Mutation Analysis

Security Testing
Rahul Gopinath, Saarland University
Story so far
Story so far

Fuzzing
Story so far

- Blackbox

Fuzzing

- White box
Story so far

- Blackbox
  - Unstructured input
  - Input coverage guided

Fuzzing

- White box
  - Search based input
  - Symbolic execution
  - Grammar inference (inclusion)
  - Improved grammar inference with taints!
How good are my tests?
Coverage!
def triangle(a, b, c):
    if a == b:
        if a == c:
            Equilateral
        else:
            Isosceles
    else:
        if b != c:
            if a == c:
                Isosceles
            else:
                Scalene
        else:
            Isosceles
def triangle(a, b, c):
    if a == b:
        if a == c:
            Equilateral
        else:
            Isosceles
    else:
        if b != c:
            if a == c:
                Isosceles
            else:
                Scalene
        else:
            Isosceles

assert triangle(1, 1, 1) = Equilateral
def triangle(a, b, c):
    if a == b:
        if a == c:
            Equilateral
        else:
            Isosceles
    else:
        if b != c:
            if a == c:
                Isosceles
            else:
                Scalene
        else:
            Isosceles

assert triangle(1, 1, 1) = Equilateral
assert triangle(1, 2, 1) = Isosceles
def triangle(a, b, c):
    if a == b:
        if a == c:
            Equilateral
        else:
            Isosceles
    else:
        if b != c:
            if a == c:
                Isosceles
            else:
                Scalene
        else:
            Isosceles

assert triangle(1, 1, 1) = Equilateral
assert triangle(1, 2, 1) = Isosceles
assert triangle(1, 1, 2) = Isosceles
def triangle(a, b, c):
    if a == b:
        if a == c:
            Equilateral
        else:
            Isosceles
    else:
        if b != c:
            if a == c:
                Isosceles
            else:
                Scalene
        else:
            Isosceles

assert triangle(1, 1, 1) == Equilateral
assert triangle(1, 2, 1) == Isosceles
assert triangle(1, 1, 2) == Isosceles
assert triangle(1, 2, 2) == Isosceles
def triangle(a,b,c):
    if a == b:
        if a == c:
            Equilateral
        else:
            Isosceles
    else:
        if b != c:
            if a == c:
                Isosceles
            else:
                Scalene
        else:
            Isosceles

assert triangle(1,1,1) = Equilateral
assert triangle(1,2,1) = Isosceles
assert triangle(1,1,2) = Isosceles
assert triangle(1,2,2) = Isosceles
assert triangle(1,2,3) = Scalene
def triangle(a, b, c):
    if a == b:
        if a == c:
            Equilateral
        else:
            Isosceles
    else:
        if b != c:
            if a == c:
                Isosceles
            else:
                Scalene
        else:
            Isosceles

assert triangle(1, 1, 1) = Equilateral
assert triangle(1, 2, 1) = Isosceles
assert triangle(1, 1, 2) = Isosceles
assert triangle(1, 2, 2) = Isosceles
assert triangle(1, 2, 3) = Scalene

(Statement) Coverage = 100%!
def triangle(a, b, c):
    if a == b:
        if a == c:
            Equilateral
        else:
            Isosceles
    else:
        if b != c:
            if a == c:
                Isosceles
            else:
                Scalene
        else:
            Isosceles

assert triangle(1, 1, 1) == Equilateral
assert triangle(1, 2, 1) == Isosceles
assert triangle(1, 1, 2) == Isosceles
assert triangle(1, 2, 2) == Isosceles
assert triangle(1, 2, 3) == Scalene

Branch Coverage = 100%!
(Statement) Coverage = 100%!
def triangle(a, b, c):
    if a == b:
        if a == c:
            print('Equilateral')
        else:
            print('Isosceles')
    else:
        if b != c:
            if a == c:
                print('Isosceles')
            else:
                print('Scalene')
        else:
            print('Isosceles')

assert triangle(1, 1, 1) == 'Equilateral'
assert triangle(1, 2, 1) == 'Isosceles'
assert triangle(1, 1, 2) == 'Isosceles'
assert triangle(1, 2, 2) == 'Isosceles'
assert triangle(1, 2, 3) == 'Scalene'

