Commit-aware Selective Mutation Testing

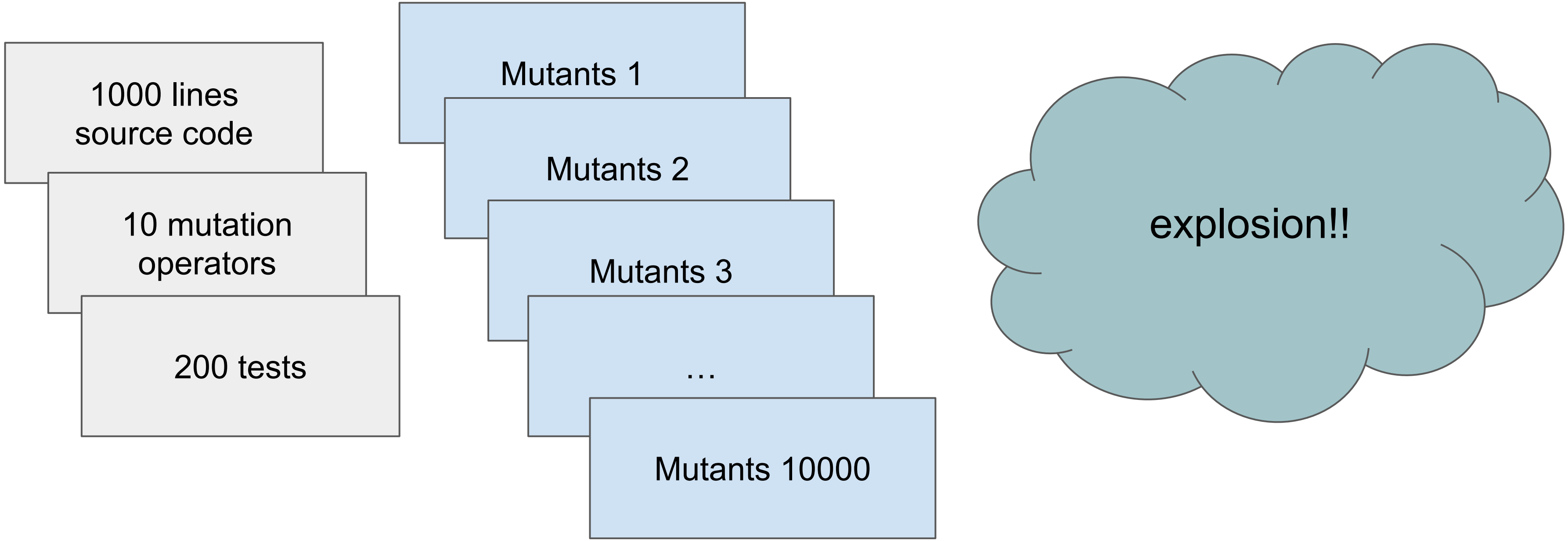
Team 3

Kyungwook Nam, Juchang Lee, Minwoo Baek, Hyeongjin Choi

Contents

Mutation Testing

- Mutation testing is a powerful testing technique
- But is mutation testing scalable in real world programming?



Mutation Testing Cost Reduction Techniques

Jia, Yue, and Mark Harman. "An analysis and survey of the development of mutation testing." *IEEE transactions on software engineering* 37.5 (2010): 649-678.

