

Do Mutations of Strongly Subsuming Second-Order Mutants Really Mask Each Other?

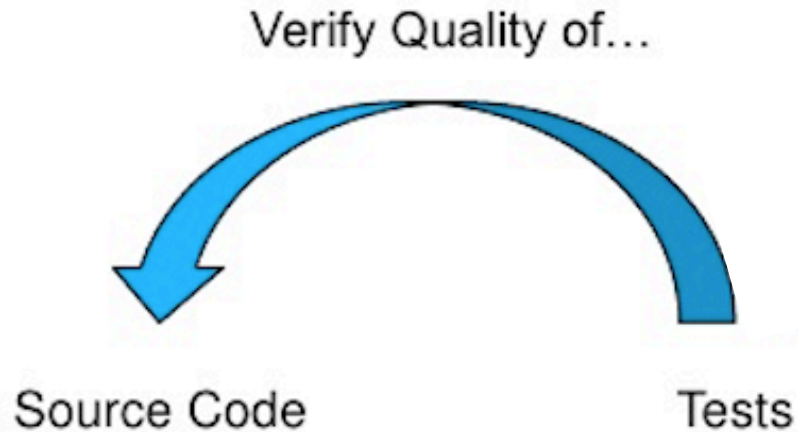
João Paulo Diniz, Fischer Ferreira,
Fabiano Ferrari, Eduardo Figueiredo

LabSoft Seminar. September 25th, 2023

Outline

- ❑ Mutation testing
- ❑ Higher-order mutants
- ❑ Mutants generation
- ❑ Strongly Subsuming Second-Order Mutants
- ❑ Study #1
- ❑ Study #2
- ❑ Related work
- ❑ Conclusion

Introduction



Mutation Testing

- Introducing **artificial changes** into the original source code
 - Syntactic changes → **mutations**
 - Changed programs → **mutants**
 - Representing real common programming mistakes
- Running test cases on mutants
 - Result different from original: mutant **killed**
 - Otherwise: **alive**

Example of a mutant

Original:

```
public class Taxes {  
  
    double simpleTax(double amount) {  
  
        return amount * 0.2;  
    }  
  
}
```

Example of a mutant

Mutation place:

