Technical Report # CS76007-R

Program Testing and Conditional Correctness

Dr. Johannes J. Martin
Virginia Polytechnic Institute
and State University

June, 1976

Abstract: It is shown that some beliefs about program testing are incorrect. A new notion of ocrrectness, conditional correctness, is defined. It is then shown that conditional correctness, which can in principle be achieved by testing, is not accomplished by such methods as "testing all branches" or "testing all paths". The latter method is proven to be not only insufficient but also highly redundant. Rules for establishing conditional correctness by testing are given and illustrated by an example.

Key Words and Phrases: program testing, program correctness, conditional correctness, testing all paths, testing all branches.

CR categories: 4.42, 5.24

Introduction

It is the purpose of this paper to demonstrate that some beliefs about program testing are incorrect and to report a few results and ideas that may eventually lead to a theory of program testing.

First, there is the conjecture that testing can only show the presence of errors but not their absence. We find this statement not only in articles that promote analytic methods of program verification [3] but also in publications concerned with program testing. Examples are papers by Huang [5] and Keirstead [6].

Here we will not dwell on a refutation of this conjecture in its general form. In [8] the author has proven that complete, finite test point sets exist and that the sizes of these sets are linearly related to the lengths of the programs tested.

Secondly, there is the assumption that testing must be exhaustive in order to be conclusive. We find two versions of exhaustiveness.

a) Exhaustive testing is testing a program at all points of its domain [3,4,5,7,9]. Assuming that testing must be exhaustive in this sense is obviously equivalent to accepting the conjecture that testing can not show the absence of errors.

b) Exhaustive testing is exhaustively examining the control flow of a program [1,2,4,5]. This requires testpoints that cause

execution to follow every possible path through the program, where a path is defined as a potential sequence of instructions executed from start to stop. If the program contains loops this may imply the necessity of infinitely many test points. We will demonstrate that this strategy is highly redundant in some sense but still insufficient for proving correctness.

Thirdly, it is assumed that testing all branches of a program (sometimes called thorough testing [5]) gives a high degree of assurance that the program is correct. We will show that a whole class of errors is not detected by this method.

Conditional correctness

Conventionally, a program is considered correct if it agrees with its specification. This definition requires a formal specification of the program's desired properties in order to make proving correctness possible.

Mostly, programs are written from quite informal problem specifications and the precise description of the program's behaviour is developed along with the development of the program itself. Also, specifications, if formalized, are subject to the same types of inaccuracies or gross blunders as programs. Furthermore, if the specifications are developed along with the program then it is quite possible that both the program and its formal specification contain the same errors as far as the original purpose of the program is concerned.

A formal verification procedure can clearly not detect this last type of error whereas tests performed at critical points would probably reveal the problem because test results can be more easily checked for consistency with the original informal description of the program than a formal specification.

We therefore suggest a new notion of correctness. We shall call it conditional correctness in contrast to strict correctness in the conventional sense. Note that our concept of conditional correctness, which will be well defined, is not based on or related to the informal use of the term by Kopetz [7].

It follows from the results described by the author in [8] that, in principle, conditional correctness can be proven by testing. Moreover, under certain conditions conditional correctness is equivalent to strict correctness.

Definition:

Given a well defined set P of programs or other formal descriptions that define a set F(P) of functions, we say a program p in P is conditionally correct over P with respect to some given function g if, with the assumption that g is an element of F(P), we can prove that p computes (describes) g.

Note that g is not required to be in F(P). It is only required that there exists a proof for "p computes g" if g in F(P) is used as a premise.

Clearly, conditional correctness becomes strict correctness if both the program and its specification are members of the set P.

A special case arises if the set P contains all finite programs and descrptions. Here conditional correctness is a priori strict correctness if, indeed, there exists a formal specification s for the given program p because both s and p are by definition members of P. Here testing for conditional correctness would require the exhaustive examination of all points x in the domain of p since F(P) contains all functions that can be specified and, therefore, for any point x we must expect at least two functions that agree everywhere except at x.

However, it has been shown in the paper mentioned above [8] that for finite sets, P, proportional to $\log|P|$ many testpoints suffice to prove correctness. Note that such a finite set P that contains both the program p and its specification s can easily be constructed:

$$P = \{ \text{ all descriptions } q \text{ such that } |q| \leq MAX(|p|,|s|) \} .$$

Since conditional correctness can be demonstrated by testing we can give it an interpretation that is independent of the function g that the tested program is supposed to compute: if the set $T=\{x_i,y_i\}$ of test points has been used to prove p conditionally correct over some set P, then we know that every program q in P that satisfies all points in T will compute the same function as p. Conversely, by specifying P and a complete set T of test points we uniquely specify a function. We might therefore consider the development of automatic methods for constructing computer programs from pairs (P,T).