Mutation Reduction Techniques

- Mutant Sampling
- Mutant Clustering
- High Order Mutation
- Selective Mutation

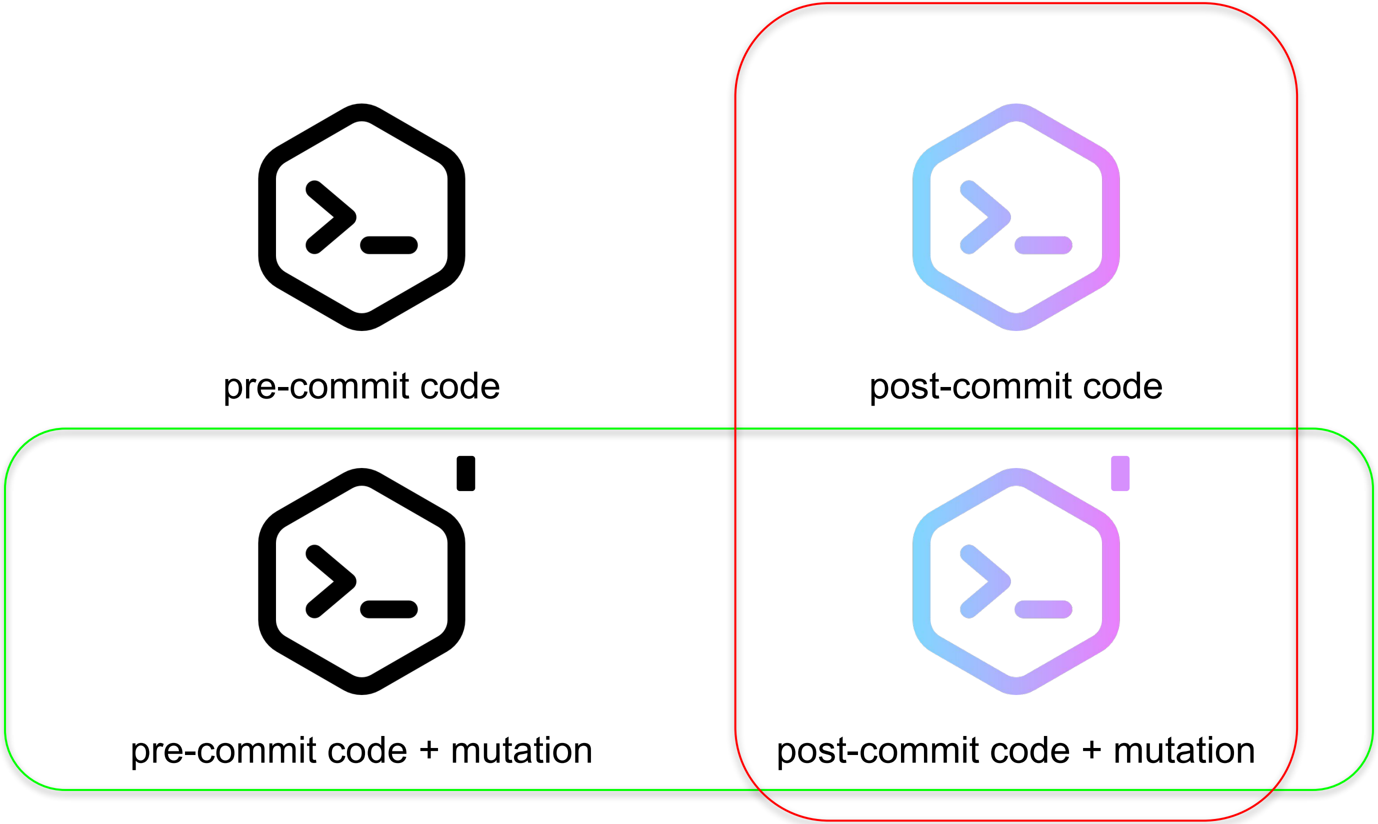
Execution Cost Reduction Techniques

- Strong, Weak and Firm Mutation
- Run-time Optimization
- Advanced Platforms Support

What we’re trying to enhance

- Situation is …
    - Continuous Integration / Continuous Development
    - During CI/CD, mutation testing is non-optimal.
    - Repeated tests on non-changed code is a waste of resource.
- So why don’t we …
    - Generate tests relevant to commits?
        - Commit-relevant Mutants
    - So that we don’t waste time on irrelevant mutants.

Commit-relevant Mutant?



Commit-relevant Mutant; an example

```
int func (int x[3], int y[3]) {
1.      int L, R, vL = 0, vR = 0;
2.      sort(x); sort(y);
3.      R = 2;          // R = 0;
4.      if (x[R] > y[R]) {
5.          vR = 1;
6.      } else if (x[R] == y[R]) {
7. -        L = 1;
7. +        L = 0;
8.          if (x[L] > y[L])
9.              vL = 1;
10.     }
11.
12.     if (x[0] > y[2])
13.         return -1;
14.
15.     return vL + vR;
}
```

Mutant M_1 (Relevant)

For test input: ***x* = *{0, 3 ,4}*** and ***y* = *{0, 2, 3}***, the return codes are following:

- Mutant post-commit: **0**
- Mutant pre-commit: **1**
- Original post-commit: **1**

```
int func (int x[3], int y[3]) {
1.      int L, R, vL = 0, vR = 0;
2.      sort(x); sort(y);
3.      R = 2;
4.      if (x[R] > y[R]) {
5.          vR = 1;     // vR = 0;
6.      } else if (x[R] == y[R]) {
7. -        L = 1;
7. +        L = 0;
8.          if (x[L] > y[L])
9.              vL = 1;
10.     }
11.
12.     if (x[0] > y[2])
13.         return -1;
14.
15.     return vL + vR;
}
```

Mutant M_2 (Non-relevant)

No test can execute both the mutated statement (line 5) and the modification (line 7) in both pre and post commit versions

```
int func (int x[3], int y[3]) {
1.      int L, R, vL = 0, vR = 0;
2.      sort(x); sort(y);
3.      R = 2;
4.      if (x[R] > y[R]) {
5.          vR = 1;
6.      } else if (x[R] == y[R]) {
7. -        L = 1;
7. +        L = 0;
8.          if (x[L] > y[L])
9.              vL = 1;
10.     }
11.
12.     if (x[0] > y[2]) // if (x[0] >= y[2])
13.         return -1;
14.
15.     return vL + vR;
}
```

Mutant M_3 (Non-relevant)

Any test that kills the mutant post-commit must fulfil the condition $x[0] == y[2]$. Any test that fulfil the above condition will make the mutant output -1 for pre and post commit versions. Thus no test can make the mutation interact with the modification.

Fig. 2. Example of relevant and non-relevant mutants. Mutant 1 is relevant to the committed changes. Mutants 2 and 3 are not relevant.

Collecting Commit-relevant Mutants

- Naive Approach
  - Mutation only on changed code?
    - Codes are complex.
- Previous Approach
  - MuDelta [1]
    - Check if mutants are commit-relevant via ML.
    - Mutants need to be created before testing.

[1] W. Ma, T. Titcheu Chekam, M. Papadakis and M. Harman, "MuDelta: Delta-Oriented Mutation Testing at Commit Time," 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), Madrid, ES, 2021, pp. 897-909, doi: 10.1109/ICSE43902.2021.00086.

Approach

- **Creating commit-relevant mutants from the start.**
- Assumption
    - We can find mutator applicable code based on data/control dependency.
- Techniques
    - Data Dependency
        - Use-define chain
        - Inter-functional
    - Control Dependency
        - Conditional Checks

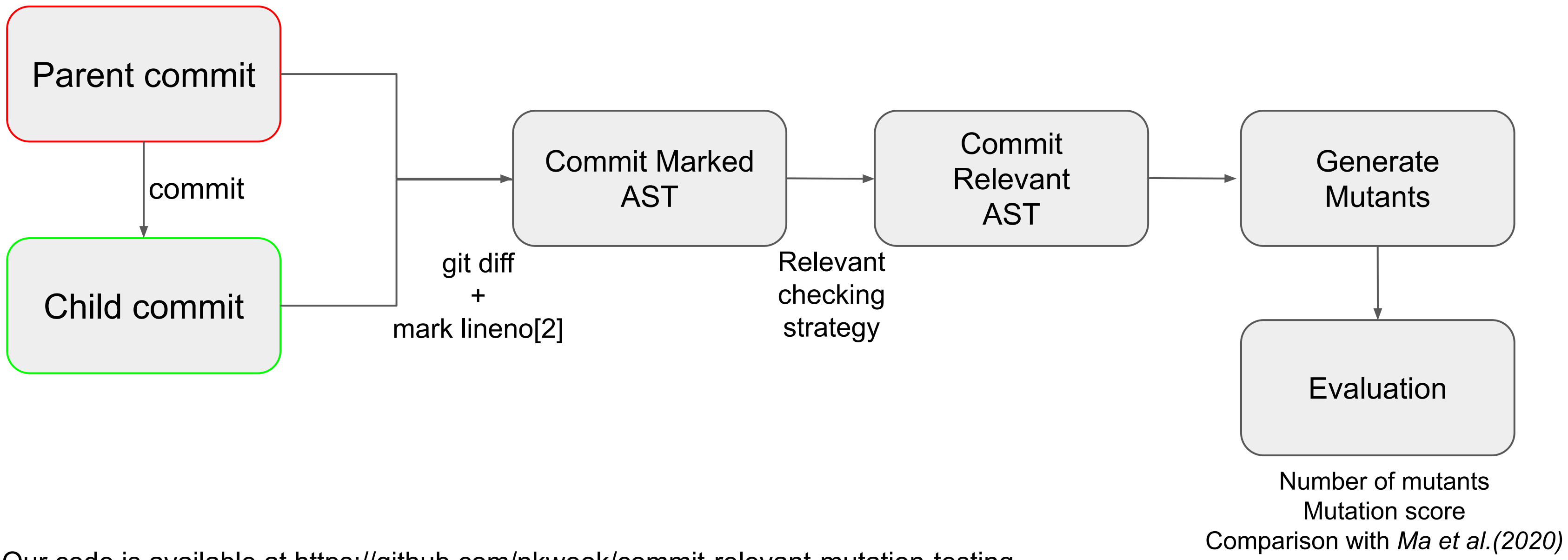
```python
def two_sum(a, b):
    c = a
    c = 2
    a = b
    c += 10
    # irrelevant code
    d = a + b
    z = c + d
```