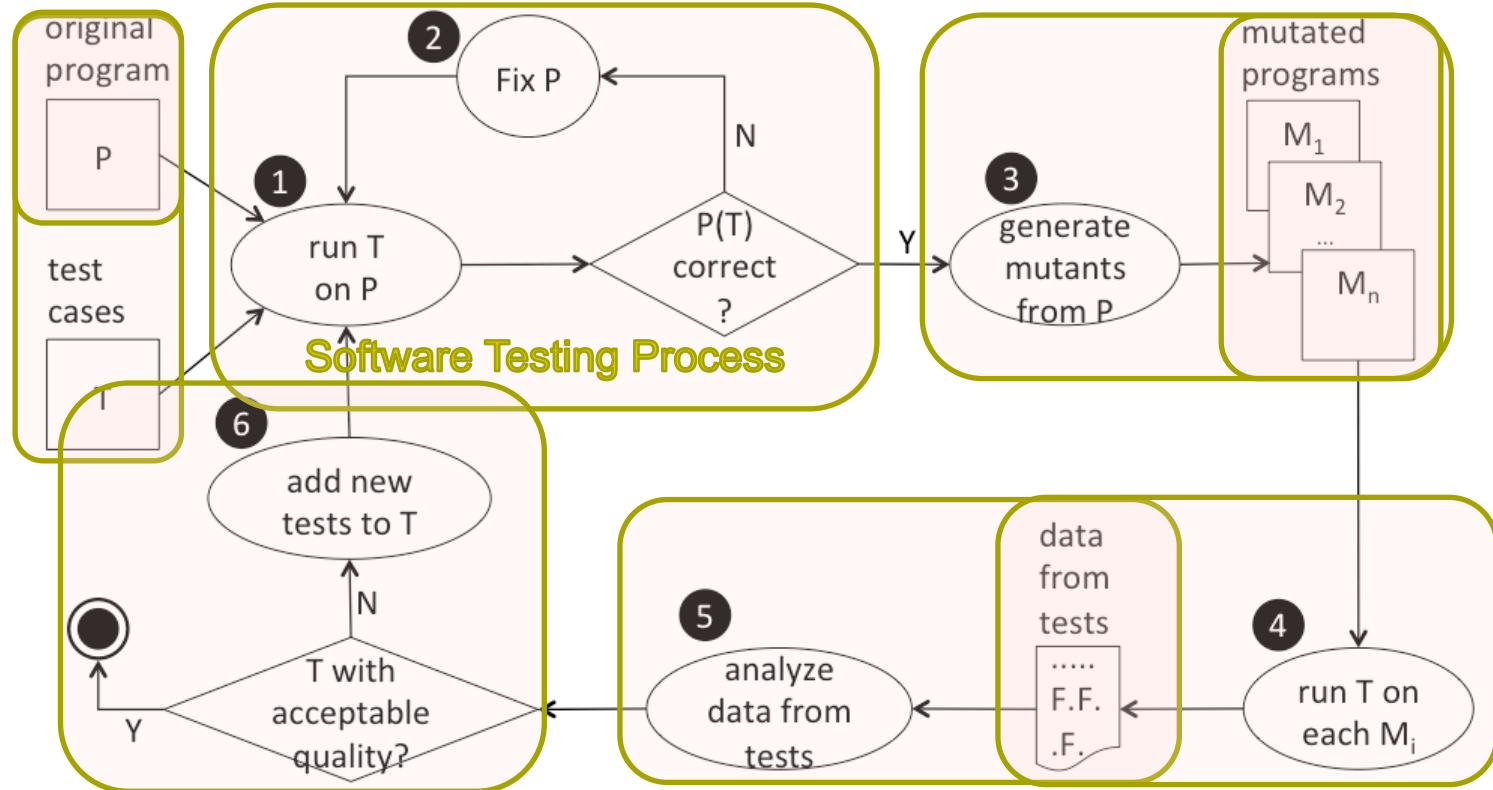
```
public class Taxes {  
  
    double simpleTax(double amount) {  
  
        return amount * 0.2;  
  
    }  
  
}
```

Example of a mutant

* → +

```
public class Taxes {  
  
    double simpleTax(double amount) {  
  
        return amount + 0.2;  
  
    }  
  
}
```

