CSE P 504

Advanced topics in Software Systems

Fall 2022

Mutation-based Testing

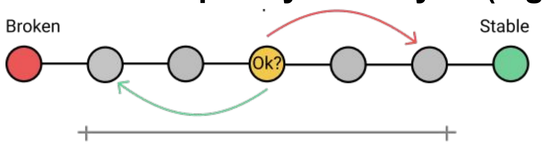
October 24, 2022

Today

- Recap: Git bisect exercise
- Mutation-based testing
  - The basics
  - Productive mutants
  - Mutant subsumption
- Coverage-based vs. mutation-based testing
- In-class exercise 3

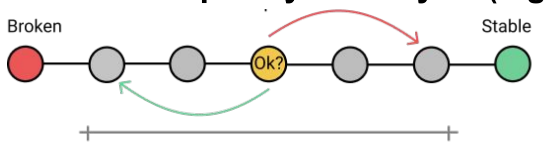
Recap: Git bisect

- **Git bisect run-time complexity is always O(log(n))**

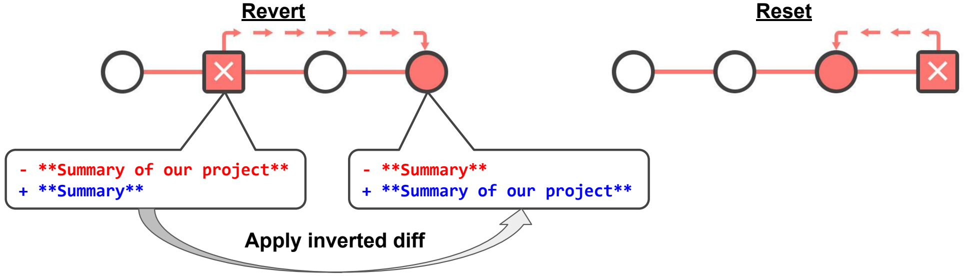


Recap: Git bisect

- **Git bisect run-time complexity is always O(log(n))**

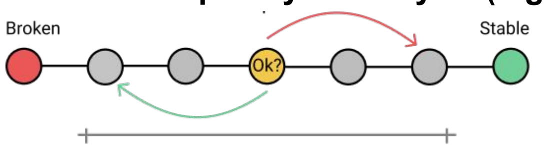


- **Git revert vs. git reset**



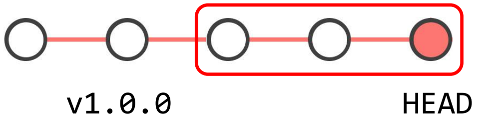
