Perturbation Testing for Domain Errors

Steven J. Zeil

COINS Technical Report 83-38
December 1983
Revised: February 1984

Department of Computer and Information Science
University of Massachusetts
Amherst, Massachusetts 01003

This work was supported by the National Science Foundation, grant no. MCS-8210084

Abstract

Perturbation testing is an approach to software testing which focuses on errors within arithmetic expressions appearing throughout a program. In this paper perturbation testing is generalized to permit analysis of individual test points rather than entire paths, and to concentrate on domain errors. Errors are modeled as perturbing functions drawn from a vector space of potential errors and added to the correct form of an arithmetic expression. It is possible to derive the set of error functions to which a given input is blind. For those errors to which the input data was not blind, sensitivity measures can be derived which limit the possible size of those errors. The combination of these new measures with standard optimization techniques opens up new possibilities for testing strategies, either alone or in concert with established path selection techniques.

These measures are used to evaluate the domain testing strategy and its variants. It is shown that both the original and the improved versions achieve their principal goal, and simple extensions are proposed to apply these strategies to more elaborate programs.

I. Introduction

Testing newly written programs for possible errors is considered a basic necessity by any professional in computer science. The sad fact is that such testing is usually conducted in an ad hoc, informal manner, guided as much by instinct as by any understanding of how to properly test software. There is an obvious need for testing techniques with more rigor and demonstrable levels of reliability. Such techniques need not, indeed, can not provide complete reliability for all programs, but techniques which can be shown to capture useful classes of errors for wide varieties of programs would be quite valuable.

A classification of errors which has proven useful is the division into domain and computation errors. A *domain error* occurs when incorrect output is generated due to executing a wrong path through the program [3]. 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. Discussions on the detection of computation errors may be found in [3,4,10], among others.

This paper will be concerned with the detection of domain errors, but the techniques to be presented here take as their starting point the author's analysis of computation errors in [10]. 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 which, 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 [3,6]. In this paper, we will be concerned with path selection errors. Test data which captures these errors should catch most missing path errors, but in general it is undecidable whether a program is missing a path.

The simplest case of a domain error is the *predicate fault*, in which a fault in a conditional statement causes execution to flow down the wrong path for some input data. Since the evaluation of a predicate usually depends on previous computations, a fault in an assignment statement can also cause domain errors. This will be referred to as an *assignment fault*. A method to detect both of these forms of path selection errors has been proposed by White and Cohen [6], with improvements suggested by Clarke et al. [1], but this method presumes that the paths to be tested have already been selected. It is hoped that the theory outlined in this paper will eventually shed some light on how to effectively choose paths for use with that method. In addition, the theory presented here is applicable to a wider range of programs, providing more flexibility in the types of functions which may be employed as program computations and errors.

Previous work by the author has introduced *perturbation testing*, a method that focuses on errors in arithmetic expressions appearing throughout a program [7,8,9,10]. Errors in arithmetic expressions are represented as perturbing functions added to the correct form of the expression. It is then possible to derive the set of potential errors in a chosen functional class which would have escaped detection with a given set of test data. For example, test runs which pass through an assignment statement "X := f(Y)" are incapable of revealing whether the perturbing function "X − f(Y)" might have been added to later expressions. Other test data might avoid that assignment statement, and so would be capable of detecting that particular error. In general, there are an infinite number of such

undetectable error perturbations for any test run. However, when the functional class of of error expressions being investigated is well-behaved, it is possible to compute a finite description of that infinite set of untested possible errors, which can then be used to guide the selection of additional test data.

Perturbation testing has been applied to selecting paths for testing in order to catch both domain and computation errors [8,9], and has more recently been extended to evaluating the power of input data points (rather than entire paths) in detecting computation errors [10]. This paper will extend the theory of perturbation testing to provide a measure of the ability of data points to detect domain errors due to predicate or assignment faults. This measure may be used as a guide to the selection of test data, or as a tool in evaluating the effectiveness of other testing techniques. Section II reviews the basic model underlying perturbation testing. Sections III and IV derive expressions for predicate and assignment faults, respectively, showing the set of untested error directions and the sensitivity with which the remaining directions have been tested after any test run. Section V discusses possible testing strategies based on these new sensitivity measures. The remaining sections contain an analysis of the domain testing method of White and Cohen [6] and its variations as proposed by Clarke, Hassell, and Richardson [1], in which a perturbational analysis is used to examine the power of these methods, and to suggest extensions of the method to more general programs.

II. The Perturbation Model

We begin by presenting a model of programs as groups of functional components and of errors as perturbing functions added to the correct components. The state of the program at any point in its execution is described in terms of the current environment $\bar{v}$, a vector containing the current values of the program inputs and program variables. In the initial environment, denoted by $\bar{v}_0$, only the input values are considered to be defined. We will often be discussing these initial environments in terms of their relation to the set of all possible inputs, in which case we will treat them as points (i.e. cartesian coordinates) within an input space.

As execution proceeds down some path, the assignment statements along that path transform the environment, computing new values for the program variables. For any subpath P_A we can designate a function C_A which represents a transformation equivalent to those computations:

$$\bar{v}_A = C_A(\bar{v}_0).$$

Predicates are represented as functions, $T(\bar{v})$, which are applied to the current environment and then compared to zero in order to determine the subsequent control flow. The comparison to zero may employ any of the conventional relational operators for real arithmetic.

Path selection errors can be caused by errors in the program predicates or by errors in assignment statements whose results are directly or indirectly used by later predicates. Turning first to the case where a predicate is in error, consider a predicate whose proper

form is given by the function T, but which has been mistakenly replaced with the function T′. The error in T′ is defined as the difference of T and T′, and will be denoted by $\bar{e}$,

$$\bar{e} = T' - T.$$

In practice, we will postulate a set of possible functions for $\bar{e}$, and that set will be chosen to be a vector space. Examples of possible classes of $\bar{e}$ would be the set of linear functions or the set of multinomials of arbitrary, fixed degree. Part of the power of perturbation testing stems from the ability of these functions to serve as approximators for other, less manageable functions. The advantage of dealing with vector spaces of functions is that a linearly independent set of characteristic functions can be chosen for any vector space such that any function in the space can be uniquely expressed as a linear combination of those characteristic functions. The coefficients of that linear combination can then be viewed as coordinates in an "error space", with the axes of the coordinate system being those characteristic functions.

Although the choice of the class of functions to be investigated as potential errors is left to the discretion of the tester, a simple inspection of the code can often suggest appropriate classes. In fact, the initial choice should probably be kept quite conservative (e.g. linear functions), since if later inspection should reveal the first choice to be to limited, a more general functional class can be substituted with no penalty for the initial mistake as long as the new class of errors entirely contains the original class.

$\bar{e}$ can be split into two parts,

$$\bar{e} = \alpha\hat{e}$$

where α is any non-zero real number and $\hat{e}$ has been normalized. The reason for this normalization is the separation of the "size" of the error, α, from the "direction" of the error $\hat{e}$.

Similarly, for assignment faults we will treat the replacement of a basic block of assignment statements C by C′ as the addition of a perturbing function $\alpha\hat{e}$, where $\hat{e}$ is, as before, a normalized direction drawn from a vector space of potential errors. An important distinction between the error functions for predicate and assignment faults is that $\hat{e}(\bar{v})$ returns a single value when used with predicates but returns a vector when representing an assignment fault, with each element in the vector representing an erroneous perturbation to a different variable in the environment.

III. Predicate Faults

Suppose that a test run has been conducted with input data $\bar{v}_0$. Under what circumstances would the predicate fault $\alpha\hat{e}$ be detected using this input data? Assuming, of course, that $\bar{v}_0$ causes execution to pass through the erroneous predicate, there are two possible ways for $\alpha\hat{e}$ to go undetected.

The first possibility is that T and T′ have exactly the same value. More formally, let C be the function representing the change in the program environment caused by the computations performed prior to reaching T′. The predicate, like any expression in a program, is evaluated on the environment $C(\bar{v}_0)$ which results from the prior computations, and so

$$T' \circ C(\bar{v}_0) = T \circ C(\bar{v}_0)$$

is a sufficient condition for the error going undetected. Since $T' = (T + \alpha\hat{e})$, this condition reduces to

$$\alpha\hat{e} \circ C(\bar{v}_0) = 0,$$

and since α is non-zero (otherwise there is no error),

$$\hat{e} \circ C(\bar{v}_0) = 0. \tag{1}$$

When this condition is satisfied, $\bar{v}_0$ will be said to be *blind* to $\alpha\hat{e}$. This is a slightly simplified form of the principle theorems of [9]. It should be noted that equation (1) depends only on the direction of the error, not on the size.

