81-61

A Close Look at Domain Testing+

Lori A. Clarke*
Johnette Hassell**
Debra J.Richardson*

January 1981

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

**Computer Science Department
Tulane University
New Orleans, Louisiana 70118

This work was supported in part by the National Science Foundation under grants NSFMCS-77-02101 and SPI-79-16639, the Air Force Office of Scientific Research under grant AFOSR 77-3287, and the IBM Computer Science Graduate Fellowship Program.

+This paper will appear in IEEE Transactions on Software Engineering, July 1982.

Abstract

White and Cohen have proposed the domain testing method, which attempts to uncover errors in a path domain by selecting test data on and near the boundary of the path domain. The goal of domain testing is to demonstrate that the boundary is correct within an acceptable error bound. Domain testing is intuitively appealing in that it provides a method for satisfying the often suggested guideline that boundary conditions should be tested.

In addition to proposing the domain testing method, White and Cohen have developed a test data selection strategy, which attempts to satisfy this method. Further, they have described two error measures for evaluating domain testing strategies. This paper takes a close look at their strategy and their proposed error measures. It is shown that inordinately large domain errors may remain undetected by the White and Cohen strategy. Two alternative domain testing strategies, which improve on the error bound, are then proposed and the complexity of each of the three strategies is analyzed. Finally, several other issues that must be addressed by domain testing are presented and the general applicability of this method is discussed.

1. Introduction

A testing method should provide guidance in the selection of test data for a program. Ideally, executing the program on this data reveals errors in the program or provides confidence in its correctness. Several testing methods have been developed that assist in the detection of either data flow or control flow errors. White and Cohen have developed a method called domain testing [WHI80], which focuses on the detection of control flow errors. Domain testing attempts to uncover errors in a path domain by selecting test data on and near the boundaries of the path domain. This method appeals to our intuition in that it provides a formal approach for satisfying the often suggested guideline that boundary conditions be tested. In addition to the general method, White and Cohen have proposed a specific domain testing strategy that selects points to test the boundary of a path domain. They have also defined two error measures for evaluating domain testing strategies [WHI78,WHI80].

In attempting to apply the White and Cohen domain testing strategy to some programs, we encountered several problems. This led us to examine this strategy and the associated error measures more closely. This paper describes some of the problems we encountered, proposes two alternative domain testing strategies that improve on the error bound, and discusses the general applicability of the domain testing method. Section two presents some general testing terminology and an overview of some of the more formal approaches to program testing. The third section describes the general domain testing method, some related terminology, and the White and Cohen domain testing strategy. The restrictions on domain testing that are assumed and the notation that is used throughout this paper are presented. The fourth section introduces the two error measures proposed for domain testing and analyzes the White and Cohen strategy. In section five, we propose two alternative domain testing strategies and show that these strategies improve on the error bound. In comparing all three strategies, trade-offs between the size of the error bound, the necessary number of test points, and the amount of effort that must be expended to find the test points are all considered. The sixth section describes several additional issues that must be addressed by domain testing and discusses the general applicability of this method.

2. Testing Terminology and Related Work

The domain testing method is a modification of a more general testing method called path analysis testing [HOW76a], which constructs test data sets for selected paths in a program. A path through a program corresponds to some possible flow of control. Associated with each path is the path domain, the subset of the program's domain that causes execution of the path, and the path computation, the function that is computed by the execution of the statements along the path. Path analysis testing strategies typically use symbolic execution [CLA76] to provide a symbolic representation of a path. Symbolic execution assigns symbolic names to a program's input values and "executes" a path through the program. Throughout this execution, variables are maintained as algebraic expressions in terms of these symbolic names. The algebraic expressions for the output values provide a symbolic representation of the path computation. The conditional branches encountered along a path are represented as constraints in terms of the algebraic expressions of the variables referenced within the condition. The conjunction of these constraints provides a symbolic representation of the path domain. The symbolic representations of the path computation and path domain are used by path analysis testing strategies to direct the selection of data to test a particular path. The appropriate selection of test data for a path can increase the probability of detecting errors in that path.

Program errors can be considered from two perspectives - cause and effect. Program testing detects errors by discovering the effects, while debugging searches for the associated cause. It is possible, however, that an error on an executed path may not produce erroneous results, thus complicating the testing process. When an error exists on a path, but execution of the path produces correct output for some selected test data, coincidental correctness is said to occur. The effects, if any, of an error on a path can be related to its effects on the path domain and path computation [HOW76a,GOO76]. A computation error is reflected by an incorrect path computation. Such an error may be caused, for example, by the execution of an inappropriate assignment statement that affects the function computed by the path. A domain error is reflected by an incorrect path domain. This type of error may occur, for instance, when a branch predicate is expressed incorrectly or an assignment statement that affects a branch predicate is wrong, thus affecting the conditions under which the path is executed. Domain errors can be further divided into path selection errors and missing path errors. A path selection error occurs when a program recognizes the need for a path but incorrectly determines the condition under which the path is selected. A missing path error occurs when a special case requires a unique sequence of actions, but the program does not contain a corresponding path. Missing path errors are particularly insidious. If only one point in a patn domain should be in the missing domain, the missing path error will not be detected unless that point happens to be selected for testing.

Path analysis testing does not assure the detection of any of these types of errors, although certain techniques can be applied to reveal some types of domain and computation errors. Symbolic testing [CLA76,HOW77] attempts to detect errors by the examination of the symbolic representations of the path domains and computations. These representations can also be used to guide in the selection of test data that are sensitive to the path and likely to expose errors. More rigorous techniques, which detect or verify the absence of computation or domain errors under rigidly defined conditions, have been proposed. When the path computation can be correctly specified by multinomial functions, the correct execution of a path on an appropriate number of test points demonstrates that no computation errors exist on that path [HOW76b]. The number of such test points depends on the degree of the multinomial functions and may be quite large. Probabilistic arguments have also been made for detecting computation errors for such functions using fewer test points [DEM77]. The domain testing method [WHI80] attempts to uncover errors in a path domain by selecting test data on and near the boundary of the path domain. This method attempts to demonstrate that the boundary either is in error or is correct within an acceptable error bound. Domain testing does not address missing path errors but concentrates solely on detecting path selection errors.

3. The Domain Testing Method and the Cohen and White Strategy

The domain testing method guides in the selection of test data by analyzing the boundary of a path domain. The program is tested with this data in order to reveal domain errors or provide confidence in the correctness of the path domain. A domain error is manifested by a shift in some section of the path domain boundary. Domain testing exploits the often observed fact that points near the boundary of a domain are most sensitive to domain errors. The method proposes the selection of test data on and slightly off the domain boundary of each path to be tested. If the program yields correct results for the chosen test data, domain testing concludes that the path domain boundary is correct within a quantifiable error bound.

The boundary of a path domain is determined by the conditional branches that are taken along the path. Associated with each conditional branch is a <u>predicate</u>, a logical combination of relational expressions. When a conditional branch is taken during symbolic execution, the corresponding predicate is evaluated and simplified providing a <u>predicate interpretation</u>, a logical expression in terms of the symbolic names for the input values.

The section of the path domain boundary determined by a single predicate interpretation is a <u>border</u>. Each border of a path domain is either closed or open, with respect to that domain. A <u>closed border</u> is in the path domain and is formed by a relational expression with a <=, >=, or = operator. An open border is not in the path domain and results from a relational expression with a <, >, or /= operator. Since an open border of a path domain is a closed border of an adjacent path domain, only the closed borders of a path domain need be tested by the domain testing method.