Conventional testing rules and conditional correctness

Proponents of conventional testing rules such as testing all branches or testing all paths never claim that these rules assure correctness in the strict sense. We will now examine if or to what extent they achieve conditional correctness.

In order to do so we first have to choose a set P over which conditional correctness is to be proven. This is a rather important step; the choice of P determines how meaningful the test results are going to be. The more we restrict P the fewer test points we are going to need. For example, in the trivial special case P={p} we may rightly claim that p is conditionally correct over P without any test. On the other hand, the more we enlarge P the more assurance we are going to have that g is in F(P). Also, if we define P by defining the properties of its elements (rather than by enumeration) we should make sure that there exists a simple procedure that decides if p is in P. Simplicity of the definition of P may also help a later attempt to prove that g is in F(P). Thus, we should choose P such that we can be confident that g is in F(P) and sure that p is in P.

We will pursue this goal by choosing the set of all programs that have the same control schemata and use the same types of functions in their branches as the given program p (e.g. rational functions with not more than k constant coefficients for some value k).

This choice will, for example, give us perfect assurance if we want to verify that an algorithm that is known to be correct has been properly implemented.

Straight line programs

We will start with a simple straight line program. Such a program consists of only one branch and there is only one path that control can follow. By conventional rules, a single test point should be sufficient to give us a good assurance of correctness.

For the sake of simplicity, we assume that only primitives for real arithmetic operating with unlimited accuracy are provided by the programming language used. Therefore, every straight line program will compute a rational function.

As the set P we choose the set of all programs that

(i) are applicable to the same domain X and map into the same range Y as the given program p,

(ii) use the same number k of constant coefficients different from 0 and 1 as p, and

(iii) are limited in length by some number $L \geq |p|$.

Given p, these properties of P can be determined by a rather simple auotmatic scanning mechanism ensuring p in P. Now, with the assumption that some subset Q of P computes the desired function g we want to demonstrate that p is in Q.

It is obvious that a single test point can not ensure p in Q because the programs q in P compute all kinds of rational functions each depending on k parameters. Given some point t=(x,y) we will

be able to find many different functions that satisfy this point. However, with k points at almost arbitrary locations (except for a set of "bad" test configurations whose measure is zero) we will obtain k independent conditions for k parameters and thus get a unique solution or, if the conditions turn out to be nonlinear, a finite set of solutions for the parameters. Now the number of functions computable by members of P is finite because the number of structurally different expressions is finite for their lengths are bounded by L. Since the set of common points of any two distinct rational functions is of measure zero there must exist points (in fact most points will qualify) where the desired function differs from all those other functions that are still candidates after k tests.

Therefore, k+1 tests are (usually necessary and) sufficient to ensure conditional correctness.

Although the probability of choosing a bad test point is extremely small and decreases with increasing accuracy in the limit down to zero, a systematic method for avoiding bad choices would be desirable. Since, at present, we have no such method the example discussed further down falls short of being a correctness proof.

If one is only after a procedure that ensures conditional correctness with a high <u>probability</u> rather than certainty one single test point does not perform too badly. Suppose we are given a correct straight line program p and some randomly chosen point x in the domain of p. Certainly, almost all changes (i.e. errors) that could be introduced to p will change the behavior of F(p) at x.

IF - THEN - ELSE segments

The severe shortcomings of conventional test mehtods become apparent if these methods are applied to conditional statements. Consider the example

```
p:  IF x1 - 2*x2 + 3 ≧ 0
    THEN (straight line code p1) ELSE (straight line code p2);
```

Here the set P contains all program of the form

```
IF f(x1,x2) ≧ 0 THEN p1 ELSE p2 ;
```

with f(x1,x2) being rational over x1 and x2 and containing not more than two constant coefficients. Since the program consists of two branches (and two paths) conventional rules of testing suggest two test points, one for each branch.

As conceded before, this procedure will test the segments p1 and p2 reasonably well (for conclusive testing we would have to use k1 + k2 + 2 testpoints according to the previous argument) but it will examine the condition itself only superficially. Suppose the two test points chosen are u = (0,0) and v = (-2,1) where u and v are vectors with x1 and x2 as components. Then any conditional expression of the form f(x1,x2) = x1 - A*x2 + B ≧ 0 with A ≧ 0 and B > A - 2 would yield the same test results and, in addition, there is a large number of structurally different conditional expressions that would equally work.