(Path) Coverage = 100%!
(Statement) Coverage = 100%!
(Branch) Coverage = 100%!
def triangle(a, b, c):
    if a == b:
        if a == c:
            Equilateral
        else:
            Isosceles
    else:
        if b != c:
            if a == c:
                Isosceles
            else:
                Scalene
        else:
            Isosceles

assert triangle(1, 1, 1) == Equilateral
assert triangle(1, 2, 1) == Isosceles
assert triangle(1, 1, 2) == Isosceles
assert triangle(1, 2, 2) == Isosceles
assert triangle(1, 2, 3) == Scalene

What are we missing?
def triangle(a, b, c):
    if a == b:
        if a == c:
            Equilateral
        else:
            Isosceles
    else:
        if b != c:
            if a == c:
                Isosceles
            else:
                Scalene
        else:
            Isosceles

assert triangle(1, 1, 1) == Equilateral
assert triangle(1, 2, 1) != Equilateral
assert triangle(1, 1, 2) != Equilateral
assert triangle(1, 2, 2) != Equilateral
assert triangle(1, 2, 3) != Equilateral
def triangle(a, b, c):
    if a == b:
        if a == c:
            Equilateral
        else:
            Isosceles
    else:
        if b != c:
            if a == c:
                Isosceles
            else:
                Scalene
        else:
            Isosceles

assert triangle(1,1,1) == Equilateral
assert triangle(1,2,1) != Equilateral
assert triangle(1,1,2) != Equilateral
assert triangle(1,2,2) != Equilateral
assert triangle(1,2,3) != Equilateral

Evaluate Assertions!
def triangle(a, b, c):
    if a == b:
        if a == c:
            Equilateral
        else:
            Isosceles
    else:
        if b != c:
            if a == c:
                Isosceles
            else:
                Scalene
        else:
            Isosceles

assert triangle(1, 1, 1) == Equilateral
assert triangle(1, 2, 1) != Equilateral
assert triangle(1, 1, 2) != Equilateral
assert triangle(1, 2, 2) != Equilateral
assert triangle(1, 2, 3) != Equilateral

Evaluate Assertions!

Oracle Strength
Fault Seeding
def triangle(a, b, c):
    if a == b:
        if a == c:
            Equilateral
        else:
            Isosceles
    else:
        if b != c:
            if a == c:
                Isosceles
            else:
                Scalene
        else:
            Isosceles

assert triangle(1, 1, 1) = Equilateral
assert triangle(1, 2, 1) != Equilateral
assert triangle(1, 1, 2) != Equilateral
assert triangle(1, 2, 2) != Equilateral
assert triangle(1, 2, 3) != Equilateral
assert triangle(1, 2, 3) != Equilateral

Faults are identified by line number sequences
```python
def triangle(a, b, c):
    if a == b:
        if a != c:
            Equilateral
        else:
            Isosceles
    else:
        if b != c:
            if a == c:
                Isosceles
            else:
                Scalen
        else:
            Isosceles

assert triangle(1,1,1) = Equilateral
assert triangle(1,2,1) != Equilateral
assert triangle(1,1,2) != Equilateral
assert triangle(1,2,2) != Equilateral
assert triangle(1,2,3) != Equilateral
assert triangle(1,2,3) != Equilateral
```

