Using Conditional Mutation to Increase the Efficiency of Mutation Analysis

René Just\textsuperscript{1} & Gregory M. Kapfhammer\textsuperscript{2} & Franz Schweiggert\textsuperscript{1}

\textsuperscript{1}Ulm University, Germany
\textsuperscript{2}Allegheny College, USA

6th International Workshop on the Automation of Software Test
Waikiki, Honolulu, Hawaii, USA
May 23 - 24, 2011
Overview of the Presentation

Efficient Mutation Analysis
Overview of the Presentation

Efficient Mutation Analysis

Challenges

Just & Kapfhammer & Schweiggert

Using Conditional Mutation to Increase the Efficiency of Mutation Analysis
Overview of the Presentation

Efficient Mutation Analysis

Challenges
Solutions

Using Conditional Mutation to Increase the Efficiency of Mutation Analysis
Overview of the Presentation

- Efficient Mutation Analysis
- Challenges
- Solutions

Conditional Mutation

Just & Kapfhammer & Schweiggert
Ulm University, Allegheny College

Using Conditional Mutation to Increase the Efficiency of Mutation Analysis
Overview of the Presentation

- Efficient Mutation Analysis
- Challenges
- Solutions
- Conditional Mutation
- Syntax Tree Transformation

Just & Kapfhammer & Schweiggert

Using Conditional Mutation to Increase the Efficiency of Mutation Analysis
Overview of the Presentation

- Efficient Mutation Analysis
  - Conditional Mutation
    - Syntax Tree Transformation
  - Expressions and Statements
- Challenges
- Solutions

Just & Kapfhammer & Schweiggert

Ulm University, Allegheny College

Using Conditional Mutation to Increase the Efficiency of Mutation Analysis
Overview of the Presentation

Efficient Mutation Analysis

Challenges

Conditional Mutation

Compiler Integrated

Syntax Tree Transformation

Expressions and Statements

Solutions
Overview of the Presentation

Comprehensive Empirical Study

Efficient Mutation Analysis

Challenges

Conditional Mutation

Syntax Tree Transformation

Solutions

Compiler Integrated

Expressions and Statements

Just & Kapfhammer & Schweiggert

Ulm University, Allegheny College

Using Conditional Mutation to Increase the Efficiency of Mutation Analysis
Using Conditional Mutation to Increase the Efficiency of Mutation Analysis
Overview of Mutation Analysis

```java
public int eval(int x) {
    int a = 3, b = 1, y;

    y = a * x;

    y += b;
    return y;
}

public int max(int a, int b) {
    int max = a;

    if (b > a) {
        max = b;
    }

    return max;
}
```

Just & Kapfhammer & Schweiggert
Ulm University, Allegheny College
Using Conditional Mutation to Increase the Efficiency of Mutation Analysis
Overview of Mutation Analysis

```java
public int eval(int x) {
    int a=3, b=1, y;
    y = a * x;
    y += b;
    return y;
}

public int max(int a, int b) {
    int max = a;
    if (b>a) {
        max = b;
    }
    return max;
}
```

Methodically inject small syntactical faults into the program under test

Just & Kapfhammer & Schweiggert

Ulm University, Allegheny College

Using Conditional Mutation to Increase the Efficiency of Mutation Analysis
Overview of Mutation Analysis

```java
public int eval(int x) {
    int a=3, b=1, y;

    y = a * x;
    y += b;
    return y;
}

public int max(int a, int b) {
    int max = a;

    if(b>a) {
        max=b;
    }

    return max;
}
```

Just & Kapfhammer & Schweiggert

Ulm University, Allegheny College

Using Conditional Mutation to Increase the Efficiency of Mutation Analysis
Overview of Mutation Analysis

```java
public int eval(int x) {
    int a=3, b=1, y;

    y = a * x;
    y += b;
    return y;
}

public int max(int a, int b) {
    int max = a;

    if(b>a) {
        max=b;
    }

    return max;
}
```

![Mutations and Conditional Mutations](image.png)
Overview of Mutation Analysis

```
public int eval(int x) {
    int a = 3, b = 1, y;
    y = a * x;
    y += b;
    return y;
}

public int max(int a, int b) {
    int max = a;
    if (b > a) {
        max = b;
    }
    return max;
}
```

Unbiased and powerful method for assessing oracles and input values
Overview of Mutation Analysis

```java
public int eval(int x){
    int a=3, b=1, y;
    y = a * x;
    y += b;
    return y;
}

public int max(int a, int b){
    int max = a;
    if(b>a){
        max=b;
    }
    return max;
}
```

Unbiased and powerful method for assessing oracles and input values.