Recap: Git bisect

- **Git bisect run-time complexity is always O(log(n))**



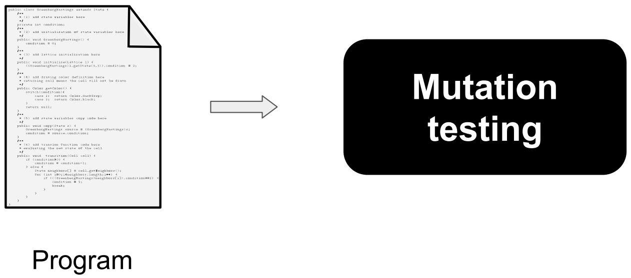
- **Git revert vs. git reset**

- `git rev-list v1.0.0..HEAD (or HEAD ^v1.0.0)`

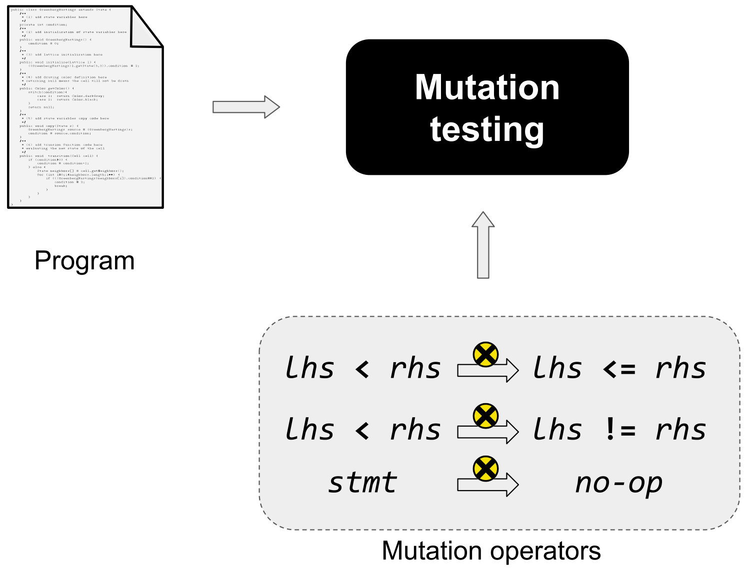


Mutation-based testing: the basics

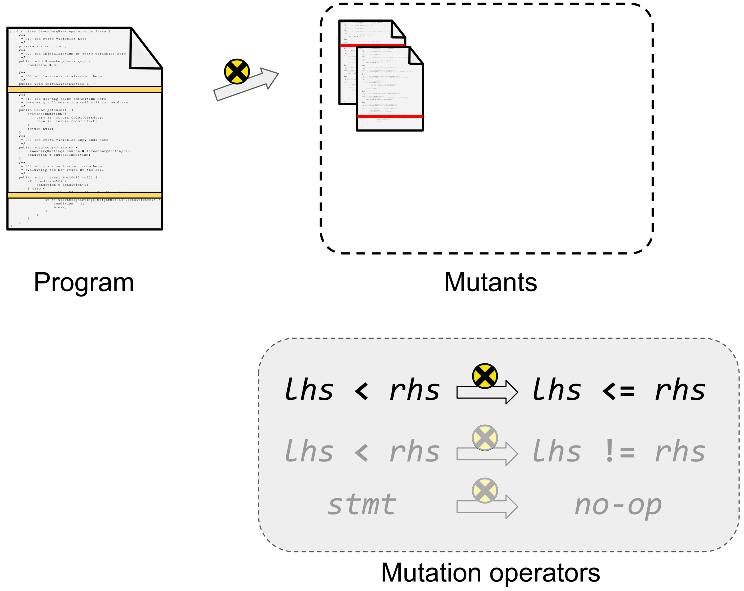
Mutation testing



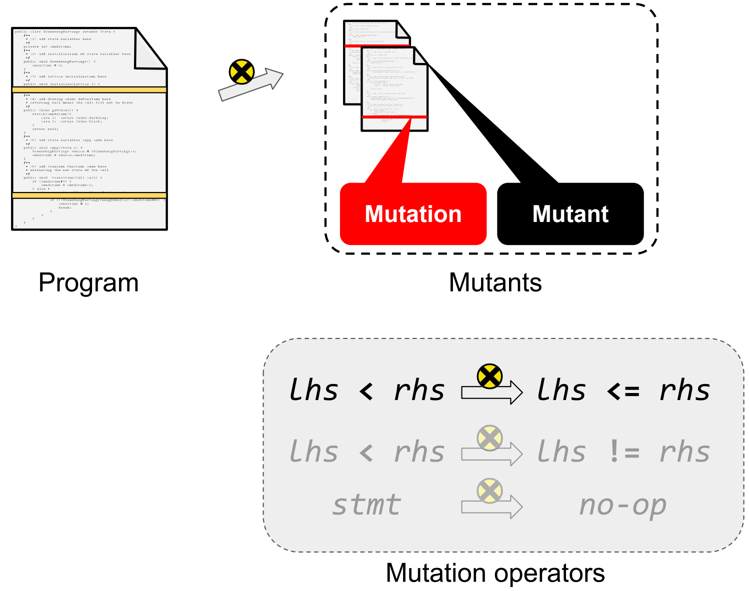
Mutation testing: mutant generation



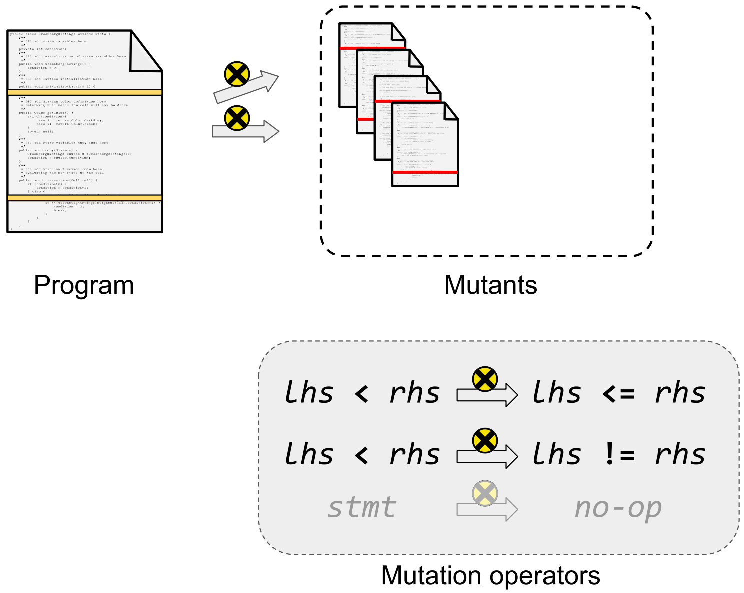
Mutation testing: mutant generation



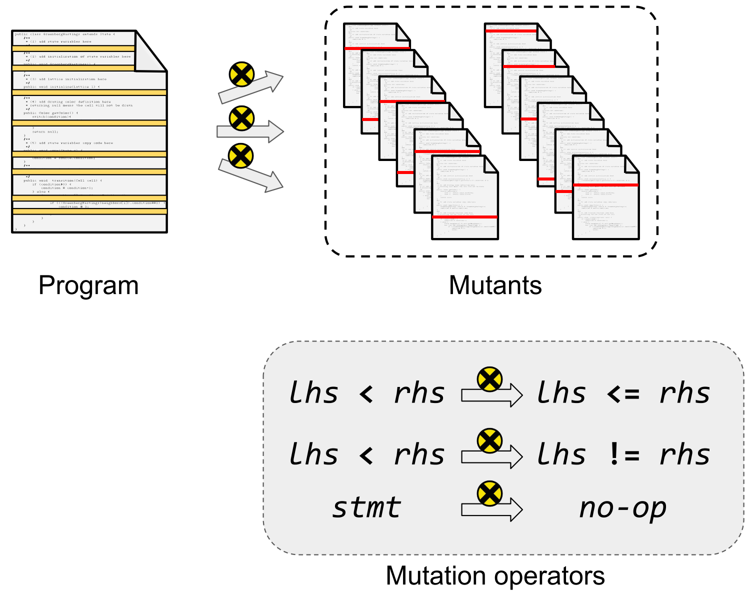
Mutation testing: mutant generation



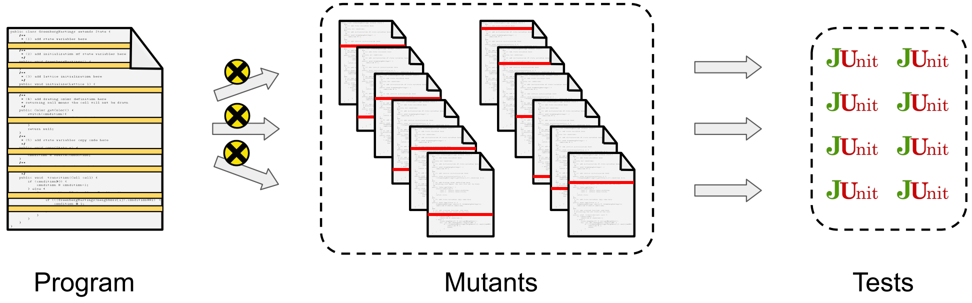
Mutation testing: mutant generation



Mutation testing: mutant generation



Mutation testing: test creation



Assumptions

- Mutants are coupled to real faults
- Mutant detection is correlated with real-fault detection

https://homes.cs.washington.edu/~rjust/publ/mutants_real_faults_fse_2014.pdf,
https://homes.cs.washington.edu/~rjust/publ/mutation_testing_practices_icse_2021.pdf

Mutation testing: a concrete example

Original program:

```
public int min(int a, int b) {
    return a < b ? a : b;
}
```

Mutant 1:

```
public int min(int a, int b) {
    return a;
}
```

Mutation testing: another example

Original program:

```
public int min(int a, int b) {
    return a < b ? a : b;
}
```