Faults are identified by line number sequences.
def triangle(a, b, c):
    if a == b:
        if a != c:
            Equilateral
        else:
            Isosceles
    else:
        if b != c:
            if a == c:
                Isosceles
            else:
                Scalene
        else:
            Isosceles

assert triangle(1, 1, 1) == Equilateral
assert triangle(1, 2, 1) != Equilateral
assert triangle(1, 1, 2) != Equilateral
assert triangle(1, 2, 2) != Equilateral
assert triangle(1, 2, 3) != Equilateral
assert triangle(1, 2, 3) != Equilateral
def triangle(a, b, c):
    if a == b:
        if a == c:
            Equilateral
        else:
            Isosceles
    else:
        Scalene

assert triangle(1, 1, 1) == Equilateral
assert triangle(1, 2, 1) != Equilateral
assert triangle(1, 1, 2) != Equilateral
assert triangle(1, 2, 2) != Equilateral
assert triangle(1, 2, 3) != Equilateral
assert triangle(1, 2, 3) != Equilateral
Faults are identified by line number sequences

def triangle(a,b,c):
1: if a == b:
2:   if a == c: [!=]
3:     Equilateral                        assert triangle(1,1,1) = Equilateral
4:   else:
5:     Isosceles                          assert triangle(1,2,1) != Equilateral
6: else:
7:   if b != c: [Scalene]
8:     if a == c:                         assert triangle(1,1,2) != Equilateral
9:       Isosceles                        assert triangle(1,2,2) != Equilateral
10:      else:                           assert triangle(1,2,3) != Equilateral
11:        Scalene
12:       else:
13:         Isosceles
Faults are identified by line number sequences

```python
def triangle(a, b, c):
    if a == b:
        if a == c:  # Equilateral
            assert triangle(1,1,1) = Equilateral
        else:
            Isosceles
    else:
        if b != c:
            if a == c:
                Isosceles
            else:
                Scalene
        else:
            Isosceles
assert triangle(1,2,1) != Equilateral
assert triangle(1,1,2) != Equilateral
assert triangle(1,2,2) != Equilateral
assert triangle(1,2,3) != Equilateral
assert triangle(1,1,1) = Equilateral
assert triangle(1,2,1) != Equilateral
assert triangle(1,1,2) != Equilateral
assert triangle(1,2,2) != Equilateral
assert triangle(1,2,3) != Equilateral
```

Nothing catches fault `[7,8,9,10,11,12,13]`
Estimating defect density
How will you estimate the number of fishes in a lake?
Mark release recapture
Catch
Release
Recapture
Estimate!

Caught and tagged = Tagged in the second catch

Fishes in the lake

Second catch
Residual Defect Density

Residual defects in the program =

Detected seeded faults

Seeded
Fault Seeding

Problems

• How to generate faults?
• Which faults to generate?

Given a program of \( n \) tokens, there are at least \( 2^n \) alternative programs if each token can be replaced by another
Mutation Analysis
Mutation Analysis

Seed

all simple faults
Mutation Analysis

Seed

all simple faults

Detection ratio = \frac{\text{# faults detected}}{\text{# faults seeded}}
Seed  all simple faults

all first order mutants

Mutation score = \[
\frac{\text{# mutants killed (detected)}}{\text{# mutants seeded}}
\]
Mutation Analysis: Assumptions

Competent programmer hypothesis

• Most faulty programs contain numerous independent small faults (rather than faults that require complex conditions to manifest)
Mutation Analysis: Assumptions

Competent programmer hypothesis
• Most faulty programs contain numerous independent small faults

Coupling effect
• Tests detecting these small faults in isolation can detect most of the composite faults.
Mutation Analysis: Assumptions

Competent programmer hypothesis

- Most faulty programs contain numerous independent small faults

Coupling effect

- Tests detecting these small faults in isolation can detect most of the complex faults.

How much is most?

What is a small fault?

What is a complex fault?
Mutation Analysis: Formal definitions

Finite neighborhood hypothesis

- Most faulty programs contain numerous independent faults in the syntactic neighborhood (δ token changes) of the correct program.