The border being tested is called the given border. Although the domain testing method does not require knowledge about the correct program, it is convenient to refer to the border that results from the associated predicate interpretation in a correct program as the correct border. When the given border differs from the correct border, a border shift is said to occur, causing a domain error. The displaced domain is the set of elements that are placed in the wrong path domain by a border shift.

The goal of the domain testing method is to select test points that either lead to the detection of a border shift or provide a limit on the maximum displaced domain. The strategies discussed in this paper direct the selection of test data on and slightly off each of the closed borders of the path domain. An ON test point lies on the given border and thus in the path domain being tested. An OFF test point lies on the open side of the given border and thus in some other path domain. The analysis presented in this paper assumes that coincidental correctness does not occur. Under this assumption, if the program produces correct results for all of the chosen ON and OFF test points, then the given border is considered "close" to the correct border. An undetected border shift can only occur when the ON test points and the OFF test points lie on opposite sides of the correct border. The undetectable border shifts are kept "small" by choosing the OFF test points as close to the given border as possible. The strategies discussed here select the OFF test points within some distance d of the given border. For continuous input space, the ideal choice for d is the machine tolerance. This, however, may cause underflow problems when computing the test points. Further problems can occur when the size of the given border is comparable to d. These are complex problems, requiring additional investigation. For now, the assumption is made that d is not so small as to cause underflow, but small compared to the size of the borders. The discussion of domain testing that follows also assumes that the input space is continuous. This restriction is not a limitation inherent to the domain testing method, but allows OFF test points to be selected arbitrarily close to the given border and thus simplifies the ensuing discussion.

If the predicate interpretations are linear in the symbolic names, the path domain boundary consists of linear borders. In addition, if each of the predicates is a single inequality, the path domain is a linear convex polyhedron. In two dimensions, such a path domain is a convex plane segment, whose borders are straight line segments. In N dimensions, such a path domain is a convex N-dimensional hyperplane segment - the generalization of a plane - with (N-1)-dimensional hyperplane segments as borders. The domain testing method is simplified considerably when the path domains are linear convex polyhedra; this occurs when the evaluation of any predicate results in a single linear inequality. In practice many programs satisfy this restriction, and the domain testing strategies discussed in this paper first address those programs. Section 6 considers the applicabilty of the domain testing method to programs with unrestricted predicate interpretations (including equality, nonequality, nonlinear, and complex predicate interpretations) as well as to programs with discrete input space and in which coincidental correctness might occur.

For convenience, we adopt a standard notation for describing the domain testing strategies. The standard labels of the relevant items are:

G	given border
C	correct border
D_T	path domain being tested
D_A	path domain adjacent to D_T
P,Q,R,S,T	ON test points
U,V,W,X,Y	OFF test points
d	distance from G to an OFF point
L	hyperplane segment parallel to G and a distance d from G on the open side.

For two dimensions, some additional standard labels are:

J	extension of border adjacent to G on left
K	extension of border adjacent to G on right

When of interest, the displaced domain in two dimensions is shown as a shaded area. An arrow on an adjacent border indicates whether the border is closed (pointing into D_T) or open (pointing out of D_T). Figure 1 illustrates the notation for two dimensions.

In testing programs in two dimensions, White and Cohen propose selecting two ON test points and one OFF test point

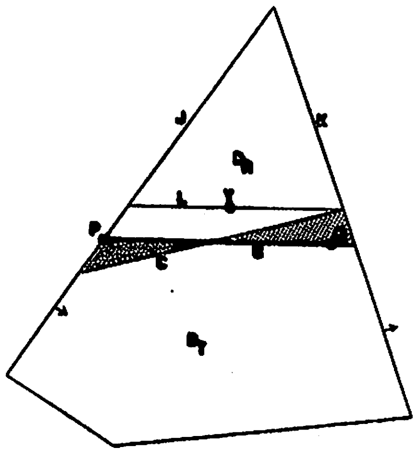

FIGURE 1.
STANDARD NOTATION FOR THE DIMENSIONS

for each closed border of a path domain [WHI80]. We refer to this domain testing strategy as the <u>2x1 strategy</u>. The two ON points are selected as close as possible to the ends of the given border. If G is closed at an end, then the corresponding endpoint is chosen as an ON point. If G is open at an end, the corresponding endpoint is not included in D_T, so an ON point at some small distance from the endpoint is chosen; we assume this distance is no larger than d. The OFF point is selected a distance d from G and is chosen so that its projection onto G lies strictly between the two ON points. Further, the OFF point must satisfy all the inequalities defining D_T except the inequality corresponding to G; thus, it lies within the side extensions J and K. In Figure 1, for instance, the 2x1 strategy might select the ON points P and Q and OFF point Y to test the given border G.

Figure 2 illustrates the ability of the 2x1 strategy to detect border shifts. Disallowing coincidental correctness, testing the program for a point that lies within the wrong path domain is sufficient to detect the domain error. In each of the cases in Figure 2, at least one of P, Q, or Y lies on the wrong side of the correct border, so these types of border shifts are detected.

The 2x1 strategy, however, is unable to reveal all border shifts. Any shift for which the correct border lies completely between the line segment PQ and the point Y will be undetected, as shown in Figure 3. Note that P, Q, and Y all lie in their correct path domains. Assuming that the