Mutant 2:

```
public int min(int a, int b) {
    return b;
}
```

Mutation testing: yet another example

Original program:

```
public int min(int a, int b) {
    return a < b ? a : b;
}
```

Mutant 3:

```
public int min(int a, int b) {
    return a >= b ? a : b;
}
```

Mutation testing: last example (I promise)

Original program:

```
public int min(int a, int b) {
    return a < b ? a : b;
}
```

Mutant 4:

```
public int min(int a, int b) {
    return a <= b ? a : b;
}
```

Mutation testing: exercise

Original program:

```
public int min(int a, int b) {
    return a < b ? a : b;
}
```

Mutants:

```
M1: return a;
M2: return b;
M3: return a >= b ? a : b;
M4: return a <= b ? a : b;
```

For each mutant, provide a test case that detects it
(i.e., passes on the original program but fails on the mutant)

Mutation testing: exercise

Original program:

```
public int min(int a, int b) {
    return a < b ? a : b;
}
```

Mutants:

```
M1: return a;
M2: return b;
M3: return a >= b ? a : b;
M4: return a <= b ? a : b;
```

M4 cannot be detected (equivalent mutant).

a	*b*	Original	M1	M2	M3	M4
1	2	1	1	**2**	**2**	1
1	1	1	1	1	1	1
2	1	1	**2**	1	**2**	1

Mutation testing: exercise

Original program:

```
public int min(int a, int b) {
    return a < b ? a : b;
}
```

Mutants:

```
M1: return a;
M2: return b;
M3: return a >= b ? a : b;
M4: return a <= b ? a : b;
```

Which mutant(s) should we show to a developer?

a	*b*	Original	M1	M2	M3	M4
1	2	1	1	**2**	**2**	1
1	1	1	1	1	1	1
2	1	1	**2**	1	**2**	1

Mutation testing: summary

Original program:

```
public int min(int a, int b) {
    return a < b ? a : b;
}
```

Mutants:

```
M1: return a;
M2: return b;
M3: return a >= b ? a : b;
M4: return a <= b ? a : b;
```

Redundant (M3)

Equivalent (M4)

a	*b*	Original	M1	M2	M3	M4
1	2	1	1	**2**	**2**	1
1	1	1	1	1	1	1
2	1	1	**2**	1	**2**	1

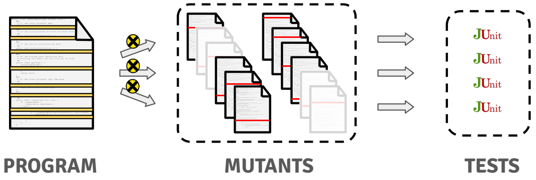
Mutation testing: challenges

- Redundant mutants
  - Inflate the mutant detection ratio
  - Hard to assess progress and remaining effort
- Equivalent mutants
  - Max mutant detection ratio != 100%
  - Waste resources (CPU and human time)

a	*b*	Original	M1	M2	M3	M4
1	2	1	1	**2**	**2**	1
1	1	1	1	1	1	1
2	1	1	**2**	1	**2**	1

