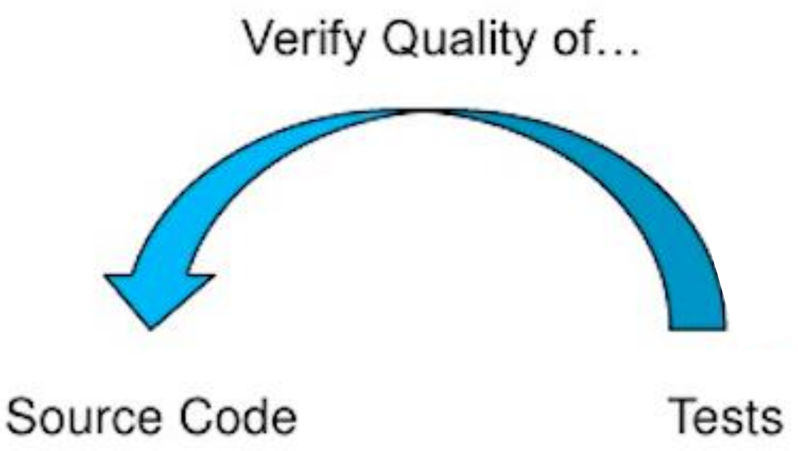
Mutant Subsumption in First- and Second-Order Mutation Testing

João Paulo Diniz

LabSoft Seminar. Mar 28th, 2025

Introduction



Mutation Testing

- Introducing **artificial syntactic changes** (**mutations**) into original source code
  - Intending to represent real common programming bugs
  - Changed programs are called **mutants**
- Running the test suite on mutants
  - Result different from original: mutant **killed**
  - Otherwise: **alive**

Example of a mutant

Mutation place:

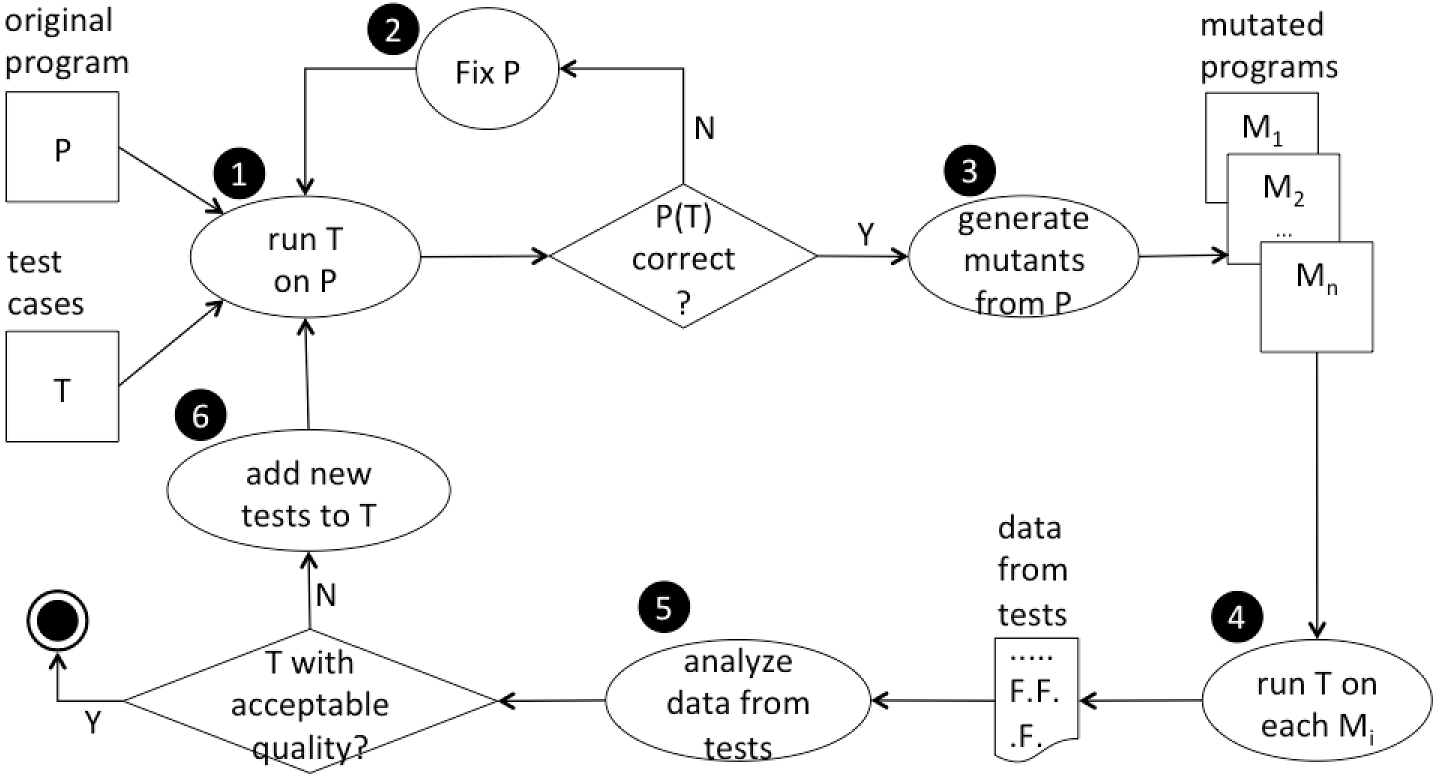
```
public class Taxes {

      double simpleTax(double amount) {

            return amount * 0.2;
      }
}
```

