RELIABLE TEST DATA SELECTION STRATEGIES -- AN INTEGRATED APPROACH*

Lori A. Clarke
Debra J. Richardson

COINS Technical Report 82-13
July 1982

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

*This paper appears in the Fourth Israel Conference on Quality Assurance, 1982.

This research was funded in part by the National Science Foundation under grant NSFMCS 81-04202.

```
   procedure TRIANGLE( A, B, C: in natural;
         CLASS: out integer; AREA: out real) is
   ASQRD, BSQRD, CSQRD: integer; S: real;
s  begin
1  if (A < B) or (B < C) then
      -- illegal input
2     CLASS := -1;
3     AREA := 0.0;
   else -- A >= B >= C
      -- legal input
4     if (A /= B) and (B /= C) then
         -- triangle is scalene
5        ASQRD := A*A;
6        BSQRD := B*B;
7        CSQRD := C*C;
8        if (ASQRD = BSQRD + CSQRD) then
            -- right triangle
9           CLASS := 3;
10          AREA := B * C / 2.0;
         else -- ASQRD /= BSQRD + CSQRD
            -- not right triangle
11          S := (A + B + C) / 2.0;
12          AREA := sqrt( S*(S-A)*(S-B)*(S-C) );
13          if (ASQRD < BSQRD + CSQRD) then
               -- acute triangle
14             CLASS := 4;
            else -- ASQRD > BSQRD + CSQRD
               -- obtuse triangle
15             CLASS := 5;
            endif;
         endif;
16    elsif (A = B) and (B = C) then
         -- equilateral triangle
17       CLASS := 1;
18       AREA := A*A*sqrt(3.0)/4.0;
      else -- (A /= B) or (B /= C)
         -- isosceles triangle
19       CLASS := 2;
20       if (A = B) then
21          AREA := C * sqrt(4*A*B-C*C) / 4.0;
         else -- B = C
22          AREA := A * sqrt(4*B**C-A*A) / 4.0;
         endif;
      endif;
   endif;
f  end TRIANGLE;
```

Figure 1. Procedure TRIANGLE

to 4 is the compliment of the condition at node 1. This evaluated branch predicate is first simplified and then conjoined to the previously generated path condition, resulting in the path condition

true and ~((a<b) or (b<c)) = (a>b) and (b>c).

Symbolic interpretation of the statements on a path P_J provides a symbolic representation of the path computation and path domain. The path computation $C[P_J]$ consists of the symbolic representation of the output values. The symbolic representation of the path domain $D[P_J]$ is provided by the path condition. Note that only the input values that satisfy the path condition could cause execution of the path. Figure 2 shows the symbolic representations of the path domains and path computations resulting from symbolic evaluation of all paths in TRIANGLE.

Unlike the TRIANGLE example, most programs contain loops. A symbolic representation of all executable paths through such a program is usually unreasonable since there may be a large, or even infinite, number of executable paths. One approach to this problem is to replace each loop with a closed form expression that captures the effect of that loop[2,4]. Using this technique, a path may then represent a class of paths that differ only by their number of loop iterations. While this is a powerful technique, it is not always

```
P1:     s,1,4,16,19,20,22,f
D[P1]:  (a - b > 0) and (b - c = 0)
C[P1]:  CLASS = 2
        AREA = a * sqrt(4.0*b**c - a**2) / 4.0
P2:     s,1,4,16,19,20,21,f
D[P2]:  (a - b = 0) and (b - c > 0)
C[P2]:  CLASS = 2
        AREA = c * sqrt(4.0*a*b - c**2) / 4.0
P3:     s,1,4,16,17,18,f
D[P3]:  (a - b = 0) and (b - c = 0)
C[P3]:  CLASS = 1
        AREA = a**2*sqrt(3.0)/4.0
P4:     s,1,4,5,6,7,8,11,12,13,15,f
D[P4]:  (a - b > 0) and (b - c > 0) and
              (a**2 - b**2 - c**2 > 0)
C[P4]:  CLASS = 5
        AREA = sqrt((-a**4 + 2*a**2*b**2 +
                     2*a**2*c**2 - b**4 +
                     2*b**2*c**2 - c**4) / 16.0)
P5:     s,1,4,5,6,7,8,11,12,13,14,f
D[P5]:  (a - b > 0) and (b - c > 0) and
              (a**2 - b**2 - c**2 < 0)
C[P5]:  CLASS = 5
        AREA = sqrt((-a**4 + 2*a**2*b**2 +
                     2*a**2*c**2 - b**4 +
                     2*b**2*c**2 - c**4) / 16.0)
P6:     s,1,4,5,6,7,8,9,10,f
D[P6]:  (a - b > 0) and (b - c > 0) and
              (a**2 - b**2 - c**2 = 0)
C[P6]:  CLASS = 3
        AREA = b * c / 2.0
P7:     s,1,2,3,f
D[P7]:  ((a - b < 0) or (b - c < 0))
C[P7]:  CLASS = -1
        AREA = 0.0
```