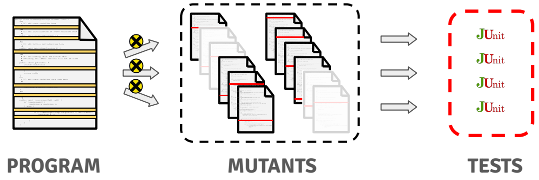
Mutation Testing vs. Mutation Analysis

Mutation Testing



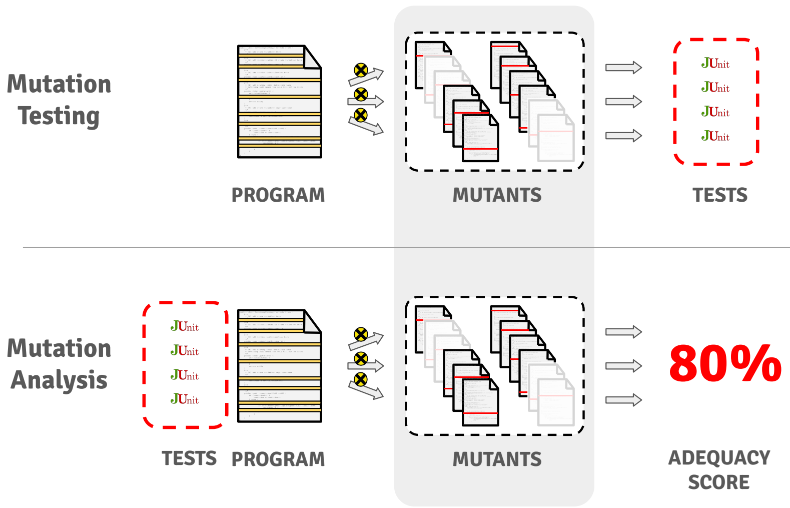
Mutation Testing vs. Mutation Analysis

Mutation Testing



Primary output is **new tests.**

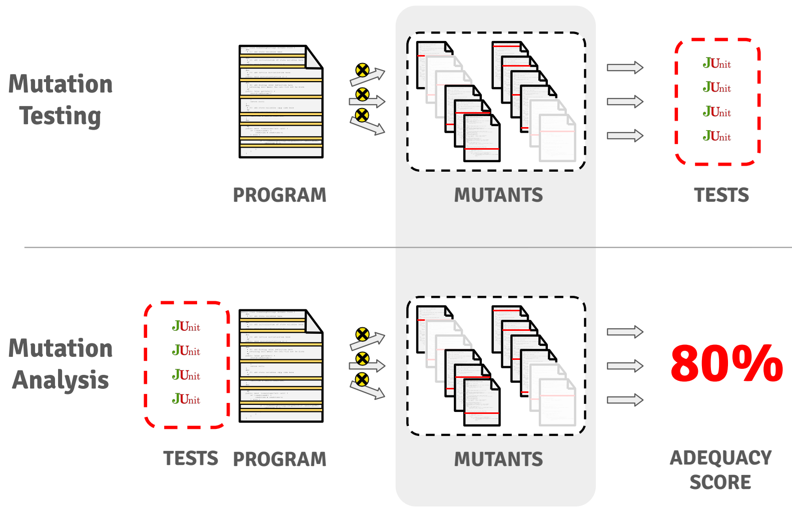
Mutation Testing vs. Mutation Analysis



Mutation Testing: Primary output is **new tests.**

Mutation Analysis: Primary output is **adequacy score** for **existing tests.**

Mutation Testing vs. Mutation Analysis



Mutation Testing: Primary output is **new tests.**

Mutation Analysis: Primary output is **adequacy score** for **existing tests.**

How expensive is mutation testing?
Is the mutation score meaningful?

Mutation-based testing: productive mutants

Detectable vs. productive mutants

Historically

- **Detectable** mutants are **good** ⟹ **tests**
- **Equivalent** mutants are **bad** ⟹ **no tests**

A more nuanced view

- **Detectable vs. equivalent** is **too simplistic**
- **Productive mutants** elicit effective tests, but
  - detectable mutants can be useless, and
  - equivalent mutants can be useful!

The core question here concerns test-goal utility
(applies to any adequacy criterion).

An Industrial Application of Mutation Testing: Lessons, Challenges, and Research Directions (Reading 1)

Detectable vs. productive mutants

Historically

- **Detectable** mutants are **good** ⟹ **tests**
- **Equivalent** mutants are **bad** ⟹ **no tests**

A more nuanced view

- **Detectable vs. equivalent** is **too simplistic**
- **Productive mutants** elicit effective tests, but
  - detectable mutants can be useless, and
  - equivalent mutants can be useful!

The notion of productive mutants is fuzzy!