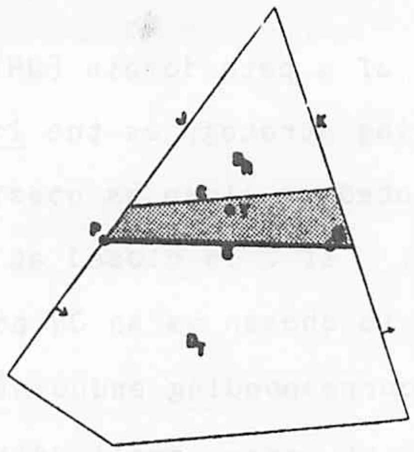

FIGURE 2A.
BORDER SHIFT DETECTED BY 2X1 STRATEGY

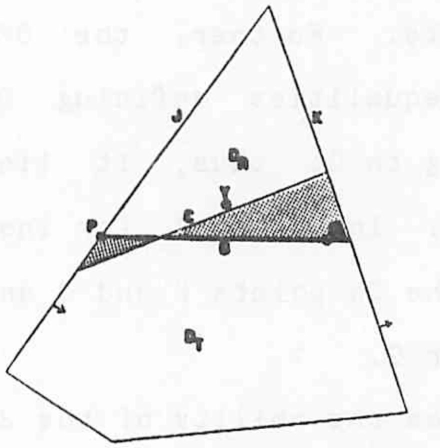

FIGURE 2B.
BORDER SHIFT DETECTED BY 2X1 STRATEGY

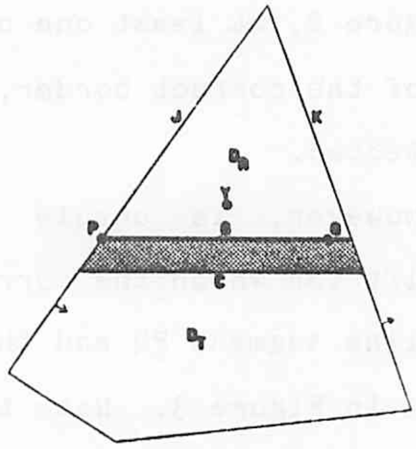

FIGURE 2C.
BORDER SHIFT DETECTED BY 2X1 STRATEGY

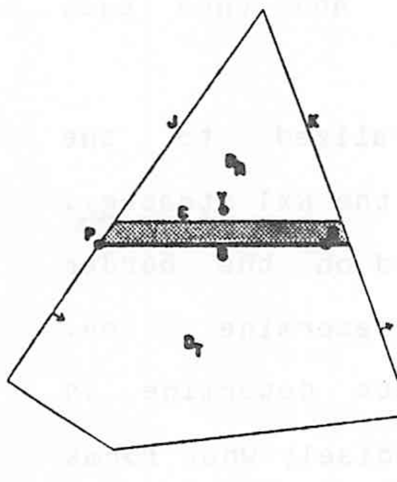

FIGURE 9A.
BORDER SHIFT UNDETECTED BY 2X1 STRATEGY

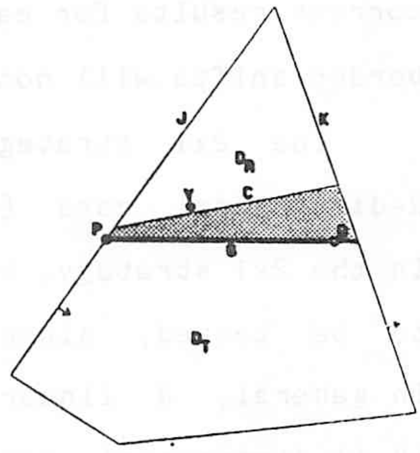

FIGURE 9B.
BORDER SHIFT UNDETECTED BY 2X1 STRATEGY

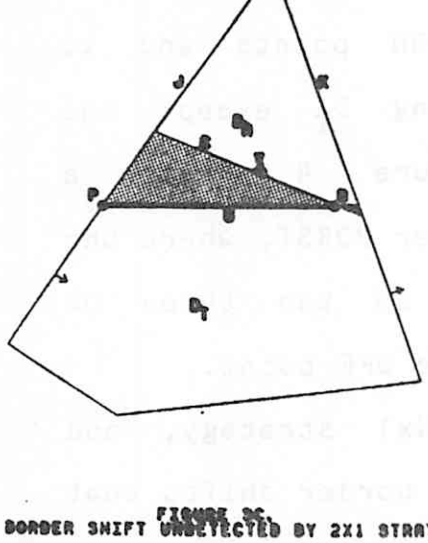

FIGURE 9C.
BORDER SHIFT UNDETECTED BY 2X1 STRATEGY

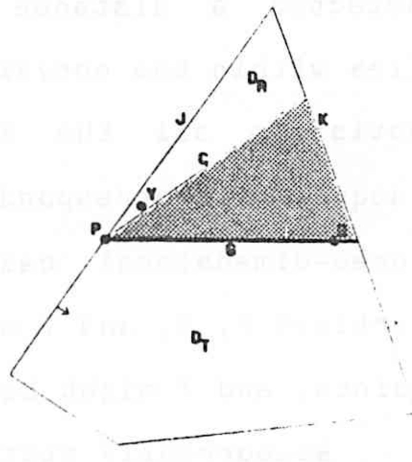

FIGURE 9D.
BORDER SHIFT UNDETECTED BY 2X1 STRATEGY

path computations are correct, the program will yield correct results for each of P, Q, and Y, and thus such border shifts will not be detected.

The 2x1 strategy has been generalized to the N-dimensional case [WHI80], resulting in the Nx1 strategy. In the 2x1 strategy, two points are selected on the border to be tested, since two points uniquely determine a line. In general, N linearly independent points determine an (N-1)-dimensional hyperplane, which is precisely what forms a border in N-dimensional space. Thus, to test a border in N dimensions, the Nx1 strategy selects N independent ON points and one OFF point. Any N vertices of G are linearly independent and thus the ON points are selected at, or as close as possible to, the vertices of G. The OFF point is selected a distance d from G so that its projection onto G lies within the convex hull of the N ON points and it satisfies all the inequalities defining D_T except the inequality corresponding to G. Figure 4 shows a three-dimensional case with given border PQRST, where the vertices P, R, and T might be selected as the three ON points, and Y might be selected as the one OFF point.

As currently stated [WHI80], the Nx1 strategy, and hence the 2x1 strategy, allows undetected border shifts that result in unnecessarily large displaced domains. In Figure 3D, note that Y is selected near one of the side extensions, thus allowing a larger displaced domain than if Y had been selected near the center. A similar situation occurs in N dimensions, as shown in the 3-dimensional example of Figure

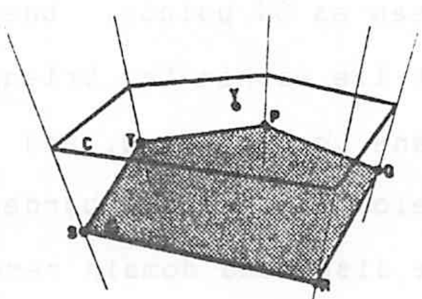

FIGURE 4.
NX1 STRATEGY IN THREE DIMENSIONS
(GIVEN BORDER IS SHADED)

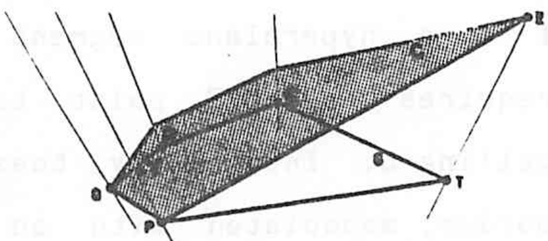

FIGURE 5.
LARGE DISPLACED DOMAIN ALLOWED BY NX1 STRATEGY
(CORRECT BORDER IS SHADED)

5, where the given border is an assymetric pentagon, PQRST. If P, Q, and R are chosen as ON points, the projection of the OFF point Y must lie within the triangle PQR. If the correct border is a plane through P, Q, and Z rising from the base and passing below Y, then the border shift will not be detected and a large displaced domain results.

To control this situation, additional constraints must be placed on the ON and OFF test points. Two factors contribute to the large displaced domain in Figure 5: the ON points are clustered close together on one side of the given border, and the OFF point is not near the center. The strategy can be improved by the addition of two constraints. These additional constraints are the <u>centroid constraints</u>: the figure formed by the N ON points must contain the centroid of the given border, and the OFF point must be at the centroid of the hyperplane segment L. In two dimensions, this requires the OFF point to be at the midpoint of the ceiling L. Intuitively, these constraints force the correct border, associated with an undetectable border shift, to "be closer to" the given border than if the OFF point is allowed to be far from the center. In most cases, employing the centroid constraints reduces the displaced domain associated with an undetectable border shift. The analysis of the Nx1 strategy, which follows, assumes the centroid constraints are satisfied.

4. Error Analysis of the Nx1 Domain Testing Strategy

This section describes two error measures that have been proposed for domain testing. The characteristics and usefulness of these measures are discussed. The Nx1 strategy is then shown to be inadequate according to the more useful error measure.

