Do Redundant Mutants Affect the Effectiveness and Efficiency of Mutation Analysis?

René Just[1] & Gregory M. Kapfhammer[2] & Franz Schweiggert[1]

[1]Ulm University, Germany
[2]Allegheny College, USA

7th International Workshop on Mutation Analysis
Montreal, Canada
April 17, 2012

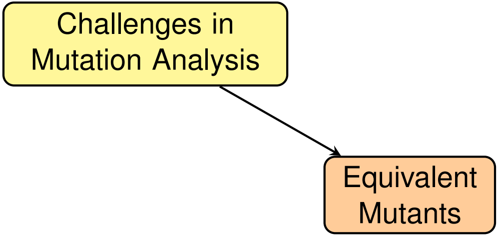
ulm university universität ulm

Allegheny College

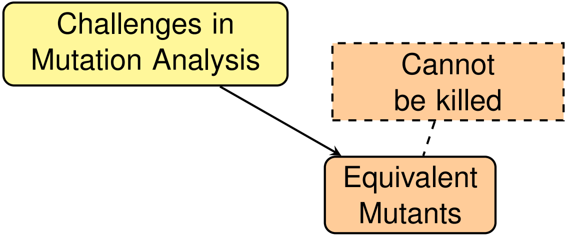
Overview of the Presentation

Challenges in Mutation Analysis

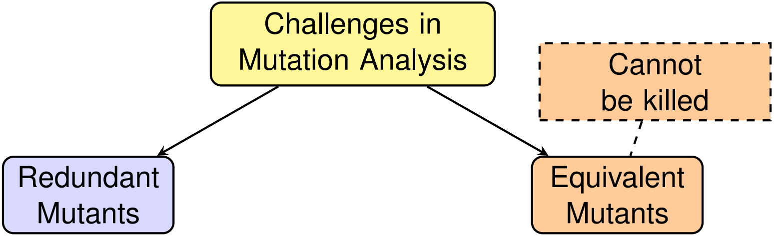
Overview of the Presentation



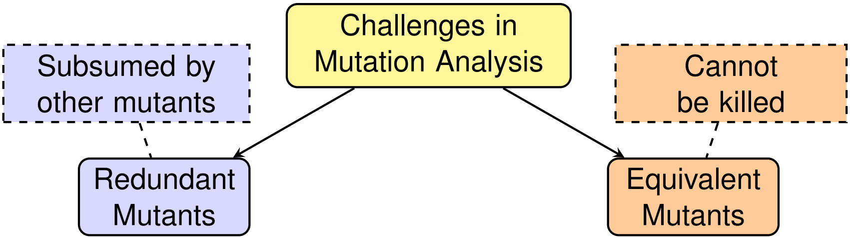
Overview of the Presentation



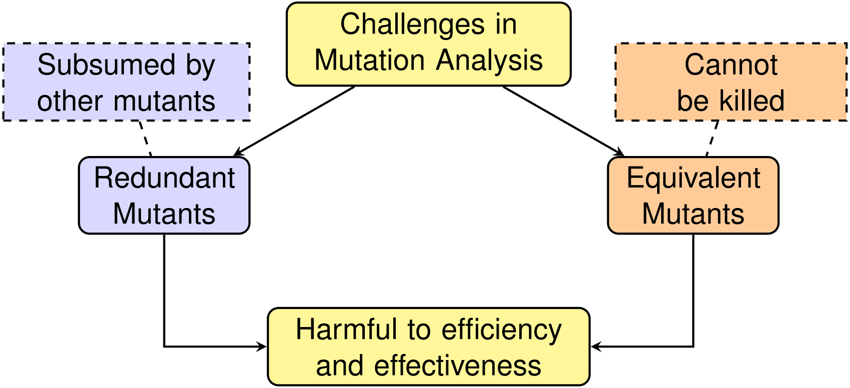
Overview of the Presentation



Overview of the Presentation



Overview of the Presentation



Overview of the Presentation

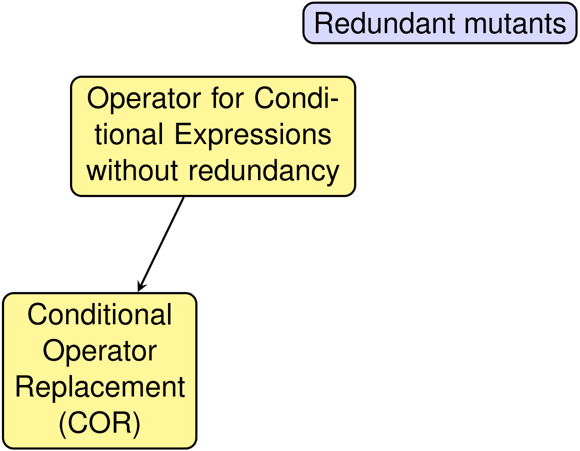
Redundant mutants

