X <= D then
8           if Y <= 1.001*D then
            ⋮
```

Figure 13: Sample Program

The first statement for which test data is generated is statement 2. The linear fault space for this statement is the set of faults of the form $\alpha_0 + \alpha_1 X + \alpha_2 Y$. Initially, there are no constraints upon this fault space, because no tests have yet been run.

The first choice for the target direction $\bar{e}_t$ might therefore be $(1, 1, 1)$ or, alternatively, $1 + X + Y$. Normalizing yields $\hat{e}_t = (0.57735, 0.57735, 0.57735)$, and so we begin by trying to find a point $\bar{x}$ maximizing $\hat{e}_t \cdot \hat{e}_0$, where $\hat{e}_0 = \frac{(1,X,Y)}{\sqrt{1^2+X^2+Y^2}}$, subject to the constraint $X = 0.001$. The constraint in this case reflects the desire to have the first test point for this statement lie slightly off of, but on the accepted side of the border. The subpath condition is empty (true) since this is the first conditional statement reached. In practice, we need to add implied bounds for X and Y describing the minimum and maximum values they can take on. In this example, we will use $-10000 \leq X, Y \leq 10000$.

The maximum value of $\hat{e}_t \cdot \hat{e}_0$ is found approximately at $X = 0.001, Y = 0.96647$. At this point, $\hat{e}_t \cdot \hat{e}_0 = 0.816793$, an acceptably large value. $\hat{e}_0 = 0.71906 + 0.0071906X + 0.69495Y$, and $T'(\bar{v}) = 0.001$, so by Theorem 5, the use of this test point imposes the constraint

$$\bar{e} \cdot (0.71906, 0.0071906, 0.69495) \leq 0.00719006$$

upon the untested fault region. The input point (0.001, 0.96647) becomes the first test for this program.

The next step is to determine the effect of this test on the other statements in the program. For example, statement 3 is reached by this test, and so certain possible assignment faults may be eliminated by this test. To determine which faults are eliminated, we apply Theorem 6 to statement 3. This requires finding a predicate or output statement reached after executing statement 3 that depends upon the value of Z. Statement 5 is executed twice on this test path, and each occurrence can be used. Since there are no new assignments between statements 2 and 3, $\hat{e}_0$ and $\hat{e}_0(\bar{v})$ are the same for statement 3 as for statement 2. The function $T \circ C_P$ for the first occurrence of statement 5 is $Y - X$ and for the second is $Y - X - 1.0$. Consequently, the constraints

$$\bar{e} \cdot (0.71906, 0.0071906, 0.69495) < 0.69423$$

Stmt	Test Point
2	(0.001, 0.96647)
	(0.0, -10000.)
	(0.0, 10000.)
	(-0.001, 0.091766)
3	(10000., 10000.)
	(9990., 10000.)
	(0.0, 0.001)
4	(0.0, 0.0)
7	(1.001, 2.001)
	(1.0000, 1.0060)
8	(0.30148, 1.0020)
	(1.0000, 1.0010)

Table 3: Test Data

and

$$\bar{e} \cdot (0.71906, 0.0071906, 0.69495) \geq -0.024829$$

are added to $\texttt{Single}_s$ for statement 3. The effects of this test on the remaining statements are found in a similar fashion.

Returning to the problem of choosing tests for statement 2, the next target direction could be $\bar{e}_t = (-1, -1, -1)$. The maximum value of $\hat{e}_t \cdot \hat{e}_v$ for this target, subject to the constraint $\texttt{X} = 0.0$, is obtained at $\texttt{X} = 0, \texttt{Y} = -10000$. For this point, $\hat{e}_t \cdot \hat{e}_v = 0.577293$. This point is added to the test set, and its effects on the other statements determined.

