Lase: Locating and Applying Systematic Edits by Learning from Examples

Na Meng      Miryung Kim      Kathryn S. McKinley
The University of Texas at Austin
Motivating Scenario

Pat needs to update database transaction code to prevent SQL injection attacks
Systematic Editing

• *Similar but not identical changes* to multiple contexts
• Manual, tedious, and error-prone
• Source transformation tools require describing edits in a formal language
• Bug fixing tools locate and apply simple or limited stylized code changes
  – Coccinelle, CFix, FixMeUp
• Sydit applies an edit inferred from a code example to user-selected targets
Workflow of Lase

User selects examples

\[ A_{\text{old}} \rightarrow A_{\text{new}} \quad B_{\text{old}} \rightarrow B_{\text{new}} \]

LASE selects methods & suggests edits

\[ D_{\text{old}} \rightarrow D_{\text{suggested}} \quad I_{\text{old}} \rightarrow I_{\text{suggested}} \quad X_{\text{old}} \rightarrow X_{\text{suggested}} \]
Approach Overview

Phase: I. Create edit script

- Syntactic program diff
- Identify common edit
- Generalize identifier
- Extract context
- Find edit location
- Apply edit script

Object next = e.next();
Object next = $0.next();
Object next = iter.next();

Phase:

A_{old}
A_{new}
B_{old}
B_{new}
C_{old}
D_{old}
D_{new}

I.
II.
III.

✗ no match
✔ match
Step 1. Syntactic Program Diff

Input: $m_{old}$, $m_{new}$

Output: Edit operations

<table>
<thead>
<tr>
<th>operation</th>
<th>definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$insert(u, v, k)$</td>
<td>insert node $u$ and position it as the (k+1)th child of node $v$</td>
</tr>
<tr>
<td>$delete(u)$</td>
<td>delete node $u$</td>
</tr>
<tr>
<td>$update(u, v)$</td>
<td>replace $u$ with $v$</td>
</tr>
<tr>
<td>$move(u, v, k)$</td>
<td>delete $u$ from its current position and insert $u$ as the (k+1)th child of $v$</td>
</tr>
</tbody>
</table>
Step 2: Identify Common Edit

- Longest Common Edit Operation Subsequence

Edit script A:
- `insert(Object next = e.next(...))`
- `insert(if(next instanceof MVAction))`
- `insert(((MVAction)next).update())`
- `update(print(next.toString())) to ...`

Edit script B:
- `insert(Object next = iter.next(...))`
- `update(print(next.getString())) to ...`
- `insert(if(next instanceof MVAction))`
- `insert(((MVAction)next).update())`
- `delete(System.out.println(...))`
Step 3: Generalize Identifier

- Keep the original identifiers if examples agree
- Abstract identifiers if examples disagree

<table>
<thead>
<tr>
<th>Generalized Identifier</th>
<th>Identifier in mA</th>
<th>Identifier in mB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable Map</td>
<td></td>
<td></td>
</tr>
<tr>
<td>next</td>
<td>next</td>
<td>next</td>
</tr>
<tr>
<td>v$0</td>
<td>e</td>
<td>iter</td>
</tr>
<tr>
<td>Method Map</td>
<td></td>
<td></td>
</tr>
<tr>
<td>next</td>
<td>next</td>
<td>next</td>
</tr>
<tr>
<td>Type Map</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Object</td>
<td>Object</td>
<td>Object</td>
</tr>
<tr>
<td>Iterator</td>
<td>Iterator</td>
<td>Iterator</td>
</tr>
</tbody>
</table>

Object next = e.next();
Object next = iter.next();
Object next = v$0.next();
Step 4: Extract Context

Iterator e = fActions.values().iterator();
... ...
while(e.hasNext())

Object next = e.next();

Iterator iter = getActions().values().iterator();
... ...
while(iter.hasNext())

Object next = iter.next();

Iterator v$0 = u$0:FieldAccessOrMethodInvocation.values().iterator();
... ...
while(v$0.hasNext())

