Proposal of Execution Paths Indication Method for Integration Testing by Using an Automatic Visualization Tool ‘Avis’

Yoshihiro Kita, Tetsuro Katayama, and Shigeyuki Tomita
University of Miyazaki, Japan
Agenda

1. Background
2. Goal & Approach
3. Avis
4. Evaluation of Proposal method
5. Discussion
6. Conclusion & Future works
Background

• Integration Testing
  ▫ A part of software testing process
  ▫ Verification targets
    • Completeness of functionality
    • Data manipulation
    • Inter-module interfaces
Background

- Verification of inter-module interfaces by black-box testing.

Module A (Calling module)

Module B (Called module)
Background

- Verification of inter-module interfaces by white-box testing.
Background

• A problem of integration testing by white-box testing
  ▫ Explosive increasing number of execution paths for covering all modules and call-pairs.
  
  It is impossible to execute all paths.
Explosive increasing number of execution paths

Module A
Recursive call

Module B
Infinite iteration

Loop

Explosive increasing
Goal & Approach

• Goal
  Verification support of inter-module interfaces by white-box testing in integration testing

• Approach
  Indicate the minimum set of execution paths by using automatic visualization tool ‘Avis’.
Avis

• **Automatic Visualization Tool for Programs**

• **Input**
  ▫ *Source code of Java program*

• **Outputs**
  ▫ *Flowchart*
  ▫ *Sequential execution paths*
  ▫ *Inter-module execution paths*
View of Avis

Sequential execution path

Inter-module execution path

Code

Flowchart
Paths Generation

Flowchart

if
\[ a \quad b \quad c \]

Execution Paths

Path1

if
\[ a \quad b \quad c \]

Path2

if
\[ a \quad b \quad c \]

Separation
Solution of explosive increasing number of paths

- Criteria for generating execution paths
- Criteria for integration testing
- Algorithm for abstracting execution paths
Explosive increasing number of execution paths

Module A

Module B

Loop

Infinite iteration

Explosive increasing
Criteria for generating execution paths

• Avis’s criteria for reducing the number of paths based on branch coverage (C1)
  ▫ Criterion for branch covering
  ▫ Criterion for loop covering
Criterion for branch covering

- All branches are executed at least once.
Criterion for loop covering

- All loops are iterated zero times (i.e. no iteration) and once.
Solution of explosive increasing number of paths

- Criteria for generating execution paths
- Criteria for integration testing
- Algorithm for abstracting execution paths
Explosive increasing number of execution paths

Module A

Recursive call

Module B

Infinite iteration

Explosive increasing
Criteria for integration testing

- Avis’s new criteria for reducing the number of paths based on Module coverage($S_0$) and Call-pair coverage($S_1$) in integration testing.
  - Criterion for module covering
  - Criterion for recursive call covering
Criterion for module covering

- All modules are executed at least once.
Criterion for recursive call covering

• All recursive calls are iterated once.
Solution of explosive increasing number of paths

- Criteria for generating execution paths
- Criteria for integration testing
- Algorithm for abstracting execution paths
Useless of execution paths

• The execution paths are generated by Avis include useless paths for indicating Abstraction of the useful execution paths
abstracting execution paths

Path1, Path2 and Path3 are abstracted paths for covering all call-pairs.

Path1 and Path3 are abstracted paths for covering all modules.
Practicality of execution paths

- Infeasible paths may be generated in the following cases.
  - Paths disregard the dependence of branch conditions.
  - Paths include the code that it is impossible to execute (called ‘dead-code’).
Evaluation of proposal method

- **Practicality of Avis**
  - Runtime of Avis

- **Abstraction of execution paths**
  - Comparison between the number of modules (or call-pairs) and the number of paths generated by Avis

- **Rate of executable paths**
  - Number of executable paths among execution paths

- **Usefulness of indicating infeasible paths**
  - Number of dead-code included in infeasible paths
Practicality of Avis

- Runtime of Avis

<table>
<thead>
<tr>
<th>Java Program</th>
<th>Lines of code</th>
<th>Runtime(sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program A: Calendar</td>
<td>80</td>
<td>0.21</td>
</tr>
<tr>
<td>Program B: Simulation of bounding balls</td>
<td>155</td>
<td>0.36</td>
</tr>
<tr>
<td>Program C: Card game (Black jack)</td>
<td>722</td>
<td>1.22</td>
</tr>
<tr>
<td>Program D: Parser (A part of Avis)</td>
<td>8,034</td>
<td>9.73</td>
</tr>
</tbody>
</table>
Abstraction of execution paths