Next, the target direction $\bar{e}_t = (-1, -1, 1)$ yields a maximum $\hat{e}_t \cdot \hat{e}_v = 0.577293$ at $\texttt{X} = 0.0, \texttt{Y} = 10000$. After adding this point to the test set, the next target direction $\bar{e}_t = (-1, -1, -0.0001)$ yields a maximum value of $\hat{e}_t \cdot \hat{e}_v = 0.0000$ for any *on* point. An *off* point can be found, however, that yields a minimal value of $\hat{e}_t \cdot \hat{e}_v = -0.70345$ at $\texttt{X} = -0.001, \texttt{Y} = 0.091766$. After this point is added, the largest target $\bar{e}_t$ is (0, -1, 0), which is already singly bounded and cannot be doubly bounded. Hence neither an acceptable *on* nor *off* point is found, and $\bar{e} \cdot (0, -1, 0) = 0$ is added to `Unsuccessful`. There are then no remaining target directions other than $(0, 0, 0)$, and we can move on to the choice of test points for statement 3, subject to the constraints imposed on the untested fault region by the previous tests and subject to the path condition $\texttt{X} \geq 0$ imposed by our designated test path.

The remaining tests chosen for this test path are shown in Table 3. These tests are also shown in Figure 14, although the points at ±10000. are not shown to proper scale. Note that no tests were chosen explicitly for statements 5 and 6, because the tests chosen for the earlier statements proved sufficient. If additional test paths are chosen, only statements 7 and 8 will require additional test points, and no more than two points each will be required. This is because the fault direction $1 - \texttt{D}$ has yet to be bounded for those two statements, since they have only been executed on paths for

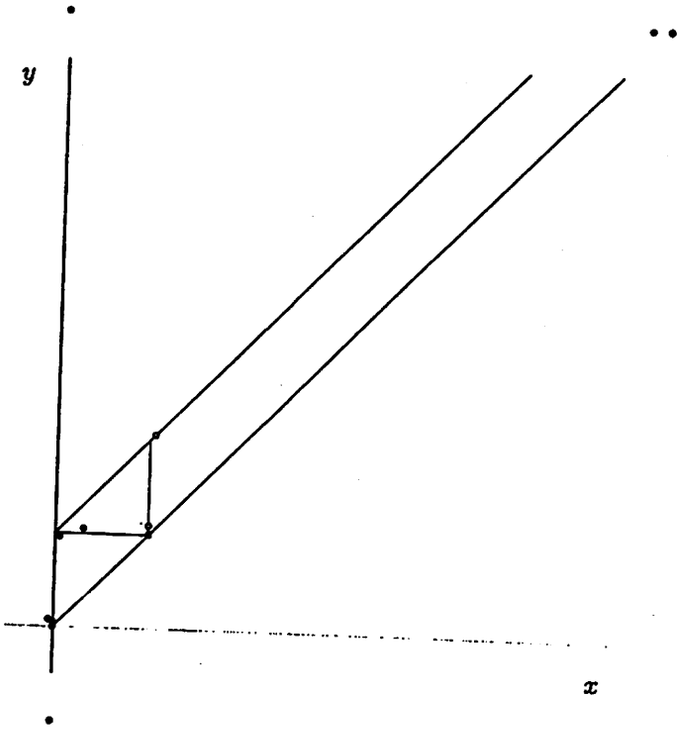

Figure 14: Distribution of Test Points

which $D = 1$. All other linear fault directions that can be bounded have been given reasonably tight bounds.

7. Conclusions

The domain testing strategies have been proposed as a means of detecting errors in the control flow through programs consisting primarily of numerical calculations. This paper has shown that the arguments that have been used to establish the power of domain testing have been flawed, but that their conclusion was nonetheless correct, in that all of the variants of domain testing can be shown to establish an upper bound on the size of any linear error in a domain border.

On the basis of this analysis, extensions to domain testing can be proposed. This paper has shown how domain testing can be employed on linear borders in programs containing non-linear calculations. Furthermore, with the addition of only one more test point, domain testing can be employed to detect linear errors in non-linear borders.

Conventional domain testing does not take into account the fact that different paths through the program usually share many statements, so that tests along one path provide information about the possible errors along other paths. By shifting the focus of attention from errors along paths to faults in individual statements, an algorithm is obtained that detects linear faults in arithmetic expressions that could result in either domain errors or computation errors. The resulting testing method is similar to domain testing in the kinds of faults and errors detected, but requires many fewer test points, at the cost of additional computations to determine each point.