Example of a mutant

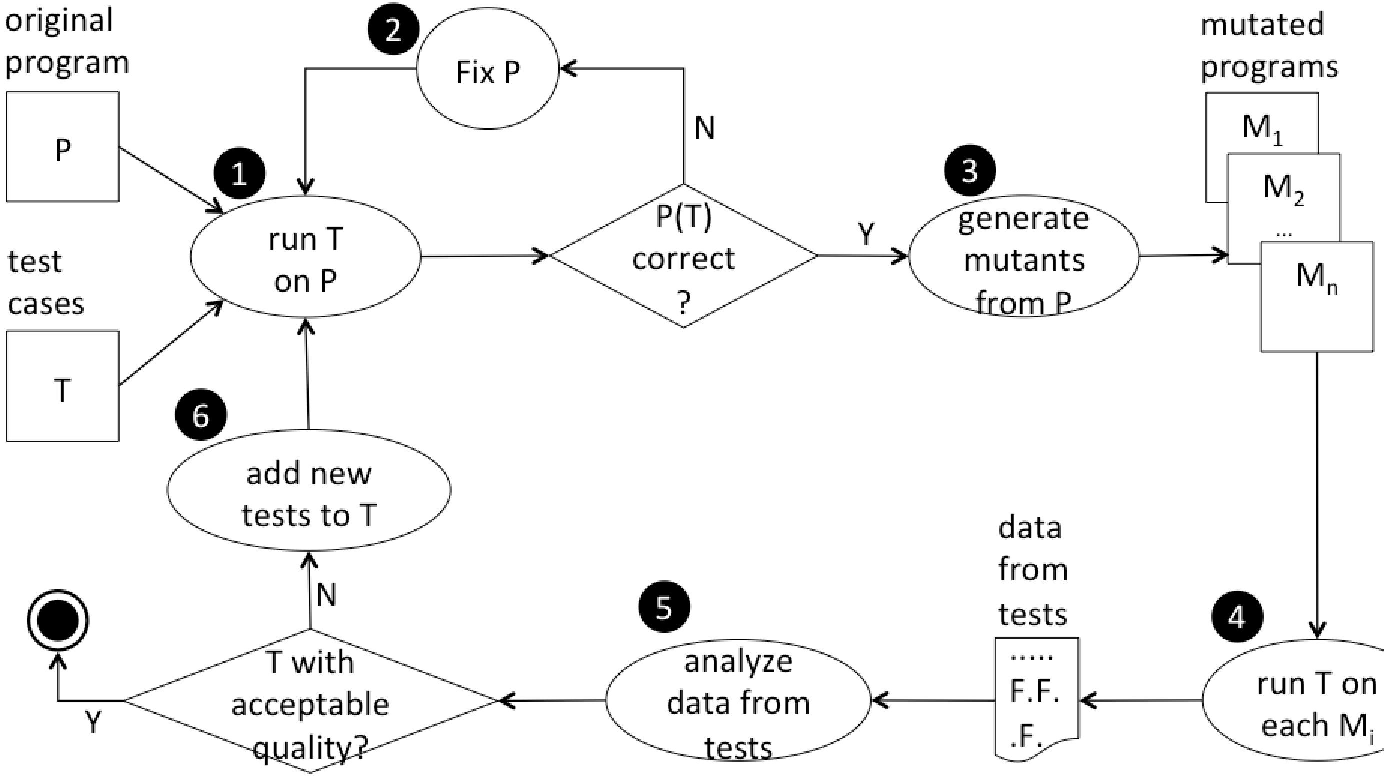
Mutation: * → +

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

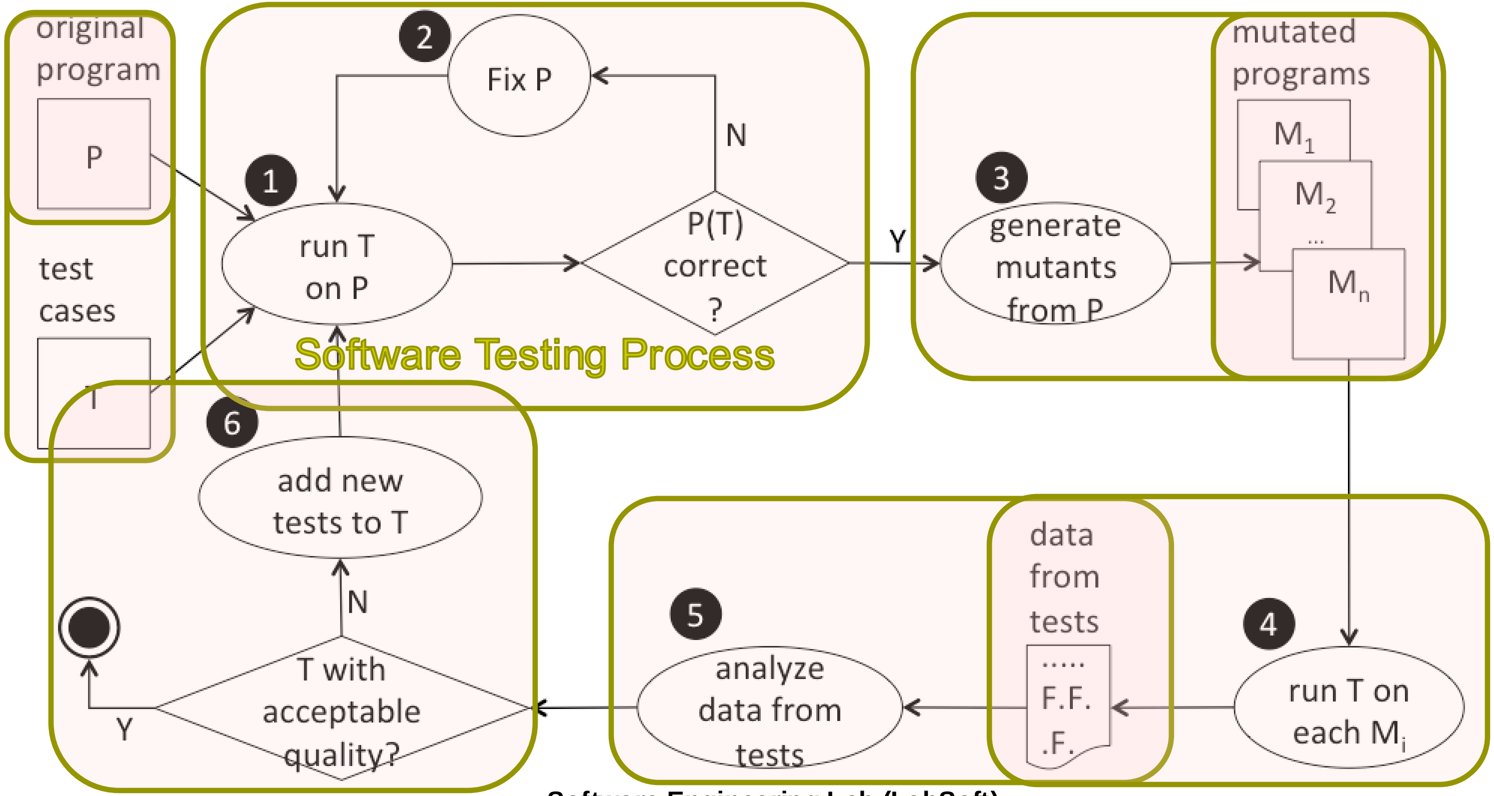
Mutation testing process



Mutation testing process

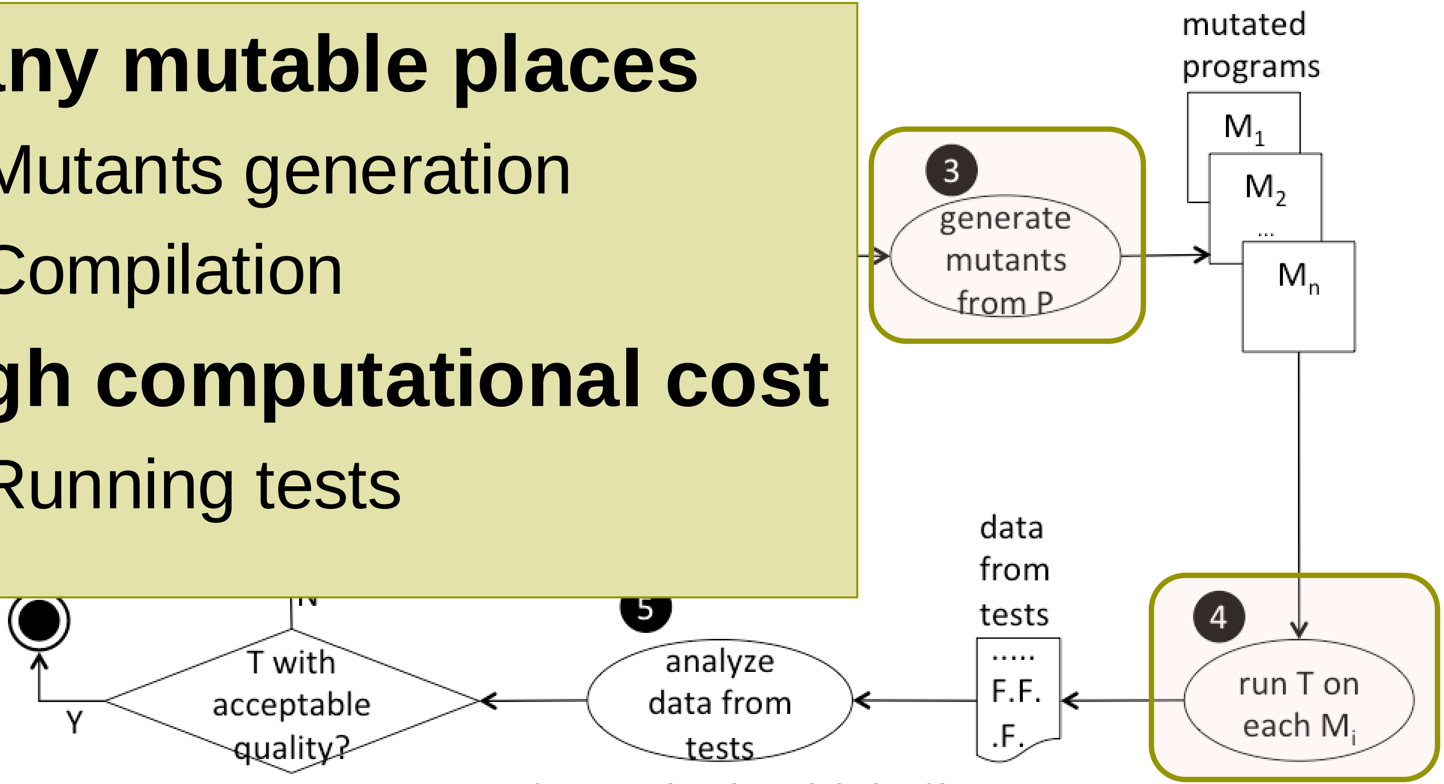


Mutation testing process



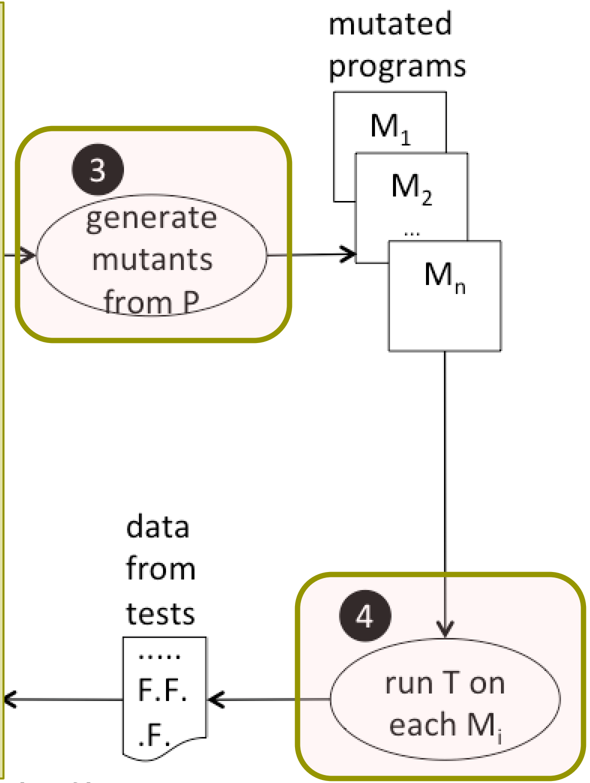
Mutation testing drawbacks

3. **Many mutable places**
   - Mutants generation
   - Compilation
4. **High computational cost**
   - Running tests



Mutation testing drawbacks

- Cost reduction techniques
  - Number of test cases