- Comparison between the number of modules and the number of execution path satisfying S0

<table>
<thead>
<tr>
<th>Java Program</th>
<th>Number of modules</th>
<th>Number of execution paths satisfying S0</th>
<th>Reduction ratio(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program A</td>
<td>7</td>
<td>3</td>
<td>57.1</td>
</tr>
<tr>
<td>Program B</td>
<td>23</td>
<td>14</td>
<td>39.1</td>
</tr>
<tr>
<td>Program C</td>
<td>76</td>
<td>38</td>
<td>50.0</td>
</tr>
<tr>
<td>Program D</td>
<td>573</td>
<td>352</td>
<td>38.6</td>
</tr>
</tbody>
</table>
Abstraction of execution paths

- Comparison between the number of call-pairs and the number of execution path satisfying S1

<table>
<thead>
<tr>
<th>Java Program</th>
<th>Number of call-pairs</th>
<th>Number of execution paths satisfying S1</th>
<th>Reduction ratio(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program A</td>
<td>17</td>
<td>3</td>
<td>82.4</td>
</tr>
<tr>
<td>Program B</td>
<td>66</td>
<td>16</td>
<td>75.8</td>
</tr>
<tr>
<td>Program C</td>
<td>259</td>
<td>45</td>
<td>82.6</td>
</tr>
<tr>
<td>Program D</td>
<td>2,277</td>
<td>372</td>
<td>83.7</td>
</tr>
</tbody>
</table>
Rate of executable paths

- Number of executable paths among execution paths satisfying So

<table>
<thead>
<tr>
<th>Java Program</th>
<th>Num. of paths</th>
<th>Num. of executable paths</th>
<th>Num. of covering modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program A</td>
<td>3</td>
<td>3 (100.0%)</td>
<td>7 (100.0%)</td>
</tr>
<tr>
<td>Program B</td>
<td>14</td>
<td>13 (92.0%)</td>
<td>22 (95.7%)</td>
</tr>
<tr>
<td>Program C</td>
<td>38</td>
<td>27 (71.1%)</td>
<td>61 (80.3%)</td>
</tr>
<tr>
<td>Program D</td>
<td>352</td>
<td>217 (61.6%)</td>
<td>409 (71.4%)</td>
</tr>
</tbody>
</table>
Rate of executable paths

- Number of executable paths among execution paths satisfying S1

<table>
<thead>
<tr>
<th>Java Program</th>
<th>Num. of paths</th>
<th>Num. of executable paths</th>
<th>Num. of covering call-pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program A</td>
<td>3</td>
<td>3 (100.0%)</td>
<td>7 (100.0%)</td>
</tr>
<tr>
<td>Program B</td>
<td>16</td>
<td>14 (87.5%)</td>
<td>63 (95.5%)</td>
</tr>
<tr>
<td>Program C</td>
<td>45</td>
<td>29 (64.4%)</td>
<td>217 (83.8%)</td>
</tr>
<tr>
<td>Program D</td>
<td>372</td>
<td>223 (59.9%)</td>
<td>1,558 (68.4%)</td>
</tr>
</tbody>
</table>
Usefulness of indicating infeasible paths

- Number of dead-code included in infeasible paths

<table>
<thead>
<tr>
<th>Java Program</th>
<th>Num. of infeasible paths(S0)</th>
<th>Num. of dead-codes</th>
<th>Num. of infeasible paths(S1)</th>
<th>Num. of dead-codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program A</td>
<td>0</td>
<td>-</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>Program B</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Program C</td>
<td>11</td>
<td>0</td>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>Program D</td>
<td>135</td>
<td>7</td>
<td>149</td>
<td>9</td>
</tr>
</tbody>
</table>
Discussion

• Result of evaluation
  ▫ Reducing the number of execution paths by proposed criteria.
  ▫ Minimum set of execution paths can cover all modules and call-pairs.
  ▫ Infeasible paths include dead-code.
Conclusion

• Goal
  Verification support of inter-module interfaces by white-box testing in integration testing

• Proposal
  Indicate the minimum set of execution paths by using automatic visualization tool ‘Avis’.

• Out Comes
  ▫ Minimum set of execution paths can cover both all modules and call-pairs.
  ▫ The set of execution paths contain dead-code can be specified.
Future works

- Measures for infeasible paths

- Applications of Avis to software testing education

- Applications of Avis to other programming languages for integration testing
Thank you for your attention!!