Useful method for fault seeding during the empirical study of testing techniques.

Just & Kapfhammer & Schweiggert
Ulm University, Allegheny College

Using Conditional Mutation to Increase the Efficiency of Mutation Analysis
Mutation Analysis Challenges

Mutant Generation
Mutation Analysis Challenges

Mutation Operators → Mutant Generation
Mutation Analysis Challenges

Program

Mutant Generation

Mutation Operators
Mutation Analysis Challenges

Program

Mutation Operators → Mutant Generation → Mutants

Just & Kapfhammer & Schweiggert
Ulm University, Allegheny College
Using Conditional Mutation to Increase the Efficiency of Mutation Analysis
Mutation Analysis Challenges

- Often Yields a Substantial Number of Mutants

Program

Mutation Operators

Mutant Generation

Mutants
Mutation Analysis Challenges

High Time Overhead for Generation
Mutation Analysis Challenges

Program → Mutant Generation → Mutants

- Often Yields a Substantial Number of Mutants
- High Time Overhead for Generation

Just & Kapfhammer & Schweiggert
Ulm University, Allegheny College

Using Conditional Mutation to Increase the Efficiency of Mutation Analysis
Using Conditional Mutation to Increase the Efficiency of Mutation Analysis

Mutation Analysis Challenges

- Mutant Generation
- Mutation Operators
- Program
- Mutants
- Tests
- Mutation Analysis

High Time Overhead for Generation

Often Yields a Substantial Number of Mutants

Just & Kapfhammer & Schweiggert
Ulm University, Allegheny College
Introduction

Conditional Mutation

Implementation

Conclusion

Mutation Analysis Challenges

Mutation Operators → Mutant Generation → Mutants → Mutation Analysis → Results

Program

Often Yields a Substantial Number of Mutants

High Time Overhead for Generation

Just & Kapfhammer & Schweiggert

Ulm University, Allegheny College

Using Conditional Mutation to Increase the Efficiency of Mutation Analysis
Mutation Analysis Challenges

High Time Overhead for Generation

Individually Executing the Mutants is Too Expensive

Often Yields a Substantial Number of Mutants

Using Conditional Mutation to Increase the Efficiency of Mutation Analysis
Mutation Analysis Challenges

Prior Solutions?

Mutation Operators

Program

Mutant Generation

Mutants

Tests

Mutation Analysis

Results

Often Yields a Substantial Number of Mutants

High Time Overhead for Generation

Individually Executing the Mutants is Too Expensive
Prior Work in Mutation Analysis

Improving Mutation Analysis
Prior Work in Mutation Analysis

Improving Mutation Analysis

Offutt and Untch
Prior Work in Mutation Analysis

Improving Mutation Analysis

Do Fewer

Offutt and Untch

Just & Kapfhammer & Schweiggert

Using Conditional Mutation to Increase the Efficiency of Mutation Analysis
Prior Work in Mutation Analysis

Improving Mutation Analysis

Offutt and Untch

Do Fewer

Sampling

Selection

Using Conditional Mutation to Increase the Efficiency of Mutation Analysis

Just & Kapfhammer & Schweiggert

Ulm University, Allegheny College
Prior Work in Mutation Analysis

Improving Mutation Analysis

Offutt and Untch

Do Fewer

Do Smarter
Prior Work in Mutation Analysis

- Improving Mutation Analysis
  - Do Fewer
  - Distributed
  - Do Smarter
  - Weak Mutation

Offutt and Untch
Prior Work in Mutation Analysis

Improving Mutation Analysis

- Offutt and Untch
- Do Fewer
- Do Smarter
- Do Faster

Just & Kapfhammer & Schweiggert
Ulm University, Allegheny College

Using Conditional Mutation to Increase the Efficiency of Mutation Analysis
Prior Work in Mutation Analysis

Improving Mutation Analysis

Offutt and Untch

Do Fewer

Do Smarter

Do Faster

Compiler Integrated

Bytecode Transformation

Mutant Schemata

Using Conditional Mutation to Increase the Efficiency of Mutation Analysis
Prior Work in Mutation Analysis