Equation (1) can be solved for those errors to which a given $\bar{v}_0$ is blind whenever $\hat{e}$ is believed to lie within a vector space. Even when $\hat{e}$ and C are nonlinear functions, (1) is still a linear equation in the vector space coordinates and hence is solvable with standard Gaussian methods.

The second possible way for $\alpha\hat{e}$ to go undetected is associated with the size of the error and represents the major addition to the theories previously developed in [8,9,10]. If (1) does not hold, then $\hat{e}(\bar{v})$ is non-zero, where $\bar{v}=C(\bar{v}_0)$ is the environment when the predicate is reached. There are two ways in which the error term can be non-zero without changing the result of the predicate at $\bar{v}_0$. First, if $T(\bar{v})$ and $\bar{e}(\bar{v})$ have the same sign (The possibility that $T(\bar{v})=0$ is discussed later.), then $T' = T + \alpha\hat{e}$ must have the same sense as T at $\bar{v}$ and the result of comparing the two functions to zero must be the same. Second, if $T(\bar{v})$ and $\alpha\hat{e}(\bar{v})$ have opposite signs, but $|T(\bar{v})| > |\alpha\hat{e}(\bar{v})|$, then T′ will still have the same sense as T, in this case because the change wrought by the error was not large enough to affect the sense of the predicate. More formally, the error can be missed when

$$T'(\bar{v}) \neq T(\bar{v}) \tag{2}$$

and

$$(T'(\bar{v}) \text{ inequ } 0) \quad \rightarrow \quad (T(\bar{v}) \text{ inequ } 0)$$

where "inequ" can be replaced with "<", "≤", ">", or "≥", whichever originally appears in the predicate (The "=" and "≠" operators will be examined later). Equation (2) implies that $\alpha\hat{e}(\bar{v})$ is non-zero. If $T(\bar{v})$ and $\alpha\hat{e}(\bar{v})$ have the same sign, then T′ must have the same sense as T at $\bar{v}$ and the result of comparing the two functions to zero must be the same.

Alternatively, if $T(\bar{v})$ and $\alpha\hat{e}(\bar{v})$ have opposite signs, but $|T(\bar{v})| > |\alpha\hat{e}(\bar{v})|$, then $T' = T + \alpha\hat{e}$ will still have the same sense as T, in this case because the change wrought by the error was not large enough to affect the sense of the predicate.

This suggests that, for a given error direction $\hat{e}$ and a given test input $\bar{v}_0$, there is a critical size α below which errors of the form $\alpha\hat{e}$ cannot be detected. This critical value will be denoted by $\alpha_c(\hat{e},\bar{v}_0)$, and is defined by the point at which T and T′ acquire opposite senses as $|\alpha|$ is increased. Since T′ is given in the source code, we view T as varying with α and $\hat{e}$ and ask, for a given input, when is $T(\alpha,\hat{e}) \circ C(\bar{v}_0)=0$? (For our purposes it will not be necessary to worry about whether the sense changes at the border point $T(\bar{v}) = 0$ or just beyond it at $T(\bar{v}) = \pm\epsilon$.) Substituting for T in terms of T′ and the error:

$$(T' - \alpha_c(\hat{e},\bar{v}_0)\hat{e}) \circ C(\bar{v}_0) = 0.$$

Solving for α_c gives

$$\alpha_c(\hat{e},\bar{v}_0) = T' \circ C(\bar{v}_0) \,/\, \hat{e} \circ C(\bar{v}_0). \qquad (3)$$

The critical value α_c separates the errors $\alpha\hat{e}$ into tested and untested ranges. Any error $\alpha\hat{e}$ for which α has the opposite sign of α_c or a smaller absolute value than α_c will go undetected. Thus it should be possible to define a region within the postulated space of possible errors which contains all as yet untested errors.

As a first step towards describing such a region, consider some special cases of equation (3). One such case occurs when the input data $\bar{v}_0$ lies exactly on the domain border $T'\circ C$. In this case, $T'\circ C(\bar{v}_0) = 0$, and so α_c will be zero, indicating that any shift of the proper direction in the value of the predicate should be detectable. This is correct, but is the error detected for positive or for negative shifts? The answer will depend on the relational operator used with the predicate. If, for example, the condition were "$T'(\bar{v}) < 0$", then any $\bar{v}_0$ which lies on the border is yielding a "false" result for this condition. An error in T′ is detected only if the answer should have been "true", so $T(\bar{v})$ must be less than zero. Since we have chosen the input $\bar{v}_0$ so that $T'(\bar{v}) = 0$, we conclude that the error is detected only when $\alpha\hat{e}(\bar{v}) = T'(\bar{v}) - T(\bar{v})$ is positive. Thus if α has the same sign as $\hat{e}(\bar{v})$, the error is detected no matter how close to zero α might be, but if α and $\hat{e}(\bar{v})$ have opposite signs, the error goes undetected. By similar arguments, we reach the same conclusion for "$\geq$", but the signs of α and $\hat{e}(\bar{v})$ must be opposite to detect errors in "$>$" and "$\leq$" using points lying on the border.

The relational operators "$=$" and "$\neq$" can be treated by a simple extension of these arguments. It should be clear, for example, that test points exactly on an equality border give an α_c of 0 for both positive and negative α's, since any change, positive or negative, in the predicate value would be detectable as being not equal to zero. By contrast, points

chosen within the "≠" domain of a path yield almost no information at all about the "≠" predicate. All that can be said after such a test is that errors have been eliminated which have the form $\alpha\hat{e}$ and have α exactly $T'\circ C(\bar{v}_0) \;/\; \hat{e}\circ C(\bar{v}_0)$. In other words, choosing points slightly off an equality border can only prove that the border should not have passed through those particular points. (Such points may still be useful for other purposes, however. They may be required in order to exercise some other statements, especially statements lying along the "≠" branch of the predicate, or may be required to detect the substitution of one relational operator for another, as described later.)

Another special case of equation (3) which is worth examining is that of the blindness errors. For those errors satisfying equation (1), α_c approaches infinity, indicating that no bound is imposed on the size of those errors. This is consistent with our interpretation of a blindness error as being an error that is undetectable because of its direction $\hat{e}$, no matter what its size.

Figure 1 shows a program containing a predicate fault. The predicate in the WHILE statement should have been "B*(1+R/12.) > A" to include the final month's interest in the stopping criterion. The error term is therefore −B*R/12. Clearly this error will be revealed only if the value of −B*R/12 is large enough to necessitate another month's iteration. Taking B*R as the normalized error direction, we can ask whether various test data would reveal the error.

Consider first the input (P,R,A) = (1000., 0.14, 100.). At the first execution of the WHILE statement, α_c = (B−A) / (B*R) = 900./140. which has the wrong sign and so cannot detect the error. A negative α_c is obtained only on the final execution of the WHILE, at which time B=68.82 and α_c = (B−A) / (B*R) = −31.18/.3089 = −100.9. The actual error has α = −1/12 = −0.0833, so this test is not nearly sensitive enough.

```
(*  Loan History Program                                  *)
(*                                                        *)
(*  This program computes the monthly balances            *)
(*  for a loan issued on principal P at annual            *)
(*  interest R with monthly payments A.  Interest         *)
(*  is figured monthly, with the final payment            *)
(*  being no greater than A.                              *)
(*                                                        *)
READ  P, R, A
B := P
WHILE  B > A  DO
   B := B * (1. + R/12.)
   B := B − A
   WRITE "Balance: ", B
   END DO
WRITE "Final Payment is ", B * (1. + R/12.)
END
```

Figure 1: Loan History Program.

On the other hand, the input (P,R,A) = (385.90, 0.18, 100.) results in a final execution of the WHILE statement with B=99.00 so that α_c = (B−A) / (B*R) = −0.0561, which is sensitive enough to detect the error. For this test, the final balance of 99. is close enough to the monthly payment that failing to figure in the last month's interest causes an early exit from the loop.

The discussion of α_c has, so far, been concerned with a single, known $\hat{e}$. In fact, the test using $\bar{v}_0$ imposes constraints on many error directions. To interpret the effect of constraining α over a variety of directions, we need to understand the blindness errors more fully. Equation (1) defines a subspace of the space of all possible error terms, a subspace containing only errors which cannot be detected using $\bar{v}_0$, no matter how large those errors might be. If the space of potential error terms is of dimension N, then the blindness space defined by (1) will have dimension N−1. There exists, therefore, a unique critical direction $\hat{e}_c$ which is orthogonal to the entire blindness space. A necessary condition for detecting $\alpha\hat{e}$ is that it have a non-zero component in the $\hat{e}_c$ direction. If not, then $\alpha\hat{e}$ lies entirely within the blindness space and so satisfies equation (1).