  - Test case prioritization
  - Number of mutants
    - **subsumption**



Mutants subsumption

Contextualizing

```
def greaterThan(a, b):
    return a > b   # original

def greaterThan(a, b):
    return a >= b # mutant 1

def greaterThan(a, b):
    return a <= b # mutant 2
```

Contextualizing

```
def greaterThan(a, b):
    return a > b   # original
```

	Test	orig
t1	`assertTrue(greaterThan(6, 5))`	✅
t2	`assertFalse(greaterThan(5, 5))`	✅
t3	`assertFalse(greaterThan(5, 6))`	✅

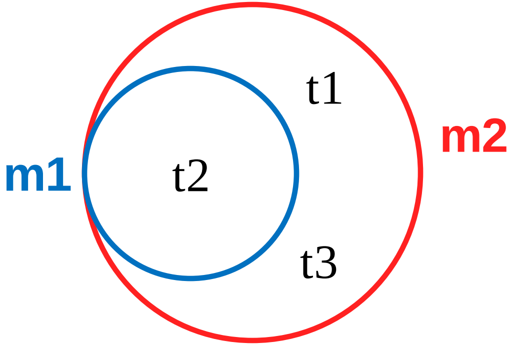
Contextualizing

```
def greaterThan(a, b):
    return a > b   # original
    return a >= b # mutant 1
    return a <= b # mutant 2
```

	Test	orig	m1	m2
t1	`assertTrue(greaterThan(6, 5))`	✅	✅	❌
t2	`assertFalse(greaterThan(5, 5))`	✅	❌	❌
t3	`assertFalse(greaterThan(5, 6))`	✅	✅	❌

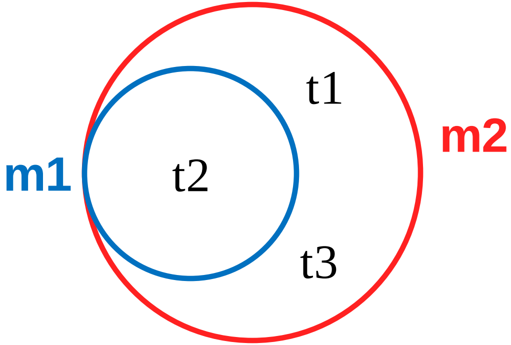
Contextualizing

- Killing tests



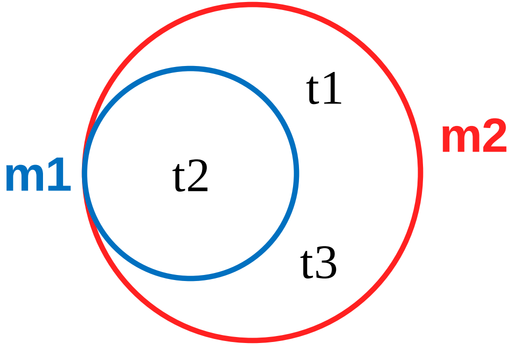
Contextualizing

- All test sets that kill **m1** also kill **m2**



Definition

- **m1 subsumes m2**



In summary

- If we know beforehand that
  - **m1** subsumes **m2**
- Therefore,
  - **m2** should not have been generated