Improving Mutation Analysis

- Do Fewer
- Do Smarter
- Do Faster

Jia and Harman
Higher Order Mutation

Offutt and Untch
Conditional Mutation

Using Conditional Mutation to Increase the Efficiency of Mutation Analysis

Just & Kapfhammer & Schweiggert

Ulm University, Allegheny College
Conditional Mutation

Encapsulates all mutants within the same block
Conditional Mutation

**Encapsulates all mutants within the same block**

**Can be integrated within the compiler**

---

Just & Kapfhammer & Schweiggert

Ulm University, Allegheny College

Using Conditional Mutation to Increase the Efficiency of Mutation Analysis
Using Conditional Mutation to Increase the Efficiency of Mutation Analysis
Using Conditional Mutation to Increase the Efficiency of Mutation Analysis
Working Example

```java
public int eval(int x) {
    int a = 3, b = 1, y;

    y = a * x;

    y += b;
    return y;
}
```
public int eval(int x){
    int a=3, b=1, y;
    y = a * x;
    y += b;
    return y;
}

⇓

ASSIGN
IDENT
y
BINARY
*
a x

Using Conditional Mutation to Increase the Efficiency of Mutation Analysis
Working Example

```java
public int eval(int x) {
    int a = 3, b = 1, y;

    y = a * x;
    y += b;
    return y;
}
```

Just & Kapfhammer & Schweiggert

Ulm University, Allegheny College

Using Conditional Mutation to Increase the Efficiency of Mutation Analysis
Working Example

```java
public int eval(int x) {
    int a=3, b=1, y;
    y = a * x;
    y += b;
    return y;
}
```

Diagram:
```
ASSIGN
   | IDENT
   | y
   | BINARY
   | *
   | a
   | x
```

```
COND-EXPR
   | THEN
   | +
   | a
   | x
   | M_NO == 2
   | ELSE
   | -
   | a
   | x
   | M_NO == 1
```

Just & Kapfhammer & Schweiggert
Ulm University, Allegheny College
Using Conditional Mutation to Increase the Efficiency of Mutation Analysis
Conditional Mutation Algorithm

```java
public int eval(int x){
    int a=3, b=1, y;

    y = a * x;
    y += b;
    return y;
}
```

1. Define mutation operators
   \[ MOP(x \times y) = \{x - y, x + y, x/y\} \]

2. Determine whether current expression or statement is affected by mutation

3. Apply mutation operators

Just & Kapfhammer & Schweiggert
Ulm University, Allegheny College

Using Conditional Mutation to Increase the Efficiency of Mutation Analysis
Conditional Mutation Algorithm

```java
public int eval(int x) {
    int a = 3, b = 1, y;

    y = a * x;

    y += b;
    return y;
}
```

1. Define mutation operators
   \[ MOP(x \times y) = \{ x - y, x + y, x/y \} \]

2. Determine whether current expression or statement is affected by mutation

3. Apply mutation operators

---

Just & Kapfhammer & Schweiggert

Ulm University, Allegheny College

Using Conditional Mutation to Increase the Efficiency of Mutation Analysis
# Conditional Mutation Algorithm

```java
public int eval(int x){
    int a=3, b=1, y;

    y = a * x;
    y += b;
    return y;
}
```

1. **Define mutation operators**

   \[ MOP(x \ast y) = \{x - y, x + y, x/y\} \]

2. **Determine whether current expression or statement is affected by mutation**

3. **Apply mutation operators**
Conditional Mutation Algorithm

```java
public int eval(int x) {
    int a = 3, b = 1, y;
    y = (M_NO == 1) ? a - x : a * x;
    y += b;
    return y;
}
```

1. Define mutation operators
   \[ MOP(x \ast y) = \{ x - y, x + y, x/y \} \]

2. Determine whether current expression or statement is affected by mutation

3. Apply mutation operators
Conditional Mutation Algorithm

```java
public int eval(int x){
    int a=3, b=1, y;
    y = (M_NO==2)? a + x:
        (M_NO==1)? a - x:
                     a * x;
    y += b;
    return y;
}
```

1. Define mutation operators
   \[ MOP(x \ast y) = \{ x - y, x + y, x/y \} \]

2. Determine whether current expression or statement is affected by mutation

3. Apply mutation operators

Just & Kapfhammer & Schweiggert
Ulm University, Allegheny College

Using Conditional Mutation to Increase the Efficiency of Mutation Analysis
**Conditional Mutation Algorithm**

```java
public int eval(int x) {
    int a = 3, b = 1, y;
    y = (M_NO==3)? a / x :
        (M_NO==2)? a + x :
        (M_NO==1)? a - x :
            a * x;
    y += b;
    return y;
}
```