Figure 2. Paths of TRIANGLE

successful. Other methods that guide in the selection of a subset of paths such as data flow[14,16,17], mutation analysis[7], and blindness testing[21] are currently being explored but are not discussed further here. In the next section, it is assumed that a reasonably powerful method of path selection is being applied and the test data selection strategies are thus described for a selected set of paths.

Test Data Selection Strategies

A test data selection strategy should provide guidance in the selection of test data for a program. Ideally, executing the program on the selected data reveals errors in the program or provides confidence in the program's correctness. In general, program testing detects an error by discovering the effect of that error. It is possible, however, that an error on an executed path may not produce erroneous results for some selected test data; this is referred to as coincidental correctness. For example, suppose that a computation z=a*2 is incorrect and should be z=a**2; if no test data other than a=0 or a=2 are selected, the error will not be detected. Although this appears to be a contrived example, coincidental correctness is a very real phenomenon. Test data selection strategies must address this problem.

The testing literature has classified errors into two types according to their effect on the path domains and path computations. If an incorrect path computation exists, a computation error is said to have occured. Such an error may be caused by an inappropriate or missing assignment statement that affects the function computed by the path. If a path domain is incorrect, a domain error is said to have occured. Domain errors can be further divided into path selection errors and missing path errors. A path

takes on

a) nonextremal values (erroneous processing of typical structures)
a) extremal values (erroneous processing of atypical structures or insufficient storage);

5) a compound structure referenced in $C[P_J]$ takes on
   a) an empty value (erroneous initialization or processing of underflow)
   b) a full value (erroneous processing of overflow)

These guidelines are not applicable to the path computations in TRIANGLE, since they do not contain data manipulations.

A path computation may contain both arithmetic and data manipulations, in which case all applicable guidelines should be considered. It is important to note that the guidelines may not all be satisfiable due to the condition defining $D[P_J]$ or the representation of $C[P_J]$. In selecting test data for path P_1 of TRIANGLE, for example, several guidelines could not be satisfied due to the constraints that a, b, and c be positive and that b=c. These computation testing guidelines subsume those proposed by Howden[12] for special values testing and extremal output values testing, as well as the error-sensitive test case analysis proposed by Foster[9].

When the path computations fall into specialized categories, the general computation testing guidelines can be tuned to guide in the selection of an even more comprehensive set of test data. For example, if a path computation involves trigonometric functions, then guidelines dependent upon their properties should be exploited. Polynomial functions are another category for which the guidelines can be refined. Under certain assumptions, it is possible to demonstrate the correctness of a polynomial path computation by means of testing. This is called polynomial testing and is based on algebraic results, applicable only when an upper bound on the algebraic complexity of the "correct" path computation is known. If the path computation $C[P_J]$ should be a univariate polynomial of maximal degree T-1, the selection of T linearly independent test points is sufficient to determine whether $C[P_J]$ is correct. If the path computaton $C[P_J]$ should be a multivariate polynomial in K input values of maximal degree T-1, $C[P_J]$ must be tested for T^K linearly independent test points in order to determine that it is correct[11]. The practicaltiy of polynomial testing is limited to polynomials in few variables and of low degree since the number of test points required to determine correctness increases rapidly with the number of variables and the degree.