In general, therefore, an error $\alpha\hat{e}$ can be written as

$$\alpha\hat{e} = \alpha_1\hat{e}_b + \alpha_2\hat{e}_c$$

where $\hat{e}_b$ is some direction entirely within the blindness space. The critical value of α for this error is therefore

$$\alpha_c(\alpha\hat{e},\bar{v}_0) = T \circ C(\bar{v}_0) \,/\, (\alpha_1\hat{e}_b + \alpha_2\hat{e}_c) \circ C(\bar{v}_0).$$

By equation (1), the $\hat{e}_b$ term goes to zero,

$$\alpha_c(\alpha\hat{e},\bar{v}_0) = T \circ C(\bar{v}_0) \,/\, \alpha_2\hat{e}_c \circ C(\bar{v}_0),$$

and finally by equation (3),

$$\alpha_c(\alpha\hat{e},\bar{v}_0) = \alpha_c(\hat{e}_c,\bar{v}_0) \,/\, \alpha_2. \tag{4}$$

The sensitivity of a test point $\bar{v}_0$ to an error depends therefore on its sensitivity to $\hat{e}_c$ and to the size of that error's component along the direction $\hat{e}_c$. An interesting implication of this is that $|\alpha_c(\hat{e},\bar{v}_0)|$ is minimized (over unit length errors) by $\hat{e}_c$. This observation in turn indicates a simple method of computing $\hat{e}_c$. If the error space E is spanned by a linearly independent set of functions $\{\hat{e}_i\}$, then

$$\bar{e}_c = \sum_i [\hat{e}_i \circ C(\bar{v}_0)]\ \hat{e}_i \tag{5}$$

is the (unnormalized) critical error direction for $\bar{v}_0$.

The restriction imposed on the error space by a given test point $\bar{v}_0$ can now be described geometrically. For any $\bar{v}_0$ there exists some direction $\hat{e}_c$ which is orthogonal to

the solutions of equation (1). In an N dimensional error space, there is an N−1 dimension (hyper)plane orthogonal to $\hat{e}_c$ and at a distance $\alpha_c(\hat{e}_c)$ from the origin. Any error term on the opposite side of the plane from the origin will be detected, since all points on that side of the plane have components in the $\hat{e}_c$ direction which are larger than α_c.

As additional tests are performed, an error is undetected only if it is missed on all tests. Each test contributes another plane in the error space, with the total set of planes describing a system of linear inequalities restricting the set of untested potential errors. If the straight line from an error to the origin crosses any of those planes, the error is detected by the corresponding test point. If all directions were bounded, these planes would define a convex polyhedron containing the origin and all error terms to which the tests were not sufficiently sensitive. The greater the volume of this polyhedron, the more potential error terms remain unchecked.

Figure 2 shows a program which computes the greatest common factor of two positive numbers by Euclid's algorithm. Consider the testing of the first "IF" statement. We will postulate an error space of two dimensions (in order to permit the graphing of the untested error space). This potential error space will consist of all linear combinations of P and Q.

If this program is tested with inputs "4" and "6", then when the "IF" statement is first encountered, (P,Q) = (4,6). By equation (5), we have $\bar{e}_c = 4P + 6Q$. Normalizing gives $\hat{e}_c = 0.55P + 0.83Q$, and by equation (3),

---

```
(*  This program computes the greatest common factor  *)
(*  of two positive integers by Euclid's algorithm.   *)
INPUT A, B
S := A
T := B
WHILE  S ≠ T DO
   IF S > T THEN
      S := S − T
   ELSE
      U := S
      S := T
      T := U
   END IF
END DO
IF S = 1 THEN
   PRINT A, " AND ", B, " ARE RELATIVELY PRIME."
ELSE
   PRINT "THE GCF OF ", A, " AND ", B, " IS ", S
END IF
END
```

Figure 2: Euclid Algorithm.

$$\begin{aligned}\alpha_c(\hat{e}_c,\bar{v}_0) &= (S - T) / \hat{e}_c(S,T) \\ &= (4 - 6) / (0.55 * 4 + 0.83 * 6) \\ &= -0.277\end{aligned}$$

Figure 3 shows the restriction imposed on the error space by this test as the inequality whose edge is labeled "1". As always, the untested errors are those lying on the same side of the line as the origin. Because this "IF" statement is inside a loop, it is executed several more times before the program halts. On the second execution of that statement, the values of P and Q have been exchanged. As a result, $\hat{e}_c = 0.83P + 0.55Q$ and $\alpha_c = 0.277$, resulting in inequality "2" shown in figure 3. On the third iteration of the loop, (P,Q) = (2,4), resulting in $\hat{e}_c = 0.45P + 0.89Q$ and $\alpha_c = -0.45$, while the fourth and final iteration again exchanges the components of $\hat{e}_c$ and changes the sign of α_c, adding inequalities "3" and "4" to the figure. As it turns out, both of these inequalities are implied by the first two, and so add no new information to the test.

In general, a given direction in the error space might remain unbounded after a number of tests. If a given error has satisfied (1) for all test points (or, equivalently, if it lies within the intersection of the blindness spaces for all test points), then none of the planes will intersect that direction. Expressions which are invariantly equal to zero will, if in the original error space, always be solutions to (1) and hence those directions will never

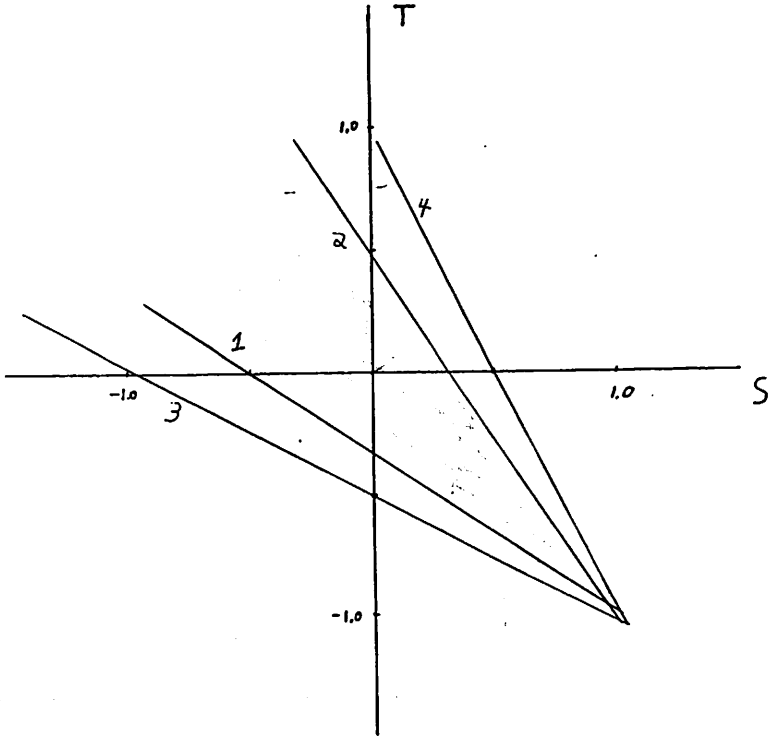

Figure 3: Constraints In the Error Space.

be bounded. Such invariant expressions will occur quite frequently but should usually be easily identified. If, for example, the predicate being tested immediately follows an assignment statement "X := f(Y)", then the expression "X − f(Y)" is invariantly zero at that statement. Note, however, that substitution of f(Y) for X in the predicate could not possibly cause an error (assuming f does not involve hidden side effects).

It is also possible that a direction has so far been bounded only on one side, and that test points have not yet been found which will yield an α_c of the opposite sign. A special case of this situation occurs for errors of the form $\hat{e} = kT$ for any real number k (assuming T is in the original error space). It is impossible to obtain a negative α_c for this error. This is because T + kT for positive k and T − kT for negative k must always have the same sense as T. An example of this is seen in figure 3 where the direction P–Q is bounded, but Q–P is not.

IV. Assignment Faults

Turning next to domain errors caused by faults in assignment statements, a similar but more restricted sensitivity measure can be derived. Assume that a block of assignment statements C has been perturbed by the addition of the error function $\alpha\hat{e}$, affecting one or more variables and yielding an erroneous computation C′. Assume that the path taken by the test input $\bar{v}_0$ passes through computations C_A, then the incorrect statements C′, and finally through additional computations C_B until coming to a predicate T. Does $\alpha\hat{e}$ cause a domain error by its effects on T?