Theory of composite faults

- Tests detecting these faults in isolation can detect > 99% of the composite faults in an average software.
def triangle(a,b,c):
    if a == b:
        if a == c:
            Equilateral
        else:
            Isosceles
    else:
        if b != c:
            if a == c:
                Isosceles
            else:
                Scalene
        else:
            Isosceles

Mutations: Simple (single token changes)

substitute == with one of !=, <, <=, >, >=

substitute != with one of ==, <, <=, >, >=
def triangle(a,b,c):
    if a == b:
        if a == c:
            Equilateral
        else:
            Isosceles
    else:
        if b != c:
            if a == c:
                Isosceles
            else:
                Scalene
        else:
            Isosceles

where value may be:
    true, false, not(cond)
def triangle(a, b, c):
    if a == b:
        if a == c:
            Equilateral
        else:
            Isosceles
    else:
        if b != c:
            if a == c:
                Isosceles
            else:
                Scalene
        else:
            Isosceles

Mutations: Simple (single token changes)
replace: strings with _
or other strings
Mutations: Simple (single token changes)

def triangle(a,b,c):
1:     if a == b:
2:         if a == c:
3:             Equilateral
4:         else:
5:             Isosceles
6:     else:
7:         if b != c:
8:             if a == c:
9:                 Isosceles
10:             else:
11:                 Scalene
12:         else:
13:             Isosceles

And pretty much any simple change that you can think of!
def triangle(a, b, c):
1:    if a == b:
2:        if a == c:
3:            Equilateral
4:        else:
5:            Isosceles
6:    else:
7:        if b != c:
8:            if a == c:
9:                Isosceles
10:            else:
11:                Scalene
12:        else:
13:            Isosceles

Mutations: Simple (single token changes)
Mutations: Simple (single token changes)

def triangle(a,b,c):
1: if a == b:
2:   if a == c:
3:     Equilateral
4:   else:
5:     Isosceles
6: else:
7:   if b != c:
8:     if a == c:
9:       \textcolor{red}{Isosceles} Scalene
10:   else:
11:    Scalene
12: else:
13:   Isosceles
Enough theory!
Review: Symbolic Execution

Hijack the Interpreter

- Assign proxy values to some or all inputs.
- On each conditional, fork the execution.
  - On child, continue with predicate true
  - On parent, continue with predicate false
- Collect and print predicates at the end
Hijack the Interpreter
def triangle(a, b, c):
    if a == b:
        if b == c:
            return 'Equilateral'
        else:
            return 'Isosceles'
    else:
        if b == c:
            return 'Isosceles'
        else:
            if a == c:
                return 'Isosceles'
            else:
                return 'Scalene'
def triangle(a, b, c):
    if a == b:
        if b == c:
            return 'Equilateral'
        else:
            return 'Isosceles'
    else:
        if b == c:
            return 'Isosceles'
        else:
            if a == c:
                return 'Isosceles'
            else:
                return 'Scalene'

def triangle(a, b, c):
    if mutate((4, 7), (a == b), f):
        if mutate((5, 11), (b == c), f):
            return 'Equilateral'
        else:
            return 'Isosceles'
    else:
        if mutate((10, 11), (b == c), f):
            return 'Isosceles'
        else:
            if mutate((13, 15), (a == c), f):
                return 'Isosceles'
            else:
                return 'Scalene'
def triangle(a, b, c):
    if a == b:
        if b == c:
            return 'Equilateral'
        else:
            return 'Isosceles'
    else:
        if b == c:
            return 'Isosceles'
        else:
            if a == c:
                return 'Isosceles'
            else:
                return 'Scalene'

def mutate(myid, cond_result, f):
    global registry
    if f.is_parent():
        if myid in registry:  # have we spawned this mutant before?
            return cond_result
        else:  # No we have not spawned.
            f.fork(myid)
            registry[myid] = result_mutate(cond_result) if f.is_child() else cond_result
    else:
        return registry.get(myid) or cond_result
def triangle(a, b, c):
    if a == b:
        if b == c:
            return 'Equilateral'
        else:
            return 'Isosceles'
    else:
        if b == c:
            return 'Isosceles'
        else:
            if a == c:
                return 'Isosceles'
            else:
                return 'Scalene'