Mutation testing process



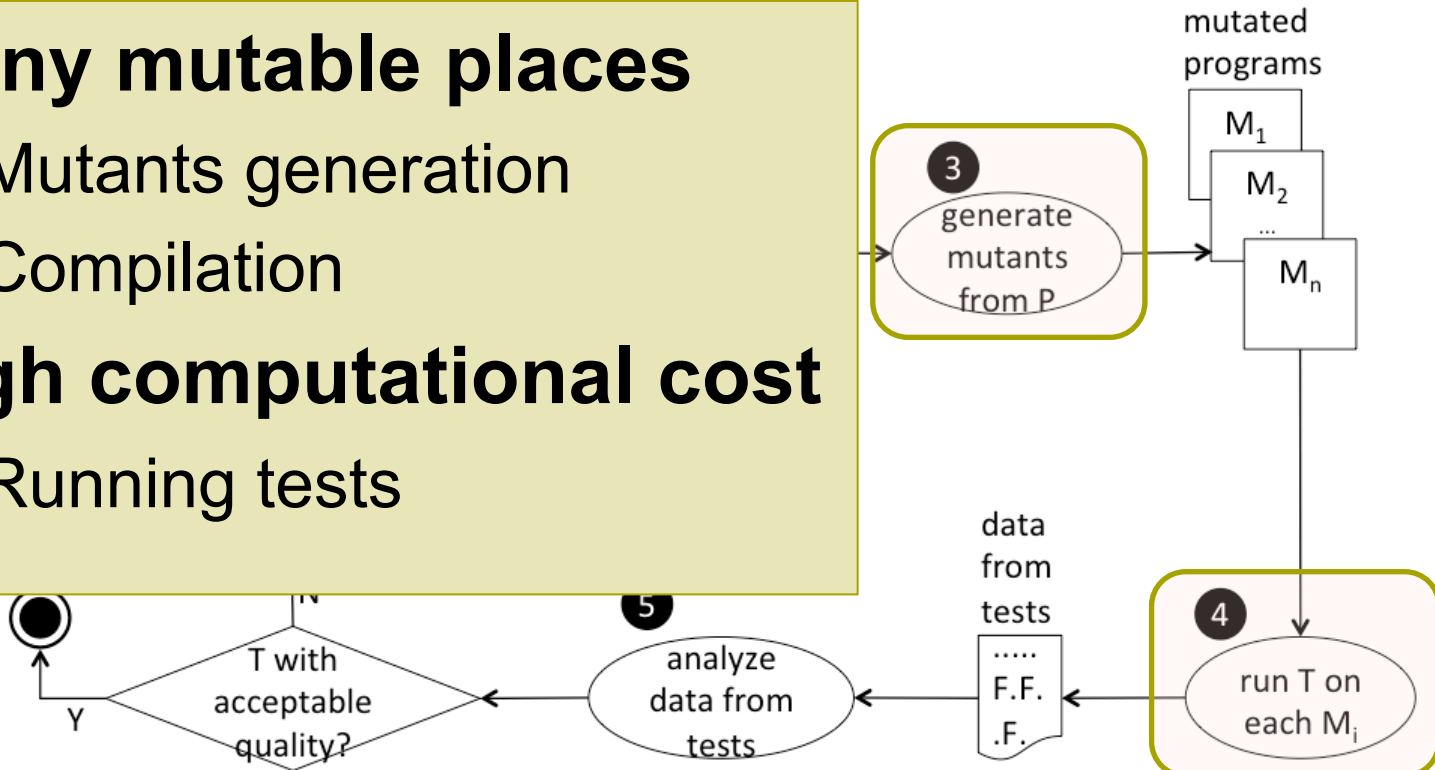
Mutation testing drawbacks

3. Many mutable places

- Mutants generation
- Compilation

4. High computational cost

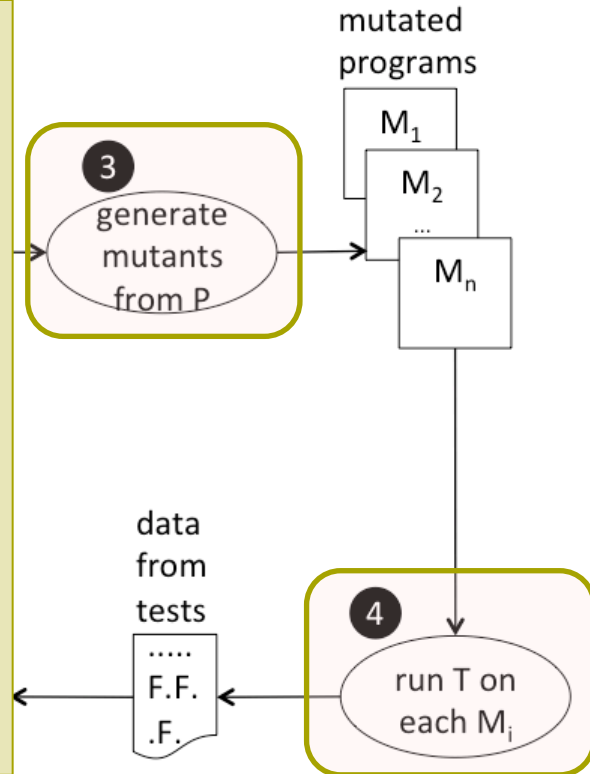
- Running tests



Mutation testing drawbacks

□ Cost reduction techniques

- Number of mutants
- Number of test cases
- Test case prioritization
- **Higher-order mutation testing**