Overview of the Presentation

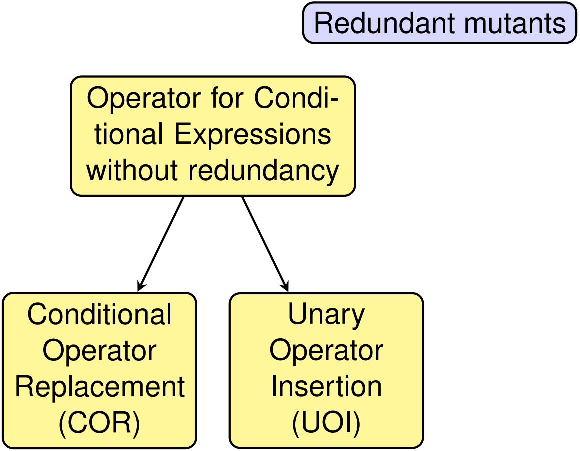
Redundant mutants

Operator for Conditional Expressions without redundancy

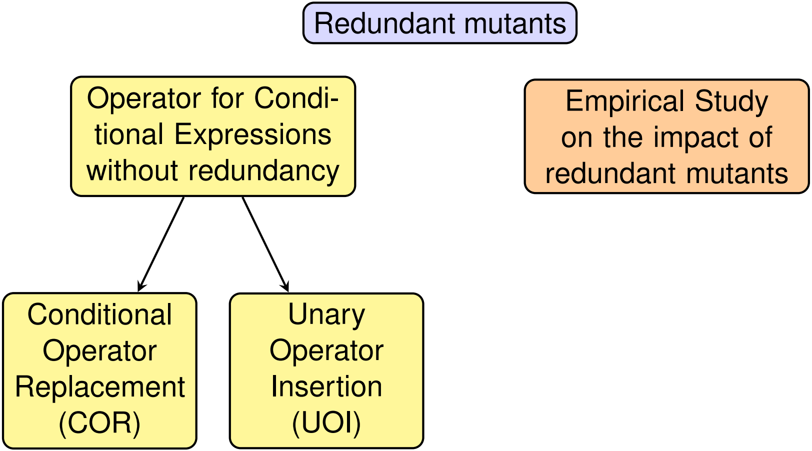
Overview of the Presentation



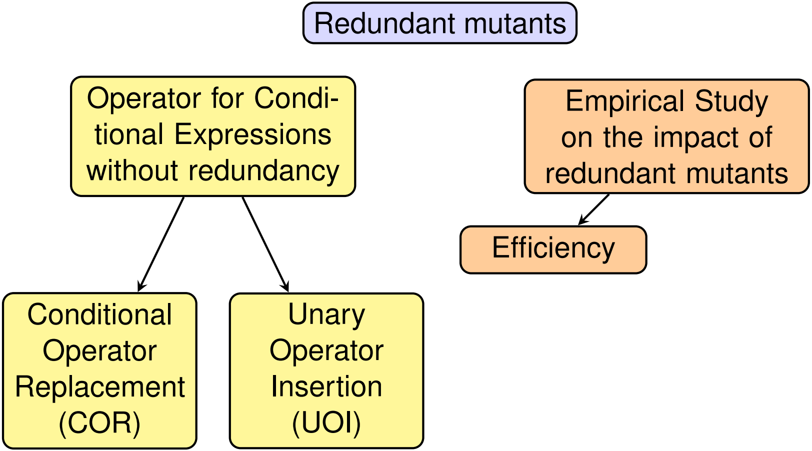
Overview of the Presentation



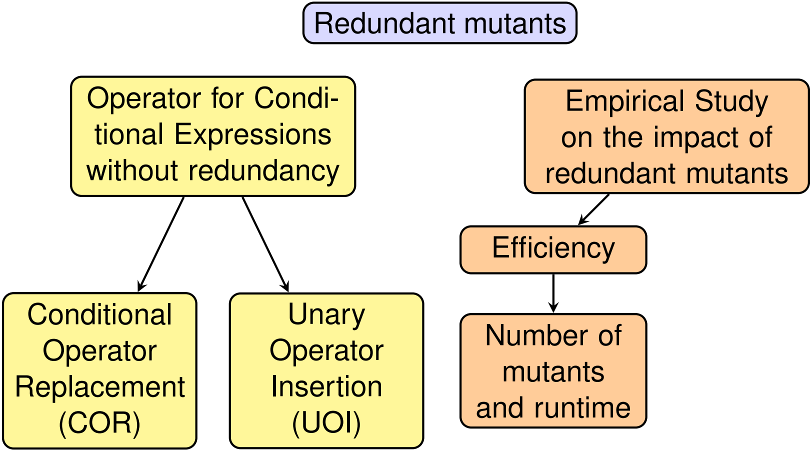
Overview of the Presentation



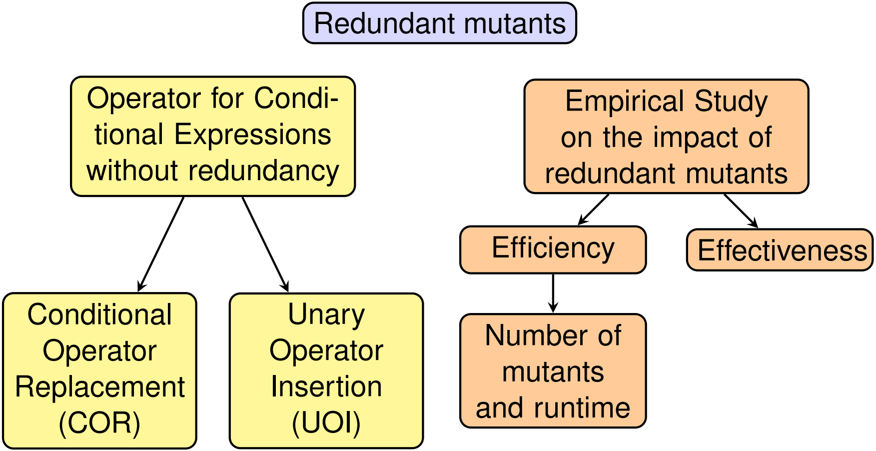
Overview of the Presentation



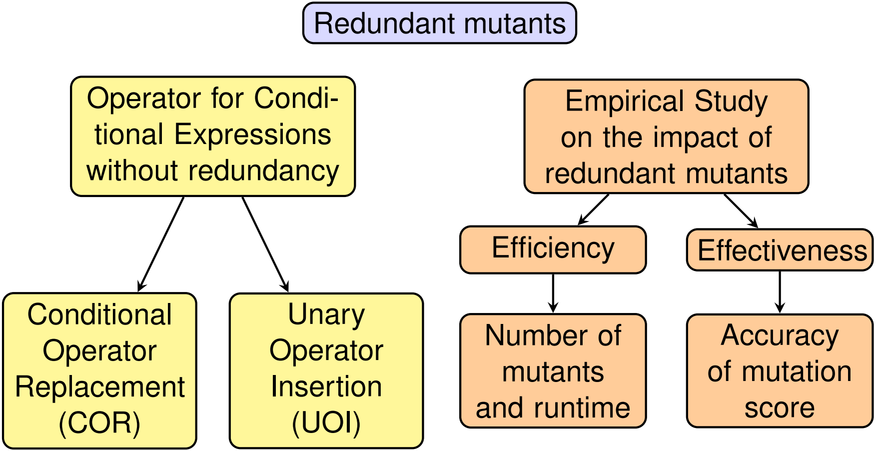
Overview of the Presentation



Overview of the Presentation



Overview of the Presentation

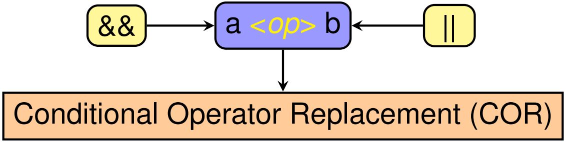


Mutating Conditional Expressions

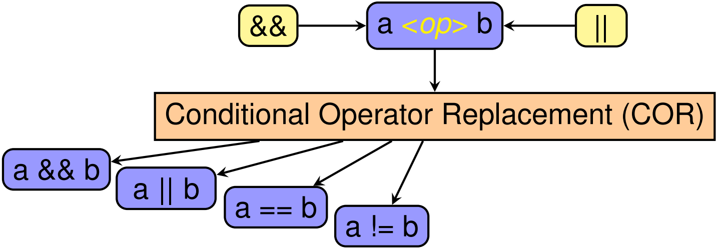
a <*op*> b

Mutating Conditional Expressions

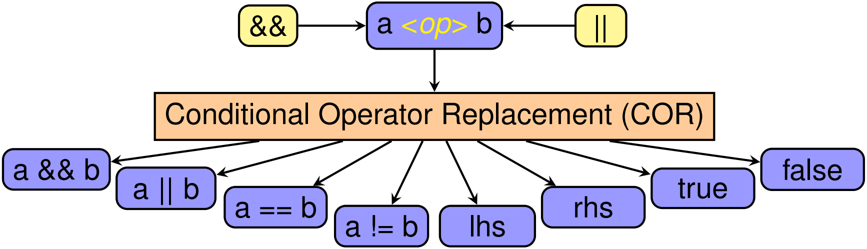
Mutating Conditional Expressions



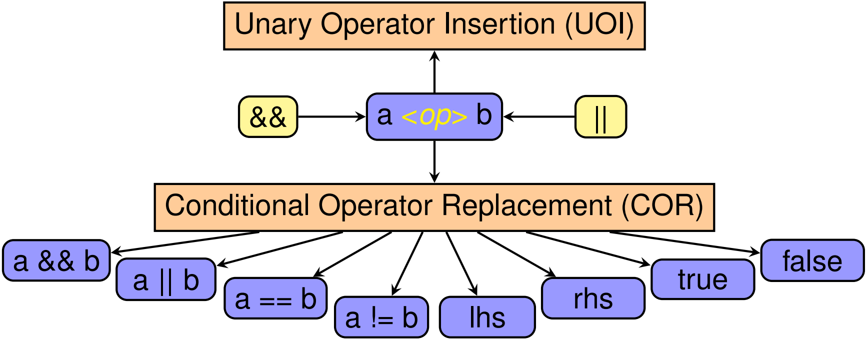
