Enhancing Mutation Based Fault Localization

Team 5
Hyungjoon Yoon, Sanghyun Park,
Banseok Woo, Hyeonseok Lee, Somin Kim

TABLE OF CONTENTS

Project Introduction

Mutation Based Fault Localization (1/2)

MBFL : Blend of Mutation Testing and Fault Localization

Key Idea : Mutating an already faulty program can reveal insights about the fault (further damage vs partial fix)

Project Introduction

Mutation Based Fault Localization (2/2)

Case1: Mutating Correct Statements

- Equivalent	P	F
- New Fault	P-	F+
- Mask	P+	F-

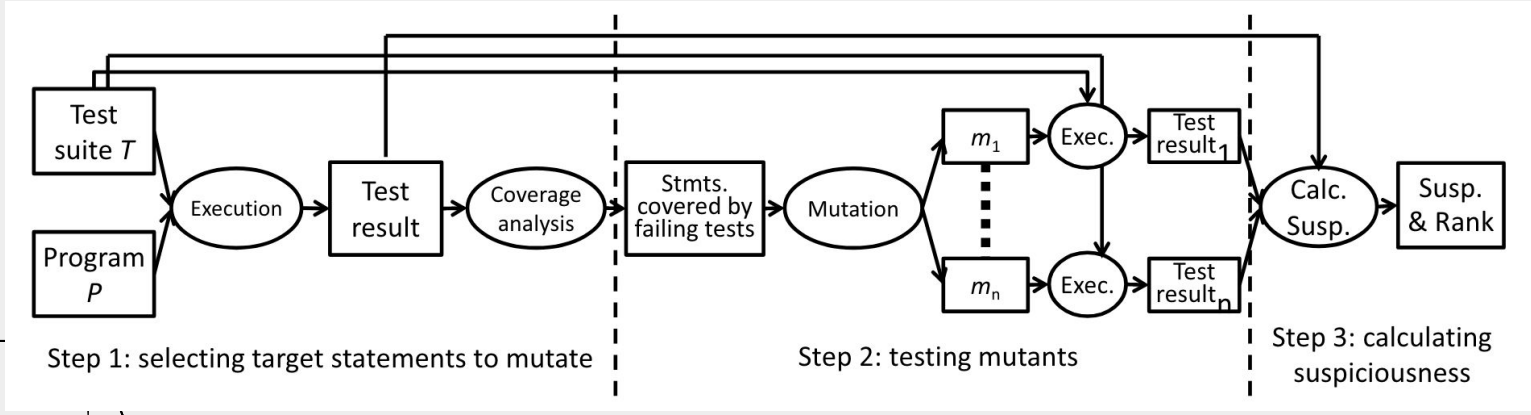
Case2: Mutating Faulty Statements

- Equivalent	P	F
- (New) Fault	P?	F?
- Mask	P+	F-
- (Partial) Fix	P+	F-

Project Introduction

Our Objective : Enhancing MBFL[MUSE (Moon et al., 2014)]

Framework of MUtation-baSEd fault localization technique



Project Introduction

Our Objective : Enhancing MBFL[MUSE (Moon et al., 2014)]

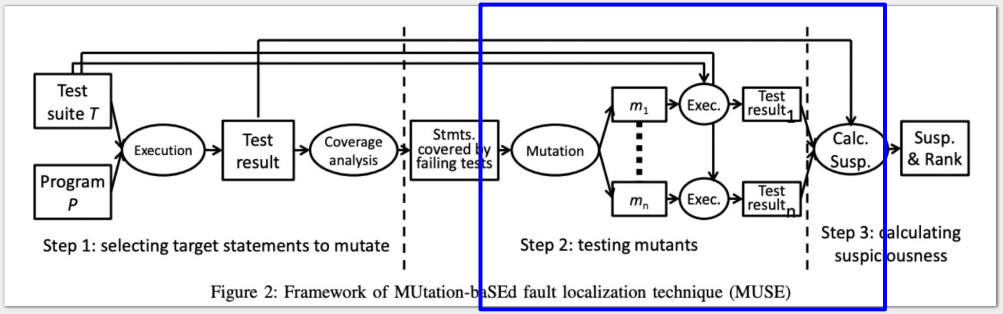


Figure 2: Framework of MUtation-baSEd fault localization technique (MUSE)

Cost of mutating every elements ↑

Dynamically Eliminating Statements that have a **low probability** of being buggy

*Probability is ranked with Spectrum Based Fault Localization Ranks

Before beginning...