A mutant is **productive** if it is

1. **detectable** and **elicits an effective test** or
2. **equivalent** and **advances code quality or knowledge**

An Industrial Application of Mutation Testing: Lessons, Challenges, and Research Directions (Reading 1)

Productive mutants: mutation testing at Google

```
int RunMe(int a, int b) {
  if (a == b || b == 1) {
```

▾ Mutants
14:25, 28 Mar

```
Changing this 1 line to

  if (a != b || b == 1) {

does not cause any test exercising them to fail.

Consider adding test cases that fail when the code is mutated to
ensure those bugs would be caught.

Mutants ran because goranpetrovic is whitelisted
```

Please fix   Not useful

Practical Mutation Testing at Scale: A view from Google (Reading 3)

Productive mutants: mutation testing at Google

```
int RunMe(int a, int b) {
  if (a == b || b == 1) {
```

▾ Mutants
14:25, 28 Mar

```
Changing this 1 line to

  if (a != b || b == 1) {

does not cause any test exercising them to fail.

Consider adding test cases that fail when the code is mutated to
ensure those bugs would be caught.

Mutants ran because goranpetrovic is whitelisted
```

Please fix   Not useful

Practical Mutation Testing at Scale: A view from Google (Reading 3)

Detectable vs. productive mutants (1)

Original program

```
public double getAvg(double[] nums) {
  double sum = 0;
  int len = nums.length;

  for (int i = 0; i < len; ++i) {
      sum = sum + nums[i];
  }

  return sum / len;
}
```

Mutant

```
public double getAvg(double[] nums) {
  double sum = 0;
  int len = nums.length;

  for (int i = 0; i < len; ++i) {
      sum = sum * nums[i];
  }

  return sum / len;
}
```

Is the mutant is **detectable?**

Detectable vs. productive mutants (1)

Original program

```
public double getAvg(double[] nums) {
  double sum = 0;
  int len = nums.length;

  for (int i = 0; i < len; ++i) {
      sum = sum + nums[i];
  }

  return sum / len;
}
```

Mutant

```
public double getAvg(double[] nums) {
  double sum = 0;
  int len = nums.length;

  for (int i = 0; i < len; ++i) {
      sum = sum * nums[i];
  }

  return sum / len;
}
```

The mutant is **detectable, but** is it **productive?**

Detectable vs. productive mutants (1)

Original program

```
public double getAvg(double[] nums) {
  double sum = 0;
  int len = nums.length;

  for (int i = 0; i < len; ++i) {
      sum = sum + nums[i];
  }

  return sum / len;
}
```

Mutant

```
public double getAvg(double[] nums) {
  double sum = 0;
  int len = nums.length;

  for (int i = 0; i < len; ++i) {
      sum = sum * nums[i];
  }

  return sum / len;
}
```

The mutant is **detectable, but** is it **productive? Yes!**

Detectable vs. productive mutants (2)

Original program

```
public double getAvg(double[] nums) {
  int len = nums.length;
  double sum = 0;
  double avg = 0;

  for (int i = 0; i < len; ++i) {
      avg = avg + (nums[i] / len);
      sum = sum + nums[i];
  }

  return sum / len;
}
```

Mutant

```
public double getAvg(double[] nums) {
  int len = nums.length;
  double sum = 0;
  double avg = 0;

  for (int i = 0; i < len; ++i) {
      avg = avg * (nums[i] / len);
      sum = sum + nums[i];
  }

  return sum / len;
}
```

Is the mutant **detectable?**

Detectable vs. productive mutants (2)

Original program

```
public double getAvg(double[] nums) {
  int len = nums.length;
  double sum = 0;
  double avg = 0;

  for (int i = 0; i < len; ++i) {
      avg = avg + (nums[i] / len);
      sum = sum + nums[i];
  }

  return sum / len;
}
```

Mutant

```
public double getAvg(double[] nums) {
  int len = nums.length;
  double sum = 0;
  double avg = 0;

  for (int i = 0; i < len; ++i) {
      avg = avg * (nums[i] / len);
      sum = sum + nums[i];
  }

  return sum / len;
}
```

The mutant is **not detectable, but** is it **unproductive?**

Detectable vs. productive mutants (2)

Original program

```
public double getAvg(double[] nums) {
  int len = nums.length;
  double sum = 0;
  double avg = 0;

  for (int i = 0; i < len; ++i) {
      avg = avg + (nums[i] / len);
      sum = sum + nums[i];
  }

  return sum / len;
}
```

Mutant

```
public double getAvg(double[] nums) {
  int len = nums.length;
  double sum = 0;
  double avg = 0;

  for (int i = 0; i < len; ++i) {
      avg = avg * (nums[i] / len);
      sum = sum + nums[i];
  }

  return sum / len;
}
```