def mutate(myid, cond_result, f):
    global registry
    if f.is_parent():
        if myid in registry:  # have we spawned this mutant before?
            return cond_result
        else:  # No we have not spawned.
            f.fork(myid)
            registry[myid] = result_mutate(cond_result) if f.is_child() else cond_result
    else:
        return registry.get(myid) or cond_result
class ForkingTransformer(ast.NodeTransformer):
    def visit_If(self, node):
        muid = (node.test.lineno, node.test.col_offset)
        node = ast.If(ast.Call(ast.Name('mu.mutate', None),
                           [ast.Str(muid), node.test, ast.Name('f', None)],
                           []), node.body, node.orelse)
        return self.generic_visit(node)

def visit_Assert(self, node):
    node = ast.Expr(ast.Call(ast.Name('mu.verify', None),
                             [node.test, ast.Name('f', None), ast.Num(node.test.lineno)],
                             []))
    return self.generic_visit(node)

def forking_transform(src):
    return astunparse.unparse(ForkingTransformer().visit(ast.parse(src)))

def main(args):
    tmpl = ""
    import mu
    f = mu.Forker()
    %s
    f.waitfor()
    ""
    t = ast.parse(slurp(args[1]))
    print(dedent(tmpl) % forking_transform(t))
class Forker():
    def fork(self, myid):
        self.r = os.fork()
        if self.r != 0:
            self.pids.append(self.r)
        else:
            self.myid = myid
            self.pids = []

    def is_child(self): return self.r == 0
    def is_parent(self): return self.r != 0

    def waitfor(self):
        for i in self.pids:
            if i == -1: continue
            os.waitpid(i, 0)

    def mypid(self):
        return os.getpid()

    def result_mutate(v):
        return not v

    def verify(tcond, forker, ln):
        with open(".pids/%s" % forker.mypid(), 'a+') as f:
            print("%s: %s (True?) at %d" % (forker.myid, str(tcond), ln), file=f)
Back to theory!
Are all mutants equally usable?
Are all mutants usable?

A

def triangle(a,b,c):
    if a == b:
        if a != c:
            Equilateral
        else:
            Isosceles
    else:
        if b != c:
            if a == c:
                Isosceles
            else:
                Scalene
        else:
            Isosceles

B

def triangle(a,b,c):
    if a == b:
        if true:
            Equilateral
        else:
            Isosceles
    else:
        if b != c:
            if a == c:
                Isosceles
            else:
                Scalene
        else:
            Isosceles
Are all mutants usable?

def triangle(a,b,c):
    if a == b:  
        if a != c:  
            Equilateral  
        else:  
            Isosceles  
    else:  
        if b != c:  
            if a == c:  
                Isosceles  
            else:  
                Scalene  
        else:  
            Isosceles  

assert triangle(1,1,1) = Equilateral  
assert triangle(1,2,1) = Isosceles  
assert triangle(1,1,2) = Isosceles  
assert triangle(1,2,2) = Isosceles  
assert triangle(1,2,3) = Scalene
Are all mutants usable?

```python
def triangle(a,b,c):
    if a == b:
        if a != c:
            Equilateral
        else:
            Isosceles
    else:
        if b != c:
            if a == c:
                Isosceles
            else:
                Scalene
        else:
            Isosceles

assert triangle(1,1,1) = Equilateral
assert triangle(1,2,1) = Isosceles
assert triangle(1,1,2) = Isosceles
assert triangle(1,2,2) = Isosceles
assert triangle(1,2,3) = Scalene
```
Are all mutants usable?