Challenges

Unfeasible Tools for MUSE Reproduction

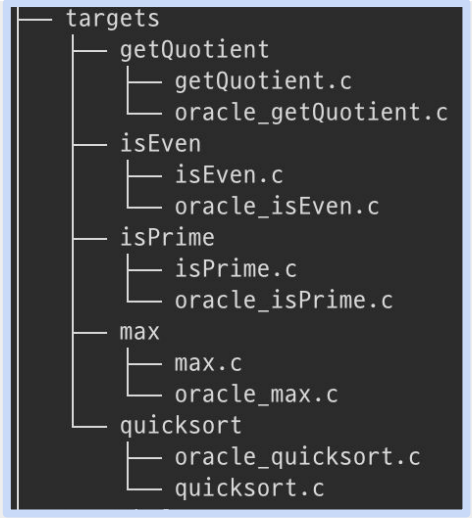
- Proteum, the C code mutator not working
  - Used mull for mutation

- SIR benchmark not accessible
  - Started with small codes for Proof-of-Concept

Progress Report

MUSE Reproduction (1/4)

Small-scale experiment



Progress Report

MUSE Reproduction (2/4)

Small-scale experiment

```
├── targets
│   ├── getQuotient
│   │   ├── getQuotient.c
│   │   └── oracle_getQuotient.c
│   ├── isEven
│   │   ├── isEven.c
│   │   └── oracle_isEven.c
│   ├── isPrime
│   │   ├── isPrime.c
│   │   └── oracle_isPrime.c
│   ├── max
│   │   ├── max.c
│   │   └── oracle_max.c
│   └── quicksort
│       ├── oracle_quicksort.c
│       └── quicksort.c
```

Generated mutants w/ mull

```
├── max
│   ├── max.c
│   ├── max.exec
│   ├── mutants
│   │   ├── mu0.c
│   │   ├── mu0.exec
│   │   ├── mu0_for_gcov.exec
│   │   ├── mu0_for_gcov.exec-mu0.gcda
│   │   ├── mu0_for_gcov.exec-mu0.gcno
│   │   ├── mu1-L4.c
│   │   ├── mu1-L4.exec
│   │   ├── mu2-L5.c
│   │   ├── mu2-L5.exec
│   │   ├── mu3-L5.c
│   │   ├── mu3-L5.exec
│   │   ├── mu4-L7.c
│   │   ├── mu4-L7.exec
│   │   ├── mu5-L7.c
│   │   ├── mu5-L7.exec
│   │   ├── mu6-L7.c
│   │   ├── mu6-L7.exec
│   │   └── oracle_max.c
│   ├── oracle_max.c
│   └── reports
```

Progress Report

MUSE Reproduction (3/4)

Program under test

```c
#include "oracle_max.c"

int setmax(int x, int y) {
    int max = -x; // should be 'max = x;'
    if (max < y) {
        max = y;
        if (x * y < 0) {
            printf("diff.sign\n");
        }
    }
    printf("max: %d\n", max);
    return max;
}

```

Analysed the test result changes

```
                (3, 1)  (5, -4) (0, -4) (0, 7)  (-1, 3)
4:    mu1       F->P    F->P

5:    mu2                       P->F    P->F    P->F
 :    mu3

7:    mu4
 :    mu5
 :    mu6
```

```
4:    mu1       P->F: 0, F->P: 2

5:    mu2       P->F: 3, F->P: 0
5:    mu3       P->F: 0, F->P: 0

7:    mu4       P->F: 0, F->P: 0
7:    mu5       P->F: 0, F->P: 0
7:    mu6       P->F: 0, F->P: 0
```