Cost reduction: fewer mutants to run the test suite against

Dynamic mutant subsumption graphs

Example

test	m1	m2	m3	m4	m5
t1	✕	✕		✕	✕
t2	✕		✕	✕	
t3				✕	
t4		✕		✕	✕

Subsumption relationships

test	m1	m2	m3	m4	m5
t1	✕	✕		✕	✕
t2	✕		✕	✕	
t3				✕	
t4		✕		✕	✕

m1 → m4

m2 → m4

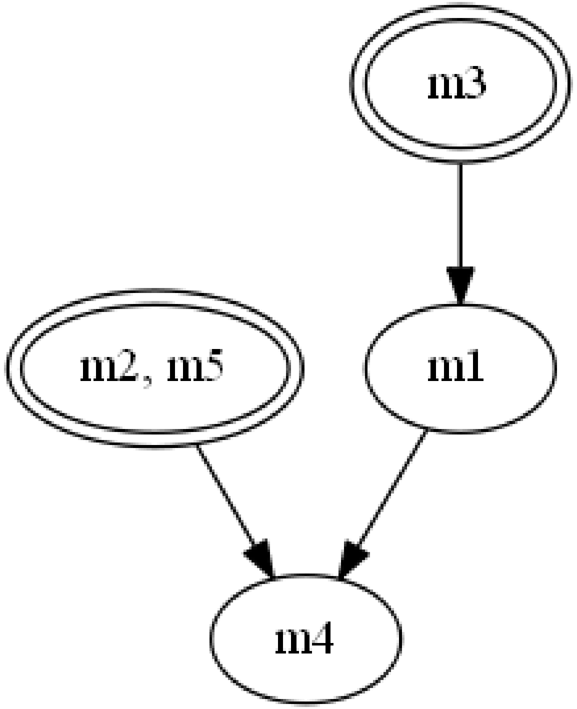
m3 → m1

m3 → m4

m5 → m4

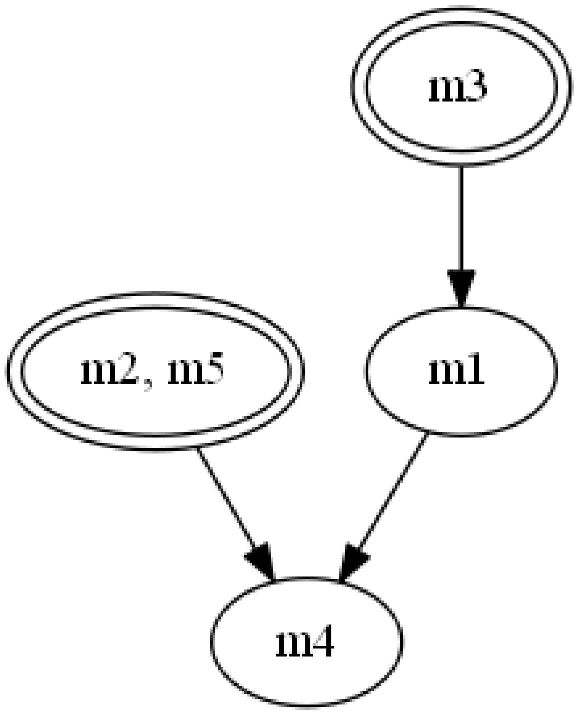
Subsumption graph

Test	m1	m2	m3	m4	m5
t1	✕	✕		✕	✕
t2	✕		✕	✕	
t3				✕	
t4		✕		✕	✕



Conclusion

- **Root nodes are kept**
  - 2 minimal
  - 3 mutants
- Remaining nodes
  - are disregarded
  - (redundants)



Second-order mutants subsumption

A 2OM subsumption example

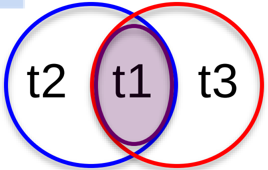
Original
```
bool f(int a, int b):
  if (a == 1):
    return a < b
  return a > b
```

Mutation 1 (FOM)
```
bool f(int a, int b):
  if (a != 1):
    return a < b
  return a > b
```

Mutation 2 (FOM)
```
bool f(int a, int b):
  if (a == 1):
    return a >= b
  return a > b
```

Both (HOM)
```
bool f(int a, int b):
  if (a != 1):
    return a >= b
  return a > b
```

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



A 2OM subsumption example

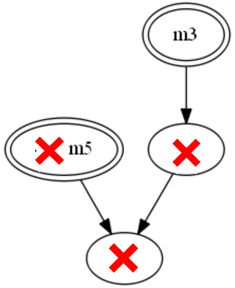
Strongly subsuming second-order mutant

(SS2OM)

An SS2OM can replace their constituent FOMs without loss of effectiveness

Mutants reduction via subsumption

From FOMs

- Remove all FOMs in subsumed nodes
- Randomly pick one FOM in each minimal node
- 