A domain error will occur only when $T \circ C_B \circ C' \circ C_A(\bar{v}_0)$ has a different sense than $T \circ C_B \circ C \circ C_A(\bar{v}_0)$. As before, we can argue that one case where this does not occur is when

$$T \circ C_B \circ C' \circ C_A(\bar{v}_0) = T \circ C_B \circ C \circ C_A(\bar{v}_0). \tag{6}$$

Unfortunately, this equation is not, in general, solvable. There are, however, two important situations in which it can be solved. First, if the error $\hat{e}$ involves the value assigned to a single variable, then $\hat{e}$ goes undetected when

$$\hat{e} \circ C_A(\bar{v}_0) = 0 \tag{7}$$

or when $T \circ C_B$ is not partially dependent on the variable being computed in C′. (A function is *partially dependent* on a variable if a change in the value of that variable results in a change in the function value, or, formally, if there exists some integer k such that the *k*th partial derivative of the function with respect to that variable is not zero [10].) Equation (7) is a linear equation, and hence solvable, whenever $\hat{e}$ lies within a vector space.

The second situation in which (6) is solvable is when $T \circ C_B$ is a linear function of the variables whose assignments were affected by the error. $\hat{e}$ may involve assignments to any number of variables. Then $\hat{e}$ goes undetected when

$$T \circ C_B \circ \hat{e} \circ C_A(\bar{v}_0) = 0. \quad (8)$$

This is a linear equation for $\hat{e}$, even though $\hat{e}$ and C_A might be non-linear, since these are replaced by their values at $\bar{v}_0$.

Again, as for predicate faults, an assignment fault may fail to satisfy (6) but still not cause a domain error if the change in the predicate value is not sufficiently large. The critical point occurs when

$$T \circ C_B \circ C \circ C_A(\bar{v}_0) = 0.$$

We cannot solve here for α without restricting T and C_B. In fact, even for relatively simple forms of $T \circ C_B$, α_c may not be unique. If, however, $T \circ C_B$ is linear, then

$$\alpha_c = [T \circ C_B \circ C' \circ C_A(\bar{v}_0)] \; / \; [T \circ C_B \circ \hat{e} \circ C_A(\bar{v}_0)].$$

This permits the description of the restrictions imposed upon the possible error terms in an assignment statement whenever that statement is linearly referenced by a later predicate. These restrictions appear as inequalities forming planar boundaries α_c from the origin and orthogonal to $\hat{e}_c$, the unique unit length vector which is orthogonal to all solutions of equation (8). The requirement that $T \circ C_B$ be linear means that we must search for linear references to the variables assigned in C′.

V. Implications for Testing

With the derivation of the sensitivity measure α_c for both predicate and assignment statement faults, we can now outline the two goals for a successful domain error testing procedure:

1. Impose bounds on as many error directions as possible.

2. Reduce the size of the bounded region as much as possible.

It is not at all clear what the proper measure of "size" should be for these untested regions within the potential error space. The most obvious measure would be the volume of the enclosed region. This is intuitively satisfying, since the volume is a direct measure of the number of points and hence the number of errors in the undetected region. In its strictest sense, however, the volume is not a very useful measure. Allowances can be made for the unbounded directions by initially imposing an implicit bound, ±M, on all directions, with M being a very large number (probably related to the largest representable number on a given processor). Even then, however, the volume measure has problems because it is too easily dominated by a few directions. Consider, for example, the volume of a rectangular area in which the two dimensions are several orders of magnitude apart. Undetected errors in that region are most likely associated with the longer axis, but we could reduce the volume to any target level by squeezing the narrow axis still further. The worst case occurs when the rectangle has degenerated to a line. The volume would then be zero, yet there

would still be errors lying on that line which could be tested. If testing were continued, and the endpoints of that line segment were moved closer to the origin, this increase in confidence would not be mirrored by a change in the volume, which would remain a constant (zero). This worst case can be handled by performing the volume computations in the largest dimensioned subspace in which the volume is non-zero. Then, for example, the volume measure of a degenerate rectangle would become the length of the remaining line segment. This solution, however, does not help in the case where dimensions are orders of magnitude apart, and it is also not clear what action should be taken when one dimension of the bounded region is only approximately zero.

For these reasons, a more plausible measure of the size of the untested region is probably the maximum distance from the origin to any point on the border of the untested region. A hypothetical testing method which attempted to minimize this measure with each new test point would always attack those error directions to which the previous tests had been least sensitive. This would be a very desirable approach. It is worth noting that any new test point which improves the maximum distance measure must also reduce the volume (in the largest subspace where the volume is non-zero). The major point of difference between the methods is the emphasis assigned to different directions which are all capable of reducing the volume.

For example, a plausible volume-reducing strategy would be to consider each coordinate axis of the error space in turn. For each axis direction, find a path which is not blind to that direction (setting aside for the moment the difficulties in actually finding this path). Then apply any standard numerical method for constrained optimization to find points within that path domain which reduce α_c to an acceptably small level. An alternative to the use of arbitrary axes would be to choose successive error directions to be orthogonal to the e_c's for all previous tests.

This strategy, which we shall term *coordinate reduction*, will indeed reduce the volume with each test point selected. It may be quite good at initially bounding the error space, but, depending on the angles of the bounding planes, there can still be untested directions at arbitrary distances from the origin. Hence coordinate reduction is not optimal with respect to that maximal distance criteria. In fact, there can and usually will be completely unbounded directions intersected by none of the bounding planes.

The maximal distance measure is not without its own problems. The chief difficulty lies in the unbounded directions, which will tend to dominate this measure. Some of these will correspond to testable directions, and it does indeed make sense that these should be a high priority for future testing. Many others, however, can correspond to untestable directions, as outlined above. To which of these classes a given unbounded direction corresponds is, in general, undecidable. Even when decidable, it is not clear what should be done with those directions. They could be assigned an arbitrary bound, $\alpha_c=0$, but this may unwisely turn attention away from those errors which have a component in an untestable direction, but are still testable themselves due to their other components. If some maximum acceptable distance can be postulated, then assigning that distance as the bound on untestable directions would probably be preferable. Once a reasonable way of dealing with untestable directions is formulated, we can define another testing strategy, *maximal distance reduction*, in which the error direction associated with the maximal distance from the origin is chosen and a test point selected to place a better bound on that direction in the same manner as was done for the axis directions in coordinate reduction. An interesting variation

on this strategy would be to begin by choosing the path to be tested, then choose the maximal distance direction testable for that path. This variation would be a potent means for improving a previously selected point, choosing a new point within the same path which is more sensitive to domain errors. Many proposed testing strategies in fact only specify paths to be tested without further requirements on the points chosen except that they force the execution of those desired paths [2,5]. This variation on maximal distance reduction would be of particular utility when combined with such path selection schemes.

The notion of a maximum acceptable distance is even more important as a stopping criterion for testing. One possibility is the arbitrary choice of some number close to zero, 10^{-k}, with k a reasonably small integer. It should be noted that very small values for α_c can be achieved. If the predicate being tested is inclusive ($\leq$, $\geq$, or $=$), then by choosing points exactly on the border we get $\alpha_c=0$. If the predicate is exclusive, then we can choose points within ϵ of the border. For reasonably well behaved computations and predicates we will then have $T' \circ C$ very small. If $\hat{e}$ is not too closely related to T', so that we can find a point within ϵ of the border at which $\hat{e}(\bar{v}_0)$ is reasonably large, then α_c will be very small. In practice, when attempting to minimize α_c for a given error and a given path, it will probably be best to confine the search to the points on or within ϵ of the border, and simply attempt to maximize $\hat{e} \circ C$ over that set of points. While this approach is not guaranteed to produce the minimum α_c, it should produce acceptably small values. Similarly, an iterative optimization of α_c should probably not be overly concerned with finding the actual minimum, but should halt as soon as reasonably small values are achieved.

In fact, the maximum acceptable distance may be fairly large. Since the error directions are normalized, the largest coefficient of any axis component will be ± 1.0. If we expect the coefficients in the predicates to always have magnitude k or greater, then we will be satisfied by a maximum distance from the origin of k. This is of particular importance if a predicate expression should always have integer coefficients (k=1.) or coefficients expressible as rational numbers with small integer denominators.

Some notes about the computational effort required for the strategies outlined above may be in order. The computation of the blindness directions involves a single solution to a system of linear equations [9,10]. Because the constraints in the error space form convex polyhedra, many of the remaining problems relating to the untested errors can be formulated as linear programming problems. For example, determining whether a given test point reduces the untested volume is probably best done by using $\pm\hat{e}_c$ for that point as the linear evaluation function in a linear program and attempting to show the existence of (previously) untested points on each side of the new border. Note that since the origin is a known feasible point, half of this existence proof can be accomplished immediately. Finding the direction of maximum distance from the origin is a linearly constrained optimization problem with a quadratic evaluation function. However, since the maximum distance from the origin must occur at a vertex of the convex polyhedron formed by the constraint planes, the author has been able to devise polynomial-time algorithms for this purpose. One such algorithm, bearing a strong resemblance to hidden-line algorithms for computer graphics, is currently being added to the author's system for computing blindness errors for FORTRAN programs.