Progress Report

MUSE Reproduction (4/4)

Program under test

```
1   #include "oracle_max.c"
2
3   int
4
5
6
7
8
9
10
11
12
13  }
14
```

Ranked the statements

```
        (3, 1)  (5, -4) (0, -4) (0, 7)  (-1, 3)
4:      1       1       1       1       1       f_P(s): 2, p_P(s): 3, suspiciousness: 1.0
5:      1       1       1       1       1       f_P(s): 2, p_P(s): 3, suspiciousness: -0.4444444444444444
7:      1       1       0       1       1       f_P(s): 2, p_P(s): 2, suspiciousness: 0.0
        F       F       P       P       P
```

$$\mu(s) = \frac{1}{|mut(s)|} \sum_{m \in mut(s)} \left(\frac{|f_P(s) \cap p_m|}{|f_P|} - \alpha \cdot \frac{|p_P(s) \cap f_m|}{|p_P|} \right)$$

Analysed the test result changes

```
(3, 1)  (5, -4) (0, -4) (0, 7)  (-1, 3)
F->P    F->P                             P->F

7:      mu6     P->F: 0, F->P: 0
```

Progress Report

SBFL

```
hslee@hsmac ~/workspace/muse/mbfl-muse  main python sbfl.py
```

Line	Jaccard		Ochiai		Op2	
	Susp.	Rank	Susp.	Rank	Susp.	Rank
4	0.40	5	0.63	5	1.25	5
5	0.40	5	0.63	5	1.25	5
6	0.50	2	0.71	2	1.50	2
7	0.50	2	0.71	2	1.50	2
8	0.33	6	0.50	6	0.75	6
11	0.40	5	0.50	5	0.75	5

Mapping the journey ahead

Remaining Tasks / Challenges

Targeting Complex Problems

+ Combine IRFL Methods (if possible) to the results
+ Comparing results (bug finding capability, ranking top-n) with SOTA Fault Localization Techniques

Mapping the journey ahead

Remaining Tasks / Challenges

Targeting Complex Problems

+ Combine IRFL Methods (if possible) to the results
+ Comparing results (bug finding capability, ranking top-n) with SOTA Fault Localization Techniques

+) Thinking about better ways to combine different FL techniques

Maybe moving to another language domain (python, java) ..?!

Wrapping up

Summary

Objective: Trying to enhance Mutation Based Fault Localization

Project Introduction

Our Objective : Enhancing MBFL[MUSE (Moon et al., 2014)]

Dynamically Eliminating Statements that have a **low probability** of being buggy

*Probability is ranked with Spectrum Based Fault Localization Ranks

Cost of mutating every elements

Progress Report

SBFL

```
hslee@hsmac ~/workspace/muse/mbfl-muse  main python sbfl.py
Line |    Jaccard    |     Ochiai    |       Op2     |
     | Susp. | Rank  | Susp. | Rank  | Susp. | Rank  |
   4 |  0.40 |    5  |  0.63 |    5  |  1.25 |    5  |
   5 |  0.40 |    5  |  0.63 |    5  |  1.25 |    5  |
   6 |  0.50 |    2  |  0.71 |    2  |  1.50 |    2  |
   7 |  0.50 |    2  |  0.71 |    2  |  1.50 |    2  |
   8 |  0.33 |    6  |  0.50 |    6  |  0.75 |    6  |
  11 |  0.40 |    5  |  0.50 |    5  |  0.75 |    5  |
```

Progress Report

MUSE Reproduction (4/4)

Program under test | Analysed the test result changes

Ranked the statements

$$\mu(s) = \frac{1}{|mut(s)|} \sum_{m \in mut(s)} \left(\frac{|f_P(s) \cap p_m|}{|f_P|} - \alpha \cdot \frac{|p_P(s) \cap f_m|}{|p_P|} \right)$$