Higher-order Mutants

Higher-order mutants

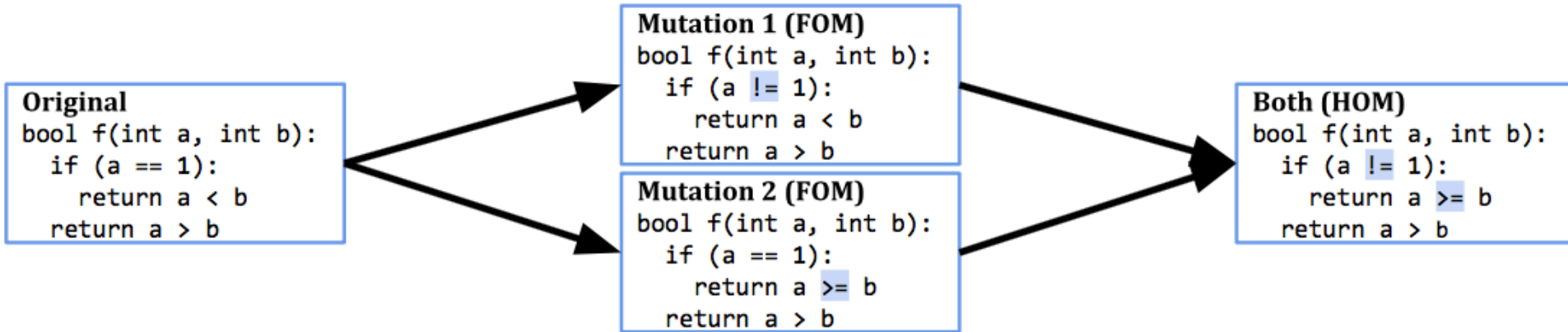
- 1 mutation → first-order mutant (**FOM**)
- > 1 mutations → higher-order mutant (**HOM**)
 - 2 mutations → 2nd order mutant (**2OM**)

- **n** mutations can generate
 - $\sim C_2^n$ 2OMs
 - $\sim C_3^n$ 3OMs
 - $\sim 2^n$ HOMs (exponential)

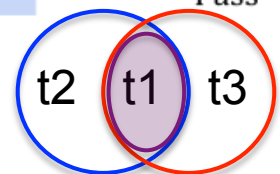
Higher-Order Mutants

- Most are **more likely to be killed** than FOMs
- But few are **harder** to be killed
 - are **more representative**
 - represent **subtler faults**
 - mutations can (partially) **mask** each other
 - Strongly Subsuming Higher-Order Mutants (**SSHOMs**)

An SS2OM example



Test	Original	Mutation 1	Mutation 2	Both
assert f(1, 2) ←t1	Pass	Fail	Fail	Fail
assert !f(0, 3) ←t2	Pass	Fail	Pass	Pass
assert !f(1, 1) ←t3	Pass	Pass	Fail	Pass



Mutants reduction - example

FOMs

m1

m2

m3

m4

m5

m6

m7

m8

Mutants reduction - example

FOMs

SS2OMs

m1

[m1,m2]

m2

[m3,m4]

m3

[m5,m6]

m4


m5

m6

m7

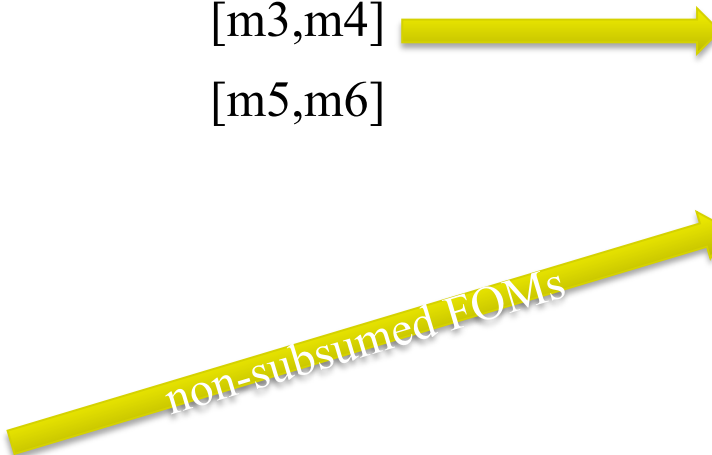
m8

Mutants reduction - example

FOMs	SS2OMs	Resulting mutants
m1	[m1,m2]	[m1,m2]
m2	[m3,m4] 	[m3,m4]
m3	[m5,m6]	[m5,m6]
m4		
m5		
m6		
m7		
m8		