The first error measure, called the domain error magnitude, attempts to quantify the distance between the given and correct borders [WHI80]. In two dimensions, when an undetected border shift occurs, the correct border must intersect each of the two ON-OFF line segments formed by the three test points, where the points of intersection are called H and I. The domain error magnitude (DEM) of an undetected border shift is the maximum of the distances from H to G and from I to G. Clearly, in all cases, the DEM is strictly less than d. Figure 6 illustrates the construction of the domain error magnitude.

We argue that the DEM is inadequate as an error measure. Its use is limited because one must know the correct border in order to evaluate the DEM. Furthermore, the DEM fails to reflect how much of the program domain is processed incorrectly. Consider Figure 7, in which two given borders result from two incorrect versions of a program in which C is the correct border. The DEM and displaced domain have been computed in arbitrary units to show their relative size. The DEM in 7A is slightly larger than the DEM in 7B, but the displaced domain in 7A is much smaller than the displaced domain in 7B. Our intuition

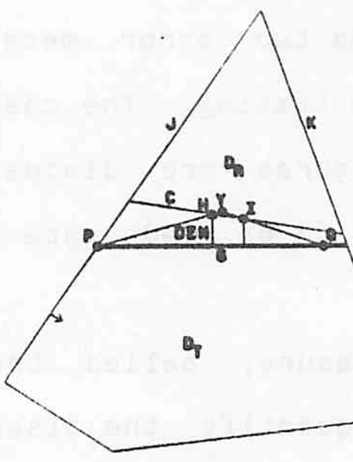

FIGURE 6.
CONSTRUCTION OF DOMAIN ERROR MAGNITUDE (DEM)

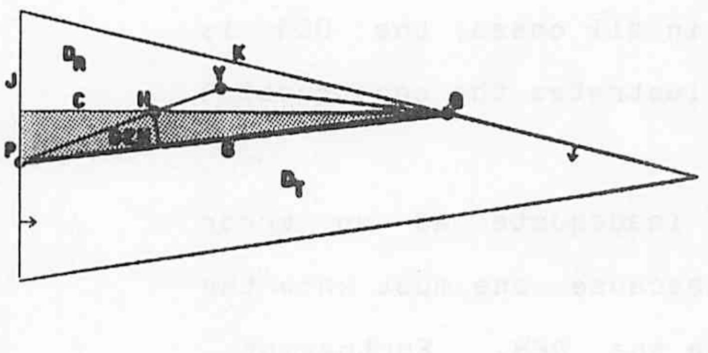

FIGURE 7A.
DOMAIN ERROR MAGNITUDE IS 2.02
AREA OF DISPLACED DOMAIN IS 360.0

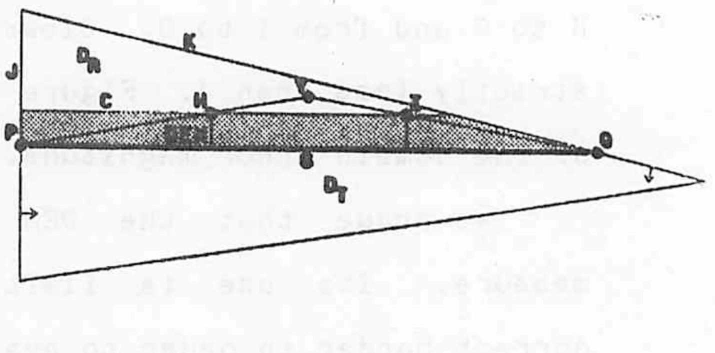

FIGURE 7B.
DOMAIN ERROR MAGNITUDE IS 2.00
AREA OF DISPLACED DOMAIN IS 560.0

tells us that there is more error in 7B, since the program corresponding to 7B processes more of the program domain incorrectly than the program for 7A. We contend that the concept of area is more appropriate as an error measure than the distance measure given by the DEM; thus, the DEM is not discussed further.

The second error measure, which we call the border shift error, models the maximum displaced domain that could result from an undetectable border shift [WHI78]. To define this error measure, it is convenient to consider the _border shift contour_, which is the shape of the figure formed by the given and correct borders and the side extensions, when an undetected border shift exists. When a correct border does not intersect both side extensions, an _open border shift contour_ occurs, as shown in Figure 8. When both ON points are endpoints of G, three types of closed border shift contours - a _border shift trapezoid_, a _border shift triangle_, and a _border shift quadrilateral_ - can occur. If one or more of the ON points are not endpoints of G, these three types of closed border shift contours and a fourth type - a _border shift double-triangle_ - can occur. A border shift double-triangle is a pair of vertical triangles. The undetected border shifts in Figure 3 show the four types of closed border shift contours - border shift trapezoid (3A), border shift quadrilateral (3B), border shift double-triangle (3C), and border shift triangle (3D). Note that a border shift trapezoid is a special case of a border shift quadrilateral, but it is distinguished because it

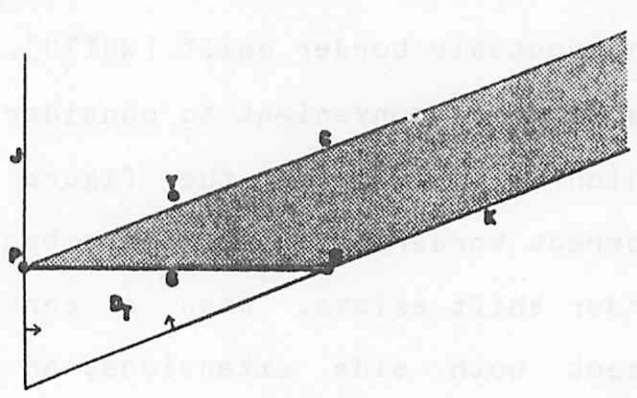

FIGURE 8.
OPEN BORDER SHIFT CONTOUR
FOR 2X1 STRATEGY

plays an important role in the analysis that follows.

The consideration of border shift contours was motivated by displaced domains. Despite the intuitive appeal of displaced domains, their exact determination is complicated by many factors, especially the interaction among borders. The border shift contours are bounded by the side extensions, rather than some other arbitrary lines, because the extensions are sensitive to the shape of the path domain being tested. Moreover, the side extensions frequently are the borders of the adjacent path domain and when this occurs, the area of the border shift contour is likely to be the same as that of the displaced domain. It is important to note that while a border shift contour may not correspond exactly to the boundary of a displaced domain, both the displaced domain and border shift contour behave the same. If a change to a border increases or decreases the displaced domain, the area of the border shift contour increases or decreases accordingly.

In general, the correct border is not known (otherwise it would be used in the program!) and thus the area of the corresponding border shift contour cannot be found. It is natural, however, to ask about the area of the largest contour that could result from an undetectable border shift; such contours are called the <u>limit contours</u>. The <u>limit trapezoid</u> is the convex hull of the border shift trapezoids - that is, the smallest trapezoid containing all border shift trapezoids. Thus, it is the trapezoid with G and L as bases and the side extensions J and K as sides. The two limit triangles, one with an edge along J and one with an edge along K, are the convex hulls of the border shift triangles with an edge along J or K, respectively. Similarly, the two limit double-triangles, are the convex hulls of the border shift double-triangles with an edge along J or K. Note that the limit double-triangle with an edge along J exists only when the ON point Q is not an endpoint of G, and likewise for P and K. A limit of the border shift quadrilaterals need not be considered since we are concerned with the area of the maximum displaced domain; it can be shown that the maximum area of the border shift quadrilaterals is bounded by the area of either the limit trapezoid or one of the limit triangles, whichever is largest. Note that the limit contours are not border shift contours, because each contains an OFF point, which would reveal the corresponding border shift when tested. Limit contours for the 2x1 strategy are illustrated in Figure 9. The concepts of border shift contour and limit contour generalize to N dimensions in a straightforward way. For instance, the 3-dimensional generalization of a limit trapezoid is a solid with two parallel plane bases and any number of plane sides; an example appears in Figure 10.

