Extension, Abbreviation and Refinement

-Identifying High-Level Dependence Structures Using Slice-Based Dependence Analysis

Zheng Li
CREST, King’s College London, UK

Supervised by Mark Harman and Nicolas Gold
Overview

• Motivation
• Three techniques
  – Extension
  – Abbreviation
  – Refinement
• Empirical study
• Results
Many analysis techniques for program comprehension have been proposed.
<table>
<thead>
<tr>
<th></th>
<th>High-level</th>
<th>Low-level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalability</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Accuracy</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Human Knowledge</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
If combine the two?

- High-level techniques can provide a reasonable analysis scope with domain knowledge for low-level analysis techniques, then avoiding the scalability problem of low-level techniques.

- Low-level techniques can improve the accuracy of high-level techniques.
In this thesis

Concept Assignment

Program Slicing
Concept Assignment

• First defined in 1993 and aimed at comprehension tasks
• Locates the implementation of domain and software engineering concepts in source code
• Hypothesis-Based Concept Assignment (HB-CA)
  – Existing implementation
  – Uses domain and program semantics
  – Good quality assignments
Program Slicing

which other lines affect the selected line?

we only care about this line
Concept Assignment

Program Slicing

Contiguous?

Executable?

High/low level?
Combination 1: Extension

- **Concept Slice**
  - Using program slicing to ‘extend’ a concept binding by tracing its dependencies

- **Algorithm**
  - Using concepts as slicing criteria, the concept slice is the union of slices for each program point in the concept
Combination 2: Abbreviation

- Extract **key statements** within concept bindings
  - The statements that capture most impact with highest cohesion
  - Help to focus attention more rapidly on the core of a concept binding

- Algorithm
  - Intersection of slices with respect to principal variables within a concept binding
D=2*r;
perimeter=PI*D;
undersurface=PI*r*r;
sidesurface=perimeter*h;
area=2*undersurface+sidesurface;
volume=undersurface*h;
printf("\nThe Area is %d\n", area);
printf("\nThe Volume is %d\n", volume);
Combination 3: Refinement

• A more accurate dependence based concept binding by removing non-concept-dependent statements

• Algorithm
  – Build dependence graph
  – Vertex Rank Model (Google page rank)
  – program chopping
D=2*r;
perimeter=PI*D;
undersurface=PI*r*r;
sidesurface=perimeter*h;
area=2*undersurface+sidesurface;
volume=undersurface*h;
printf("\nThe Area is %d\n", area);
printf("\nThe Volume is %d\n", volume);
Empirical Study

• Tools
  – WeSCA and CodeSurfer
• 10 Subject programs
  – Open source and industry code
  – More than 600 concept bindings are extracted
• Dependence based metrics are defined
• Statistical analysis
Main Results

• The concept slice has no size explosion
• The identified key statements have high Impact and Cohesion, but some concept bindings do not contain key statements
• Refinement can rule out the non-concept-dependent statements
Thesis is available at
http://www.dcs.kcl.ac.uk/staff/zheng/


Questions?

(zheng.li@kcl.ac.uk)