def triangle(a, b, c):
    if a == b:
        if a != c:
            Equilateral
        else:
            Isosceles
    else:
        if b != c:
            if a == c:
                Isosceles
            else:
                Scalene
        else:
            Isosceles

assert triangle(1,1,1) = Equilateral
assert triangle(1,2,1) = Isosceles
assert triangle(1,1,2) = Isosceles
assert triangle(1,2,2) = Isosceles
assert triangle(1,2,3) = Scalene
Are all mutants usable?

```
def triangle(a,b,c):
    if a == b:
        if a != c:
            Equilateral
        else:
            Isosceles
    else:
        if b != c:
            if a == c:
                Isosceles
            else:
                Scalene
        else:
            Isosceles

assert triangle(1,1,1) = Equilateral
assert triangle(1,2,1) = Isosceles
assert triangle(1,1,2) = Isosceles
assert triangle(1,2,2) = Isosceles
assert triangle(1,2,3) = Scalene
```

```python
A
```
Are all mutants usable?

```
def triangle(a, b, c):
    if a == b:
        if a != c:
            Equilateral
        else:
            Isosceles
    else:
        if b != c:
            if a == c:
                Isosceles
            else:
                Scalene
        else:
            Isosceles

assert triangle(1, 1, 1) = Equilateral
assert triangle(1, 2, 1) = Isosceles
assert triangle(1, 1, 2) = Isosceles
assert triangle(1, 2, 2) = Isosceles
assert triangle(1, 2, 3) = Scalene
```
Are all mutants usable?

def triangle(a,b,c):
    if a == b:
        if a != c:
            Equilateral
        else:
            Isosceles
    else:
        if b != c:
            if a == c:
                Isosceles
            else:
                Scalene
        else:
            Isosceles

assert triangle(1,1,1) = Equilateral
assert triangle(1,2,1) = Isosceles
assert triangle(1,1,2) = Isosceles
assert triangle(1,2,2) = Isosceles
assert triangle(1,2,3) = Scalene
Are all mutants usable?

```python
def triangle(a, b, c):
    if a == b:
        if a != c:
            Equilateral
        else:
            Isosceles
    else:
        if b != c:
            if a == c:
                Isosceles
            else:
                Scalene
        else:
            Isosceles

assert triangle(1, 1, 1) = Equilateral
assert triangle(1, 2, 1) = Isosceles
assert triangle(1, 1, 2) = Isosceles
assert triangle(1, 2, 2) = Isosceles
assert triangle(1, 2, 3) = Scalene
```
Are all mutants usable?

def triangle(a,b,c):
    if a == b:
        if a != c:
            Equilateral
        else:
            Isosceles
    else:
        if b != c:
            if a == c:
                Isosceles
            else:
                Scalene
        else:
            Isosceles

assert triangle(1,1,1) = Equilateral
assert triangle(1,2,1) = Isosceles
assert triangle(1,1,2) = Isosceles
assert triangle(1,2,2) = Isosceles
assert triangle(1,2,3) = Scalene
Are all mutants usable?

```python
def triangle(a,b,c):
    if a == b:
        if a != c:
            Equilateral
        else:
            Isosceles
    else:
        if b != c:
            if a == c:
                Isosceles
            else:
                Scalene
        else:
            Isosceles

assert triangle(1,1,1) = Equilateral
assert triangle(1,2,1) = Isosceles
assert triangle(1,1,2) = Isosceles
assert triangle(1,2,2) = Isosceles
assert triangle(1,2,3) = Scalene
```
Are all mutants usable?

```python
def triangle(a, b, c):
    if a == b:
        if a != c:
            Equilateral
        else:
            Isosceles
    else:
        if b != c:
            if a == c:
                Isosceles
            else:
                Scalene
        else:
            Isosceles

assert triangle(1, 1, 1) = Equilateral
assert triangle(1, 2, 1) = Isosceles
assert triangle(1, 1, 2) = Isosceles
assert triangle(1, 2, 2) = Isosceles
assert triangle(1, 2, 3) = Scalene
```
Are all mutants usable?

def triangle(a,b,c):
    if a == b:
        if a != c:
            Equilateral
        else:
            Isosceles
    else:
        if b != c:
            if a == c:
                Isosceles
            else:
                Scalene
        else:
            Isosceles

assert triangle(1,1,1) = Equilateral  A
assert triangle(1,2,1) = Isosceles      _
assert triangle(1,1,2) = Isosceles      A  B
assert triangle(1,2,2) = Isosceles      _
assert triangle(1,2,3) = Scalene        _

def triangle(a,b,c):
    if a == b:
        if true:
            Equilateral
        else:
            Isosceles
    else:
        if b != c:
            if a == c:
                Isosceles
            else:
                Scalene
        else:
            Isosceles
Are all mutants usable?