The limit contours are used to model the maximum displaced domain allowed by a domain testing strategy. The border shift error (BSE) of a tested border is the least upper bound (if it exists) of the set of (generalized) volumes of the border shift contours resulting from undetectable shifts of that border; the border shift error

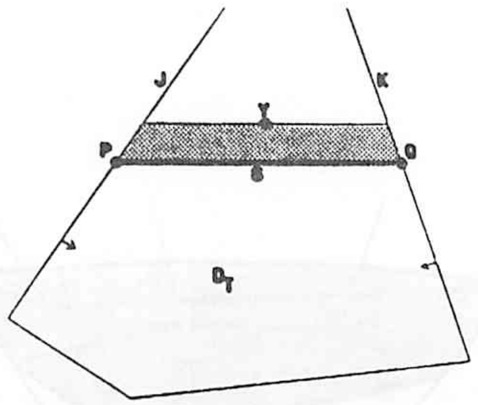

FIGURE 9A.
LIMIT TRAPEZOID
FOR NX1 STRATEGY

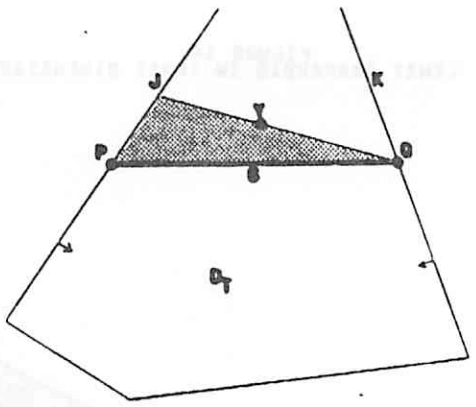

FIGURE 9B.
LIMIT TRIANGLE WITH EDGE ALONG J
FOR 2X1 STRATEGY

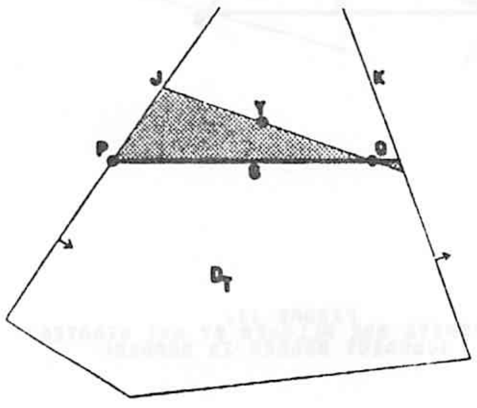

FIGURE 9C.
LIMIT DOUBLE-TRIANGLE WITH EDGE ALONG J
FOR 2X1 STRATEGY

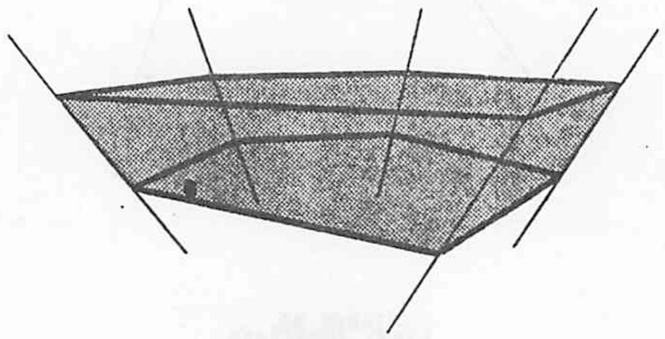

FIGURE 10.
LIMIT TRAPEZOID IN THREE DIMENSIONS

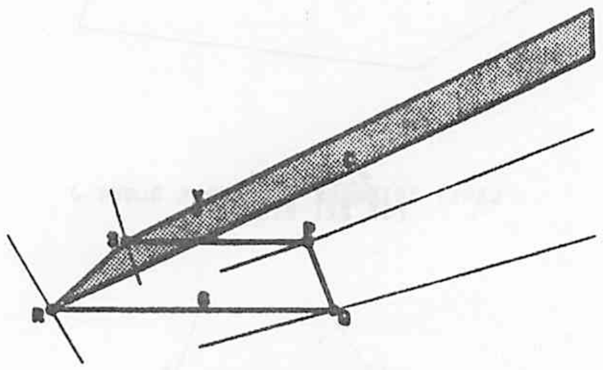

FIGURE 11.
INFINITE OSE ALLOWED BY MX1 STRATEGY
(CORRECT BORDER IS SHADED)

is infinite if the least upper bound does not exist. Thus, if an open border shift contour can occur the BSE is infinite, otherwise it is the volume of the largest limit contour. In absolute terms, it is not clear how to interpret the BSE. What does it mean for a border to have a BSE of 450.0? Nevertheless, the BSE can be used comparatively. In general, if one domain testing strategy always provides a smaller BSE than another strategy, we conclude the former is better.

We are concerned that the Nx1 domain testing strategy allows open border shift contours and thus an infinite border shift error. This occurs when a side extension "tilts outward" so far or the correct border "tilts upward" so far that they fail to intersect. An infinite BSE results from the open border shift contour shown in Figure 8. Figure 11 illustrates an infinite BSE in three dimensions, where P, R, and S are the ON points. Note that although the centroid constraints are used in selecting the test points in both examples, an infinite BSE still occurs. An infinite BSE, however, can be avoided in domain testing. Section 5 presents an alternative domain testing strategy that always yields a finite error as measured by BSE.

5. Alternative Strategies for Domain Testing

In this section, two alternatives to the Nx1 domain testing strategy are proposed. Both strategies, when applied to 2-dimensional space, coincide in the 2x2 strategy, which is presented first. The 2x2 strategy, which selects two ON and two OFF test points, is shown to have finite BSE that can be found in a straightforward manner. Each of the generalized strategies is described for higher dimensions, and its associated BSE is discussed.

Recall that in the 2x1 strategy an infinite BSE occurs when the correct border is allowed to tilt upward so far that it fails to intersect a side extension. The 2x2 strategy proposed here prevents this by selecting two OFF points, one near each end of the border being tested. These two OFF points are chosen on the line L, which is parallel to the given border, thus L forms a "ceiling" for undetectable border shifts. Using the standard notation presented in Section 2, the two ON points, P and Q, are selected as in the 2x1 strategy, and the two OFF points, U and V, are selected as vertices where L intersects J and K. Figure 12 shows test points that might be chosen using the guidelines of the 2x2 strategy, and demonstrates that the 2x2 strategy does not detect all border shifts. Note that the allowable border shift trapezoids are the same as in the 2x1 strategy. By the selection of test points at the ends of L, however, the 2x2 strategy disallows the large border shift triangles, quadrilaterals, and double-triangles allowed by the 2x1 strategy.