Mutants reduction - example

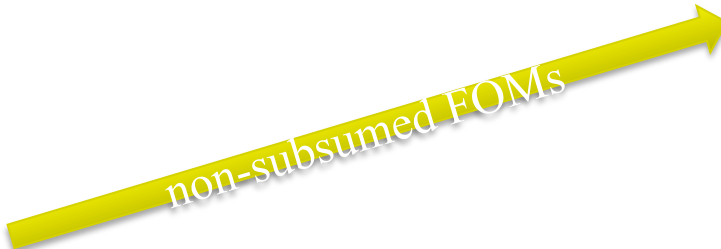
FOMs	SS2OMs	Resulting mutants
m1	[m1,m2]	[m1,m2]
m2	[m3,m4]	[m3,m4]
m3	[m5,m6]	[m5,m6]
m4		m7
m5		m8
m6		
m7		
m8		



non-subsumed FOMs

Mutants reduction - example

FOMs	SS2OMs	Resulting mutants	Reduction
m1	[m1,m2]	[m1,m2]	3 out of 8 mutants (37.5%)
m2	[m3,m4] 	[m3,m4]	
m3	[m5,m6]	[m5,m6]	
m4		m7	
m5		m8	
m6			
m7			
m8			



SSHOMs research

- Exponential search space
- Dynamic search (running test suite)
- Trends:
 - Find as many SSHOMs as possible
 - Evolutionary algorithms
 - Genetic Algorithms



Goals

Main goal

- Understand which **characteristics** make SSHOMs more difficult to kill than their constituent FOMs
- One of the explanations in the literature is that their mutations **partially mask** the others
 - Lack of studies
 - Scope limited to SS2OMs



Research Method

Dataset - Java subject systems

System	Version	LOC	# Tests	JUnit
Vending Machine	Exceptions	~100	35	4
Triangle	n/a	34	12	4
Monopoly	n/a	1,181	124	3
Commons CSV	1.8	~2k	325	4
Commons CLI	1.4	2,699	318	4
ECal	2003.10	3,626	224	3
Commons Validator	1.6	7,409	536	4
Gson	2.9.0	> 10k	1,089	3 and 4
Chess	n/a	4,924	930	3 and 4

Mutation operators implemented

Operator	Name	Examples	
		Original	Mutants
AOR	Arithmetic Operator Replacement	<code>a + b</code>	<code>a - b</code> <code>a * b</code> <code>a / b</code> <code>a % b</code>
ROR	Relational Operator Replacement	<code>a < b</code>	<code>a > b</code> <code>a <= b</code> <code>a >= b</code> <code>a == b</code> <code>a != b</code>
LCR	Logical Connector Replacement	<code>a b</code>	<code>a && b</code>
SBR	Statement Block Removal	<code>a = b+1;</code> <code>if (...) {</code> <code>...</code> <code>}</code>	<code>∅</code> <code>∅</code>

FOMs generated

System	AOR	LCR	ROR	SBR	Total
Vending Machine	4	1	40	12	57
Triangle	36	7	85	10	138
Monopoly	120	8	380	358	866
Commons CSV	208	56	250	411	925
Commons CLI	108	80	355	539	1,082
ECal	136	9	315	753	1,207
Commons Validator	324	174	1,675	1,024	3,197
Gson	536	211	1,430	1,535	3,712
Chess	1,876	244	1,890	1,309	5,287
Overall	3,348	759	6,420	5,944	16,471

SS2OMs found

System	FOMs	2OMs	SS2OMs
Vending Machine	57	1,488	25
Triangle	138	9,086	393
Monopoly	866	372,885	3,324
Commons CSV	925	425,531	4,430
Commons CLI	1,082	582,685	1,852
ECal	1,207	727,556	1,421
Commons Validator	3,197	5,102,330	17,546
Gson	3,712	6,880,416	6,970
Chess	5,287	13,952,357	8,959