2 out of 5 FOMs → 60%

From SS2OMs

FOMs

m1

m2

m3

m4

m5

m6

m7

m8

From SS2OMs

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

From SS2OMs

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

From SS2OMs

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

non-subsumed FOMs

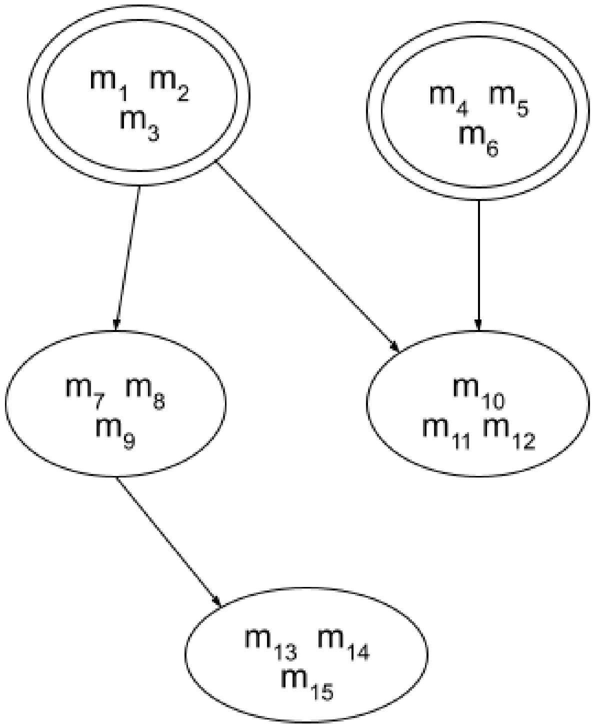
Study design

RQ1

- Which *subsumption* can reduce more mutants?
  - From FOMs
  - From SS2OMs

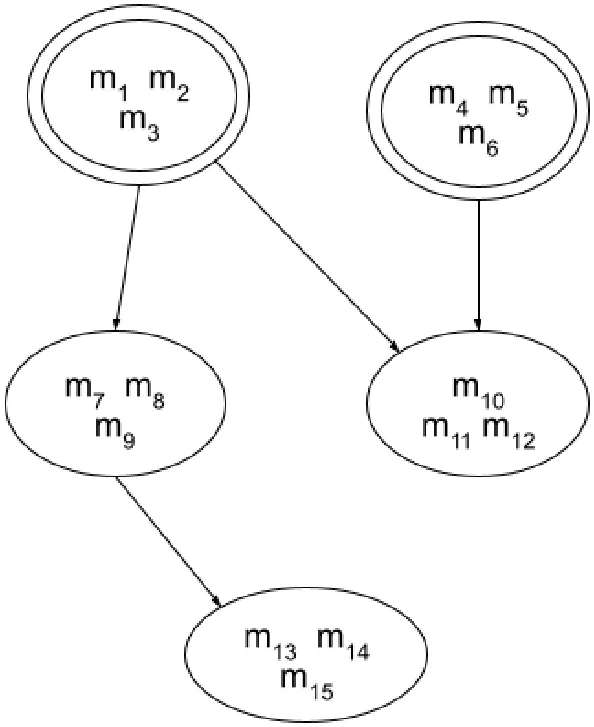
RQ2

- What is the distribution of the SS2OMs **constituent FOMs** in the subsumption graphs?



RQ2 – example of SS2OMs

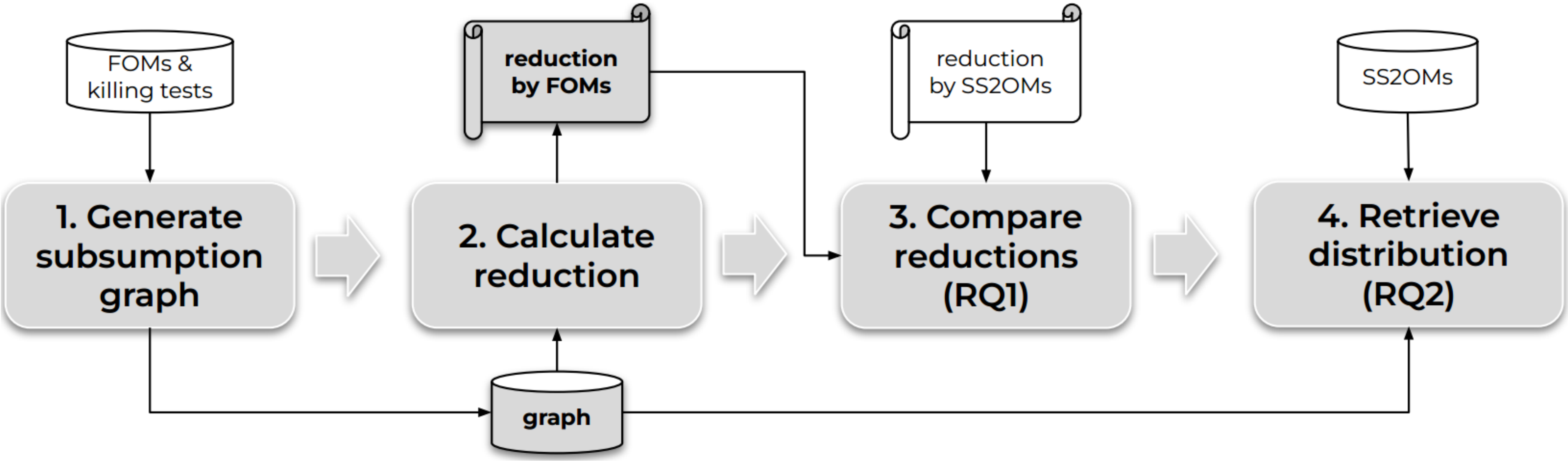
- $[m_1, m_3]$: same minimal node
- $[m_1, m_4]$: ≠ minimal nodes
- $[m_4, m_7]$: 1 minimal, 1 subsumed
- $[m_{10}, m_{11}]$: same subsumed nodes
- $[m_{10}, m_{13}]$: ≠ subsumed nodes