VI. The Domain Testing Method

In this section we will examine the domain testing method proposed by White and Cohen [6] and modified by Clarke, Hassell, and Richardson [1]. This method is intended to detect any predicate or assignment domain errors which could have occurred on a previously selected test path. The method is limited for practical purposes to programs in which the borders of the path domains are linear functions of the program inputs.

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. They claim that this strategy, henceforth referred to as the Nx1 strategy, constrains the perpendicular distance through the off point by which the border might have been shifted without changing the path taken for any of the N+1 test points.

Clarke et al. point out that the perpendicular distance is not a very meaningful measure. They propose the volume of test points through which the border might move without crossing any of the chosen test points as a superior measure. Since this measure seems a reasonable approximation to the worst-case volume of the input space on which the border might yield incorrect results (and hence is proportional to the probability of an input for that path resulting in an error), it does appear to be more reasonable. They then show that the Nx1 strategy allows this volume measure to be unbounded. 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 strategies which perform better according to their volume measure, the NxN strategy in which N points each are chosen on and off the border, and the VxV strategy in which a test point is chosen exactly on and just off each of the vertices of the border segment.

All three of these strategies define their desired testing attributes in terms of the dynamic structure of the path domains. This makes it difficult to reason back from the test results to the actual program code, to say exactly what errors in which statements have been eliminated. To a certain degree, this is understandable since the domain testing method does not specify which paths are to be tested. It is, however, particularly annoying that no use or acknowledgment is made of the fact that a given predicate may be responsible for a number of border segments for a variety of paths, or even for a single path. Clearly any tests done on one such segment provide information regarding the other segments. Perturbation testing, on the other hand, ties its information directly to the statements being tested. This in turn makes it easy to evaluate the effect on one path of testing a statement along another.

The failure to consider other paths is especially important to the interpretation of Clarke's volume measure. This measure would be a true indicator of the probability of a fault causing an error only if integrated over all paths through the affected statements. In particular, it is entirely possible that the NxN or VxV strategies might bound the volume of error-producing inputs for one path, but that the volumes associated with other paths through one or more of the same statements might still be unbounded. It is difficult, therefore, to determine how serious a problem is really implied by the failure to restrict this measure. This difficulty is compounded by the fact that an unbounded volume of input space may still be a negligible fraction of the entire program input domain. Consequently, the analysis afforded by perturbation testing, measured in terms of actual faults in the

code, appears to be a more compelling basis for evaluating domain testing.

In order to examine domain testing using perturbational techniques, we note that domain testing takes a black box view of the functions representing the domain borders. In effect, a very simple program model is being considered:

input $\bar{v}_0$

{constrain $\bar{v}_0$}

if $T \circ C(\bar{v}_0) + e \circ C(\bar{v}_0)$ r.o. 0

then print $f_1(\bar{v}_0)$

else print $f_2(\bar{v}_0)$

The line "{constrain $\bar{v}_0$}" allows for the effects of predicates which are encountered prior to the predicate whose border is being tested. Predicates which are encountered after T can be ignored, as they cannot possibly affect the interpretation of T, a point which was missed in [1] and [6] but which can significantly simplify the practical implementation of the domain strategy. The function C represents the computations performed along the chosen path up to the predicate responsible for the border being tested. It is not written as an explicit transformation of the environment (i.e. $\bar{v} = C(\bar{v}_0)$) because domain testing is really dealing with the functions $T \circ C$ and $e \circ C$, both of which are required to be linear (which is actually less restrictive than requiring that T and C be linear).

This model does not, however, cover all possible predicate errors, since not all predicate faults are manifested as border shifts. The important exception occurs when equality operators are swapped with inequalities. If the observed predicate is an equality, then the Nx1 strategy (and by implication also the NxN and VxV strategies) will detect the error. If, on the other hand, the observed predicate is an inequality, the point selection rules given in both [1] and [6] cannot determine whether an equality operator should have been used. In order to detect this condition, at least one point should be chosen which is close to but not on the border, and lies on the accepted side of the inequality (the side opposite the normal *off* points). This point need not be an additional test. It can instead substitute for one of the *on* points.

Other operator substitutions are either trivially detected or are representable as border shifts in this model. Substitution of "$\neq$" for "=", for example, is trivially easy to detect. Substitution of $<$ for $\leq$ involves a border shift by some small ϵ. Substitution of $<$ for $\geq$, on the other hand, is equivalent to $\alpha\hat{e} = -2^{*}T$.

In the remainder of this section, we will attempt to evaluate the domain strategies in terms of their ability to restrict the size of the untested error region. We will be testing initially for predicate faults in the above program model, with $e = \alpha\hat{e}$ selected from the set of linear functions of the program inputs without constant terms, which will be added later. Since $\hat{e}$ is a function of the input values, $\hat{e} \circ C = \hat{e}$. Equation (2) simplifies, therefore, to

$$\alpha_c = T' \circ C(\bar{v}_0) \,/\, \hat{e}(\bar{v}_0). \tag{9}$$

For this special case, we can solve immediately for $\hat{e}_c$ by remembering that $\hat{e}_c$ minimizes α_c for a given $\bar{v}_0$. This occurs when $\hat{e}$ is simply the normalized form of $\bar{v}_0$,

$$\hat{e}_c = \bar{v}_0 / |\bar{v}_0|. \tag{10}$$

So, when the error space being investigated is expressible entirely in terms of the program inputs, there is a simple and direct relationship between $\hat{e}_c$ and the test data.

This relationship makes it possible to infer some relationships between the coordinate and maximal distance reduction strategies proposed in the previous sections and the domain testing strategies. Beginning with coordinate reduction, choose an arbitrary direction in the error space. Find the point in the "THEN" subpath domain which minimizes $|\alpha_c|$ for that direction, and for any possible sign of α_c (choosing arbitrarily if both signs can be constrained). Then find a second test point in the "ELSE" subdomain which minimizes $|\alpha_c|$ for the opposite sign (such a point is guaranteed to exist for this class of error terms). Repeat for any direction $\hat{e}$ which is orthogonal to the $\hat{e}_c$ of the first test point. Continue, choosing error directions which are orthogonal to the critical directions of all previous test data, until it is no longer possible to find new error directions orthogonal to those already used. This will occur when 2N points have been chosen, half on each side of the new border.

If the path domain has not been reduced in dimension by previous equality predicates, this procedure will constrain the entire error space, except possibly in the direction T∘C which, as explained earlier, cannot be constrained in the negative direction. It is possible, however, that T∘C may not be a linear function in the program inputs only, and will not lie within the space being tested.

If previous equalities have occurred, there will be one unconstrained direction corresponding to each such equality. These directions must be constrained using different paths, since there exist no points in this path domain which are not blind to these directions.

The above procedure seems reasonable from the perturbation standpoint, if our goal is to eliminate all linear errors involving the program inputs. Now let us consider exactly what points will be selected by this strategy. Clearly this strategy generates two linearly independent sets of $\hat{e}_c$'s (one set on each side of the new border. Because of the special relationship between $\hat{e}_c$ and $\bar{v}_0$, it follows that the input points selected in each subdomain will be in general position. (A group of n+1 points $x_0,\ldots,x_n$ is in general position if the set of vectors $x_i - x_0$, i=1,...,n, is linearly independent. In geometric terms, the n+1 points cannot be contained in a hyperplane of dimension less than n.) The border segment being tested is linear, and its edges are linear. Given a direction $\hat{e}$, the expression given above for α_c is minimized when $T \circ C(\bar{v}_0)$ is minimized and $\hat{e} \circ C(\bar{v}_0)$ is maximized. The first occurs when $\bar{v}_0$ is chosen exactly on the border, if the border itself is in the subdomain, or else within ϵ of the border if the border itself is not in the subdomain. We then must maximize $\hat{e} \circ C(\bar{v}_0)$ within a hyperplane lying on or very close to the border. Since this is a linear programming problem, the solution must occur at (or just ϵ from) a vertex of the border

segment.

To summarize, the proposed procedure selects a set of points in general position at or within ϵ of the vertices of the border segment. If the path domain is of dimension N, the procedure will choose N points on each side. This of course satisfies the NxN strategy of Clarke et al. It represents a special case of that strategy in which each *on* point has a nearby *off* point corresponding to the same vertex of the domain border. A similar procedure could be defined based on maximal distance reduction, and would yield a special case of the Nx1 strategy. Either of these procedures would successfully impose finite bounds on all possible linear error directions. It is more difficult to establish that either the Nx1 or the NxN strategy will, in general, be this powerful.