1. **Define mutation operators**
   
   \[ MOP(x \cdot y) = \{ x - y, x + y, x/y \} \]

2. **Determine whether current expression or statement is affected by mutation**

3. **Apply mutation operators**

- Versatile approach, can be combined with prior solutions
- Formal description and implementation details in the paper

Just & Kapfhammer & Schweiggert

Ulm University, Allegheny College

Using Conditional Mutation to Increase the Efficiency of Mutation Analysis
Introduction

Conditional Mutation

Implementation

Conclusion

Conditional Mutation Algorithm

```java
public int eval(int x){
    int a=3, b=1, y;
    y = (M_NO==3)? a / x :
        (M_NO==2)? a + x :
        (M_NO==1)? a - x :
        a * x;
    y += b;
    return y;
}
```

1. Define mutation operators
   \[ MOP(x \times y) = \{ x - y, x + y, x/y \} \]

2. Determine whether current expression or statement is affected by mutation

3. Apply mutation operators

- Versatile approach, can be combined with prior solutions
- Formal description and implementation details in the paper

Just & Kapfhammer & Schweiggert
Ulm University, Allegheny College
Using Conditional Mutation to Increase the Efficiency of Mutation Analysis
Conditional Mutation Algorithm

```java
public int eval(int x){
    int a=3, b=1, y;
    y = (M_NO==3)? a / x : 
        (M_NO==2)? a + x : 
        (M_NO==1)? a - x : 
        a * x;
    y += b;
    return y;
}
```

1. Define mutation operators
   \[ MOP(x \ast y) = \{ x - y, x + y, x/y \} \]

2. Determine whether current expression or statement is affected by mutation

3. Apply mutation operators

- Versatile approach, can be combined with prior solutions
- Formal description and implementation details in the paper
Mutation Coverage

```java
public int eval(int x) {
    int a=3, b=1, y;

    y = (M_NO==3)? a / x :
        (M_NO==2)? a + x :
        (M_NO==1)? a - x :
                   a * x;

    y += b;
    return y;
}
```
Mutation Coverage

public int eval(int x) {
    int a = 3, b = 1, y;

    y = (M_NO == 3) ? a / x :
        (M_NO == 2) ? a + x :
        (M_NO == 1) ? a - x :
            a * x;

    y += b;
    return y;
}
Mutation Coverage

```java
public int eval(int x) {
    int a=3, b=1, y;

    y = (M_NO==3)? a / x : 
        (M_NO==2)? a + x : 
        (M_NO==1)? a - x : 
        (M_NO==0 && COVERED(1,3))?
            a * x : a * x;

    y += b;

    return y;
}
```

Mutants not executed cannot be killed

Determine covered mutants with additional instrumentation

Just & Kapfhammer & Schweiggert
Ulm University, Allegheny College

Using Conditional Mutation to Increase the Efficiency of Mutation Analysis
Mutation Coverage

```java
public int eval(int x) {
    int a = 3, b = 1, y;

    y = (M_NO == 3) ? a / x :
        (M_NO == 2) ? a + x :
        (M_NO == 1) ? a - x :
        (M_NO == 0 && COVERED(1, 3)) ? a * x :
                                        a * x;

    y += b;

    return y;
}
```

- Mutants not executed cannot be killed
- Determine covered mutants with additional instrumentation
- Only execute and investigate the covered mutants

Using Conditional Mutation to Increase the Efficiency of Mutation Analysis

Just & Kapfhammer & Schweiggert

Ulm University, Allegheny College
MAJOR: Mutation Analysis in a Java Compiler

Using Conditional Mutation to Increase the Efficiency of Mutation Analysis
MAJOR: **Mutation Analysis in a Java Compiler**

Just & Kapfhammer & Schweiggert
Ulm University, Allegheny College

Using Conditional Mutation to Increase the Efficiency of Mutation Analysis
MAJOR: Mutation Analysis in a Java Compiler

Using Conditional Mutation to Increase the Efficiency of Mutation Analysis
MAJOR: Mutation Analysis in a Java Compiler

Configuration:
- Common compiler options
- Domain specific language (DSL)

