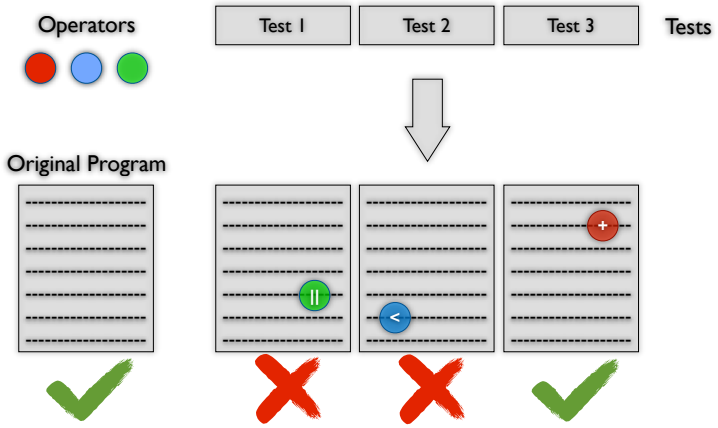
The Oracle Problem

- Executing all the code is not enough
- We need to check the functional behavior
- Does this thing actually do what we want?
- Automated oracles can be spec, model
- Else, manual oracles have to be defined

1

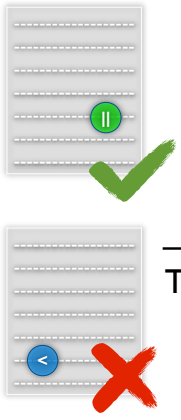
Coverage misses one important aspect: The Oracle Problem. A test oracle is the entity that decides whether a test case passed or failed.

Mutation Testing



2

The mutation testing process: Mutants are generated from a program by applying different mutation operators. The test cases are executed against the program and each of its mutants. If a mutant passes all tests, it is live. If it fails a test, it is killed.



Mutation Score:

$$\frac{\text{Killed Mutants}}{\text{Total Mutants - Equivalent mutants}}$$

3

The mutation score is used to quantify how good a test suite is at detecting the faults represented by the mutants. We only count non-equivalent mutants, otherwise 100% mutation score would not be possible.

Mutation testing focuses on First Order Mutants

Competent Programmer Hypothesis

Coupling Effect

4

Because of the competent programmer hypothesis and coupling effect mutation testing in general only considers first order mutants.

Improvements

Do fewer

- Mutant sampling
- Selective mutation

Do smarter

- Parallelize
- Weak mutation
- Use coverage
- Impact

Do faster

- Mutate bytecode
- Mutant schemata

5

Mutation testing is costly - each mutation operator results in many different mutants. Each of the mutants needs to be compiled, and all tests potentially have to be executed against every mutant. There are several improvements over the basic approach to reduce the overall costs.

```
int trian(int a, int b, int c) {
  if(a<=0||b<=0||c<=0)
    return INVALID;
  int trian = 0;
  if(a==b) trian = trian+1;
  if(a==c) trian = trian+2;
  if(b==c) trian = trian+3;
  if (trian == 0)
    if (a + b < c || a + c < b || b + c < a)
      return INVALID;
    else
      return SCALENE;
  if (trian > 3)
    return EQUILATERAL;
  if(trian==1 && a+b>c)
    return ISOSCELES ;
  else if(trian==2 && a+c>b)
    return ISOSCELES;
  elseif(trian==3 && b+c>a)
    return ISOSCELES;

  return INVALID;
}
```

a==c && a+b>c && a+c<=b

b==c && a+b>a && b+c<=a

a==b && a+b>c

a==b && a+b<=c

```
if(trian >  1 && a + b > c)

if(trian == 1 && a + b <= c)

if(trian >  1 && a + b <= c)
```


6

Here is an example of a strongly subsuming HOM on a different implementation of the triangle program. Only test cases that satisfy the constraints in the intersection of the constraints of the FOMs can kill the HOM.