44,920

Mutants reduction

System	FOMs	Resulting mutants	Mutants reduction
Vending Machine	57	49	14.04%
Triangle	138	88	36.23%
Monopoli	866	657	24.13%
Commons CSV	925	737	20.32%
Commons CLI	1,082	856	31.05%
ECal	1,213	934	22.54%
Commons Validator	3,197	2,408	24.52%
Gson	3,712	2,954	20.42%
Chess	5,319	4,234	20.38%
Overall	16,471	12,928	22.37%



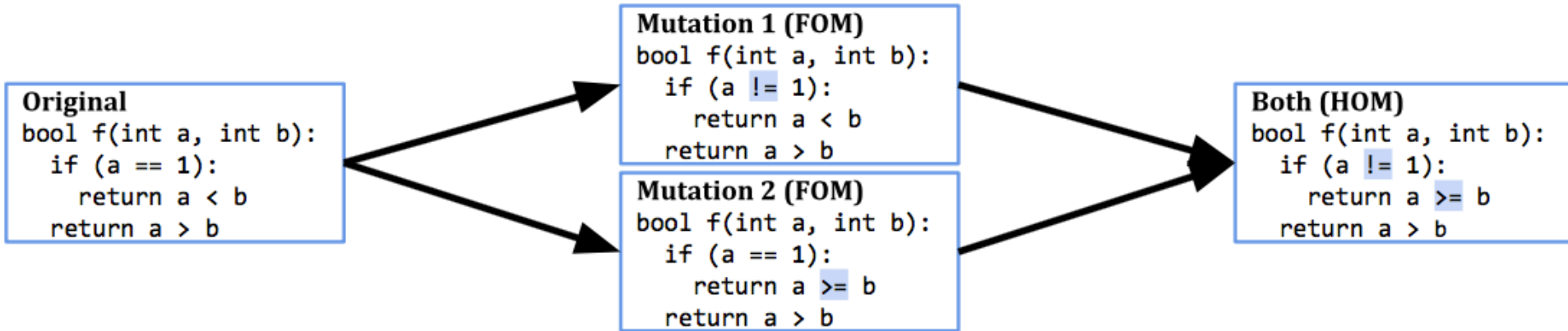
Study #1

Motivation

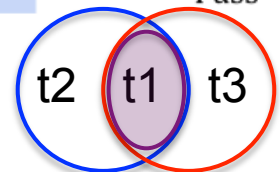
- While **manually inspecting** sampling source codes, we discovered:

For some SS2OMs, their killing tests do not reach both of their mutations

An SS2OM example



Test	Original	Mutation 1	Mutation 2	Both
assert f(1, 2) ←t1	Pass	Fail	Fail	Fail
assert !f(0, 3) ←t2	Pass	Fail	Pass	Pass
assert !f(1, 1) ←t3	Pass	Pass	Fail	Pass



SS2OMs *reaching* possibilities

SS2OMs	Killing tests size	Tests reaching both mutations	Tests reaching mutation 1	Tests reaching mutation 2
SS2OM ₁	9	9	-	-
SS2OM ₂	10	-	8	2
SS2OM ₃	6	-	6	-
SS2OM ₄	4	-	-	4
SS2OM ₅	8	4	3	1
SS2OM ₆	7	4	-	3
SS2OM ₇	5	2	3	-

} can mask

} cannot mask

Research questions

- **RQ1:** What is the **proportion** of SS2OMs so that their mutations **can mask** each other?
- **RQ2:** What is the **reduction** achieved by SS2OMs that their mutations can mask or do not mask each other?