REFERENCES

[1] T. A. Budd, "Mutation Analysis: Ideas, Examples, Problems and Prospects," *Computer Program Testing*, B. Chandrasekaran and S. Radichi (eds.), North-Holland Publishing Co., 1981, pp. 129–148

[2] T. A. Budd and D. Angluin, "Two Notions of Correctness and Their Relation to Testing," *Acta Informatica*, vol. 18, (1982), pp. 31–45

[3] T. A. Budd, *The Portable Mutation Testing Suite*, University of Arizona technical report TR 83–8, March 1983

[4] T. E. Cheatham Jr., G. H. Holloway, and J. A. Townley, "Symbolic Evaluation and the Analysis of Programs", *IEEE Transactions on Software Engineering*, vol. SE–5, 4, July 1979, 402–417

[5] L. A. Clarke, J. Hassell, and D. J. Richardson, "A Close Look at Domain Testing," *IEEE Transactions on Software Engineering*, vol. SE–8, no. 4, 380–390, July 1982

[6] L. A. Clarke and D. J. Richardson, "Symbolic Evaluation — an Aid to Testing and Verification," *Proceedings of the Symposium on Software Validation*, North-Holland Publishing Co., H.–L. Hausen (ed.), 1983

[7] E. I. Cohen, *A Finite Domain-Testing Strategy for Computer Program Testing*, Ph.D. dissertation, 1978, Ohio State University

[8] R. A. DeMillo, F. G. Sayward, and R. J. Lipton, "Program Mutation: A New Approach to Program Testing," *State of the Art Report on Program Testing*, 1979, Infotech International

[9] K. A. Foster, "Error Sensitive Test Cases Analysis (ESTCA)", *IEEE Transactions on Software Engineering*, vol. SE–6, no. 3, 258–264, May 1980

[10] J. S. Gourlay, "A Mathematical Framework for the Investigation of Testing," *IEEE Transactions on Software Engineering*, vol. SE–9, no. 6, pp. 686–709, November 1983

[11] W. E. Howden, "Methodology for the Generation of Program Test Data", *IEEE Transactions on Computers*, vol. C–24, no. 5, 554–560, May 1975

[12] W. E. Howden, "Reliability of the Path Analysis Testing Strategy", *IEEE Transactions on Software Engineering*, vol. SE–2, no. 3, 280–215, September 1976

[13] W. E. Howden, "Functional Program Testing",*IEEE Transactions on Software Engineering*, vol. SE–6, no. 2, March 1980, 162–169

[14] J. W. Laski and B. Korel, "A Data Flow Oriented Program Testing Strategy", *IEEE Transactions on Software Engineering*, vol. SE–9, no. 3, 347–354, May 1983

[15] S. J. Ntafos, "On Required Elements Testing," *IEEE Transactions on Software Engineering*, vol. SE-10, no. 6, 795 803, November 1984

[16] S. Rapps and E. J. Weyuker, "Data Flow Analysis Techniques for Test Data Selection", *Proceedings of the Sixth International Conference on Software Engineering*, 1982, IEEE Computer Society, 272–278

[17] D. J. Richardson and L. A. Clarke, "A Partition Analysis Method to Increase Program Reliability," *Fifth International Conference on Software Engineering*, March 1981, pp. 244–253

[18] L. J. White and E. I. Cohen,"A Domain Strategy for Computer Program Testing",*IEEE Transactions on Software Engineering*, vol. SE–6, no. 3, 247–257, May 1980

[19] L. J. White and I. A. Perrera, "An Error Analysis of the Domain Testing Strategy", *Workshop on Software Testing*, July 1986, IEEE

[20] S. J. Zeil, *Selecting Sufficient Sets of Test Paths for Program Testing*, Ph.D. dissertation, 1981, Ohio State University, also technical report OSU–CISRC–TR–81–10

[21] S. J. Zeil, "Testing for Perturbations of Program Statements", *IEEE Transactions on Software Engineering*, SE–9, No. 3, May 1983, pp. 335–346

[22] S. J. Zeil, *Perturbation Testing for Domain Errors*, COINS Technical Report 83–38, University of Massachusetts, December 1983

[23] S. J. Zeil, "Perturbation Testing for Computation Errors", *Seventh International Conference on Software Engineering*, March 1984, IEEE, also University of Massachusetts Technical Report 83–23, July 1983

[24] S. J. Zeil, *Perturbation Testing for Domain Errors*, submitted to IEEE Transactions on Software Engineering