Mutating Conditional Expressions



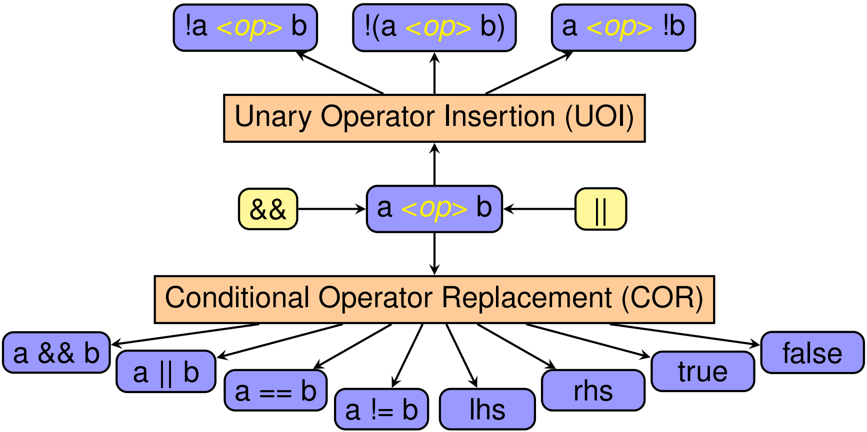
Mutating Conditional Expressions



Mutating Conditional Expressions



Mutating Conditional Expressions



Mutating Conditional Expressions

Literals		Expression
a	b	a && b
0	0	0
0	1	0
1	0	0
1	1	1

Literals		Expression
a	b	a || b
0	0	0
0	1	1
1	0	1
1	1	1

Mutating Conditional Expressions

Literals		Expression										
a	b	a && b	false	lhs	rhs	a == b	a || b	a != b	true	!a && b	!(a && b)	a && !b
0	0	0	0	0	0	1	0	0	1	0	1	0
0	1	0	0	0	1	0	1	1	1	1	1	0
1	0	0	0	1	0	0	1	1	1	0	1	1
1	1	1	0	1	1	1	1	0	1	0	0	0

Literals		Expression										
a	b	a || b	a != b	rhs	lhs	true	a && b	a == b	false	!a || b	!(a || b)	a || !b
0	0	0	0	0	0	1	0	1	0	1	1	1
0	1	1	1	1	0	1	0	0	0	1	0	0
1	0	1	1	0	1	1	0	0	0	0	0	1
1	1	1	0	1	1	1	1	1	0	1	0	1

Mutating Conditional Expressions