Domain Testing

Domain testing is based on the observation that points satisfying boundary conditions are most sensitive to domain errors. A path selection error is manifested by a shift in some section of a path domain boundary. A missing path error typically corresponds to a missing path domain along some section of the boundary of an existing path domain. Missing path errors are particularly insidious, however, since it is possible that only one point in a path domain should be in the missing path domain. In this case the error will not be detected unless that point happens to be selected for testing. Missing path errors cannot be found systematically unless a specification is employed by the test data selection strategy, as is done by the partition analysis method[18].

The domain testing strategy[5,20] selects test data on and near the boundaries of each path domain. The boundary of a path domain is composed of borders with adjacent path domains. For each closed border, the strategy selects "on" test points, which lie on the border and thus in the path domain being tested, and "off" test points, which lie on the open side of the border and thus in an adjacent path domain. In such a way, domain testing attempts to detect border shifts, which occur when the border being tested is incorrect -- that is, it differs from the correct border. If the correct results are produced for each of the on and off test points, the border must be "close" to the correct border. An undetected border shift can only occur if 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 border being tested as possible. In fact, with the proper selection of on and off test points, a quantified error bound measuring the set of elements placed in the wrong domain by an undetected border shift can be provided. Figure 4 illustrates a border shift, where G is the border being tested, C is the correct border, and the set of elements placed in the wrong domain is shaded. This border shift is revealed by testing the on points P and Q and the off points U and V, since the off point V is in the wrong domain. For a path domain border resulting from an inequality predicate in two-dimensions (two input values), the selection of four test data points (two on points and two off points is most effective for detecting border shifts. For an inequality border in higher dimensions, 2*V (where V is the number of vertices of the border) test data points (V on points and V off points) must be selected for best results. For an equality border, twice as many off points, divided between the two sides of the border, must be selected. A thorough description of the domain testing strategy and its effectiveness is provided in[5]. Figure 3 shows the test data selected for path P_1 to satisfy the domain testing strategy. The only closed border of the path domain is (b-c=0), which has three vertices. The figure indicates whether each datum is an onpoint or an offpoint (above or below the equality border). Notice that several of the points selected reveal the missing path error, which is also detected by computation testing.

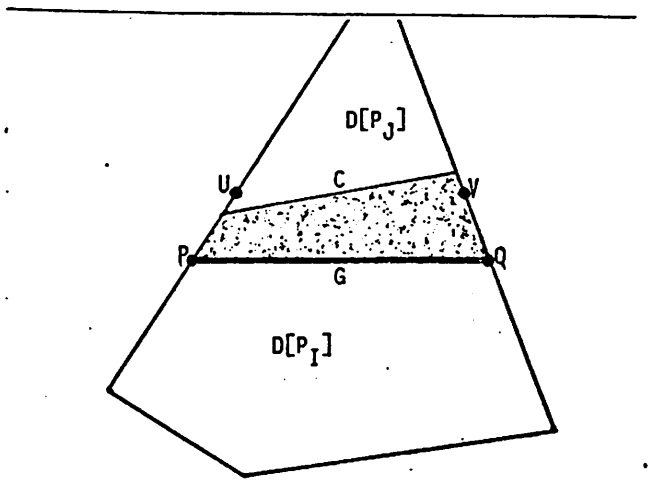

Figure 4. Border Shift Detected by Domain Testing

The basic domain testing strategy described is useful for testing path domain borders that involve both arithemetic manipulations and data manipulations in which the values of component selectors are known. Complications in applying the strategy arise when the values of component selectors depend on input values. Due to the dependencies among components of a compound structure and the component selectors, it may not be possible to find good on and off test points for a particular border. The intuitive concepts underlying domain testing can be used as heuristics to test the borders of a path domain. For instance, if a path domain border references a component of a compound structure with a selector of unknown value it is

important to test values both inside and just outside the domain for both the selector and the component. With the application of such heuristics, however, a bound on the error cannot be quantified.

