Hyperheuristic Observation Based Slicing of Guava

Seongmin Lee and Shin Yoo

Korea Advanced Institute of Science and Technology
COINSE Lab
Program Slicing

• Generates a subset of the original program, while preserving the specific behavior of the original program.

• Specific behavior: Slicing Criterion $< i, V >$ ($i$ : line number, $V$ : variable name)

• Testing, Debugging, Maintenance, etc.
Program Slicing

• Generates a subset of the original program, while preserving the specific behavior of the original program.

• Specific behavior: Slicing Criterion < i, V > (i : line number, V : variable name)

• Testing, Debugging, Maintenance, etc.
Program Slicing

• Generates a **subset** of the original program, while preserving the **specific behavior** of the original program.

• Specific behavior: Slicing Criterion < i, V > ( i: line number, V: variable name)

• Testing, Debugging, Maintenance, etc.

• Limitations:
  - scalability of static analysis
  - lack of supports on multi-lingual systems.
Observation Based Slicing (ORBS)

- Purely dynamic & Language Independent
- Makes a series of deletions of code lines, which
  
  1) leaves the code (still) compilable, and
  
  2) preserves the trajectory of the slicing criterion.

- Approximate the program dependence via observations of test executions.
Observation-Based Slicing (ORBS)

```c
int main()
{
    int sum = 0;
    int i = 1;
    while (i<11) {
        sum = sum+i;
        i = i + 1;
    }
    printf("%d \n", sum);
    printf("%d \n", i);
}
```
Observation-Based Slicing (ORBS)

```c
int main()
{
    int sum = 0;
    int i = 1;
    while (i<11) {
        sum = sum+i;
        i = i + 1;
    }
    printf("%d \n", sum);
    printf("d \n", i);
}
```
Observation-Based Slicing (ORBS)

```c
int main()
{
    int sum = 0;
    int i = 1;
    while (i<11) {
        sum = sum+i;
        i = i + 1;
    }
    printf("%d \n", sum);
    printf("%d \n", i);
}
```
Observation-Based Slicing (ORBS)

```c
int main()
{
    int sum = 0;
    int i = 1;
    while (i<11) {
        sum = sum+i;
        i = i + 1;
    }
}
```

"ORBS: Language-Independent Program Slicing", FSE14
Observation-Based Slicing (ORBS)

```c
int main()
{
    int sum = 0;
    int i = 1;
    while (i<11) {
        sum = sum+i;
        i = i + 1;
    }
    printf("%d \n", sum);

    //
}
```
Observation-Based Slicing (ORBS)

```c
int main()
{
    int sum = 0;
    int i = 1;
    while (i<11) {
        sum = sum + i;
        i = i + 1;
    }

    printf("d \n", i);
}
```
Observation-Based Slicing (ORBS)

```c
int main()
{
    int sum = 0;
    int i = 1;
    while (i<11) {
        sum = sum+i;
        i = i + 1;
    }
    printf("d \n", i);
}
```
Observation-Based Slicing (ORBS)

```c
int main()
{
    int sum = 0;
    int i = 1;
    while (i<11) {
        sum = sum+i;
        i = i + 1;
    }

    printf("d \n", i);
}
```

Window-Deletion
Observation-Based Slicing (ORBS)

```c
int main()
{
    int i = 1;
    while (i<11) {
        i = i + 1;
    }
    printf("d \n", i);
}
```

Window-Deletion
Observation-Based Slicing (ORBS)

- Purely dynamic & Language Independent
- Able to slice programs on which
  - static slicers are guaranteed to err,
    [3] ORBS and the Limits of Static Slicing, SCAM15
  - have highly unconventional semantics.
    [9] Observational slicing based on visual semantics, JSS17
Limitations of ORBS

- Scalability
Limitations of ORBS

- Scalability

```c
int main(){
    int sum = 0;
    int i = 1;
    while (i<11) {
        sum = sum+i;
        i = i + 1;
    }
    printf("%d \n", sum);
    printf("d \n", i);
}
```
Limitations of ORBS

- Scalability

  - Takes around 7200 s to delete 220 lines.
    \[ \Rightarrow 0.03 \text{ del/s} \]
    \[ \Rightarrow 32.7 \text{ s/del} \]

  (‘escape’ package’ on Guava)

```java
int main(){
    int sum = 0;
    int i = 1;
    while (i<11) {
        sum = sum+i;
        i = i + 1;
    }
    printf(“%d \n”, sum);
    printf(“d \n”, i);
}
```
Scalability
Efficiency
Efficiency

\[
\text{Number of Deleted Lines} \div \text{Time spent}
\]
Efficiency

\[ \frac{\text{Number of Deleted Lines}}{\text{Deletion Attempt}} \]
Efficiency

Number of Deleted Lines

Deletion Attempt
Deletion based on Lexical Similarity
Deletion based on Lexical Similarity

“Delete all lines of code that are related to a word ‘log’!”
Deletion based on Lexical Similarity

“Delete all lines of code that are related to a word ‘log’!”