Literals		**Expression**	**Sufficient mutations**				**Subsumed mutations**			**Subsumed operator UOI**		
a	b	a && b	false	lhs	rhs	a == b	a || b	a != b	true	!a && b	!(a && b)	a && !b
0	0	0	0	0	0	1	0	0	1	0	1	0
0	1	0	0	0	1	0	1	1	1	1	1	0
1	0	0	0	1	0	0	1	1	1	0	1	1
1	1	1	0	1	1	1	1	0	1	0	0	0

Literals		**Expression**	**Sufficient mutations**				**Subsumed mutations**			**Subsumed operator UOI**		
a	b	a || b	a != b	rhs	lhs	true	a && b	a == b	false	!a || b	!(a || b)	a || !b
0	0	0	0	0	0	1	0	1	0	1	1	1
0	1	1	1	1	0	1	0	0	0	1	0	0
1	0	1	1	0	1	1	0	0	0	0	0	1
1	1	1	0	1	1	1	1	1	0	1	0	1

Mutating Conditional Expressions

4 Mutants are sufficient

Literals		Expression	Sufficient mutations				Subsumed mutations			Subsumed operator UOI		
a	b	a && b	false	lhs	rhs	a == b	a || b	a != b	true	!a && b	!(a && b)	a && !b
0	0	0	0	0	0	1	0	0	1	0	1	0
0	1	0	0	0	1	0	1	1	1	1	1	0
1	0	0	0	1	0	0	1	1	1	0	1	1
1	1	1	0	1	1	1	1	0	1	0	0	0