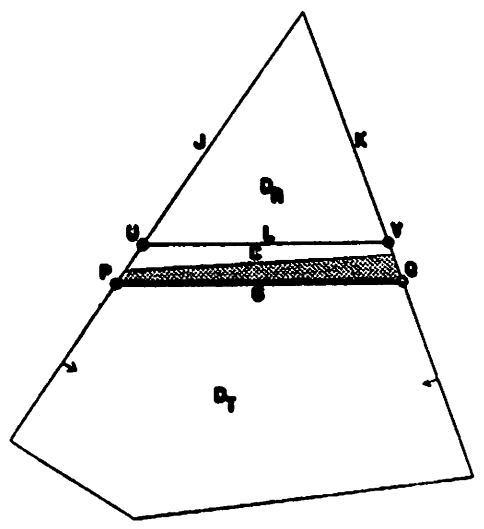

FIGURE 12.
BORDER SHIFT UNDETECTED BY 2X2 STRATEGY

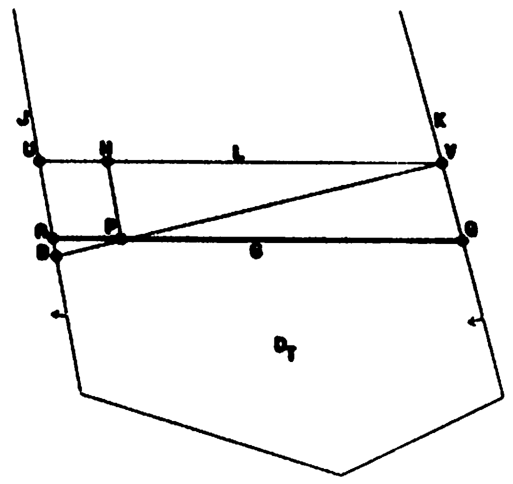

FIGURE 13.
REFERENCE FOR THEOREM 1

The BSE of the 2x2 strategy is proven, under reasonable assumptions, to be finite and given by the limit trapezoid. Because of the importance of the existence of a finite BSE and the ease of the technique for finding it, our results are stated formally.

THEOREM 1. Assuming that $|L| > 2d$, the BSE of a border tested by the 2x2 domain testing strategy is finite and given by the area of the limit trapezoid.

Proof: Refer to Figure 12, where G is a closed line segment. Both endpoints of G (P and Q) and both endpoints of L (U and V) are tested. Thus, no open border shift contours can occur and all border shift trapezoids, triangles, and quadrilaterals are contained in the limit trapezoid, UVQP. The theorem follows immediately.

Otherwise, assume at least one end of G is open. Refer to Figure 13, where P/= A. It is necessary to show that the area of the limit double-triangle (APB,PVQ) does not exceed the limit trapezoid UVQA. It is sufficient to show that Area(APB) < Area(UVPA). Construct the line segment from P on G to L and parallel to J; suppose W is the intersection with L. Since L is parallel to G, angle(WVP) = angle(APB), and thus WVP and APB are similar triangles. $|L|>2d$ and $|UW|=|AP|<d$ implies that $|WV|>d$. Thus, Area(APB) < Area(WVP) < Area (UVPA) ::

In those cases in which the BSE of a border tested by the 2x1 strategy is infinite, the 2x2 strategy clearly yields a better result. Theorem 2 shows that, in terms of the BSE, the 2x2 strategy is always at least as good as the 2x1 strategy.

THEOREM 2. In testing any 2-dimensional border, the BSE resulting from the 2x2 strategy is less than or equal to the BSE resulting from the 2x1 strategy.

Proof: The BSE resulting from the application of the 2x1 strategy is the least upper bound, if it exists, of the set of areas of all border shift contours. For the 2x2 strategy, the BSE is the least upper bound of the set of areas of border shift trapezoids. Since the set of border shift trapezoids is a proper subset of the set of all border shift contours, the least upper bound of the areas of the former set cannot exceed that of the latter. ::

In considering generalizations of the 2x2 strategy to higher dimensions, two interpretations result. One interpretation provides the NxN strategy, where N is the dimension of the domain; another provides the VxV strategy, where V is the number of vertices of the given border.

The NxN strategy arises by interpreting "2" as the dimension of the domain. In choosing two OFF points, the 2x2 strategy defines the line segment L, which forms a "ceiling" for undetectable border shifts. Thus, for N-dimensional spaces, the NxN strategy would choose N linearly independent ON points and N linearly independent OFF points. The N OFF points are chosen a uniform distance d from the border G and define the hyperplane segment L, which thus forms a "ceiling" for undetectable border shifts. The ON points are chosen so that their convex hull contains the centroid of G. Likewise, the convex hull of the OFF points must contain the centroid of L. This insures that the test points are not clustered at one end of either the border or ceiling. Furthermore, selecting ON test points that are on or near the vertices of G and selecting OFF test points on the vertices of L guarantees the linear independence of both sets of points. In Figure 14, for example, S, P, and R might be selected as the ON points. If the side extensions that form the border edges are perpendicular to G, then L and G are congruent and X, U, and W might be chosen as the OFF test points.

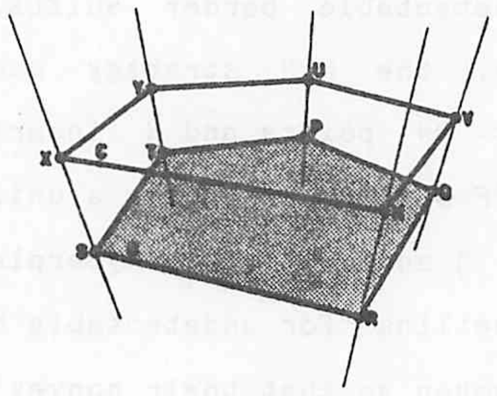

FIGURE 14.
ALTERNATIVE STRATEGIES IN THREE DIMENSIONS
(GIVEN BORDER IS SHADED)

A troubling characteristic of the NxN strategy, which is also true of the Nx1 strategy, is its insensitivity to changes in the shape of the path domain, including those that affect the shape of the tested border. Many changes could occur in border shape without requiring a change in the test points. For example, changes to the vertices that do not move the selected test points or the centroid will not necessitate the selection of new test points. The number of vertices could even change without requiring a change in test points! This does not happen in two-space because a line segment has only two vertices, and two test point are selected near them. It is a coincidence of geometry that two points determine a line and a line segment has two vertices. On the other hand, a (N-1)-dimensional hyperplane needs only N points to be defined, but a hyperplane segment may have any number of vertices. A strategy more sensitive to path domain shape is desired.

Another disturbing problem with both the Nx1 and NxN strategies is that each allows many sets of acceptable test points, but neither provides sufficient guidance in choosing the best set. To minimize the BSE, the test points should not be clustered but should be scattered with respect to each other and their associated centroid. There are several techniques for selecting test points to achieve this scattering affect, such as maximizing the sum of the distances between the ON points as well as the distances between the OFF points. There does not appear to be any technique, however, that guarantees a minimum BSE without computing the BSE for all possible sets of test points selected from the vertices. To be effective, therefore, the NxN strategy requires the consideration of all vertices plus additional computation to determine the best test point set.