The mutant is **not detectable, but** is it **unproductive? No!**

Detectable vs. productive mutants (3)

Original program

```
...
Set cache = new HashSet(a * b);
...
```

Mutant

```
...
Set cache = new HashSet(a + b);
...
```

Is the mutant **detectable?**

Detectable vs. productive mutants (3)

Original program

```
...
Set cache = new HashSet(a * b);
...
```

Mutant

```
...
Set cache = new HashSet(a + b);
...
```

The mutant is **detectable, but** is it **productive?**

Detectable vs. productive mutants (3)

Original program

```
...
Set cache = new HashSet(a * b);
...
```

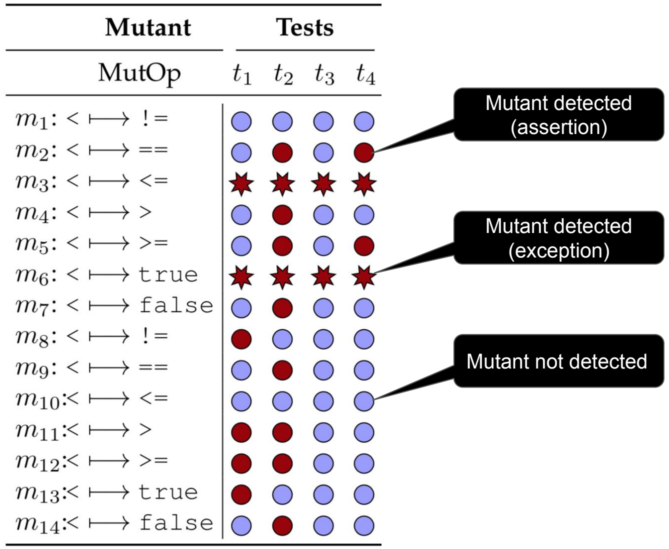
Mutant

```
...
Set cache = new HashSet(a + b);
...
```

The mutant is **detectable, but** is it **productive? No!**

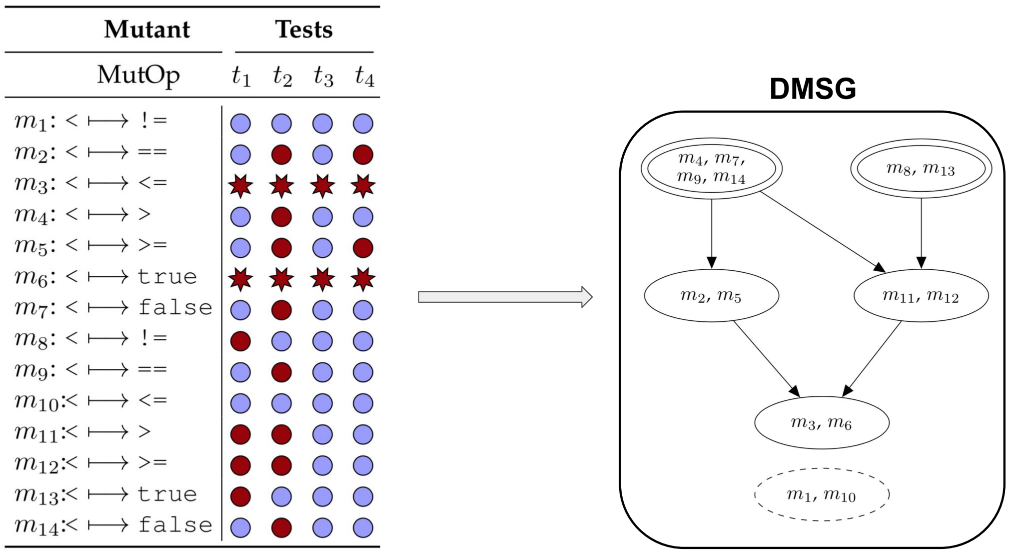
Mutation-based testing: mutant subsumption

Mutant subsumption



Prioritizing Mutants to Guide Mutation Testing ([Reading 2](#))

DMSG: Dynamic Mutant Subsumption Graph



Prioritizing Mutants to Guide Mutation Testing ([Reading 2](#))

Coverage-based vs. mutation-based testing

See dedicated Slides (4 pages).