Literals		Expression	Sufficient mutations				Subsumed mutations			Subsumed operator UOI		
a	b	a || b	a != b	rhs	lhs	true	a && b	a == b	false	!a || b	!(a || b)	a || !b
0	0	0	0	0	0	1	0	1	0	1	1	1
0	1	1	1	1	0	1	0	0	0	1	0	0
1	0	1	1	0	1	1	0	0	0	0	0	1
1	1	1	0	1	1	1	1	1	0	1	0	1

Mutating Conditional Expressions

4 Mutants are sufficient

UOI Operator completely subsumed

Literals		Expression	Sufficient mutations				Subsumed mutations			Subsumed operator UOI		
a	b	a && b	false	lhs	rhs	a == b	a || b	a != b	true	!a && b	!(a && b)	a && !b
0	0	0	0	0	0	1	0	0	1	0	1	0
0	1	0	0	0	1	0	1	1	1	1	1	0
1	0	0	0	1	0	0	1	1	1	0	1	1
1	1	1	0	1	1	1	1	0	1	0	0	0

Literals		Expression	Sufficient mutations				Subsumed mutations			Subsumed operator UOI		
a	b	a || b	a != b	rhs	lhs	true	a && b	a == b	false	!a || b	!(a || b)	a || !b
0	0	0	0	0	0	1	0	1	0	1	1	1
0	1	1	1	1	0	1	0	0	0	1	0	0
1	0	1	1	0	1	1	0	0	0	0	0	1
1	1	1	0	1	1	1	1	1	0	1	0	1

Mutating Conditional Expressions

4 Mutants are sufficient

UOI Operator completely subsumed

Literals		Expression	Sufficient mutations				Subsumed mutations			Subsumed operator UOI		
a	b	a && b	false	lhs	rhs	a == b	a || b	a != b	true	!a && b	!(a && b)	a && !b
0	0	0	0	0	0	1	0	0	1	0	1	0
0	1	0	0	0	1	0	1	1	1	1	1	0
1	0	0	0	1	0	0	1	1	1	0	1	1
1	1	1	0	1	1	1	1	0	1	0	0	0

Literals		Expression	Sufficient mutations				Subsumed mutations			Subsumed operator UOI		
a	b	a || b	a != b	rhs	lhs	true	a && b	a == b	false	!a || b	!(a || b)	a || !b
0	0	0	0	0	0	1	0	1	0	1	1	1
0	1	1	1	1	0	1	0	0	0	1	0	0
1	0	1	1	0	1	1	0	0	0	0	0	1
1	1	1	0	1	1	1	1	1	0	1	0	1

A reduction of exactly 60% ?

Mutating Conditional Expressions

Two common patterns for short-circuit operators

Mutating Conditional Expressions

Two common patterns for short-circuit operators

```
public void foo(int x){
Var v;
    if(flag&&(v=getVar())!=null)
    {
        v.bar(x);
    }
    ...
}
```

Mutating Conditional Expressions

Two common patterns for short-circuit operators

```
public void foo(int x){
Var v;
    if(flag&&(v=getVar())!=null)
    {
        v.bar(x);
    }
    ...
}
```

Mutating Conditional Expressions

Two common patterns for short-circuit operators

```
public void foo(int x){
Var v;
    if(flag&&(v=getVar())!=null)
    {
        v.bar(x);
    }
    ...
}
```

Mutating Conditional Expressions

Two common patterns for short-circuit operators

```
public void foo(int x){
Var v;
    if(flag&&(v=getVar())!=null)
    {
        v.bar(x);
    }
    ...
}
```

Mutating Conditional Expressions

Two common patterns for short-circuit operators

```
public void foo(int x){
Var v;
    if(flag&&(v=getVar())!=null)
    {
        v.bar(x);
    }
    ...
}
```

```
public void foo(int x){
Var v;
    if(flag||(v=getVar())!=null)
    {
        v.bar(x);
    }
    ...
}
```

Mutating Conditional Expressions

Two common patterns for short-circuit operators

```
public void foo(int x){
Var v;
    if(flag&&(v=getVar())!=null)
    {
        v.bar(x);
    }
    ...
}
```

```
public void foo(int x){
Var v;
    if(flag||(v=getVar())!=null)
    {
        v.bar(x);
    }
    ...
}
```

Mutating Conditional Expressions

Two common patterns for short-circuit operators

```
public void foo(int x){
Var v;
    if(flag&&(v=getVar())!=null)
    {
        v.bar(x);
    }
    ...
}
```

```
public void foo(int x){
Var v;
    if(flag||(v=getVar())!=null)
    {
        v.bar(x);
    }
    ...
}
```

Mutating Conditional Expressions

Two common patterns for short-circuit operators

```
public void foo(int x){
Var v;
    if(flag&&(v=getVar())!=null)
    {
        v.bar(x);
    }
    ...
}
```