The domain testing stategy subsumes both the boundary value testing and condition coverage guidelines proposed by Myers[15] as well as the extremal input values testing proposed by Howden[12]. Domain testing is a relatively new test data selection strategy for which much further research is needed. The strategy has been well defined for domains that are continuous, linear convex polyhedra. This assumes that the input space is continous and that none of the interpreted branch predicates contain a disjunction and all relational expressions are linear. Adequate modifications have been proposed for both nonconvex and discrete domains, although several problems remain to be addressed[5,20]. As yet, however, the strategy has only been sufficiently defined for linear borders. Modifications have been proposed that require the selection of on and off test points near each of the local minima and maxima of a nonlinear border. Unfortunately, the practical applicability of domain testing is limited to interpreted branch predicates of low degree.

An Integrated Approach

Combining the computation and domain testing strategies results in the selection of data that more rigorously test a path than other strategies proposed to date. There are two major drawbacks, however, associated with this approach. First, these strategies often produce a prohibitively excessive number of test points. Second, selecting data to satisfy these strategies is often a complex process. This section discusses the need to integrate these strategies so that the overall number of test points is reduced; and explores the possibility of providing automated support for the test data selection process.

When considering the number of test points associated with either strategy, it is important to note that many of the test data that satisfy one selection criterion also satisfy others. This overlap occurs within a strategy as well as between the two strategies. For instance, it commonly occurs that data chosen to test all the borders of a path domain also satisfy the polynomial testing criterion for a path computation. For path P_1 in TRIANGLE, many of the data points selected for domain testing satisfied computation testing criteria.

Although the guidelines outlined for both computation and domain testing are fairly well-defined, a systematic procedure for actually selecting the data needs to be derived. Such a procedure could be designed so as to maximize the number of criteria satisfied by each selected data point, thereby minimizing the total number of selected test points. It is improbable, however, that the cost of finding a minimal set of test data would be cost-effective in the long run. A heuristic approach, which exploits the overlap among the criteria in an attempt to reduce the number of selected points, should certainly be developed. The domain testing criteria are generally very explicit about the test data, whereas some of the computation criteria can be satisfied by a number of different data points. Thus, domain testing criteria should probably be satisfied before the computation testing criteria are applied.

Another consideration for reducing the number of test points is to provide various levels of testing. Only the highest level of testing would require that all the test data selection criteria be satisfied. Life critical software would utilize this testing level but less critical software could utilize lower levels. In developing these testing levels, some of the more costly criteria would only be associated with the higher testing levels. Moreover, some of the criteria could be weakened for the lower levels. For example, probabilistic arguments have been made for greatly reducing the number of test points that must be selected for polynomial testing without sacrificing too much reliability[7]. Similarly, a weaker version of domain testing, requiring considerably fewer test points, has also been evaluated[5]. The development of testing levels must consider the cost and cost benefits associated with each test data selection criteria. Moreover, these testing levels should also be associated with appropriate path selection strategies. It is not reasonable to pair a weak path selection criterion, such as statement coverage, with the highest level of test data selection.

Even if well defined procedures are available for satisfying a testing level, automated support for the testing process is required. Evidence supports this need, since even a weak testing criterion such as statement coverage is difficult to achieve without a tool to monitor program coverage. Symbolic evaluation tools provide a symbolic representation of the program, but automatic support for path selection and test data selection are needed as well. While these tasks can not always be completely automated, the need for human interaction can be minimized. Moreover, bookkeeping support tools are needed to keep track of all the information, such as stubs, drivers, input/output pairs, and test results, associated with a large testing endeavor. In some instances, specifications describing the expected output can be utilized so that the results from testing a program can be automatically verified. As is evident by the guidelines provided for computation and domain testing, reliable testing is a complex process. It is unrealistic to expect to achieve a reliable level of testing without providing programmers with appropriate evaluation and bookkeeping tools to support this process.