As in two dimensions, the NxN strategy always gives at least as good as, and often better, BSE than does the Nx1 strategy. The most serious flaw in the NxN strategy, however, is that even the best N test points may fail to provide a finite BSE. When the number of vertices of a border is much greater than the dimensionality of the domain, some consecutive vertices will go untested. This may allow a correct border to tilt upward so far that it fails to intersect a side extension corresponding to the untested vertices, resulting in an infinite BSE. Thus, we are led to the VxV strategy.

The VxV strategy is an alternative way of generalizing the 2x2 Strategy, arising from the interpretation of "2" as the number of vertices of the tested border. Recall that border vertices are formed by the intersections of side extensions (hyperplanes) with the given border. If there are V vertices to the given border, V ON points are selected at or as close as possible to each vertex of G and V OFF points are chosen at the vertices of L. These V ON and V OFF points correspond to the ends of the lines G and L in two dimensions. For the example in Figure 14, the VxV strategy would select 5 ON points, P, Q, R, S, and T, and 5 OFF points, U, V, W, X, and Y.

Although it is difficult to prove since the number of vertices per border is unconstrained, we believe that the BSE for the VxV strategy is always finite and bounded by the volume of the generalized limit trapezoid. Moreover, the VxV strategy is completely sensitive to path domain shape. Any change in border shape requires a change in test points. The strategy is even influenced by subtle features, such as the angles of intersection with adjacent borders, which do not affect border shape but do change the shape of the path domain and the tilt of the side extensions. This strategy captures our intuition about how a path domain should be tested. Each vertex represents an extremal value for a border. Thus, arguments for testing extremal values support a strategy for testing all vertices. In addition to guaranteeing a finite BSE, the complexity of the VxV strategy is often less than that of the NxN.

There are a number of ways to measure the complexity of a testing strategy. Perhaps the most straightforward is to count the number of test points required. In this light, the NxN strategy does not seem much more costly than the Nx1 strategy, but the cost of the sensitivity provided by the VxV strategy appears high. The NxN strategy requires 2*N test points per border and 2*N*B per path domain, where B is the number of closed borders in the path domain. In contrast, the Nx1 strategy requires N+1 points per border and (N+1)*B per path domain. The VxV strategy requires 2*V test points for each closed border, where V is the number of vertices of the border. For an entire path domain, the number of test points required is $\sum_{I=1,B} 2*V_I$, where V_I is the number of vertices of the Ith border. For even a simple program with relatively few variables, this total explodes quickly.

The number of test points along with information about the run time of the program, however, only provides information about the time necessary to actually test the program. Another major contributor to the complexity of a testing strategy is the time required to select the test points. Serious concerns arise in considering the complexity of each of the three domain testing strategies with respect to this measure. For each strategy, determining the OFF points is on the same order of complexity as determining the ON points and thus only the latter is described. To select the ON points, the VxV strategy requires finding the vertices of the given border, which involves solving V systems of N equations and requires polynomial time, on the order of N^3 for each vertex, and hence, $V*N^3$ for each border. Both the Nx1 and NxN strategies require solving at least N systems of N equations in order to identify suitable ON points. In fact, in order to satisfy the centroid constraints, all vertices must be found and thus V systems of N equations must be solved. Thus, to locate the ON points, the Nx1 and NxN strategies also require time on the order of $V*N^3$ per border. The dominating factor, however, is the time required to select the best set of test points for the Nx1 and NxN strategies; all combinations of vertices must be considered to determine the minimum BSE, which in general requires nonpolynomial time.

Clearly, both the time to select the test data and the time to execute the program on the chosen data should be considered in measuring the complexity of a testing strategy. When both are taken into account, the VxV strategy is usually cheaper than the Nx1 and the NxN strategies, since executing the program on additional points usually takes less time than selecting the best set of vertices. The improvement in complexity as well as the guarantee of a finite BSE indicates that the VxV strategy is the better domain testing strategy.

6. Other Considerations

The domain testing strategies described above are limited in their scope by certain restrictions. The strategies all assume that the predicate interpretations are simple, linear inequalities. Moreover, the input space is assumed to be continuous and coincidental correctness is disallowed. With slight modifications to these strategies, some of these restrictions can be dropped and others weakened. This section examines these restrictions and discusses possible modifications. The need for the integration of domain testing and other path analysis testing methods into an overall scheme for testing is also discussed. Finally, an underlying flaw of domain testing, which is common to all path analysis methods, is addressed.

The domain testing strategies can easily be modified to handle equality and nonequality predicates. White and Cohen have proposed such a modification for the Nx1 strategy [WHI80]. This modification leaves the testing of inequality predicates unchanged, but the testing of equality predicates requires the selection of two OFF points, one on each side of the corresponding border. Nonequality predicates form open borders and thus will be tested in an adjacent path domain. Similar modifications to the NxN and VxV strategies require that twice as many OFF points, divided between the two sides of the border, be chosen for equality predicates.

The restriction that borders result from simple predicates insures that the path domains are convex. Complex predicates that are the conjunction of inequalities cause no problems, but the complement of such predicates or any predicate with a disjunction may require special processing. Again, White and Cohen have proposed a modification to the Nx1 strategy, which can also be applied to the NxN and VxV strategies. When a predicate interpretation contains a disjunction that produces a nonconvex path domain, the modified strategies divide the path domain into convex subsets and then test each subset independently. If two of these subsets are adjacent (as is often the case), there is no need to test the "imaginary" border between them, since the same function is computed for both subsets.

The domain testing strategies assume that the predicate interpretations corresponding to both the given and correct borders are linear. This assumption is frequently satisfied [CLA76], especially for programs solving nonnumeric problems. To handle a wider range of applications, however, the linearity restriction must be dropped. An appropriate modification is dependent on the degree of and number of terms in the predicate interpretation and requires the selection of both ON and OFF test points at each of the local minima and maxima of a nonlinear border. Unfortunately, this often produces an inordinate number of test points and requires substantial effort. Moreover, any modification that does less than this allows an inordinate error bound. This is an inherent weakness of domain testing, and thus the practical applicability of the method is limited to predicate interpretations of low degree.

The described domain testing strategies assume a continuous input space. This guarantees that ON points can be selected at or near the edges of the given border and OFF points can be selected arbitrarily close to the border. Application of domain testing to discrete space requires additional analysis [WHI78] to locate appropriate ON and OFF test points. In pathological cases, the border may not contain points to select as ON points, or there may not be any points between the side extensions that are candidates for OFF points. All the strategies can be extended, however, to handle discrete space in such a way that they can successfully test most borders without much additional effort.

The domain testing method has an underlying assumption that a test oracle is available, which can determine if execution of the program on the chosen test points produces correct results. The restriction that coincidental correctness cannot occur implies that when correct results are produced, the correct path was executed. Coincidental correctness is a strong and perhaps unrealistic restriction to impose, since in most cases its existence cannot be determined. A weaker and more reasonable restriction is that the path computation associated with each path domain surrounding the given border be recognizably different than the computation of the path being tested. Suppose an ON test point should lie in one of the surrounding path domains, but executing the program for this point produces correct results. Comparing the path computations associated with this domain and the domain being tested would demonstrate that both computations produce the same result for this point. This would alert the tester to the possibility of coincidental correctness and to the need to test additional points on the border.