Properly handled by MAJOR

Mutating Conditional Expressions

Two common patterns for short-circuit operators

```
public void foo(int x){
Var v;
    if(flag&&(v=getVar())!=null)
    {
        v.bar(x);
    }
    ...
}
```

Properly handled by MAJOR

```
public void foo(int x){
Var v;
    if(flag||(v=getVar())==null)
    {
        return;
    }
    v.bar(x);
}
```

Properly handled by MAJOR

Mutating Conditional Expressions

Two common patterns for short-circuit operators

```
public void foo(int x){
Var v;
    if(flag&&(v=getVar())!=null)
    {
        v.bar(x);
    }
    ...
}
```

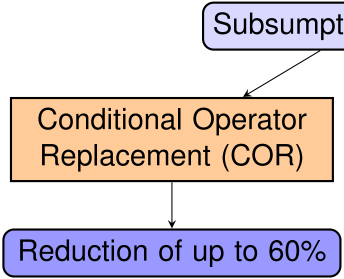
Properly handled by MAJOR

```
public void foo(int x){
Var v;
    if(flag||(v=getVar())==null)
    {
        return;
    }
    v.bar(x);
}
```

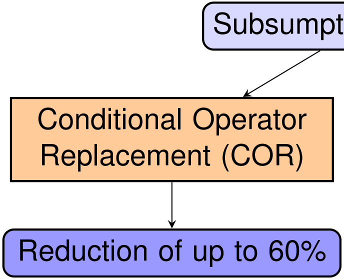
Properly handled by MAJOR

A reduction of up to 60%

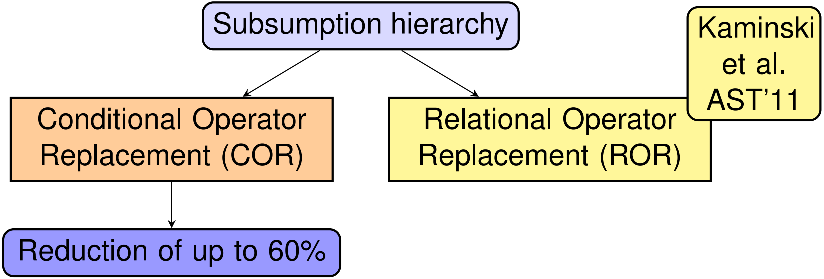
COR and ROR Mutation Operators



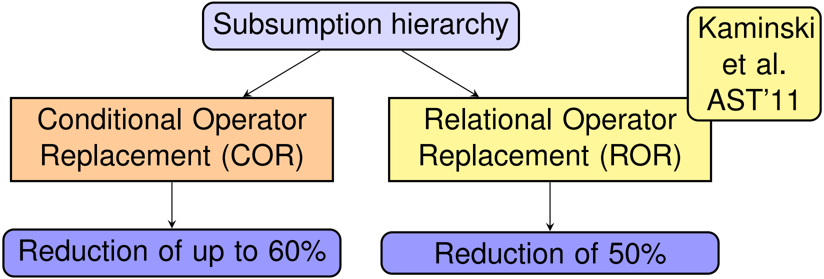
COR and ROR Mutation Operators



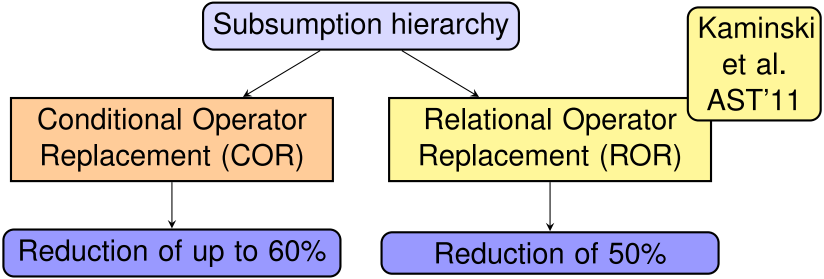
COR and ROR Mutation Operators



COR and ROR Mutation Operators



COR and ROR Mutation Operators



How prevalent are COR and ROR mutants?

Investigated Applications

	Files	LOC	Tests	Mutants Generated	Mutants Covered
commons-math	408	39,991	2,169	80,372	72,203
commons-lang	99	19,495	2,039	31,130	29,069
commons-io	100	7,908	309	9,547	4,935
numerics4j	73	3,647	218	6,835	6,547

Investigated Applications

	Files	LOC	Tests	Mutants Generated	Mutants Covered
commons-math	408	39,991	2,169	80,372	72,203
commons-lang	99	19,495	2,039	31,130	29,069
commons-io	100	7,908	309	9,547	4,935
numerics4j	73	3,647	218	6,835	6,547

Application differ in size and complexity

Investigated Applications

	Files	LOC	Tests	Mutants Generated	Mutants Covered
commons-math	408	39,991	2,169	80,372	72,203
commons-lang	99	19,495	2,039	31,130	29,069
commons-io	100	7,908	309	9,547	4,935
numerics4j	73	3,647	218	6,835	6,547