Searching for a cure we notice that a condition f(x) ≧ 0 partitions the domain into two regions separated by the boundary

$f(x) = 0$. This boundary is, in general, an algebraic curve the course of which must be verified and, hence, tested similarly to a straight line program namely by $k + 1$ tests where k is again the number of constant parameters that occur in $f(x)$. Each test that is meant to ensure that a particular point is part of the boundary must actually consist of a pair of test points located on both sides of and as close as possible to the boundary.

In the above example, three point pairs are sufficient. These could be located at (-1,1),(-1,1+e); (-3,0),(-3,e); (0,1.5),(0,1.5+e) for some positive infinitesimal e. Note that the test points are also located in the domains of the two segments p1 and p2 and may therefore serve as test points for these segments as well.

A slight complication occurs if the function $f(x)$ of the condition $f(x) \geqq 0$ can be factored. In this case the sufficient number of tests is $k + q$ where q is the number of irreducible factors of $f(x)$ since we might be forced to verify every factor individually. The value of q is, of course, bounded by the degree of f.

Sometimes, p1 and p2 may compute identical values at both sides of a boundary (e.g. in splining applications). Here the program must be "instrumented" by additional test variables that will enable us to decide which path is actually being taken upon testing.

Redundancy of testing all paths

In order to show that testing all paths may involve redundant tests we consider the segment

```
IF A THEN y:=g1(x) ELSE y:=g2(x);
IF B THEN y:=f1(y) ELSE y:=f2(y);
```

There are four different paths through the program which compute

i) $h1 = f1(g1(x))$

ii) $h2 = f1(g2(x))$

iii) $h3 = f2(g1(x))$

iv) $h4 = f2(g2(x))$

A simple calculation leads to

(1) $h4 = f2(g2(x)) = h3(h1^{-1}(h2(x))) = f2(g1(g1^{-1}(f1^{-1}(f1(g2(x))))))$

Thus if the functions h1, h2 and h3 are verified h4 is determined provided that h1 is uniquely invertable over at least some small region in its domain. Moreover, a closer inspection of (1) shows that only f1 needs to be invertable over some part of its domain because $g1(g1^{-1}(z)) = z$ for all possible inverses $g1^{-1}$ as long as z is in the range of g1.

The argument can be extended in a straight forward way to multiple branches (CASE statements) on each level. If there are n branches on the first level and m on the second then the total number of paths is n.m; however, the number of paths that need to be tested is n+m-1 provided that at least one of the functions fi on the second level is uniquely invertable for some region of its domain.

Furthermore, if i levels ($i > 2$) follow each other each having n_j branches ($1 \leq j \leq i$) then the number of paths that need to be tested turns out to be

$$\left(\sum_{j=1}^{i} n_j \right) + 1 - i$$

rather than $\prod_{j=1}^{i} n_j$ as testing <u>all</u> paths would require.

We conclude that conventional testing rules are indeed both insufficient and wasteful.

In the next section we will show how the concept of conditional correctness eases the problems connected with testing loop constructs.

Conditional correctness and loops

Consider the set P that contains all programs of the form

WHILE $f(x,y) \geq 0$ DO y:= g(x,y) END

and no others, and some given program p in P. Clearly, p is proven conditionally correct over P if both f and g are determined.

In order to see how these two functions can be verified by testing we will distinguish two cases, a simple special case and the general one. Both times we will assume that f and g are rational or otherwise testable i.e. IF-THEN-ELSE or other loop constructs. We also assume that a counter i has been added to the loop; thus the augmented segment has the form

i:= 0; WHILE $f(x,y) \geq 0$ DO y:= g(x,y); i:= i+1 END

Such a loop partitions the domain (X,Y) into partitions $Z_0 \ldots Z_j$ such that for all points (x,y) in Z_j the loop takes j turns. Further, if we aim to place a test point z into the partition Z_j then the value of the counter i will tell us after the test whether we were successful.

Case A

Testing proceeds precisely as it would for straight line programs and conditional constructs if both x and y can be selected freely on entry to the loop and if all components of y are part of the range of the function that is to be computed by the loop. In this case, all test points for g can be placed into the region Z_1; f can be tested at the boundary between Z_0 and Z_1.