In addition to investigating the integration and automation of test data selection, some theoretical and experimental evaluations should be undertaken. While some of the test data selection criteria have been carefully investigated, others are only heuristics. Some criteria may subsume others and the interaction among some criteria is not well understood. As with polynomial testing, more reliable criteria can be developed for other well defined classes of computations, such as boolean expressions. Experimental studies evaluating the actual effectiveness of these strategies for detecting errors are also needed. Intuitively, it can be argued that by combining computation and domain testing, all classes of errors, including coincidental correctness, can almost be eliminated. Because of the large number of test data selected for a path and because these data points are scattered throughout the path domain, even missing path errors are unlikely. It would be beneficial to experimentally evaluate the different levels of testing so that reliability measures can be associated with each level. For example, the expected meantime between failures or expected ratio of remaining errors to statements would be useful statistics that would help managers choose the appropriate testing level for a program.

In sum, this paper provides a description of test data selection strategies aimed at detection of computation and domain errors. Combined they provide a strong basis for a reliable testing strategy. There still remain several unanswered questions on how to refine, integrate, automate, and evaluate the test data selection process.

References

1. R.S. Boyer, B. Elspas, and K.N. Levitt, "SELECT--A Formal System for Testing and Debugging Programs by Symbolic Execution," Proceedings of the International Conference on Reliable Software, April 1975, 234-244.
2. 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.
3. L.A. Clarke, "Automatic Test Data Selection Techniques," Infotech State of the Art Report on Software Testing, 2, September 1978, 43-64.
4. L.A. Clarke and D.J. Richardson, "Symbolic Evaluation Methods -- Implementations and Applications," Computer Program Testing, North-Holland Publishing Co., B.Chandrasekaran and S.Radicchi (eds.), 1981, 65-102.
5. L.A. Clarke, J. Hassell, and D.J. Richardson, "A Close Look at Domain Testing," IEEE Transactions on Software Engineering, SE-8, 4, July 1982, 380-390.
6. M. Davis, "Hilbert's Tenth Problem is Unsolvable," American Mathematics Monthly, 80, March 1973, 233-269.
7. R.A. DeMillo and R.J. Lipton, "A Probablistic Remark on Algebraic Program Testing," Information Processing Letters, 7, June 1978.
8. R.A. DeMillo, F.G. Sayward, and R.J. Lipton, "Program Mutation: A New Approach to Program Testing," State of the Art Report on Program Testing, 1979, Infotech International.
9. K.A. Foster, "Error Sensitive Test Case Analysis (ESTCA)," IEEE Transactions on Software Engineering, SE-6, 3, May 1980, 258-264.
10. W.E. Howden, "Methodology for the Generation of Program Test Data," IEEE Transactions on Computer, C-24,5, May 1975, 554-559.
11. W.E. Howden, "Algebraic Program Testing," ACTA Informatica, 10, 1978.
12. W.E. Howden, "Functional Program Testing," IEEE Transactions on Software Engineering, SE-6,2, March 1980, 162-169.
13. J.C. Huang, "An Approach to Program Testing," ACM Computing Surveys, 7,3, September 1975, 113-128.
14. J.W. Laski, "A Hierarchical Approach to Program Testing," Department of Systems Design, University of Waterloo, Waterloo, Ontario, Canada, Technical Report No.55CFW130779.
15. G.J. Myers, The Art of Software Testing, John Wiley & Sons, New York, New York, 1979.
16. S.C. Ntafos, "On Testing With Required Elements," Proceedings of COMPSAC '81, November 1981, 132-139.
17. S. Rapps and E.J. Weyuker, "Data Flow Analysis Techniques for Test Data Selection," Computer Science Department, New York University, New York, New York, Technical Report No.023, December 1981.
18. D.J. Richardson and L.A. Clarke, "A Partition Analysis Method to Increase Program Reliability," Fifth International Conferene on Software Engineering, March 1981, 244-253.
19. E.J. Weyuker, "An Error-Based Testing Strategy," Computer Science Department, New York University, New York, New York, Technical Report No.027, January 1981.
20. L.J. White and E.I. Cohen, "A Domain Strategy for Computer Program Testing," IEEE Transactions on Software Engineering, SE-6, May 1980, 247-257.
21. S.J. Zeil and L.J. White, "Sufficient Test Sets for Path Analysis Testing Strategies," Proceedings of the Fifth International Conference on Software Engineering, 1981, 184-191.