Application differ in size and complexity

Differences in mutation coverage

Investigated Applications

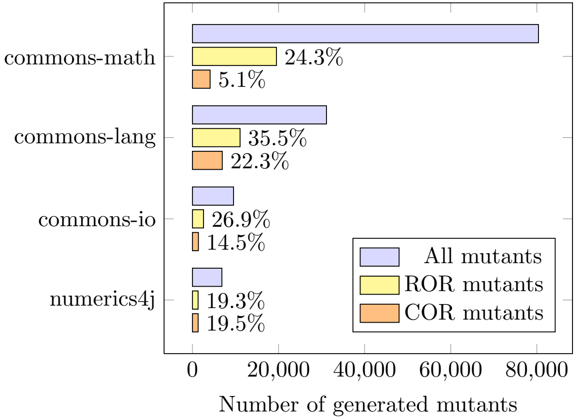
	Files	LOC	Tests	Mutants Generated	Mutants Covered
commons-math	408	39,991	2,169	80,372	72,203
commons-lang	99	19,495	2,039	31,130	29,069
commons-io	100	7,908	309	9,547	4,935
numerics4j	73	3,647	218	6,835	6,547

Application differ in size and complexity

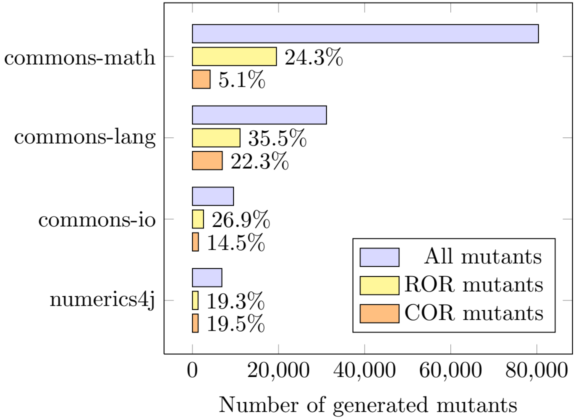
Differences in mutation coverage

How prevalent are COR and ROR mutants?

Ratio of COR and ROR Mutants

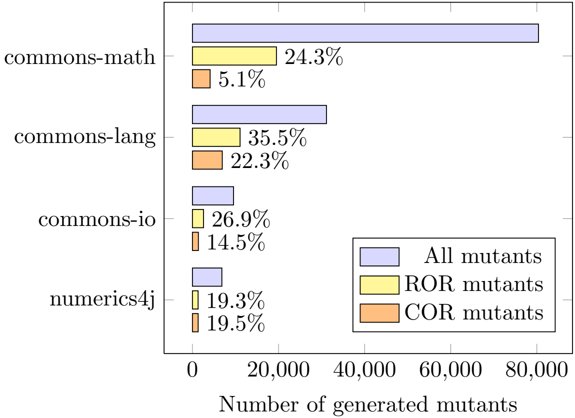


Ratio of COR and ROR Mutants



COR and ROR generate up to 58% of all mutants

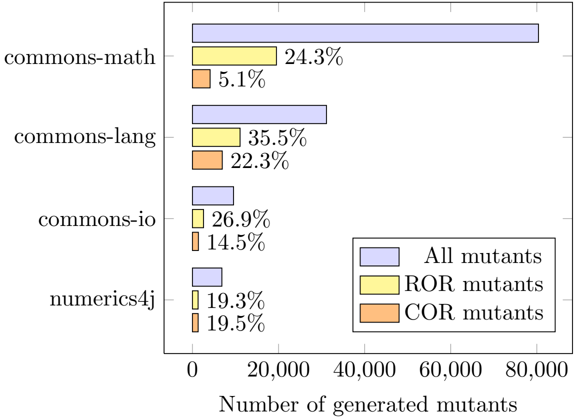
Ratio of COR and ROR Mutants



COR and ROR generate up to 58% of all mutants

Indicates notable potential for improvements

Ratio of COR and ROR Mutants



COR and ROR generate up to 58% of all mutants

Indicates notable potential for improvements

How much is the overall reduction?