Dependence Approximation

Spatial Neighborhood
Deletion based on Lexical Similarity

“Delete all lines of code that are related to a word ‘log’!”
Deletion based on Lexical Similarity

• Vector Space Model
  - Traditional method for calculating distances between text documents and a query.

• Latent Dirichlet Allocation
  - Probabilistic model that describes which topics are present in a given document.

• Consider each code lines as a document.

• Attempts to delete code lines whose similarity is above certain threshold.
Deletion based on Lexical Similarity

• Vector Space Model → VSM-Deletion
  - Traditional method for calculating distances between text documents and a query.

• Latent Dirichlet Allocation → LDA-Deletion
  - Probabilistic model that describes which topics are present in a given document.

• Consider each code lines as a document.

• Attempts to delete code lines whose similarity is above certain threshold.
Deletion based on Lexical Similarity

- Vector Space Model → VSM-Deletion
  - Traditional method for calculating distances between text documents and a query.

- Latent Dirichlet Allocation → LDA-Deletion
  - Probabilistic model that describes which topics are present in a given document.

⇒ Line Similarity based ORBS (LS-ORBS)

[7] Using source code lexical similarity to improve efficiency of observation based slicing
Deletion based on Lexical Similarity

53.3% less compilations, 34.3% less executions, 39.3% less time per 1 deleted lines.
Compare Strategies

Efficiency

# of deleted lines
Compare Strategies

Efficiency

# of deleted lines

Window-Deletion
ORBS
Compare Strategies

Efficiency

LS-ORBS
VSM-, LDA-Deletion

Window-Deletion
ORBS

# of deleted lines
Compare Strategies

LS-ORBS

ORBS
Compare Strategies

VSM-, LDA-Deletion
+ Window-Deletion

LS-ORBS

ORBS
Compare Strategies

Efficiency

# of deleted lines

LS-ORBS

ORBS

# of deleted lines
Q. How to select the operator among various kind of deletion operators?
Hyperheuristic Observation Based Slicing

(HOBBES)

(On selecting deletion operators)
HOBBES Algorithm
HOBBES Algorithm

- Initialize selection probability of deletion operators with uniform distribution
HOBBES Algorithm

- ‘Roulette Wheel Selection’
HOBBES Algorithm

- ‘Roulette Wheel Selection’
• Apply selected deletion operator on source code.
HOBBES Algorithm

Probability update formula ‘UPDATE’

\[ \text{new } P(D_k) = \begin{cases} 
\omega_{\text{comp}} \cdot P(D_k) & \text{when compile fails} \\
\omega_{\text{exec}} \cdot P(D_k) & \text{when compile succeeds, trajectory changes} \\
(1 + \log_{10} l) \cdot P(D_k) & \text{otherwise}
\end{cases} \]

\( \omega: \text{penalty value (} \omega \in [0,1] \text{)}, \ l: \# \text{ of deleted lines} \)
HOBBES Algorithm

• Update the probability.
HOBBES Algorithm

• Update the probability.

Success to delete
HOBBES Algorithm

- Update the probability.

Compilation error / Trajectory Change
HOBBES Algorithm

• Update the probability.


HOBBES - Configuration

• Studied Deletion Operators
  - Window-Deletion of size 1, 2, 3, 4.
  - VSM-, LDA-Deletion of threshold 0.6, 0.7, 0.8, 0.9.

• Subject: Guava library
  - 2 slice criteria for each of subpackage ‘escape’ and ‘net’.

• Machine
  - Intel Core i7-6700K running Ubuntu 14.04.5 LTS.
## HOBBES - Results