Case B

In reality, we will rarely deal with case A since

i) some components of x and y are usually fixed on entry to a loop by initialization and

ii) some components of y represent frequently intermediate results that do not occur in the range of the desired function. For example, increments typically belong into this category.

In both cases it is unlikely that all test points can be placed into Z_0 or Z_1. Searching for the proper test points we may therefore have to consider functions of the form

$f(x,g^i(x,y)) \geq 0$ and $y = g^i(x,y)$ with

$g^i(x,y) = y$ for $i=0$ and $g^i(x,y) = g(x,g^{i-1}(x,y))$ for $i \geq 1$.

It follows from the definition of g^i that g is determined if g^i is verified for two consecutive values of i.

An example

The following program computes the inverse y of $x = y^3$ where $e^3 < x < (x_0-e)^3$ for some positive limit x_0 and some small positive value e. The method used is a binary search.

```
p:=3; e:=1.e-6; inv:=false; [i:=0];
if  x < (x0-e)**p and x < e**p  then
     begin  low:=0; hi:=x0;
          repeat  y:=.5*(hi+low); [i:=i+1];
               case
                    (y-e)**p ≧ x:  hi:=y
                    (y+e)**p ≦ x:  low:=y
                    else         :  inv:=true
          until  inv
     end
```

The domain of the program is the plane defined by x_0 and x, and the range is defined by y, inv, and the auxiliary variable i. Simplifying our discussion we introduce the variable x' such that $x = (x')^3$.

The set P over which we will test for conditional correctness will consist of all programs of the form

p1; if f1 then begin p2; repeat p3; case f2:p4; f3:p5; else:p6
until f5 end

where all p's and f's are straight line programs or conditions of not more constant coefficients than the corresponding segments in the

given program.

Figure 1 shows the regions of the domain that correspond to the different numbers of iterations needed.

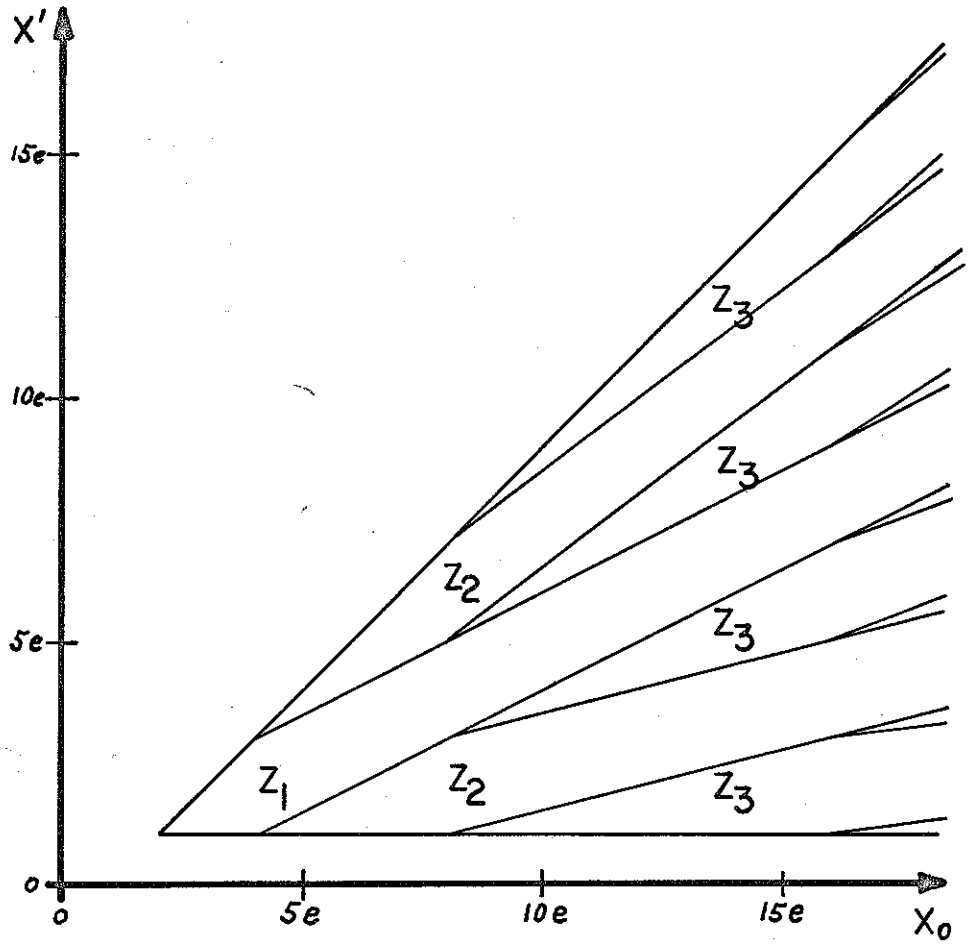

FIGURE 1 Partition of the domain

Testing will proceed as follows

a) Since the first <u>if</u> splits the plane (x_0,x') into two regions separated by the straight line boundaries $x' = x_0-e$ and $x' = e$ which depend on the two parameters $e = 1.e-6$ and $p = 3$, we will need four test point pairs.