Decrease in Number of Mutants

	Generated (original set)	Generated (reduced set)	Covered (original set)	Covered (reduced set)
commons-math	80,372	66,787 (-16.9%)	72,203	59,195 (-18.0%)
commons-lang	31,130	21,074 (-32.3%)	29,069	19,112 (-34.3%)
commons-io	9,547	7,319 (-23.3%)	4,935	4,168 (-15.5%)
numerics4j	6,835	5,437 (-20.5%)	6,547	5,149 (-21.4%)

Decrease in Number of Mutants

Overall reduction of up to 34%

	Generated (original set)	Generated (reduced set)	Covered (original set)	Covered (reduced set)
commons-math	80,372	66,787 (-16.9%)	72,203	59,195 (-18.0%)
commons-lang	31,130	21,074 (-32.3%)	29,069	19,112 (-34.3%)
commons-io	9,547	7,319 (-23.3%)	4,935	4,168 (-15.5%)
numerics4j	6,835	5,437 (-20.5%)	6,547	5,149 (-21.4%)

Decrease in Number of Mutants

Overall reduction of up to 34%

Depends on mutation coverage

	Generated (original set)	Generated (reduced set)	Covered (original set)	Covered (reduced set)
commons-math	80,372	66,787 (-16.9%)	72,203	59,195 (-18.0%)
commons-lang	31,130	21,074 (-32.3%)	29,069	19,112 (-34.3%)
commons-io	9,547	7,319 (-23.3%)	4,935	4,168 (-15.5%)
numerics4j	6,835	5,437 (-20.5%)	6,547	5,149 (-21.4%)

Decrease in Number of Mutants

Overall reduction of up to 34%

Depends on mutation coverage

	Generated (original set)	**Generated (reduced set)**	**Covered (original set)**	**Covered (reduced set)**
commons-math	80,372	66,787 (-16.9%)	72,203	59,195 (-18.0%)
commons-lang	31,130	21,074 (-32.3%)	29,069	19,112 (-34.3%)
commons-io	9,547	7,319 (-23.3%)	4,935	4,168 (-15.5%)
numerics4j	6,835	5,437 (-20.5%)	6,547	5,149 (-21.4%)

How much is the saving in runtime?

Runtime Improvement

	Runtime (original set)	**Runtime (reduced set)**
commons-math	300.77	271.10 (-09.9%)
commons-lang	28.25	18.70 (-33.8%)
commons-io	6.95	4.58 (-34.1%)
numerics4j	2.85	2.08 (-26.9%)

Runtime Improvement

Divergence due to differences in test suite runtime and coverage

	Runtime (original set)	**Runtime (reduced set)**
commons-math	300.77	271.10 (-09.9%)
commons-lang	28.25	18.70 (-33.8%)
commons-io	6.95	4.58 (-34.1%)
numerics4j	2.85	2.08 (-26.9%)

Runtime Improvement

Divergence due to differences in test suite runtime and coverage

	Runtime (original set)	Runtime (reduced set)
commons-math	300.77	271.10 (-09.9%)
commons-lang	28.25	18.70 (-33.8%)
commons-io	6.95	4.58 (-34.1%)
numerics4j	2.85	2.08 (-26.9%)

Significant speed-up for all applications

Accuracy of the Mutation Score

	Mutation Score (original set)	**Mutation Score (reduced set)**
commons-math	0.77	0.73 (- 4.5%)
commons-lang	0.76	0.67 (-10.7%)
commons-io	0.41	0.44 (8.3%)
numerics4j	0.69	0.65 (- 5.9%)

Accuracy of the Mutation Score

Mutation Score up to 10% overestimated

	Mutation Score (original set)	Mutation Score (reduced set)
commons-math	0.77	0.73 (- 4.5%)
commons-lang	0.76	0.67 (-10.7%)
commons-io	0.41	0.44 (8.3%)
numerics4j	0.69	0.65 (- 5.9%)

Accuracy of the Mutation Score

Mutation Score up to 10% overestimated

Low mutation coverage may lead to underestimation

	Mutation Score (original set)	**Mutation Score (reduced set)**
commons-math	0.77	0.73 (- 4.5%)
commons-lang	0.76	0.67 (-10.7%)
commons-io	0.41	0.44 (8.3%)
numerics4j	0.69	0.65 (- 5.9%)

Accuracy of the Mutation Score

Mutation Score up to 10% overestimated

Low mutation coverage may lead to underestimation

	Mutation Score (original set)	**Mutation Score (reduced set)**
commons-math	0.77	0.73 (- 4.5%)
commons-lang	0.76	0.67 (-10.7%)
commons-io	0.41	0.44 (8.3%)
numerics4j	0.69	0.65 (- 5.9%)

Redundant mutants tend to overestimate the mutation score

Conclusion and Future Work

Conclusion:

- Operator for conditional expressions without redundancy
- Decreased number of mutants and improved runtime
- Increased accuracy of the mutation score

Conclusion and Future Work

Conclusion:

- Operator for conditional expressions without redundancy
- Decreased number of mutants and improved runtime
- Increased accuracy of the mutation score

Future Work:

- Investigate redundancies in other mutation operators
- Analyze whether sufficient mutants tend to be equivalent
- Apply constraint solver to identify equivalent mutants

Do Redundant Mutants Affect the Effectiveness and Efficiency of Mutation Analysis?

Thank you for your attention!

Questions?

ulm university universität ulm

Allegheny College

http://www.mathematik.uni-ulm.de/sai/major