Before attempting to prove that these strategies do indeed constrain all linear error directions, it is interesting to consider the shapes of the resulting untested regions. The proposed procedures will yield a set of linearly independent $\hat{e}_c$'s for the various test points. While these critical error directions will be independent, they will not in general be orthogonal, unless the path domain features vertices lying at right angles to each other with respect to the origin. In fact, the polyhedra enclosing the untested errors will tend towards flattened, pancake-like figures (hyper-pancakes?) composed of two sets of nearly parallel planes. This arises from the fact that the constraint planes are orthogonal to the critical error directions, but these critical errors are directly based on the input points. If the path domain does not include the origin, the angles over which the vectors $\overline{v}_0$ may range will be limited. This limit (and hence the "flatness" of the untested space) grows more severe as the domains selected become smaller or more remote from the origin.

A successful test data selection strategy should constrain each of N linearly independent error directions on both signs. This does not mean, however, that 2N tests are required. In a two-dimensional input space (N=2), the simplest closed figure is a triangle. Hence all directions could be covered by a triangle, defined by only three test points. In three-space (N=3), the four sides of a tetrahedron will suffice, and so on. By implication, we could in fact be satisfied with N+1 test points. In fact, any N+1 test points forming such a closed figure in the error space would satisfy the maximal distance reduction scheme, since the maximum untested distance from the origin would occur in those directions which must still be unbounded until the first N+1 points have been chosen. It is tempting, therefore, to wonder whether the Nx1 strategy of Cohen and White might not suffice.

To see that, in fact, the Nx1 strategy does bound all error directions, we begin by noting that the N *on* points are in general position. Assume for the moment that the hyperplane containing the border being tested does not pass through or within ϵ of the origin. Then the *on* points and the origin are in general position, and we conclude from (10) that the N critical directions are linearly independent. The set of planes must therefore cover all directions for at least one sign, in the sense that for any direction $\hat{e}$, either $+\hat{e}$ or $-\hat{e}$ intersects one of the constraining planes. If, therefore, the constraint imposed by the *off* point intersects each of these N planes and lies on the "other side" of the origin, then it will result in a completely closed figure defining the untested errors.

Consider a hypothetical *on* point at the intersection of the border being tested with the vector from the origin to the *off* point. A unique such point must exist, since the border does not pass through the origin. Since $\hat{e}_c(\bar{v}_0) = \bar{v}_0/|\bar{v}_0|$, this hypothetical *on* point has the same $\hat{e}_c$ as does the *off* point. The *on* points are associated with the same sign of α_c as if they were slightly off the border on the closed side. The *off* point, however, lies on the open side of the border, giving it a different sign for $T' \circ C(\bar{v}_0)$. Consequently, the *off* point and the hypothetical *on* point have opposite signs for α_c. Since the *off* point is required to lie inside the projection of the polygon formed by the N *on* points, the $\hat{e}_c$'s for the *on* points has a positive component along the $\hat{e}_c$ for the *off* point. By equation (4) we conclude that the *off* point bounds the critical directions for each of the *on* points, but with the opposite sign. Hence all directions in the error space are constrained on both signs.

Next consider what happens when the border passes through the origin. The N *on* points are still in general position, but the *on* points and the origin are not. By implication, the $\hat{e}_c$ are no longer linearly independent. One of the *on* points is essentially useless. The remaining N−1 points and the *off* point are, however, in general position with the origin. By the arguments given in the previous paragraphs about test points in general position, we can claim that the untested region is bounded on all but one side. However, if the border T∘C passes through the origin, it must be a linear function of the program inputs only. Hence T∘C is in the error space. Since, as we have observed earlier, T∘C can be bounded on only one sign, there must always be one open side to the untested region. Hence we conclude that even in this case, the Nx1 strategy has performed properly, in the sense that it has bounded as many linearly independent directions as possible.

If the border being tested does not pass exactly through the origin, but does pass extremely close to it, it is theoretically possible to choose an *off* point close enough to the border that the analysis of the first case applies. In practice, the finite resolution of the space of input space may not permit this, but in such cases one of the bounds on the direction T∘C would be so large that for practical purposes the analysis for the border passing through the origin may be taken as correct.

Having shown that the Nx1 strategy constrains all possible linear errors, the same can be shown for NxN by noting that the N *off* points constrain all directions which are also constrained by a hypothetical *off* point at the centroid of those *off* points. Since the Nx1 system formed by the centroid and the N *on* points constrains all possible linear errors, so does the original NxN system.

The use of N-1 fewer points does not imply that the Nx1 strategy is the best possible strategy for detecting linear errors. The results of Nx1 will probably not be significantly improved upon when a completely closed untested region has been found, since in practice we expect to achieve very small α_c's. In fact, the NxN method might be worse than the Nx1 method when closed regions can be obtained. When a completely closed region cannot be attained, the angles at which the constraints intersect may be crucial. Consider a pair of angles in a two-dimensional error space as shown in figure 4a and 4b. The volume (area) of points enclosed as a function of the distance along the untestable direction depends greatly on the degree of the angle. The same can be said for the distance between the

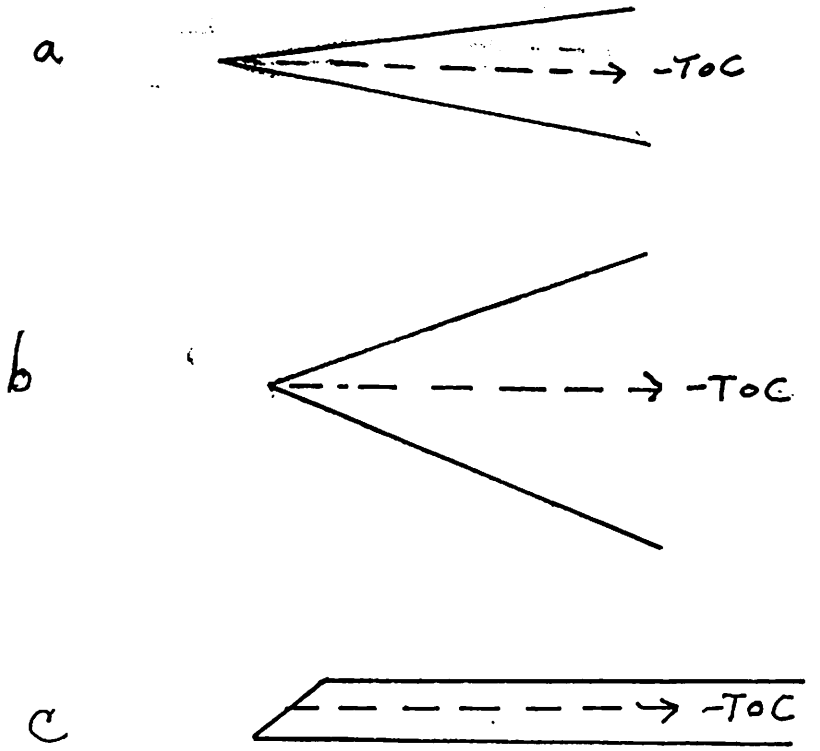

Figure 4: Variation in Volume With Angle.

two lines. Consequently both the volume and the maximal distance measures of an untested region depend strongly on the angles at which the constraints intersect. The ideal case is illustrated in figure 4c, in which the parallel constraints restrict the size of the component any error might have in the detectable direction without being detected. How can such parallelism be guaranteed? It cannot be obtained with only N constraints; in fact 2N−1 constraints would be required. Two constraining planes will be parallel only if they are generated by the same $\hat{e}_c$. This in turn requires that the inputs be chosen in pairs, $\bar{v}_0$ and $k\bar{v}_0$, such that $\bar{v}_0$ is *on* and $k\bar{v}_0$ is *off*, with k being very close to 1. This defines a simple refinement of the NxN strategy, and is equivalent to the special case of NxN in which the *off* points are chosen to correspond to the same vertices as do the *on* points.

Is the extra confidence imparted by NxN worth evaluating N−1 additional test points? Without a doubt, the gain is not on the same order of magnitude of importance as having imposed some finite bound on each detectable direction. Beyond that simple observation, only experimentation and experience can indicate whether the extra effort is worthwhile. Since, however, the NxN strategy so closely approaches the ideal of figure 4c, it does seem safe to dismiss the even more costly VxV strategy as unjustifiable.