Tests detecting A are guaranteed to detect B

assert triangle(1,1,1) = Equilateral
assert triangle(1,2,1) = Isosceles
assert triangle(1,1,2) = Isosceles
assert triangle(1,2,2) = Isosceles
assert triangle(1,2,3) = Scalene
Are all mutants usable?

```python
def triangle(a, b, c):
    if a == b:
        if a != c:
            Equilateral
        else:
            Isosceles
    else:
        if b != c:
            if a == c:
                Isosceles
            else:
                Scalene
        else:
            Isosceles
```

Tests detecting A are guaranteed to detect B

Mutant B subsumes mutant A

```
assert triangle(1,1,1) = Equilateral
A
assert triangle(1,2,1) = Isosceles
A
assert triangle(1,1,2) = Isosceles
A
assert triangle(1,2,2) = Isosceles
A
assert triangle(1,2,3) = Scalene
A B
```
Are all mutants usable?

Tests detecting A are guaranteed to detect B

Mutant B subsumes mutant A

Mutant A is redundant!
Redundant mutants

- Wasted effort in killing them
- Hard to detect (undecidable in the general case)

```python
def gcd(a, b):
    if a == 0:
        return b
    while b != 0:
        if a > b
            a = a % b
        else:
            b = b - a
    return a
```
Redundant mutants

- Hard to detect (undecidable in the general case)
- Wasted effort in killing them
- Overestimation of mutation score when detected
- Underestimation of mutation score when not

```python
def gcd(a, b):
    if a == 0:
        return b
    while b != 0:
        if a > b:
            a = a % b
        else:
            b = b - a
    return a
```
Redundant mutants

- Hard to detect (undecidable in the general case)
- Wasted effort in killing them
- Overestimation of mutation score when detected
- Underestimation of mutation score when not

Not troublesome if you consider faults with larger footprint more important

def gcd(a, b):
    if a == 0:
        return b
    while b != 0:
        if a > b:
            a = a % b
        else:
            b = b - a
    return a
Redundant mutants

Solutions:

- Identify similar mutants through
  - Clustering, location, type of fault
- Statistical methods
- Brute force (compute full matrix of kills)
Are all mutants usable?

def gcd(a, b):
    if a == 0:
        return b
    while b != 0:
        if a > b
            a = a - b
        else
            b = b - a
    return a
def gcd(a, b):
    if a == 0:
        return b
    while b != 0:
        if a > b
            a = a % b
        else
            b = b - a
    return a

Can this change be detected?
Equivalent mutants

def gcd(a, b):
    if a == 0:
        return b
    while b != 0:
        if a > b
            a = a % b
        else:
            b = b - a
    return a

Syntactic changes that do not cause semantic changes

- Hard to detect (undecidable in the general case)
- Underestimation of mutation score when present
Equivalent mutants

(Partial) Solutions:

- Semantics preserving transforms.
  E.g: compiler optimizations can eliminate \( \sim 30\% \) of equivalent mutants created \( \text{(Papadakis et al. 2015)} \)
Equivalent mutants

(Partial) Solutions:

• Semantics preserving transforms.
  E.g: compiler optimizations can eliminate ~ 30% of equivalent mutants created (Papadakis et al. 2015)

• Symbolic analysis
Equivalent mutants

(Partial) Solutions:
- Look for second degree kills (current research)
Strong Firm and Weak Mutation
Fault Detection Requirements
Fault Detection Requirements

[Diagram showing flowchart with steps involving fault injection]
Fault Detection Requirements