<table>
<thead>
<tr>
<th>Generalized identifier</th>
<th>Identifier in mA</th>
<th>Identifier in mB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uncertain Map</td>
<td>fActions</td>
<td>getActions()</td>
</tr>
<tr>
<td>Variable Map</td>
<td>e</td>
<td>iter</td>
</tr>
<tr>
<td>Method Map</td>
<td>values</td>
<td>values</td>
</tr>
<tr>
<td></td>
<td>iterator</td>
<td>iterator</td>
</tr>
<tr>
<td></td>
<td>hasNext</td>
<td>hasNext</td>
</tr>
<tr>
<td>TypeMap</td>
<td>Iterator</td>
<td>Iterator</td>
</tr>
</tbody>
</table>
Phase II. Find Edit Locations

```java
Iterator e = fActions.values().iterator();
Iterator v$0 = u$0:FieldAccessOrMethodInvocation.values().iterator();
```

<table>
<thead>
<tr>
<th>Generalized identifier</th>
<th>Identifier in mD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uncertain Map</td>
<td>u$0:FieldAccessOrMethodInvocation</td>
</tr>
<tr>
<td>Variable Map</td>
<td>v$0</td>
</tr>
<tr>
<td>Method Map</td>
<td>values</td>
</tr>
<tr>
<td></td>
<td>iterator</td>
</tr>
<tr>
<td>TypeMap</td>
<td>Iterator</td>
</tr>
</tbody>
</table>
Phase III. Applying Edit Script

• Customize general edit scripts
  – Identifier concretization
  – Edit position concretization

• Apply the customized edit scripts
Comment[] getLeadingComments(ASTNode node) {
    if (this.leadingComments != null) {
        int[] range = (int[]) this.leadingComments.get(node);
        int[] range = null;
        for (int i = 0; range == null && i <= this.leadingPtr; i++) {
            if (this.leadingNodes[i] == node) range = this.leadingIndexes[i];
        }
        if (range != null) {
            int length = range[1] - range[0] + 1;
            Comment[] leadComments = new Comment[length];
            System.arraycopy(this.comments, range[0], leadComments, 0, length);
            return leadComments;
        }
    }
}

Comment[] getTrailingComments(ASTNode node) {
    if (this.trailingComments != null) {
        int[] range = (int[]) this.trailingComments.get(node);
        int[] range = null;
        for (int i = 0; range == null && i <= this.trailingPtr; i++) {
            if (this.trailingNodes[i] == node) range = this.trailingIndexes[i];
        }
        if (range != null) {
            int length = range[1] - range[0] + 1;
            Comment[] trailComments = new Comment[length];
            System.arraycopy(this.comments, range[0], trailComments, 0, length);
            return trailComments;
        }
    }
}

public int getExtendedEnd (ASTNode node) {
    int end = node.getStartPosition() + node.getLength();
    if (this.v$1_ != null) {
        if (this.trailingPts >= 0) {
            int[] range = (int[]) this.trailingComments.get(node);
            int[] range = null;
            for (int i = 0; range == null && i <= this.v$1_; i++) {
                if (this.v$2_[i] == node) range = this.v$3_[i];
            }
            if (range[0] == -1 && range[1] == -1) {
                // ...
            } else {
                // ...
            }
        } else {
            // ...
        }
        return end - 1;
    }
}
Outline

• Phase I: Creating Abstract Edit Scripts
  – Syntactic Program Diff
  – Identify Common Edit
  – Generalize Identifier
  – Extract Context
• Phase II: Find Edit Locations
• Phase III: Apply Edit Script
• Evaluation
Test Suite

• 24 repetitive bug fixes that require multiple check-ins [Park et al., MSR 2012]
  – 2 from Eclipse JDT and 22 from Eclipse SWT
  – Each bug is fixed in multiple commits
  – Clones of at least two lines between patches checked in at different times

• 37 systematic edits that require similar changes to different methods
RQ1: Precision, Recall, and Accuracy

**Precision (P):** What percentage of all *found* locations are correctly identified?

**Recall (R):** What percentage of all *expected* locations are correctly identified?

**Accuracy (A):** How similar is Lase-generated version to developer-generated version?
On average, Lase finds edit locations with 99% precision, 89% recall, and 91% accuracy.