The above discussion has concentrated on errors which were linear functions of the program inputs only. Most discussions of the domain testing strategies have in fact been couched in terms of linear functions of program inputs and constants [1,6], a more common interpretation of phrases like "domain errors in linear borders" [6]. Adding constant offsets

to the error space results in an error space of dimension N+1. A closed figure in this space would require at least N+2 constraints, so it might seem as if the Nx1 strategy could be immediately dismissed. However, with the addition of constants to the error space, T∘C is always in the error space and so only N+1 independent directions can possibly be constrained. The arguments used above when the border passed through the origin are applicable here to show that Nx1 (and by implication, NxN) suffice to bound regions in this error space also. In this space, none of the *on* points are redundant.

To summarize, both White and Cohen's Nx1 scheme and the NxN scheme of Clarke et al for domain testing achieve the principle goal of imposing a finite limit on the size of all possible linear (in program inputs and constants) errors in linear predicates. The NxN scheme will usually give smaller bounds, but it is not clear whether the difference is significant. These schemes will also detect all operator substitutions if one of the *on* points is moved slightly off of and to the accepted side of the border.

VII. Extensions of Domain Testing

An important characteristic of perturbation testing is that the power of a set of test data to detect perturbing functions (errors) is largely independent of the complexity of the statement being tested, depending instead primarily on the properties of the chosen space of potential errors. In [10], for example, this characteristic was employed to demonstrate the usefulness of Howden's algebraic testing to a much wider variety of programs than originally believed [4]. It is natural to ask here whether similar extensions can be justified for domain testing. In particular, do the Nx1 and NxN strategies suffice for finding linear errors in non-linear borders, and can they be extended to cover non-linear errors?

When the predicates being tested can yield nonlinear borders, we may still be interested in detecting any linear errors, if only as a prelude to testing for higher-order perturbations. Both the Nx1 and NxN schemes can be extended to non-linear borders by emphasizing the need for the *on* points to be in general position and to be spread widely apart, and recognizing that the vertices where borders intersect may not be the best locations. For example, the predicate "IF X**2 + Y**2 < 1." could result in a path domain with no vertices at all! By way of contrast, the procedure for coordinate reduction given near the start of the previous section translates into non-linear domains without change, providing an example of how the search for the *on* points can be formulated.

If the border is non-linear but the space of potential error functions is linear (with constants), the function T∘C cannot lie in the error space. Consequently, N+2 points will be required to finitely bound all errors. Obviously the Nx1 strategy is not sufficient for linear errors in non-linear predicates. Can it be "patched up" by the addition of one more point?

Figure 5 shows a variety of test points chosen near a non-linear border. The points ON_1, ON_2, and OFF represent "normal" choices of the Nx1 strategy. In an N dimensional input space, the N points ON_i will be in general position and hence will impose finite bounds on N linearly independent error directions. This leaves two more bounds to be imposed.

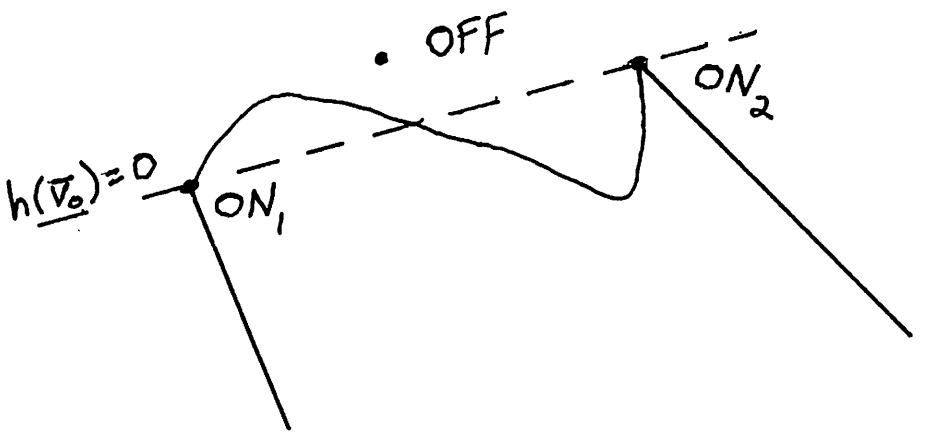

Figure 5: Testing for Linear Errors in Non-linear Borders.

Any N points can be contained in an N−1 dimension hyperplane. Consequently there must exist some linear function h such that $h(ON_i) = 0$ for all i from 1 to N (figure 5). Since h is linear, it is a potential error term, and by definition all the *on* points are blind to h. Hence h is unbounded on both signs, accounting for both of the error directions still to be covered.

If the *off* point is chosen within the same hyperplane defined by the *on* points (possible because the border is non-linear), it is useless for our purposes. It is therefore desirable to expand the requirement that "all *on* points be in general position" to "the *on* and *off* points must be in general position." Then h(OFF) will be non-zero, and the point OFF will bound one sign of h.

Recall that

$$\alpha_c(h,OFF) = T' \circ C(OFF) \,/\, h(OFF).$$

The desired final test point, then, should have a different sign than OFF for $T' \circ C$ or for h, but not both. In terms of the figure, this means that desired point must lie on the other side of the border, or on the other side of the plane $h(\bar{v}_0) = 0$, but not both. The two *off* points required to detect linear errors, therefore, must be chosen so that one point is between the border and the plane of *on* points, and the other point lies outside of both the border and the *on* point plane.

The NxN strategy fares no better in this domain. If, as recommended in the previous sections, the *off* points are very close to the *on* points but on the other side of the border, all the *off* points will lie on the same side of both the border and the *on* point plane. An additional point lying between the two would still be required. Once again the only advantage offered by the NxN strategy is the near parallelism of opposing constraints. If the *off* points are not chosen in the recommended manner, that advantage is lost, and there is still no guarantee that any of the *off* points will fall between the border and the *on* point plane.

If we wish to test for non-linear errors in non-linear borders, it is unlikely that the distribution of required points in the input space will appear geometrically significant. Equation (9) is still valid for computing α_c, but $\hat{e}$ is no longer a linear function. The goal for choosing the first n points (where n is the dimension of the error space) must still be the generation of a linearly independent set of $\hat{e}_c$'s, so that all error directions will be bounded on at least one sign. Choosing these points on or near the border and choosing additional points just across the border should still tend to give nearly parallel constraints, but it is possible that these pairs of constraints will be on the same sign. This occurs whenever $\hat{e}_c$ locally approximates $T' \circ C$. This unfortunate occurrence can easily be detected by simply checking the signs of the α_c's, and a new point selected to bound that direction, but the new constraint need not be even approximately parallel to the constraint bounding the other sign.

An analogue of the Nx1 strategy is not easily obtained for non-linear errors. We could start by choosing the first n points in the same manner as in the above paragraph, then solving for the maximal distance direction and choosing a final point to restrict that error direction. The principal difficulty appears to be that the maximal distance direction may not be unique due to the existence of local invariants, necessitating a trial-and-error search for a direction which is constrainable.

VIII. The Static View of Domain Testing

The discussion of domain testing in the previous section concentrated on possible errors in the function computed by the program without discussing the faults in the source code which could be responsible for those errors. Such an approach seems logical if the purpose of testing is to demonstrate the correctness of the program function, but such demonstrations are not, in general, possible. Concentration on the program function seems much less appropriate when the goal of testing is the increase of confidence in the program by the elimination of certain classes of errors. Simple, plausible faults in the source code can result in errors in the program function that run the gamut from subtle alterations on a few inputs to massive changes over the whole input domain. Confidence seems best associated with the knowledge that the actual code is either correct or that the correct code would be "implausibly" different from the programmer's original creation. Therefore we should like to know whether domain testing can be counted on to eliminate some large class of "plausible" faults in the source code.

Another reason for shifting the focus of this discussion from the dynamic program function to the static code is that a given fault may be encountered along a number of different paths. Faults to which one path is blind may be detectable along a different path. In addition, test data intended to constrain the set of untested errors for one statement will usually provide constraints for other statements encountered in the course of executing that data.

With regard to domain testing then, the following questions can be posed: After domain testing along one path, what possible faults remain untested? Is the full domain testing procedure required for every test path through the statements being examined? How should the additional test paths be chosen?

To simplify the following discussion, we will postulate the existence of a program P to be tested. P will have variables X and Y whose values are computed and assigned by P. In addition, P may have "input variables" A and B which are used to accept values from the input stream but which are not afterwards altered by P. Such input variables can also arise as parameters passed to P and never reassigned new values within P. The significance of input variables is that they provide a link to the earlier discussion of domain testing, which is defined in terms of functions of the program inputs. Faults written with X and Y, on the other hand, are only indirectly related to the program inputs through the partial function computed along the chosen test path, a relationship which can change with each new path. This distinction is drawn here for the sake of discussion, though it need not exist in real programs for the conclusions to be valid.