Reachability
Fault Detection Requirements
Fault Detection Requirements

Fault Propagation
Fault Detection Requirements
Fault Detection Requirements

- Fault Injection
- Reachability
- State Infection
- Fault Propagation
- Fault Detection
Weak Mutation

Check whether the injected fault infected state
Check whether the injected fault propagated to some boundary (but not necessarily detected by a test)
Weak/Firm Mutation

• No verification of assertions/oracles

• Not necessarily mutation analysis
  (can be done without generating mutants)

• A (stronger) form of coverage

• Weak mutation hypothesis: Reachability is sufficient!
Weak/Firm Mutation

- No verification of assertions/oracles
- Not necessarily mutation analysis (can be done without generating mutants)
- A (stronger) form of coverage
- Weak mutation hypothesis: Reachability is sufficient!

May or may not suffice depending on the program. E.g triangle
Subsumption of Structural Coverage Criteria

Do we need other coverage criteria along with mutation analysis?
Subsumption of statement coverage

def gcd(a, b):
    if a == 0:
        return b
    while b != 0:
        if a > b:
            a = a - b
        else:
            b = b - a
    return a

Statement deletion
mutation
Subsumption of predicate coverage

def gcd(a, b):
    if a == 0:
        return b
    while b != 0:
        if a > b:
            a = a - b
        else:
            b = b - a
    return a

(Also clause coverage!)
Time Reduction

- Do Fewer
- Do Faster (algorithmic - serial)
- Do Smarter (parallel)
Super Mutants
Super Mutants

Delta Debug the Super mutant!
Super Mutants

Detected: Yes

Delta Debug the Super mutant!
Super Mutants

Detected: Yes

Delta Debug the Super mutant!
Super Mutants

Delta Debug the Super mutant!
Super Mutants

Detected: Yes

Delta Debug the Super mutant!
Super Mutants

Detected: Yes

Delta Debug the Super mutant!
Super Mutants

Delta Debug the Super mutant!
Super Mutants

Delta Debug the Super mutant!
Super Mutants

Detected: Yes

Delta Debug the Super mutant!
Super Mutants

Detected: Yes

Program

Mutations

Yes

No

Yes

Yes

Delta Debug the Super mutant!
Super Mutants

Program Mutations

Detected: Yes

Delta Debug the Super mutant!
Super Mutants

Delta Debug the Super mutant!
Super Mutants

Program

Mutations

Detected: Yes

Yes

Yes

Yes

Yes

No

Yes

Yes

Yes

Delta Debug the Super mutant!
Super Mutants

Program

Mutations

Detected: Yes

Delta Debug the Super mutant!
Super Mutants

Detected: Yes

Program Mutations

Yes

Yes

Yes

Yes

Yes

Yes

Delta Debug the Super mutant!
Super Mutants

Detected: Yes

Delta Debug the Super mutant!
Super Mutants

 deltas Debug the Super mutant!

Best case = 1 Execution

Worst case = 2n - 1 Executions
Higher Order Mutants
Where to mutate?

- Program text
- Byte code
- Abstract Syntax Tree
- Program Model
- Specification
- Machine code
def triangle(a,b,c):
    if a == b:
        if a == c:
            Equilateral
        else:
            Isosceles
    else:
        if b != c:
            if a == c:
                Isosceles
            else:
                Scalene
        else:
            Isosceles
assert triangle(1,1,1) = Equilateral
assert triangle(1,2,1) != Equilateral
assert triangle(1,1,2) != Equilateral
assert triangle(1,2,2) != Equilateral
assert triangle(1,2,3) != Equilateral

Will this work?

Residual Defect Density

Seed all simple faults
all first order mutants

Mutation score = \# mutants killed (detected) / \# mutants seeded

Hijack the Interpreter

Demo

Subsumption of Structural Coverage Criteria

Do we need other coverage criteria along with mutation analysis?

Weak Mutation
Check whether the injected fault infected state

Firm Mutation
Check whether the injected fault propagated to some boundary (but not necessarily detected by a test)