A Theory of Fault Based Testing

Larry J. Morell

Presented by: Joonghoon Lee
A Reliable Test?

- A test whose success implies **Program Correctness**

- Unattainable in general
Quality Measures

• Desirable to have gradations of ‘goodness’
  – ‘Reliable Test’ being the ultimate
• Structural Coverage Measures
do not imply correctness
• Maximize the number of faults eliminated
  – Hopefully, eliminating all faults
Fault-Based Testing

- Determine the absence of pre-specified faults

- Based on the number of faults eliminated
A Different Perspective

• Traditional point of view
  – A test that does not find an error is useless

• Fault-Based Testing
  – Every correct program execution contains information that proves the program could not have contained particular faults
Program Verification Continuum

• Formal Verification
  – Absolute Correctness can be achieved

• **Fault-Based Testing**
  – Assume that an alternate sufficient arena is available
  – Certain faults are shown to be eliminated

• Structural Coverage
Basic Framework

- \( <P, S, D> \): Arena
- P: Program
- S: Specification
- D: Domain, source of test data
Framework

- $[P]$: Program function (input, output)
- $[P](x)\downarrow$: P halts on input $x$
- $[P](x)\uparrow$: doesn’t
- $\text{dom}([P])$: All points for which P halts
Successful Test Case

• For an arena G = <P, S, D>,

• x∈D is successful \( \text{iff} \)
\([P](x)\downarrow\) and \((x, [P](x)) \in [S]\)
Failure Sets

• The set of all failure points for $G$ is the **Failure set** of $G$.

• A Program $P$ is correct with respect to $S$ if and only if $P$’s failure set is empty.

• Failures sets are not always recursively enumerable
  – Must restrict failure set
Fault-Based Arena

- \(<P, S, D, L, A>\)
- P: Program
- S: Specification
- D: Domain, source of test data
- L: Locations in P
- A: alternative set associated with locations
Test Data

• In Fault-Based Testing, test data distinguishes the original program from its alternate programs.

• \( x \) distinguishes \( P \) from \( R \) iff

\[
\text{For a Program } P \text{ and } x \in \text{dom}([P]) \\
<P>(x) \neq <R>(x)
\]
Alternate Sufficient

- A fault based arena which contains a correct program is alternate sufficient.

- It is undecidable whether or not an arbitrary fault based arena is alternate sufficient.
Symbolic Testing

• A fault based testing strategy

• Symbolic execution
  – Use symbolic input
  – model infinitely many executions with single symbolic execution
  – 2+3, 3+3, 5+3, 7+3… => X + 3
read(x,y)
x: = x * y + 3  ➔ let’s try to ensure that no mistake was made in
write (x * 2)  in selecting the constant 3.

read(x,y)
x: = x * y + F  ➔ use F to denote infinitely many alternate programs
write (x * 2)

read(5,6)  ➔ pick  x : 5, y : 6
x: = 5 * 6 + F
write ( 30 + F ) * 2  ➔ F was propagated through the program,
                     ultimately appearing in the output

Say original program computes 66.
⇒ (30 + F)*2 = 66
Therefore, for F, no other constant than 3 will go undetected
Thus,

• \{ (5,6) \} distinguishes P from $P_E$

( $P_E$ contains all alternate programs produced by substituting any constant for 3 in P )
Another Example 1

Program ComputeArea(input,output);
vara,b,incr,area,v:real;
begin
1 read (a,b,incr); {incr>0}
2 v:=a*a+1
3 area := 0 => area := F
4 while a+incr<= b do begin
5    area := area + v*incr;
6    a := a+incr;
7    v := a*a+1;
   end
8 incr := b – a;
9 if incr>= 0 then begin
10    area := area + v*incr;
11    write(‘area by rectangular method:’,area)
   end else
12 write( ‘illegal values for a=’,a, ‘and b=’, b)
end.

Symbolic input a:A,b:B,incr:I
Assuming B>=A and A+I>B
(skipping loop)
Result is,
(A*A+1)*(B-A)
Introduce an assignment fault in 3:
area := F
Provides,
F+(A*A+1)*(B-A)
form a general propagation equation,
Thus,
F = 0
Another Example 2

Program ComputeArea(input, output);
vara, b, incr, area, v: real;
begin
1 read (a, b, incr); {incr > 0}
2 v := a * a + 1
3 area := 0
4 while a + incr <= b do begin
5    area := area + v * incr; => area := F
6    a := a + incr;
7    v := a * a + 1;
   end
8 incr := b – a;
9 if incr >= 0 then begin
10       area := area + v * incr;
11       write(‘area by rectangular method:’, area)
   end else
12 write(‘illegal values for a=’, a, ‘and b=’, b)
end.

Symbolic input a:A, b:B, incr:N

Assuming A + N <= B and A + 2N > B

(1 iteration)

Result is,

\[(A^2+1)N + [(A+N)^2+1](B-A-N)\]

Introduce an assignment fault in 5:

area := F

Provides

F + [(A+N)^2+1](B-A-N)

form a general propagation equation,

\[(A^2+1)N + [(A+N)^2+1](B-A-N)\]

Thus,

F = (A^2+1)N

A^2+1 > 0, N > 0

No clear constant substitution possible

Fault Equation.
Domain Dependent Transformations

- Domain Independent
  If $x = 1$ then $y := 1$ else $y := x^x$

- Domain Dependent
  If $x = F$ then $y := 1$ else $y := x^x$

Makes testing difficult!
• “Looking for errors”
  – misleading in two ways:
    • What errors should we find?
    • Unattainable

• Fault-Based Testing
  – <P, S, D, L, A>

• Symbolic Testing
  – Use symbolic input to represent all inputs which follow a given path