Consider first testing a predicate in P for faults taken from a postulated error space E. Assume that a path has already been subjected to domain testing designed to detect all predicate errors from a class E_{inp}. For the sake of simplicity, we will assume that either no equality restrictions occurred on that path, or else that just enough points were chosen along some other path to constrain any error directions left unbounded due to equalities. Suppose that some fault $\bar{e}$ from E remains unbounded. There are a number of possibilities:

1. $\bar{e}$ might be a function of A and B only. If, however, $\bar{e} \in E_{inp}$, then it would have been bounded by the previous domain test. Hence $\bar{e}$ must lie in E but not E_{inp}. One solution might be to domain test for a more general class of errors. Alternatively, coordinate reduction of $\bar{e}$ on the same path (or on a different one) could be applied to select a pair of points to bound $\bar{e}$. Simply conducting the same level of domain testing on a different path is not a reasonable solution, since if detection or constraint of $\bar{e}$ was not guaranteed on the first path, it would not be guaranteed on another path.

2. $\bar{e}$ might be a function of X and Y only. Let C_1 be the computation for the path tested up to the point where the predicate was encountered. $\bar{e} \circ C_1$ is a function entirely of the inputs A and B. If $\bar{e} \circ C_1$ is not zero for every point in the path domain, then we are back to the previous case, this time with the error term $\bar{e} \circ C_1$. If the path is blind to $\bar{e} \circ C_1$ (i.e., $\bar{e} \circ C_1$ is zero over the entire path domain), then clearly another test path must be chosen. Applying domain testing to this second path would be overkill, since only two new points are needed to bound $\bar{e}$ on both sides. Furthermore, the second path may fail to bound $\bar{e}$ for the same reasons as the first. It appears more economical to apply coordinate reduction in order to bound $\bar{e}$.

3. Finally, $\bar{e}$ may be a function of both regular variables and input variables. Such faults will be missed quite frequently because for each variable there exists a function describing the computation performed to compute that variable's value along the path being tested. Thus for the variable X there is some function $f_X(A,B)$ such that $X = f_X(A,B)$ for all A and B in the path domain. The expression "$X - f_X(A,B)$" is therefore untestable for all points in this path

domain. The derivation of all such expressions is described in [8,9]. If another path is selected to be tested for which $\bar{e}$ is detectable for some points, there will be a new function, $g_X(A,B)$ describing the function assigned to X on this new path. Thus the fault "$X - f_X(A,B)$" will be constrained only if the error "$(g_X - f_X)(A,B)$" is testable. If this error is in E_{inp}, domain testing along the second path suffices, but if not then we are back to the first case. Even when domain testing suffices, it is likely to be more effort than using coordinate reduction to find the two points required to bound $\bar{e}$.

Should domain testing ever be done on more than one path through a particular predicate? The above discussion stacked the deck somewhat against domain testing by postulating only a single untested error direction. When a number of directions remain untested, additional domain tests seem more plausible. From the above arguments however, it seems safe to say that domain testing should be abandoned in favor of concentrating on particular error expressions whenever the number of directions remaining to be tested is significantly less than the dimension of E_{inp}, which in turn is less than the number of points chosen by domain testing for each path. In addition, it seems safe to say that domain testing should be abandoned whenever the domain test set for the previous path fails to significantly reduce the number of remaining untested directions, since that failure implies that the interpretations of the remaining directions are tending to fall outside of E_{inp}.

The arguments presented in this section for predicate faults can be extended to assignment faults as well. Consider a path passing first through computations C_A, then through an erroneous block of computations C′, and finally through additional computations C_B and ending at a predicate T. As before, we will be interested only in those cases where $T \circ C_B$ is a linear function of the variables affected by C′. If $T \circ C_B$ is not linear, there may exist other predicates along the path in question or to which that path can be extended which do yield linearity. If no such linear uses of C′ can be found, then α_c cannot be computed, although the set of blindness directions can still be found, in order to show which directions are completely untested. The tester should also be alert to the possibility that paths can be found through output statements which depend upon the variables assigned in C′, so that C′ can also be examined for computation errors as described in [10]. Where both domain and computation errors are being assessed for the same space of possible assignment faults, any error direction considered detectable as a computation error may be considered to be constrained with $\alpha_c = 0$ for both signs.

The simplest case of analyzing domain errors due to assignment faults occurs when C′ (and C) is a single assignment statement. Then, replacing C′ with $C - \alpha\hat{e}$ gives

$$T \circ C_B \circ C' \circ C_A(\bar{v}_0) = (T \circ C_B \circ C - \alpha T \circ C_B \circ \hat{e}) \circ C_A(\bar{v}_0).$$

This may be viewed as a transformed program in which we are now concerned with detecting a predicate fault $\alpha T \circ C_B \circ \hat{e}$ which has been added to a correct predicate with function $T \circ C_B \circ C$ to yield an incorrect predicate $T \circ C_B \circ C'$. Then all of the previous arguments for predicate faults hold here as well, provided that $\alpha T \circ C_B \circ \hat{e}$ is not identically zero. Since $\hat{e}$ affects only a single variable, this in turn means that $T \circ C_B$ must make a non-trivial reference to the affected variable. Given that $T \circ C_B$ is linear, non-trivial means

simply that the coefficient of the affected variable in the linear function $T \circ C_B$ must be non-zero.

When C and C′ involve assignments to more than one variable, then the major change in the above is that the requirement that $\alpha T \circ C_B \circ \hat{e}$ be non-zero is no longer trivial. For any given $\hat{e}$, this is easily verified, but in order to assess the impact of a test point on the entire space of errors it is necessary to solve the linear system $T \circ C_B \circ \hat{e} = \bar{0}$ [8]. This system governs the combinations of simultaneous perturbations to different variables which can be detected. For example, if $T \circ C_B = X - Y$, then any fault which adds the same error term to both X and Y goes untested. This is entirely a function of the path being tested rather than the points chosen for that path, so the domain testing rules for choosing points contribute nothing to the detection of these faults. In many cases, however, a given test point may pass through so many predicates (and output statements) that all these faults will be detectable for at least one predicate, making the behavior of domain testing equivalent to the case of single assignment statements. Failing that, additional paths should be chosen which, if possible, differ only after the assignment statements being tested. These paths should be tested at the same level as the original one (i.e., domain testing if the path they were based on was the first one tested, or coordinate reduction if it was a later path), until all the solutions to $T \circ C_B \circ \hat{e} = \bar{0}$ have been eliminated as in [8,9].

IX. Conclusions

The theories underlying perturbation testing can be expanded to evaluation of individual data points as well as entire paths. This expansion has been done in part in [10], to discuss computation errors. In this paper the theory is extended to domain errors.

For both predicate and assignment faults, it is possible to derive the set of error directions to which a given input is blind. The computational effort involved is simply the solution of a single linear equation, even though the errors and program constructs in question may be nonlinear. For those errors to which the input data was not blind, sensitivity measures can be derived which limit the possible size of those errors. These sensitivity measures divide the space of possible errors into tested and untested regions, with the untested region described as a system of linear inequalities.

The combination of these new measures with standard optimization techniques opens up new possibilities for testing strategies, either alone or in concert with established path selection techniques. Two new strategies have been described, coordinate reduction and maximal distance reduction, which differ in the choice of new error directions for which improved sensitivity will be sought.

The notion of constraining the untested error space can be used as a tool for analyzing other testing strategies. The different approaches to the domain testing strategy as introduced by White and Cohen [6] and as later modified by Clarke et al. [1] can be shown to be closer in error detection capability than expected. Both the Nx1 scheme [6] and the NxN scheme [1] place finite bounds on the size of all possible linear errors in linear domain boundaries. though the NxN scheme will usually result in somewhat smaller bounds. Both schemes require the addition of one more test point to place finite bounds on

all possible linear errors in non-linear domain boundaries.

Domain testing on a single path cannot guarantee the detection of all faults in program predicates and assignment statements, even for linear classes of possible faults. As additional paths are chosen, the effectiveness of domain testing decreases, suggesting that domain testing should be restricted to early choices of test paths.

X. Bibliography

1. 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

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

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

4. W. E. Howden, "Algebraic Program Testing", *Acta Informatica*, vol. 10, 53–66, 1978

5. 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

6. 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

7. S. J. Zeil and L. J. White, "Sufficient Test Sets for Path Analysis Testing Strategies", *Proceedings of the 5th International Conference on Software Engineering*, IEEE Computer Society, 184–191, 1981

8. 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

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

10. 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