Yoyak
An “Interactive” Abstract Interpreter
Motivation & Plan

Jaeho Shin

2010-07-09
ROPAS Show&Tell
Contents

1. Motivation
2. Plan
Motivation for "Interactive"ness
Past experiences with our fully automatic analyzers
Bad User Experiences

What we have done was the "automation for Analyzer"
What we wanted perhaps was an “automation for Humans”
Plan for “Interactive”ness
Plan

1. Let’s make things **visible** to the user!

   ✴ Analysis Browser

2. User would be happy to **play** with them!

   ✴ Feedback System
Analysis Browser
Feedback System

• Control Abstraction Directives
  - Branch Discrimination
  - Loop Unrolling

• Data Abstraction Directives
  - Assertions
  - Domain Selection
Branch Discrimination
Branch Discrimination
Branch Discrimination
Other Thoughts

• Leveraging Web Technologies for Interactive Features
  - HTML5 & Javascript for analysis browser and feedback system

• Considering a re-implementation of Airac5
  - Modular & Composite Design with simple independent parts, instead of the complicated monolithic one
  - Java & Scala for worklist algorithm computing fixed-points
  - C (APRON) for core operations on abstract states and values

• Rules of Transparency & Representation
  - Designing for visibility to make inspection and debugging easier.
  - Folding knowledge into data so program logic can be stupid and robust.
<table>
<thead>
<tr>
<th>Phase 1</th>
<th>How control flow graphs of program (G form) and abstract values and states should be serialized are defined, probably in JSON format.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phase 2</td>
<td>Set of Java/Scala libraries are ready which will help us write/read G form and abstract states to/from serialized forms.</td>
</tr>
<tr>
<td>Phase 3</td>
<td>A basic analysis browser lets us navigate through C code or G form and display corresponding abstract states.</td>
</tr>
<tr>
<td>Phase 4</td>
<td>Basic implementation of operations on abstract values and states are ready in Java/Scala.</td>
</tr>
<tr>
<td>Phase 5</td>
<td>Basic implementation of work-list algorithm that computes fixed-points is ready in Java/Scala.</td>
</tr>
<tr>
<td>Phase 6</td>
<td>A front-end that converts C code into G form is ready and can initiate a full abstract interpretation.</td>
</tr>
<tr>
<td>Phase 7</td>
<td>Annotations for the feedback system are defined and their semantics are implemented in the analyzer. The analysis browser now lets user to add/remove annotations while navigating through the program.</td>
</tr>
<tr>
<td>Phase 8</td>
<td>Memory error checkers, e.g. array index range and memory leak checkers, are implemented to compare with Airac5.</td>
</tr>
<tr>
<td>Phase 9</td>
<td>Implementations of operations on abstract values and states are improved for better precision.</td>
</tr>
<tr>
<td>Phase 10</td>
<td>Analysis browser shows control flow graph diagrams with nice layout. It also visualizes the progress of on-line analyses.</td>
</tr>
<tr>
<td>Phase 11</td>
<td>Many parts are optimized for performance and scalability so that large programs can be analyzed, e.g. operations re-implemented in C/JNI, journaling implementation for abstract states, etc.</td>
</tr>
</tbody>
</table>

...
References


