Testing Refactoring Engines

Melina Mongiovi

It's difficult to implement refactorings!

Problem

Overly weak conditions allow unsafe transformations

Overly strong conditions prevent safe transformations

How can we test refactoring engines?
Writing programs as test input...

Input program

Testing of refactoring engines

Expected Output

Skipping some Test Inputs

Generating Programs

Well-Formedness Rules

fact WellFormednessRules { all f: Function | 
  f.returnType != Void => 
  f.stmt.elems & Return = 1 
}...
**Avoid State Explosion**

\[ \# \text{Valid sequences of statements} = \sum_{i=0}^{k} n^i \]

- \( k \) = max. number of statements
- \( n \) = kinds of statements

\[ \begin{align*}
  k &= 10, \ n = 4 \\
  \# \text{Valid sequences of statements} &= 1,398,101
\end{align*} \]

**Optimization**

```java
pre optimization (!) {
    all f: Function | #stmt < 5
    all f: Function | not f.stmt.hasDups
    ...
}
```

---

**SafeRefactor**

- **18** refactoring implementations

**Evaluation**

- **Time Reduction**
- **Number of detected bugs**
- **Type of detected bugs**

**Generated Programs**

<table>
<thead>
<tr>
<th>Refactoring</th>
<th>Generated Programs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Java</strong></td>
<td></td>
</tr>
<tr>
<td>Rename Method</td>
<td>8,634 809 361</td>
</tr>
<tr>
<td>Rename Local Variable</td>
<td>9,117 870 390</td>
</tr>
<tr>
<td>Rename Define</td>
<td>7,912 867 345</td>
</tr>
<tr>
<td>Rename Function</td>
<td>1,831 175 68</td>
</tr>
<tr>
<td>Rename Parameter</td>
<td>1,812 1,293 624</td>
</tr>
<tr>
<td>Rename Global Variable</td>
<td>1,034 101 31</td>
</tr>
<tr>
<td>Rename Local Variable</td>
<td>13,188 1,327 531</td>
</tr>
<tr>
<td>Rename Define</td>
<td>5,944 587 230</td>
</tr>
<tr>
<td>Rename Function</td>
<td>7,812 786 314</td>
</tr>
<tr>
<td>Rename Parameter</td>
<td>7,812 786 314</td>
</tr>
<tr>
<td>Rename Global Variable</td>
<td>7,812 786 314</td>
</tr>
</tbody>
</table>

- **Total**
  - **Java**: 96,129 9,371 3,932

---

**Refactoring**

- **No Skip**
- **Skip 10**
- **Skip 25**
Time to identify Overly Weak Conditions

- **Eclipse CDT**: 0.63%
- **Eclipse JDT**: 0.93%
- **JRRT**: 0.45%

Time to identify Overly Strong Conditions

- **Eclipse CDT**: 1.62%
- **Eclipse JDT**: 2.31%
- **JRRT**: 1.09%

Time Reduction

- No Skip: 61.61h
- Skip 10: 5.89h
- Skip 25: 2.35h

Kind of Bug

<table>
<thead>
<tr>
<th>Kind of Bug</th>
<th>Refactoring Engine</th>
<th>Failures</th>
<th>Bugs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compilation Error</td>
<td>Eclipse CDT</td>
<td>2,711</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>Eclipse JDT</td>
<td>1,907</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>JRRT</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Behavioral Change</td>
<td>Eclipse CDT</td>
<td>2,425</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>Eclipse JDT</td>
<td>1,807</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>JRRT</td>
<td>9</td>
<td>1</td>
</tr>
</tbody>
</table>

Bug in the Extract Function

```c
int g
    globalVar = 1;
}
```

```c
int func() {
    int localVar = 0;
    return globalVar;
}
```

The process

1. First Execution: using skip
1. First Execution: using skip

2. Second Execution: Executing it again using no/otherskip while inspecting the bugs

Or...

Find the first bug...

...in a few seconds

Testing Refactoring Engines

Melina Mongiovi

melina@copin.ufcg.edu.br
Federal University of Campina Grande