Data Dependency Example

```python
def f1():

    a = 1

    if a:
        b = True
    else:
        b = False

    ...
```

Control Dependency Example

Overview



Our code is available at https://github.com/nkwook/commit-relevant-mutation-testing

[2] https://github.com/jay/showlinenum

Test Suite Preparation for Evaluation

1. Code Suite: Make sample code suite + test cases from scratch
2. Commits: Which doesn't affect original test results

```python
from typing import List

def func(x: List[int], y: List[int]) -> int:
    L, R, vL, vR = 0, 0, 0, 0
    x, y = sorted(x), sorted(y)
    R = 2
    if x[R] > y[R]:
        vR = 1
    elif x[R] == y[R]:
        L = 1
        if x[L] > y[L]:
            vL = 1
    if x[0] > y[2]:
        return -1
    return vL + vR
```

```diff
@@ -7,7 +7,7 @@ def func(x: List[int], y: List[int]) -> int:
     if x[R] > y[R]:
         vR = 1
     elif x[R] == y[R]:
-        L = 0
+        L = 1
         if x[L] > y[L]:
             vL = 1
     if x[0] > y[2]:
```

Research Question

- **RQ1: Ability for finding commit-relevant mutants**

commit-included mutant	commit-relevant mutant	non-relevant mutant

  - Possibility about revealing “Hidden” mutants
- **RQ2: Correlation of Mutation Score between commit-relevant / entire mutants**
  - Strong: Two metrics have similar behaviours
  - *Ma et al.(2020)* observed correlations are relatively weak ranging from 0.15 to 0.35 on their commit-relevant mutants(which derived by actual tests)
- **RQ3: How much Our mutant and *Ma et al.(2020)*’s mutant overlap?**
  - *Ma et al.(2020)*’s commit relevant mutant selection

$$\text{PostCommitOrig} \neq \text{PostCommitMut} \wedge \text{PreCommitMut} \neq \text{PostCommitMut}$$

Evaluation

Sample	#Mutants (entire set)	#Mutants (commit-included)		#Mutants (commit-relevant + commit-included)	Mutation Score (commit-aware)	Correlation: 0.715	Mutation Score (entire set)
Conditionals1	80	**6**	$\xrightarrow{\text{Condition dependency}}$	**6**	66.67% (4 / 6)		57.50% (46 / 80)
Fibonacci	6	6		6	66.67% (4 / 6)		66.67% (4 / 6)
Sort	30	6		6	100.00% (6 / 6)		86.67% (26 / 30)
Simple	79	**22**	$\xrightarrow{\text{Data dependency}}$	**38**	78.95% (30 / 38)		88.61% (70 / 79)
Conditionals2	95	10		10	40.00% (4 / 10)		63.16% (60 / 95)
FunctionCall	74	**23**	$\longrightarrow$	**27**	29.63% (8 / 27)		50.00% (37 / 74)
List	36	5		5	00.00% (0 / 5)		61.11% (22 / 36)

Discussion

- Something works but…
- Lack of functionality
  - Conditional contexts
- Ambiguity on selecting sample / test codes / commit patterns
  - Viable mutation operators
- Approaching to **RQ3: How much Our mutant and *Ma et al.(2020)*'s mutant overlap?**
  - Even we mask the affected portion of pre / post commit codes, overall sequence of ast nodes changes
  - How to align AST nodes and automate finding procedure of commit-relevant mutants by testing?

Plan

- Approach on RQ3
  - Comparison with random selection
- Consider conditional contexts
- CI pipeline with github action
- Controlling the occurrence of diffs by formatter
  - intent, linter, ..