Refactoring with Synthesis

Veselin Raychev, Max Schäfer, Manu Sridharan, and Martin Vechev
Refactoring

- Goal - Change internal structure while maintaining external behavior
- Purpose - Improve code readability and maintainability.
- Result - Code that is easier to understand and update in the future
Research Questions

● Can refactoring procedures be automated without requiring explicit knowledge of refactoring methods?

● Is refactoring by example a valid and efficient approach to reducing the amount of work for the developer?
Why it is important

- ~ 90% of refactoring completed by hand
- IDE refactoring support underused
- Requires user knowledge
- No complex refactoring sequences
Previous work

- BeneFactor & WitchDoctor
  - Autocompletion
  - Monitor user edits for patterns suggestive of refactoring
  - Easier to apply refactorings already supported

- Limited to single step refactorings
Refactoring With Synthesis: Resynth

- Eclipse plugin, refactors through synthesis

1. “Start Refactoring” Button
2. Manually perform a few example edits
3. “Complete” Button
public class Account {
    private String name;

    void printOwing() {
        printBanner();
        System.out.println("name: " + name);
        System.out.println("outstanding: " + getOutstanding());
    }

    private double getOutstanding() {
        //...
    }

    private void printBanner() {
        //...
    }
}

public class Account {
    private String name;

    void printOwing() {
        printBanner();
        System.out.println("name: " + name);
        System.out.println("outstanding: " + getOutstanding());
    }

    private void printDetails(double outstanding) {
        System.out.println("amount: " + outstanding);
    }

    private double getOutstanding() {
        //...
    }

    private void printBanner() {
        //...
    }
}
Eclipse Vs. Resynth

● Eclipse - Requires 2 steps
  ○ EXTRACT METHOD
  ○ INTRODUCE PARAMETER

● Resynth
  ○ Replace lines 6-7 with printDetails(getOutstanding())
    ■ Resynth does the rest
How it works

- Create AST of original code & altered code
- Only use modified part of program
  - not whole code -> performance

Figure 3. Two ASTs and the change \((c_i, c_m)\) between them. The change is captured with dotted lines.
How it works (cont.)

- Apply “local refactoring”
  - Retains the original’s behavior with the edits of the altered program

- Use A* search to find refactoring sequence
  - search space: possible ref. seq.

- With local solution: Apply full refactoring
  - Resynth uses Eclipse’s built-in refactorings
How it works (cont.)

\[ P_i : \]
float T, s;
T = (a+b+c)/2
s = T*(T-a)*(T-b)*(T-c);
return Math.sqrt(s);

\[ P_m : \]
float T, s;
T = (a+b+c)/2
s = p*(p-a)*(T-b)*(T-c);
return Math.sqrt(s);

\[ P_f : \]
float p, s;
p = (a+b+c)/2
s = p*(p-a)*(p-b)*(p-c);
return Math.sqrt(s);

1. compute \( c_i = P_i \setminus P_m \Rightarrow c_i \subseteq P_i \) and \( c_m = P_m \setminus P_i \Rightarrow c_m \subseteq P_m \)
2. synthesize sequence: local rename T to p
3. perform the sequence (rename T to p) on the full program

\( c_m \subseteq P_f \) by Lemma 3.3

\( c_i : \)
= T*(T-)*

\( c_m : \)
= p*(p-)*
Evaluation

1. Produce single-step refactorings

2. Produce real, multi-step refactorings

3. User study
Single-step refactorings

1. Rename
2. Inline local
3. Inline method
4. Extract local
5. Extract method with holes
# Realistic examples

<table>
<thead>
<tr>
<th>Example</th>
<th>steps</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encapsulate Downcast</td>
<td>3</td>
<td>literature [6]</td>
</tr>
<tr>
<td>Extract Method (advanced)</td>
<td>4</td>
<td>literature [6]</td>
</tr>
<tr>
<td>Decompose Conditional</td>
<td>6</td>
<td>literature [6]</td>
</tr>
<tr>
<td>Introduce Foreign Method</td>
<td>2</td>
<td>literature [6]</td>
</tr>
<tr>
<td>Replace Temp With Query</td>
<td>3</td>
<td>literature [6]</td>
</tr>
<tr>
<td>Replace Parameter With Method</td>
<td>3</td>
<td>literature [6]</td>
</tr>
<tr>
<td>Swap Fields</td>
<td>3</td>
<td>literature [32]</td>
</tr>
<tr>
<td>Swap Field And Parameter</td>
<td>3</td>
<td>literature [25]</td>
</tr>
<tr>
<td>Introduce Parameter</td>
<td>6</td>
<td>Stack Overflow⁹</td>
</tr>
</tbody>
</table>

Table 1. Realistic examples used to test RESYNTH.
Realistic examples

- 9 real-world examples
  - Correct refactorings for 7
  - Equivalent refactorings for 2

- 100 synthetic, random sequences of edits
  - Solved 84
  - Others aborted by search space bound of 20,000 trees (sequence length > 9)
## Realistic example results

<table>
<thead>
<tr>
<th>Metric</th>
<th>Dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Real</td>
</tr>
<tr>
<td>Number of tests</td>
<td>9</td>
</tr>
<tr>
<td>Avg. number of trees searched</td>
<td>87</td>
</tr>
<tr>
<td>Avg. number of successors in a search</td>
<td>1296</td>
</tr>
<tr>
<td>Avg. search time</td>
<td>0.014s</td>
</tr>
<tr>
<td>Avg. Eclipse refactoring time</td>
<td>2.953s</td>
</tr>
<tr>
<td>Refactoring sequence length</td>
<td></td>
</tr>
<tr>
<td>1 refactoring</td>
<td>0</td>
</tr>
<tr>
<td>2 refactorings</td>
<td>1</td>
</tr>
<tr>
<td>3 refactorings</td>
<td>5</td>
</tr>
<tr>
<td>4 refactorings</td>
<td>1</td>
</tr>
<tr>
<td>5 refactorings</td>
<td>0</td>
</tr>
<tr>
<td>6 refactorings</td>
<td>2</td>
</tr>
<tr>
<td>7 refactorings</td>
<td>0</td>
</tr>
<tr>
<td>8 refactorings</td>
<td>0</td>
</tr>
<tr>
<td>9 refactorings</td>
<td>0</td>
</tr>
<tr>
<td>Failure to find sequence</td>
<td>0</td>
</tr>
<tr>
<td>after 20000 searched trees</td>
<td></td>
</tr>
</tbody>
</table>
User study

- 5 students, one professional

- Completed 3 refactorings using
  - Resynth
  - Eclipse’s built-in refactorings
  - No tools, manually

- 2 users failed on 1 refactoring each using Resynth
Discussion

- Do you think this tool can be useful? Would you use it?

- What kind of improvements to the tool would you suggest?
Discussion (cont.)

- Do you think that the use of this tool is easier than learning about the various Eclipse refactorings and composing them yourself? Is it better than performing the refactorings manually?

- Do you think that the evaluation could be improved?
Discussion (cont.)

- Could this approach be generalized? Can these techniques be applied in a different context/to a different problem?

- Would Resynth be more useful if it weren’t limited to Eclipse’s built-in refactorings? What replacement method could be used?