Mutation Testing

Main Idea

- How to evaluate the effectiveness of a testing technique or tool?
  - Both subject programs and faults need to be representative
- Mutants are created to mimic program mistakes or faults in a systematic manner
  - Each mutant is a slight variation of the original program
  - A test kills one mutant if it could distinguish the mutant from the original program.

Main Idea (2)

- Assume that we are trying to evaluate two testing techniques, A and B.
- For each subject program, create a set of mutants.
- Check how many mutants A and B could kill.
- The more mutants a technique kills, the more effective.

Basic Concepts

- Mutation Operator: a rule that specifies a slight variation of the original program.
- Mutant: The result of one application of a mutant operator
  - Stillborn mutants: syntactically illegal and caught by a compiler
  - Trivial mutants: can be killed by almost any test case
  - Equivalent mutants: functionally equivalent to the original program (and thus cannot be killed by any test case)

Example

```
int Min (int A, int B) {
   int minVal;
   minVal = A;
   if (B < A) {
      minVal = B;
   }
   return (minVal);
} // end Min
```

```
int Min (int A, int B) {
   int minVal;
   minVal = A;
*  minVal = B;
   if (B < A) {
*  if (B > A) {
*  if (B < minVal)
      minVal = B;
*     Bomb ();
*     minVal = A;
   }
   return (minVal);
} // end Min
```

Mutation Score

- Given a mutant m of a program d, a test is said to kill the mutant if and only if this test produces a different output on m than on d.
- Mutation score: the percentage of mutants that are killed (over all possible mutants)

Strongly Kill

- Given a mutant m for a program P and a test t, t is said to strongly kill m if and only if the output of t on P is different from the output of t on m.

Example

- Consider the first mutant
  - Reachability: true
  - Infection: A != B
  - Propagation: (B < A) = false
  - Full Test Spec: B > A

```
int Min (int A, int B) {
   int minVal;
   minVal = A;
   if (B < A) {
      minVal = B;
   }
   return (minVal);
} // end Min
```

```
int Min (int A, int B) {
   int minVal;
   minVal = B;
   if (B < A) {
      minVal = B;
   }
   return (minVal);
} // end Min
```

Weakly Kill

- Given a mutant m that modifies a location l in a program P, and a test t, t is said to weakly kill m if and only if the state of the execution of P on t is different from that of m immediately after l.

Example

```
boolean isEven (int x) {
   if (x < 0)
      x = 0 – x; // change to x = 0;
   if (float) (x / 2) == ((float) x) / 2.0
      return true;
   else
      return false;
}
```

Reachability: x < 0
Infection: x != 0
Propagation: x must be odd

Absolute Value Insertion

- Each arithmetic expression is modified by functions abs(), and negAbs().
- Example: x = 3 * a => x = 3 * abs(a), x = 3 * -abs(a)

Arithmetic Operator Replacement

- Each occurrence of one of the arithmetic operators +, -, *, /, **, and % is replace by each of the other operators, and special operators leftOp, rightOp, and mod.
- Example: x = a + b => x = a – b, x = a * b, x = a / b, x = a ** b, x = a, x = b, x = a % b

Relational Operator Replacement

- Each occurrence of one of the relational operators (<, <=, >, >=, =, !=) is replaced by each of the other operators and by falseOp and trueOp.
- Example: if (m > n) => if (m >= n), if (m < n), if (m <= n), if (m == n), if (m != n), if (false), if(true)

Conditional Operator Replacement

- Each occurrence of each logical operator (&&, ||, &, |, ^) is replaced by each of the other operators, and falseOp, trueOp, leftOp, and rightOp.
- Example: if (a && b) => if (a || b), if (a & b), if (a | b), if (a ^ b), if (false), if (true), if (a), if (b)

Shift Operator Replacement

- Each occurrence of one of the shift operators (<<, >>, and >>>) is replaced by each of the other operators, and the special operator leftOp.
- Example: x = m << a => x = m >> a, x = m >>> a, x = m

Logical Operator Replacement

- Each occurrence of each bitwise logical operator (&, |, and ^) is replaced by each of the other operators, and leftOp and rightOp.
- Example: x = m & n => x = m | n, x = m ^ n, x = m, x = n

Assignment Operator

- Each occurrence of one of the assignment operators (+=, -=, *=, /=, %=, &=, !=, *=, <<==, >>=, >>>=) is replaced by each of the other operators.
- Example: x += 3 => x -= 3, x *= 3, x /= 3, x %= 3, …

Unary Operator Insertion

- Each unary operator (+, -, !, ~) is inserted before each expression of the correct type.
- Example: x = 3 * a => x = 3 * +a, x = 3 * -a, x = +3 * a, x = -3 * a

Unary Operator Deletion

- Each unary operator (+, -, !, ~) is deleted.
- Example: if !(a > - b) => if (a > -b), if !(a > b)

Scalar Variable Replacement

- Each variable reference is replaced by every other variable of the appropriate type that is declared in the current scope.
- Example: x = a * b => x = a * a, a = a * b, x = x * b, x = a * x, x = b * b, b = a * b

Bomb Statement Replacement

- Each statement is replaced by a special Bomb() function
- Example: x = a * b => Bomb ()

Summary

- Mutation testing is mainly used to evaluate the effectiveness of testing techniques.
- Mutants are created to mimic programming mistakes.
- The higher the mutation score, the more effective a testing technique.
- The main challenge of mutation testing is dealing with a potentially huge number of mutants.