SS2OMs *reaching* definitions

SS2OMs	Killing tests size	Tests reaching both mutations	Tests reaching mutation 1	Tests reaching mutation 2	
SS2OM ₁	9	9	-	-	} Reach _{max}
SS2OM ₂	10	-	8	2	
SS2OM ₃	6	-	6	-	} Reach _{min}
SS2OM ₄	4	-	-	4	
SS2OM ₅	8	4	3	1	} Reach _{int}
SS2OM ₆	7	4	-	3	
SS2OM ₇	5	2	3	-	

Definitions (1/3)

- mut_i : a mutation in an arbitrary source code
- FOM_i : a FOM formed by mut_i which is part of, at least, one SS2OM
- $SS2OM_{ij}$: an SS2OM formed by mut_i and mut_j ; $i \neq j$
- t_k : a test case that kills $SS2OM_{ij}$
- KT_{ij} : the killing tests of $SS2OM_{ij}$

Definitions (2/3)

The *reach function*: r

$$r(t_k, mut_i) = \begin{cases} true & \text{if } t_k \text{ reaches } mut_i \\ false & \text{otherwise} \end{cases}$$

Definitions (3/3)

$$Reach_{max_{ij}} = \forall t_k \in KS_{ij} : r(t_k, FOM_i) \wedge r(t_k, FOM_j)$$

$$Reach_{min_{ij}} = \forall t_k \in KS_{ij} : r(t_k, FOM_i) \oplus r(t_k, FOM_j)$$

$$Reach_{int_{ij}} = \exists t_k, t_l \in KS_{ij} : r(t_k, FOM_i) \wedge r(t_k, FOM_j) \wedge (r(t_l, FOM_i) \oplus r(t_l, FOM_j))$$

RQ1: SS2OMs *reaching*

System	SS2OMs	<i>Reach</i> _{max} %	<i>Reach</i> _{int} %	<i>Reach</i> _{min} %
Vending Machine	25	12	8	80
Triangle	393	8	1	91
Monopoly	3,176	65	1	34
Commons CSV	4,430	80	1	19
Commons CLI	1,066	23	4	73
ECal	1,488	48	5	47
Commons Validator	17,571	55	4	41
Gson	6,970	47	3	50
Chess	8,977	48	2	50
Overall	44,920	53,52%	3,01%	43,47%

RQ2: reductions

System	SS2OMs %	Reach _{max} %	Reach _{min} %
Vending Machine	14.04	1.75	12.28
Triangle	36.23	13.77	34.78
Monopoly	24.13	15.59	19.40
Commons CSV	20.32	15.03	15.24
Commons CLI	31.05	13.77	28.10
ECal	22.54	15.16	15.74
Commons Validator	24.52	22.71	20.21
Gson	20.42	12.45	15.76
Chess	20.38	10.81	16.41
Overall	22.37%	14.48%	17.95%



Study #2

Forced Reach Search (FRS)

- A novel search strategy for SS2OMs
- Considers a 2OM killed, only if a test case execution:
 - fails
 - reaches its both mutations

Research questions

- **RQ3:** Are the SS2OMs found by **FRS** similar to the ones found by the Exhaustive Search (**ES**), specially the $Reach_{max}$ SS2OMs?
- **RQ4:** What is the potential reduction achieved by the FRS 2OMs?

FRS SS2OMs (RQ3 & RQ4)

System	FRS 2OMs	ES \ FRS	ES \cap FRS	FRS \ ES	Reduction
Vending Machine	28	20	5	23	12.28%
Triangle	83	359	34	49	15.94%
Monopoly	8,285	1,199	2,125	6,160	34.41%
Commons CSV	9,946	1,061	3,369	6,577	29.30%
Commons CLI	15,662	1,358	494	15,168	36.23%
ECal	4,275	668	753	3,522	25.27%
Commons Validator	27,540	7,111	10,435	17,105	26.62%
Gson	79,189	3,510	3,460	75,729	30.55%
Chess	20,981	4,661	4,298	16,683	18.24%
Overall	165,989	19,947	24,973	141,016	25,75%



Next steps

Validate FRS SS2OMs

1. generate test cases for the original test suite until it kills all non-equivalent FOMs;
2. replace the subsumed FOMs with the necessary FRS SS2OMs (from Step 5);
3. select test cases from current test suite and generate new ones until it kills all resulting mutants;
4. run the new test suite against the FOMs and verify its effectiveness via the mutation score.



Questions?