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

Overhead for generating and compiling mutants is negligible.
Performance Analysis

- Overhead for generating and compiling mutants is negligible
## Performance Analysis

<table>
<thead>
<tr>
<th>Application</th>
<th>Mutants</th>
<th>Runtime of test suite</th>
<th>Memory consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>original</td>
<td>instrumented</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WCS</td>
</tr>
<tr>
<td>aspectj</td>
<td>406,382</td>
<td>4.3</td>
<td>4.8</td>
</tr>
<tr>
<td>apache ant</td>
<td>60,258</td>
<td>331.0</td>
<td>335.0</td>
</tr>
<tr>
<td>jfreechart</td>
<td>68,782</td>
<td>15.0</td>
<td>18.0</td>
</tr>
<tr>
<td>itext</td>
<td>124,184</td>
<td>5.1</td>
<td>5.6</td>
</tr>
<tr>
<td>java pathfinder</td>
<td>37,331</td>
<td>17.0</td>
<td>22.0</td>
</tr>
<tr>
<td>commons math</td>
<td>67,895</td>
<td>67.0</td>
<td>83.0</td>
</tr>
<tr>
<td>commons lang</td>
<td>25,783</td>
<td>10.3</td>
<td>11.8</td>
</tr>
<tr>
<td>numerics4j</td>
<td>5,869</td>
<td>1.2</td>
<td>1.3</td>
</tr>
</tbody>
</table>

- Runtime overhead is application dependent
- Larger for CPU-bound applications
- Small for I/O-bound applications
- Even for large projects, applicable on commodity workstations
## Performance Analysis

<table>
<thead>
<tr>
<th>Application</th>
<th>Mutants</th>
<th>Runtime of test suite</th>
<th>Memory consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>original instrumented</td>
<td>original instrumented</td>
</tr>
<tr>
<td>aspectj</td>
<td>406,382</td>
<td>4.3 4.8 5.0</td>
<td>559 813</td>
</tr>
<tr>
<td>apache ant</td>
<td>60,258</td>
<td>331.0 335.0 346.0</td>
<td>237 293</td>
</tr>
<tr>
<td>jfreechart</td>
<td>68,782</td>
<td>15.0 18.0 23.0</td>
<td>220 303</td>
</tr>
<tr>
<td>itext</td>
<td>124,184</td>
<td>5.1 5.6 6.3</td>
<td>217 325</td>
</tr>
<tr>
<td>java pathfinder</td>
<td>37,331</td>
<td>17.0 22.0 29.0</td>
<td>182 217</td>
</tr>
<tr>
<td>commons math</td>
<td>67,895</td>
<td>67.0 83.0 98.0</td>
<td>153 225</td>
</tr>
<tr>
<td>commons lang</td>
<td>25,783</td>
<td>10.3 11.8 14.8</td>
<td>104 149</td>
</tr>
<tr>
<td>numerics4j</td>
<td>5,869</td>
<td>1.2 1.3 1.6</td>
<td>73 90</td>
</tr>
</tbody>
</table>

- Runtime overhead is application dependent
  - Larger for CPU-bound applications
  - Small for I/O-bound applications
- Even for large projects, applicable on commodity workstations
## Performance Analysis

<table>
<thead>
<tr>
<th>Application</th>
<th>Mutants</th>
<th>Runtime of test suite</th>
<th>Memory consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>original</td>
<td>instrumented WCS</td>
</tr>
<tr>
<td>aspectj</td>
<td>406,382</td>
<td>4.3</td>
<td>4.8</td>
</tr>
<tr>
<td>apache ant</td>
<td>60,258</td>
<td>331.0</td>
<td>335.0</td>
</tr>
<tr>
<td>jfreechart</td>
<td>68,782</td>
<td>15.0</td>
<td>18.0</td>
</tr>
<tr>
<td>itext</td>
<td>124,184</td>
<td>5.1</td>
<td>5.6</td>
</tr>
<tr>
<td>java pathfinder</td>
<td>37,331</td>
<td>17.0</td>
<td>22.0</td>
</tr>
<tr>
<td>commons math</td>
<td>67,895</td>
<td>67.0</td>
<td>83.0</td>
</tr>
<tr>
<td>commons lang</td>
<td>25,783</td>
<td>10.3</td>
<td>11.8</td>
</tr>
<tr>
<td>numerics4j</td>
<td>5,869</td>
<td>1.2</td>
<td>1.3</td>
</tr>
</tbody>
</table>