Dataset: 9 Java systems

System	LOC	# Tests	#FOMs	#SS2OMs
Vending Machine	~100	35	57	25
Triangle	34	12	138	393
Monopoly	1,181	124	866	3,324
Commons CSV	~2k	325	925	4,430
Commons CLI	2,699	318	1,082	1,852
ECal	3,626	224	1,207	1,421
Commons Validator	7,409	536	3,197	17,546
Gson	> 10k	1,089	3,712	6,970
Chess	4,924	930	5,287	8,959
Overall			**16,471**	**44,960**

Study steps



FOMs &
killing tests
reduction
by FOMs
reduction
by SS2OMs
SS2OMs
1. Generate
subsumption
graph
2. Calculate
reduction
3. Compare
reductions
(RQ1)
4. Retrieve
distribution
(RQ2)
graph

Results

Reduction comparison (RQ1)

RQ1 answer

Overall	#FOMs	#minimal nodes	#remaining FOMs
9 systems	16,471	1,115	3,376

Highlight on Triangle

{91}
{65, 67, 68, 70, 75, 77, 80, 52, 85, 56, 57, 58, 59, 63}
{35, 37, 38, 39, 108, 112, 114, 115, 116, 118, 119}
{129, 130, 131, 46, 122, 123, 124, 127}
{11}
{62}
{79}
{5}
{96, 132, 136, 121, 106, 111}
{69}
{97, 99, 100, 101, 103, 104, 23, 24, 25, 26, 27, 28, 93}
{0}

#FOMs	#minimal nodes	#remaining FOMs
138	12	59

RQ1 answer

Overall	FOMs	Via subsumption graph	Via SS2OMs
9 systems	16,471	77.35%	22.37%

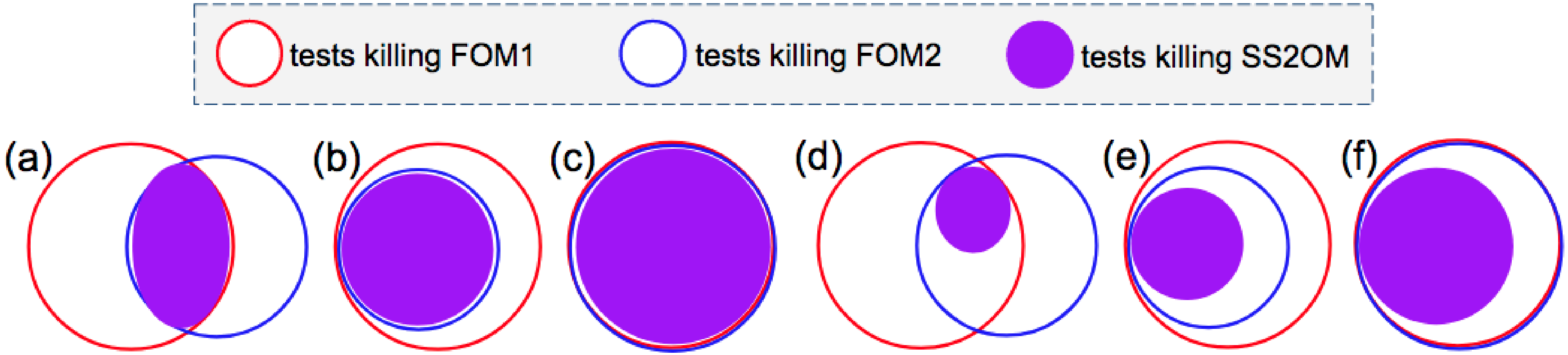
SS2OMs distribution (RQ2)

RQ2 answer

44,960 SS2OMs &
their constituent FOMs **distribution** over
the subsumption graphs

Overall	Same minimal	1 minimal 1 subsumed	Same subsumed	≠ subsumed
9 systems	20,368	1,968	18,298	4,286
	45.3%	4.4%	40.7%	9.6%

A previous result



1.1% 13.1% **84.4%** 0.03% 0.08% 0.57%

RQ2 answer

44,960 SS2OMs &
their constituent FOMs **distribution** over
the subsumption graphs

Overall	Same minimal	≠ minimal	1 minimal 1 subsumed	Same subsumed	≠ subsumed
9 systems	20,368	0	1,968	18,298	4,286
	45.3%	0%	4.4%	40.7%	9.6%