In general, the ability to examine the path computations for all surrounding domains allows dismissal of the coincidental correctness restriction. Surrounding domains include any path domain containing a selected OFF test point as well as adjacent path domains that do not contain an OFF test point. Figure 15 shows an example where it is necessary to know the path computation of an adjacent domain containing no OFF test points. (In this example, J and K are merely side extensions and not borders of any domain.) Suppose P is coincidentally correct and should lie in D_A, then the border shift would not be detected unless the path computations associated with D_A and D_T were both available. For the path domain being tested and surrounding path domains containing OFF test points, the test points determine the path and symbolic execution can easily be employed to provide representations of the associated path computations. Determining that an adjacent path domain with no OFF test points exists poses an additional problem. This problem, however, is more tractable than demonstrating that coincidental correctness does not occur.

The restrictions that were initially assumed for domain testing, therefore, can be weakened or dismissed with appropriate modifications to the strategies. This enables

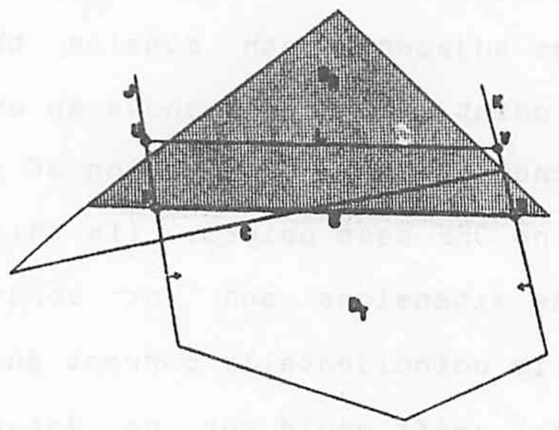

FIGURE 13
DETECTION OF COINCIDENTAL CORRECTNESS
(ADJACENT DOMAIN IS SHADED)

the application of domain testing to a larger class of programs. For domain testing to be truly effective, however, it must be integrated with other testing methods.

The domain testing method concentrates solely on path selection errors, thus other testing methods must also be employed to thoroughly test a program. Recall that program errors were divided into three types - path selection errors, missing path errors, and computation errors. The domain testing method focuses on detecting path selection errors, but may also uncover missing path and computation errors.

With domain testing, as with most testing methods, missing path errors are only detected by mere chance. In fact, missing path errors cannot be found systematically unless a specification is available. A correct specification would describe all the cases that would be handled by the program. The path domains can then be compared to the specification to determine if any cases have been neglected. The partition analysis method [RIC81] provides a technique for comparing a program to its specification, which assists in revealing missing path errors.

The domain testing method may inadvertently uncover computation errors, since the program is executed on several test points. The execution of the program for a chosen test point, however, may produce correct results, although the associated path computation is incorrect for other points in the path domain (allowing coincidental correctness).

Methods that are sensitive to computation errors should certainly be applied. Some of these computation testing methods examine the path computation to determine the number of points required to test a path. Note that the domain testing method may have selected some, and possibly the required number, of such test points for a path.

It may appear that symbolic representations of the path domain and computation are adequate to verify the correctness of the path, but this is done in a postulated environment. We believe it is imperative to execute the path on astutely selected test data to actually observe the program's performance and substantiate the correctness of the symbolic representations. Domain testing, in conjunction with these other methods, provides an overall scheme for path analysis testing.

A major drawback of path analysis testing is its dependence on the number of paths in a program. Programs often have a large, and possibly infinite, number of paths due to program loops. Practical testing methods must be developed that do not require the analysis of every path in a program. One approach to this problem is to analyze the first few iterations of a loop and then to generalize the results. Another approach [CHE79,CLA81] is based on the creation of a closed form representation of a loop, thus allowing paths that differ only by the number of loop iterations to be analyzed as a single class of paths. Both of these approaches deteriorate for complicated loops. Further research in the area of loop analysis is required and the applicability of these approaches to domain testing must be investigated.

7. Conclusion

This paper examines the domain testing method. First, the general method and the White and Cohen strategy for selecting test points for domain testing are introduced. Two error measures, DEM and BSE, are evaluated. It is shown that the DEM does not adequately reflect the size of the displaced domain and thus this error measure is rejected. The BSE is found to be a better error measure in that it models the maximum displaced domain that could result from an undetected border shift. The Nx1 strategy is analyzed with regard to this measure and it is shown that this strategy frequently allows unbounded displaced domains and thus an infinite BSE.

Two alternative domain testing strategies, which improve on the error bound, are proposed. In comparing these strategies, trade-offs between the error bound as measured by BSE, the necessary number of test points, and the time required to select the test points are all considered. The VxV strategy is determined to be better than both the Nx1 and the NxN, because, although more test points are required, these points can be selected in polynomial time and a finite BSE is always provided.

Theoretically, polynomial time is considered reasonable, but with a large number of inputs, domain testing may be too impractical to employ. Moreover, domain testing can only be reasonably applied to paths whose predicate interpretations are of low degree. Perhaps the biggest obstacle that domain testing must overcome, along with all path analysis methods, is its dependence on testing individual paths; a breakthrough on loop analysis must be achieved.

Despite these drawbacks, the domain testing method is intuitively appealing in that it formalizes the notion of testing the boundary of a path domain. Although further investigation of domain testing is needed, it appears to be a powerful path analysis testing method that can effectively be integrated into an overall testing scheme. Furthermore, it provides a theoretical basis for evaluating the limits of testing.

REFERENCES

CHE79 T.E. Cheatham, G.H. Holloway, and J.A. Townley, "Symbolic Evaluation and the Analysis of Programs," IEEE Transactions on Software Engineering, SE-5, 4, July 1979, 402-417.

CLA76 L.A. Clarke, "A System to Generate Test Data and Symbolically Execute Programs," IEEE Transactions on Software Engineering, SE-2, 3, September 1976, 215-222.

CLA81 L.A. Clarke and D.J. Richardson, "Symbolic Evaluation Methods for Program Analysis," to appear in Program Flow Analysis: Theory and Applications, Prentice Hall, Inc., Englewood Cliffs, New Jersey, 1981.

DEM77 R.A. DeMillo and R.J. Lipton, "A Probabilistic Remark on Algebraic Program Testing," School of Information and Computer Science Technical Report, Georgia Institute of Technology, May 1977.

GOO76 J.B. Goodenough and S.L. Gerhart, "Toward a Theory of Test Data Selection," IEEE Transactions on Software Engineering, SE-1, 2, September 1976, 156-173.

HOW76a W.E. Howden, "Reliability of the Path Analysis Testing Strategy," IEEE Transactions on Software Engineering, SE-2, 3, September 1976, 208-215.

HOW76b W.E. Howden, "Algebraic Program Testing," Department of Applied Physics and Information Science, University of California, San Diego, TR-14, November 1976.

HOW77 W.E. Howden, "Symbolic Testing and the DISSECT Symbolic Evaluation System," IEEE Transactions on Software Engineering, SE-3, 4, July 1977, 266-278.

RIC81 D.J. Richardson and L.A. Clarke, "A Partition Analysis Method to Increase Program Reliability," to appear in the Proceeding of the Fifth International Conference on Software Engineering.

WHI80 L.J. White and E.I. Cohen, "A Domain Strategy for Computer Program Testing," IEEE Transactions on Software Engineering, SE-6, 3, May 1980, 247-257.

WHI78 L.J. White, F.C. Teng, H. Kuo, and D. Coleman, "An Error Analysis of the Domain Testing Strategy," Ohio State University, CISRC-TR-78-2, December 1978.