Mapping the journey ahead

Remaining Tasks / Challenges

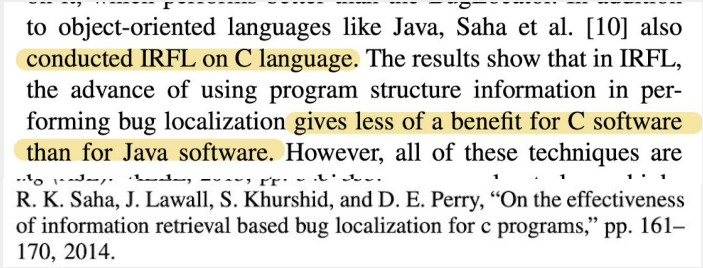
Targeting Complex Problems

+ Combine IRFL Methods (if possible) to the results
+ Comparing results (bug finding capability, ranking top-n) with SOTA Fault Localization Techniques

+) Thinking about better ways to combine different FL techniques

Maybe moving to another language domain (python, java) ..?!

Additional Slides



to object-oriented languages like Java, Saha et al. [10] also conducted IRFL on C language. The results show that in IRFL, the advance of using program structure information in performing bug localization gives less of a benefit for C software than for Java software. However, all of these techniques are

R. K. Saha, J. Lawall, S. Khurshid, and D. E. Perry, "On the effectiveness of information retrieval based bug localization for c programs," pp. 161–170, 2014.

Why we are thinking about moving to another language domain (On incorporating IRFL)

Progress Report

Subtitle

		Coverage of Test Cases (x, y)							Jaccard		Ochiai		Op2	
	`int max;` `void setmax(int x, int y){`	TC_1 (3,1)	TC_2 (5,-4)	TC_3 (0,-4)	TC_4 (0,7)	TC_5 (-1,3)	$\lvert f_P(s)\rvert$	$\lvert p_P(s)\rvert$	Susp.	Rank	Susp.	Rank	Susp.	Rank
s_1:	`max = -x; //should be 'max = x;'`	•	•	•	•	•	2	3	0.40	5	0.63	5	1.25	5
s_2:	`if(max < y){`	•	•	•	•	•	2	3	0.40	5	0.63	5	1.25	5
s_3:	`max = y;`	•	•		•	•	2	2	0.50	2	0.71	2	1.50	2
s_4:	`if(x*y<0)`	•	•		•	•	2	2	0.50	2	0.71	2	1.50	2
s_5:	`print(''diff.sign'');}`		•			•	1	1	0.33	6	0.50	6	0.75	6
s_6:	`print(max);}`	•	•	•	•	•	2	3	0.40	5	0.63	5	1.25	5
	Test Results	Fail	Fail	Pass	Pass	Pass								

			Test Result Changes						MUSE	
	Statements	Mutants	TC_1 (3,1)	TC_2 (5,-4)	TC_3 (0,-4)	TC_4 (0,7)	TC_5 (-1,3)	$\lvert f_P(s) \cap p_m\rvert$	$\lvert p_P(s) \cap f_m\rvert$	Suspiciousness	Rank
s_1:	`max = -x;`	`m1: max -= x-1;`			P→F			0	1	0.46	1
		`m2: max=x;`	F→P	F→P				2	0		
s_2:	`if(max < y){`	`m3: if(!(max<y)){`			P→F	P→F	P→F	0	3	0.09	2
		`m4: if(max==y){`	F→P			P→F		1	1		
s_3:	`max = y;`	`m5: max = -y;`				P→F	P→F	0	2	-0.16	5
		`m6: max = y+1;`				P→F	P→F	0	2		
s_4:	`if(x*y<0){`	`m7:if(!(x*y<0))`				P→F	P→F	0	2	-0.12	4
		`m8:if(x/y<0)`					P→F	0	1		
s_5:	`print(''diff.sign'');}`	`m9:return;`					P→F	0	1	-0.08	3
		`m10:;`					P→F	0	1		
s_6:	`print(max);}`	`m11:printf(0);}`				P→F	P→F	0	2	-0.20	6
		`m12:;}`			P→F	P→F	P→F	0	3		

MUSE

- Made several example codes that effectively show the strength of mutation testing
- Generated mutants w/ mull
- Analysed the test result changes
- Ranked the statements