Threats to validity

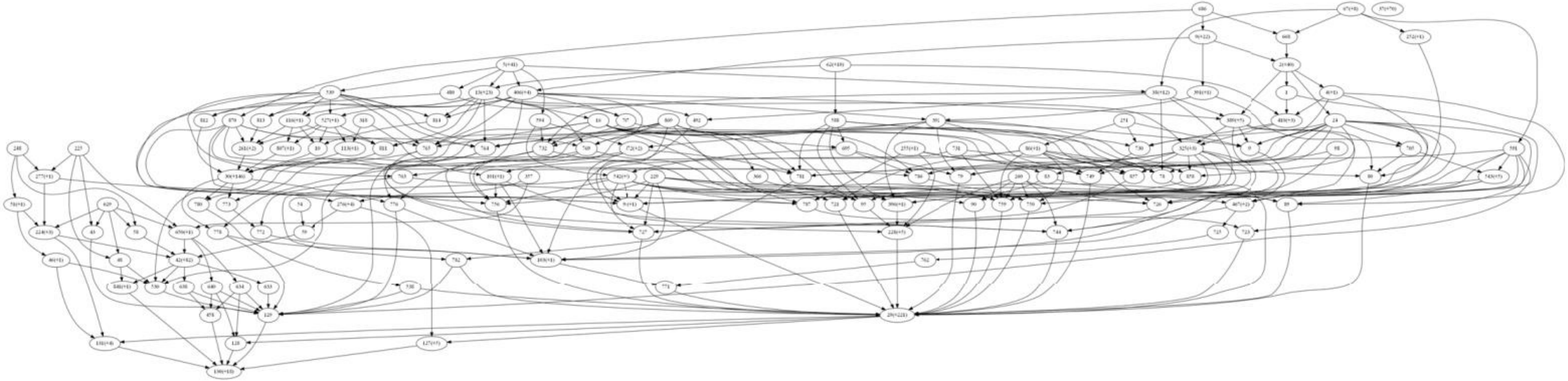
Threats to validity

- External
  - Results may not generalize
- Internal
  - Possible incorrect data from dataset
- Construction
  - Subsumption graphs correct?

Related work

Kurtz *et al*. 2014

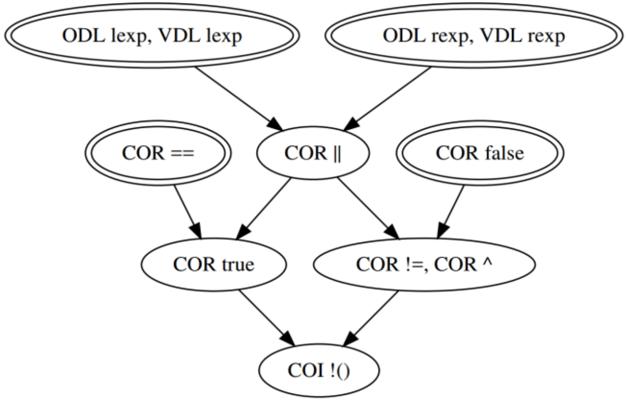
- Mutant Subsumption Graphs



Guimarães *et al*. 2020

- Optimizing Mutation Testing by Discovering Dynamic Mutant Subsumption Relations

Target	Minimal Set	Reduction
lexp + rexp	AORB %, ODL lexp, ODL rexp	62.5%
lexp + rexp (obj)	ODL lexp, ODL rexp	50.0%
lexp − rexp	AORB %, ODL lexp, ODL rexp	62.5%
lexp * rexp	AORB /, ODL lexp, ODL rexp	62.5%
lexp / rexp	AORB %, AOIS *, ODL rexp	62.5%
lexp % rexp	AORB +, AORB −, AORB /, ODL lexp	50.0%
lexp > rexp	ROR false, ROR !=, ROR >=	62.5%
lexp >= rexp	ROR true, ROR ==, ROR >	62.5%
lexp < rexp	ROR false, ROR !=, ROR <=	62.5%
lexp <= rexp	ROR true, ROR ==, ROR <	62.5%
lexp == rexp	ROR false, ROR <=, ROR >=	62.5%
lexp == rexp (obj)	ROR !=	50.0%
lexp == rexp (bool)	ROR !=, ODL lexp, ODL rexp	50.0%
lexp != rexp	ROR true, ROR <, ROR >	62.5%
lexp != rexp (obj)	ROR ==	50.0%
lexp != rexp (bool)	ROR ==, ODL lexp, ODL rexp	50.0%
lexp && rexp	COR false, COR ==	81.8%
lexp || rexp	COR true, COR !=	81.8%
lexp & rexp	ODL lexp, ODL rexp	66.7%
lexp | rexp	ODL lexp, ODL rexp, LOR ^	66.7%
lexp ˆ rexp	LOR |	83.3%
lexp ˆ rexp (bool)	COR false, COR ||	80.0%
exp	AOIU -exp	83.3%
+exp	LOI exp	75.0%
!exp	COD exp	50.0%
-exp	AODU exp	66.7%
~exp	LOD exp	75.0%
	AODS exp	75.0%
	LOI exp	75.0%
	AODS exp	75.0%
	LOI exp	75.0%
	ASRS −=, ASRS %=, ODL =	50.0%
	ASRS +=, ASRS %=, ODL =	50.0%
	ASRS /=, ASRS %=, ODL =	50.0%
	ASRS *=, ASRS %=, ODL =	50.0%
	ASRS +=, ASRS −=, ASRS *=, ASRS /=	33.3%
	ASRS >>=,	66.7%
	ASRS <<=, ODL =, SDL	0.0%
	ASRS >>=, ASRS <<=	50.0%
	ODL =, SDL	50.0%
	ODL =, ASRS ^=, SDL	25.0%
	ASRS |=	75.0%



Conclusion

Main finding

SS2OMs **do not seem to contribute to any further reduction** than the one achieved by the subsuming FOMs.

Future work

- Analyze SS3OMs
- Formal proof
  - SS2OMs constituent FOMs cannot belong to same minimal node

lab-soft
software engineering laboratory

Software Engineering Lab (LabSoft)
http://labsoft.dcc.ufmg.br/