The pairs

x_0:	3	3	;	1.7	1.7	;	4e	4e
x':	3-e	3-e-ε	;	1.7-e	1.7-e-ε	;	3e	3e-ε

may be used for testing the boundary $x' = x_0-e$ as well as the constants assigned to e and p. The value of ε is positive and as small as the computational resolution permits. The pair

x_0:	3e	3e
x':	e	e+ε

may serve to verify $x' = e$.

b) We will test the assignment y:=.5*(hi+low) and the two initialization statements low:=0 and hi:=x0 by three points in the region Z_1 (first iteration). We use three points because we want to check the two constants: .5 and 0. Since two of the above test points are already in Z_1 we will only need one more at, say, $x_0 = 2.9$, $x' = 1.45-e-\varepsilon$.

c) Finally, we will need two more pairs of test points around the boundaries $y-e = x'$ and $y+e = x'$. Since the previous test point may serve as a constituent for the pair that is to check $y-e = x'$, the other constituent becomes $x_0 = 2.9$, $x' = 1.45-e$. This point is located in some region Z_j such that $2^j*e \leq 2.9 \leq 2^{j+1}*e$. The expected result for y at this point is 1.45-2e. The pair for testing $y+e = x'$ is constructed accordingly.

Conditional and strict correctness

There is one particularly important question that needs to be resolved if testing is to be used to prove strict correctness.

Suppose we have two sets P1 and P2 of programs (descriptions) such that the program p in P1 is the program to be verified and the description s in P2 is its specification. Further suppose we have test point sets T1 and T2 that prove p and s conditionally correct over, respectively, P1 and P2. Since it follows from the definition of conditional correctness that p is equivalent to s if both are conditionally correct over the union of P1 and P2 with respect to the same complete test point set T we would like to compute T, if possible, from T1 and T2.

We can easily show that, in general, T is not the union of T1 and T2. Consider the two sets $P1 = \{p\}$ and $P2 = \{s\}$. Clearly, T1 and T2 are empty. However, the set T for $\{p,s\}$ can not be empty.

On the other hand, if P1 contains P2 (or vice versa) then the set T sufficient for testing over $P1 \cup P2$ is T1 (T2) since P1 (P2) contains both p and s.

Conclusion and outlook

We believe that the notion of conditional correctness will serve as a useful foundation for further theoretical and practical work in the area of program testing. Since the conventional method of testing all branches is insufficient and the method of testing all paths is both insufficient and wasteful it seems to be very desirable to develop the principles outlined in this paper into a complete theory and, thereafter, practical method of testing.

We should hope that for certain classes of desciptions automatic procedures for test point generation can be found that are less complex and time consuming than systems for the automatic proving of program correctness by purely analytic means.

However, a general procedure for test point selection can clearly not exist because of the well known decidability problems.

References

1. Boehm,B. Software and its impact: a quantitative assessment. Datamation 19 (May 1973), 48-59.

2. Brown,J.R. and Lipow,M. Testing for software reliability. Sigplan Notices 10-6 (June 1975) 518-527

3. Dahl,O.-J. et Al. Stuctured programming. Academic Press, 1972

4. Goodenough,J.B. and Gerhart,S.L. Toward a theory of test data selection. Sigplan Notices 10-6 (June 1975) 493-510

5. Huang,J.C. An approach to program testing. ACM Computing Surveys 7-3 (Sept.1975) 113-128

6. Keirstead,R.E. On the feasibility of software certification. Stanford Research Inst. PB-245 213 NTIS report June 1975

7. Kopetz,H. On the connections between range of variable and control structure testing. Sigplan Notices 10-6 (June 1975) 511-517

8. Martin,J.J. Finiteness and bounds of complete test point sets for program verification. VPI&SU tech. report CS76003R

9. Poole,P.C. Debugging and testing. In Bauer,F.L. Advanced course on software engineering, Springer Verlag New York 1973, 278-318