- Runtime overhead is application dependent
  - Larger for CPU-bound applications
  - Small for I/O-bound applications

Even for large projects, applicable on commodity workstations
## Performance Analysis

<table>
<thead>
<tr>
<th>Application</th>
<th>Mutants</th>
<th>Runtime of test suite</th>
<th>Memory consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>original</td>
<td>instrumented</td>
</tr>
<tr>
<td></td>
<td></td>
<td>wcs</td>
<td>wcs+COV</td>
</tr>
<tr>
<td>aspectj</td>
<td>406,382</td>
<td>4.3</td>
<td>4.8</td>
</tr>
<tr>
<td>apache ant</td>
<td>60,258</td>
<td>331.0</td>
<td>335.0</td>
</tr>
<tr>
<td>jfreechart</td>
<td>68,782</td>
<td>15.0</td>
<td>18.0</td>
</tr>
<tr>
<td>itext</td>
<td>124,184</td>
<td>5.1</td>
<td>5.6</td>
</tr>
<tr>
<td>java pathfinder</td>
<td>37,331</td>
<td>17.0</td>
<td>22.0</td>
</tr>
<tr>
<td>commons math</td>
<td>67,895</td>
<td>67.0</td>
<td>83.0</td>
</tr>
<tr>
<td>commons lang</td>
<td>25,783</td>
<td>10.3</td>
<td>11.8</td>
</tr>
<tr>
<td>numerics4j</td>
<td>5,869</td>
<td>1.2</td>
<td>1.3</td>
</tr>
</tbody>
</table>

- Runtime overhead is application dependent
  - Larger for CPU-bound applications
  - Small for I/O-bound applications

- Even for large projects, applicable on commodity workstations
### Performance Analysis

<table>
<thead>
<tr>
<th>Application</th>
<th>Mutants</th>
<th>Runtime of test suite</th>
<th>Memory consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>original</td>
<td>instrumented</td>
</tr>
<tr>
<td>aspectj</td>
<td>406,382</td>
<td>4.3</td>
<td>4.8</td>
</tr>
<tr>
<td>apache ant</td>
<td>60,258</td>
<td>331.0</td>
<td>335.0</td>
</tr>
<tr>
<td>jfreechart</td>
<td>68,782</td>
<td>15.0</td>
<td>18.0</td>
</tr>
<tr>
<td>itext</td>
<td>124,184</td>
<td>5.1</td>
<td>5.6</td>
</tr>
<tr>
<td>java pathfinder</td>
<td>37,331</td>
<td>17.0</td>
<td>22.0</td>
</tr>
<tr>
<td>commons math</td>
<td>67,895</td>
<td>67.0</td>
<td>83.0</td>
</tr>
<tr>
<td>commons lang</td>
<td>25,783</td>
<td>10.3</td>
<td>11.8</td>
</tr>
<tr>
<td>numerics4j</td>
<td>5,869</td>
<td>1.2</td>
<td>1.3</td>
</tr>
</tbody>
</table>

- Runtime overhead is application dependent
  - Larger for CPU-bound applications
  - Small for I/O-bound applications
- Even for large projects, applicable on commodity workstations
Enabling Efficient Mutation Analysis

Using Conditional Mutation to Increase the Efficiency of Mutation Analysis

Just & Kapfhammer & Schweiggert

Ulm University, Allegheny College
Enabling Efficient Mutation Analysis

Just & Kapfhammer & Schweiggert
Ulm University, Allegheny College
Using Conditional Mutation to Increase the Efficiency of Mutation Analysis
Conclusion

Conclusion:

- Largest empirical study of mutation analysis to date
- Mutant generation time reduced to a minimum
- Mutation coverage provides runtime optimization
- Versatilely applicable in every Java-based environment
- Arbitrary conditions enable support for higher order mutation

Future Work:

- Implement new mutation operators
- Enhance the domain specific language
Conclusion:
- Largest empirical study of mutation analysis to date
- Mutant generation time reduced to a minimum
- Mutation coverage provides runtime optimization
- Versatilely applicable in every Java-based environment
- Arbitrary conditions enable support for higher order mutation

Future Work:
- Implement new mutation operators
- Enhance the domain specific language
Using Conditional Mutation to Increase the Efficiency of Mutation Analysis

Thank you for your attention!

Questions?

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