<table>
<thead>
<tr>
<th>Subject</th>
<th>Strategy</th>
<th>Iter1 C</th>
<th>E D/T</th>
<th>Iter2 C</th>
<th>E D/T</th>
<th>Iter3 C</th>
<th>E D/T</th>
<th>Iter4 C</th>
<th>E D/T</th>
<th>Iter5 C</th>
<th>E D/T</th>
</tr>
</thead>
<tbody>
<tr>
<td>escape1</td>
<td>HOBBES</td>
<td>502</td>
<td>66</td>
<td>0.20</td>
<td>926</td>
<td>104</td>
<td>0.13</td>
<td>1321</td>
<td>135</td>
<td>0.11</td>
<td>1699</td>
</tr>
<tr>
<td></td>
<td>W-ORBS</td>
<td>1711</td>
<td>183</td>
<td>0.10</td>
<td>3137</td>
<td>267</td>
<td>0.06</td>
<td>4523</td>
<td>342</td>
<td>0.04</td>
<td>5840</td>
</tr>
<tr>
<td>escape2</td>
<td>HOBBES</td>
<td>1332</td>
<td>214</td>
<td>0.21</td>
<td>2424</td>
<td>309</td>
<td>0.15</td>
<td>3430</td>
<td>388</td>
<td>0.12</td>
<td>4384</td>
</tr>
<tr>
<td></td>
<td>W-ORBS</td>
<td>4179</td>
<td>655</td>
<td>0.13</td>
<td>7383</td>
<td>922</td>
<td>0.08</td>
<td>10436</td>
<td>1159</td>
<td>0.06</td>
<td>13460</td>
</tr>
<tr>
<td>net1</td>
<td>HOBBES</td>
<td>513</td>
<td>70</td>
<td>0.17</td>
<td>955</td>
<td>114</td>
<td>0.11</td>
<td>1374</td>
<td>154</td>
<td>0.09</td>
<td>1771</td>
</tr>
<tr>
<td></td>
<td>W-ORBS</td>
<td>1759</td>
<td>189</td>
<td>0.09</td>
<td>3251</td>
<td>280</td>
<td>0.06</td>
<td>4707</td>
<td>364</td>
<td>0.04</td>
<td>6141</td>
</tr>
<tr>
<td>net2</td>
<td>HOBBES</td>
<td>1341</td>
<td>222</td>
<td>0.20</td>
<td>2444</td>
<td>324</td>
<td>0.14</td>
<td>3460</td>
<td>402</td>
<td>0.11</td>
<td>4425</td>
</tr>
<tr>
<td></td>
<td>W-ORBS</td>
<td>4332</td>
<td>667</td>
<td>0.11</td>
<td>7781</td>
<td>963</td>
<td>0.07</td>
<td>11077</td>
<td>1237</td>
<td>0.05</td>
<td>14337</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
HOBBES - Results

![Graphs showing HOBBES results for Escape_1 and Escape_2]

- **Escape_1**
  - W-ORBS Deletion
  - HOBBES Deletion
  - W-ORBS Time
  - HOBBES Time

- **Escape_2**
  - W-ORBS Deletion
  - HOBBES Deletion
  - W-ORBS Time
  - HOBBES Time
HOBBES - Results

Net_1

Net_2

# of Deleted Lines

Iteration

Iteration

Time(s)

W–ORBS Deletion
HOBBES Deletion
W–ORBS Time
HOBBES Time

0
200
600
1000

0
200
600
1000

0
5000
15000
25000

0
5000
15000
25000

**HOBBES - Results**

- HOBBES can delete about **71%** of the number of lines that ORBS deletes.
- However, HOBBES only takes about **30%** of the time spent by ORBS.
Again, Compare Strategies

![Diagram showing LS-ORBS and ORBS on a graph with Efficiency and # of deleted lines axes.]
Again, Compare Strategies

Efficiency

# of deleted lines

LS-ORBS

HOBBES

ORBS
Future Work

- Investigate non-iterative application of deletions.
- Apply more sophisticated lexical analysis.
  - For example, token normalization

  ```plaintext
  ["open_file"] → ["open", "file"]
  ```
Limitations of ORBS

- Scalability
  - Takes around 7200 s to delete 220 lines.
  - \( \Rightarrow 0.03 \text{ del/s} \)
  - \( \Rightarrow 32.7 \text{ s/del} \)

```
int main()
{
    int sum = 0;
    int i = 1;
    while (i < 121) {
        sum = sum + i;
        i = i + 1;
    }
    printf("sum = %d, sum_

```

Deletion based on Lexical Similarity

- \(53.3\%\) less compilations,
- \(34.3\%\) less executions,
- \(39.3\%\) less time per 1 deleted line.

Compare Strategies

- VSM, LDA-Deletion
- Window-Deletion

- LS-ORBS
- ORBS

Hyperheuristic Observation Based Slicing (HOBBES)

(On selecting deletion operators)

HOBBES - Results

<table>
<thead>
<tr>
<th>Subject</th>
<th>Strategy</th>
<th>Bar1</th>
<th>Bar2</th>
<th>Bar3</th>
<th>Bar4</th>
<th>Bar5</th>
</tr>
</thead>
<tbody>
<tr>
<td>or221</td>
<td>HOBBES</td>
<td>502</td>
<td>69</td>
<td>50</td>
<td>114</td>
<td>123</td>
</tr>
<tr>
<td>or322</td>
<td>LS-ORBS</td>
<td>155</td>
<td>115</td>
<td>135</td>
<td>155</td>
<td>155</td>
</tr>
<tr>
<td>or322</td>
<td>ORBS</td>
<td>160</td>
<td>160</td>
<td>160</td>
<td>160</td>
<td>160</td>
</tr>
</tbody>
</table>

- HOBBES can delete about 71% of the number of lines that ORBS deletes.
- However, HOBBES only takes about 30% of the time spent by ORBS.

Again, Compare Strategies

- Efficiency
- Deletion Strength

- LS-ORBS
- HOBBES
- ORBS
How the selection probability of deletion operators changed?