For three bugs, Lase suggests in total 9 edits that developers missed and later confirmed.
RQ2: Sensitivity to number of exemplar edits

- 7 cases in the oracle data set
- Enumerate subsets of exemplar edits
<table>
<thead>
<tr>
<th>Index</th>
<th># of exemplars</th>
<th>P%</th>
<th>R%</th>
<th>A%</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1</td>
<td>100</td>
<td>17</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>100</td>
<td>51</td>
<td>72</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>100</td>
<td>82</td>
<td>67</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>100</td>
<td>96</td>
<td>67</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>100</td>
<td>100</td>
<td>67</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>100</td>
<td>59</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>100</td>
<td>83</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>100</td>
<td>84</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>100</td>
<td>88</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>100</td>
<td>92</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>100</td>
<td>96</td>
<td>100</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>100</td>
<td>54</td>
<td>92</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>78</td>
<td>90</td>
<td>85</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>49</td>
<td>98</td>
<td>83</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>31</td>
<td>100</td>
<td>82</td>
</tr>
</tbody>
</table>

As the number of exemplar edits increases,

> P does not change except for case 12

✧ R is more sensitive to the number of exemplar edits
✧ R increases as a function of exemplar edits

✓ A decreases when exemplar edits are different
✓ A remains the same or may increase when the exemplar edits are very similar
Conclusion

• Lase automates edit location search and program transformation application

• Lase achieves 99% precision, 89% recall, and 91% accuracy

• Future Work
  – Integrate with automated compilation and testing
  – Automatically detect repetitive change examples to infer program transformations
Thank You!

Questions?
References I

References II

Step 4: Common Edit Context Extraction

- Extract all potential common context
- Refine the common context
  - Consistent identifier mapping
  - Embedded subtree isomorphism
  - Program dependence equivalence
Step 4: Common Edit Context Extraction (1/4)

- Finding common *text* with clone detection (CCFinder [Kamiya et al. 2002])
Step 4: Common Edit Context Extraction (2/4)

- Identifier generalization

<table>
<thead>
<tr>
<th>Abstract identifier</th>
<th>Identifier in mA</th>
<th>Identifier in mB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uncertain Map</td>
<td>u$0:FieldAccessOrMethodInvocation</td>
<td>fActions</td>
</tr>
<tr>
<td>Variable Map</td>
<td>v$0</td>
<td>e</td>
</tr>
<tr>
<td>Method Map</td>
<td>values</td>
<td>values</td>
</tr>
<tr>
<td></td>
<td>iterator</td>
<td>iterator</td>
</tr>
<tr>
<td></td>
<td>hasNext</td>
<td>hasNext</td>
</tr>
<tr>
<td>TypeMap</td>
<td>Iterator</td>
<td>Iterator</td>
</tr>
</tbody>
</table>
Step 4: Common Edit Context Extraction (3/4)

- Maximum Common Embedded Subtree Extraction (MCESE) [Lozano et al. 2004]
Step 4: Common Edit Context Extraction (4/4)

- Program dependence analysis

```
Iterator e = fActions.values().iterator();
while (e.hasNext()) {
    Object next = e.next();
}
```

<table>
<thead>
<tr>
<th></th>
<th>Abstract identifier</th>
<th>Identifier in mA</th>
<th>Identifier in mB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable Map</td>
<td>v$0</td>
<td>e</td>
<td>iter</td>
</tr>
<tr>
<td>Method Map</td>
<td>values</td>
<td>values</td>
<td>values</td>
</tr>
</tbody>
</table>
```
```
?When more than two examples?