Fast and Precise Points-to Analysis

Jonas Lundberg, Tobias Gutzmann, and Welf Löwe

Växjö University, Sweden

September 29, 2008
Points-to analysis: (Static) dataflow analysis
- Which objects can variable $v$ possibly reference during program execution?
- Compute the points-to set $Pt(v) = \text{set of abstract objects } v \text{ may reference}$
- Abstraction: Map possible \textit{runtime objects $\rightarrow$ abstract objects}
  - usually: group objects created at the same \textit{syntactic location} together
- Provides input data for, e.g., escape analysis, virtual call resolution
- Goals: high precision, fast execution
Our approach

- Static Single Assignment (SSA) form based
- Simulated execution: inter- och intra-procedural flow-sensitivity
- *this*-sensitivity: our new context-sensitive approach, which is much faster and almost as precise as the well-known *object*-sensitivity
Our graph-based SSA program representation, designed especially for Points-to analysis

- non-pointer related operations are removed, e.g., operations related to primitive types
- variables are resolved to edges in the graph
- all dependencies are explicit
- → allows ordering of operations → local flow sensitivity
Simulated Execution

- Simulation of the actual execution of a program
- Start at one or more entry methods
  - interrupt the analysis when a call expression occurs
  - follow the call → continue analyzing the potentially called methods
  - resume with the calling method once analysis of the called method(s) is completed
- → inter- och intra-procedural flow-sensitivity
Distinguish different invocations of a method depending on calling context

Analyze method for each context separately

Calling context:
- call site - from where is the method called?
- functional - depending on current analysis state
Our new functional approach to context-sensitivity.

Contexts distinguished by the points-to set $Pt(a)$

In comparison: The well known *object-sensitive* approach analyzes a target call for each $o \in Pt(a)$

Too similar to be new?
This-Sensitivity vs. Object-Sensitivity

- two (unrelated) calls: $a_1.m()$ and $a_2.m()$
- let $Pt(a_1) = \{o_1, o_2\}$, $Pt(a_2) = \{o_1, o_2, o_3\}$
This-Sensitivity vs. Object-Sensitivity

- two (unrelated) calls: $a_1.m()$ and $a_2.m()$
- let $Pt(a_1) = \{o_1, o_2\}$, $Pt(a_2) = \{o_1, o_2, o_3\}$
- this-sensitivity:
  - need to analyze $foo()$ twice
  - both calls analyzed under different contexts

There are obviously big differences between the two approaches.
two (unrelated) calls: \( a_1.m() \) and \( a_2.m() \)

let \( Pt(a_1) = \{ o_1, o_2 \} \), \( Pt(a_2) = \{ o_1, o_2, o_3 \} \)

this-sensitivity:
- need to analyze \( foo() \) twice
- both calls analyzed under different contexts

object-sensitivity:
- need to analyze \( foo() \) five times
- in total three different contexts
two (unrelated) calls: $a_1.m()$ and $a_2.m()$

let $Pt(a_1) = \{o_1, o_2\}$, $Pt(a_2) = \{o_1, o_2, o_3\}$

this-sensitivity:
- need to analyze $foo()$ twice
- both calls analyzed under different contexts

object-sensitivity:
- need to analyze $foo()$ five times
- in total three different contexts

There are obviously big differences between the two approaches
This-Sensitivity vs. Object-Sensitivity

- Precision: We can show that neither approach is strictly more precise than the other (→ paper)
- Analysis cost:
  - this-sensitivity has, in theory, exponential analysis cost (as there may be $2^n$ contexts for each method, in regard to the number of abstract objects)
  - that doesn’t seem to happen in practice! (And we could easily implement a fail-safe)
  - object-sensitivity: multiple targets for each call
Results - Metrics

- Three precision metrics we present here:
  - **OEdge** and **Enter**:
    - a low number means better precision for *side effect analysis*, *escape analysis* etc.
  - **PCall**:
    - a low number means better precision for *virtual call resolution*
- We have some more metrics in the paper
Results indicate analysis precision relative to context insensitive analysis.

- this-sensitivity is comparably precise to object-sensitivity, except for the OEdge metric
- Other metrics (not on this slide) strengthen the observation that precision is comparable
<table>
<thead>
<tr>
<th>Program</th>
<th>Classes</th>
<th>Context</th>
<th>Time</th>
<th>Context</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>antlr</td>
<td>225</td>
<td>3.36</td>
<td>0.97</td>
<td>3.91</td>
<td>4.65</td>
</tr>
<tr>
<td>javadoc</td>
<td>416</td>
<td>4.61</td>
<td>1.33</td>
<td>10.04</td>
<td>12.65</td>
</tr>
<tr>
<td>emma</td>
<td>749</td>
<td>3.91</td>
<td>0.63</td>
<td>11.63</td>
<td>9.15</td>
</tr>
<tr>
<td>obfusc</td>
<td>688</td>
<td>2.73</td>
<td>1.23</td>
<td>3.36</td>
<td>3.67</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>average</td>
<td></td>
<td>4.15</td>
<td>1.08</td>
<td>7.48</td>
<td>11.1</td>
</tr>
<tr>
<td>median</td>
<td></td>
<td>3.45</td>
<td>1.14</td>
<td>5.37</td>
<td>9.97</td>
</tr>
</tbody>
</table>

- **Classes** is the number of classes in the program $\rightarrow$ input size (does not include library classes)
- **Context** is the average number of contexts per method $\rightarrow$ memory requirement metric
- **Time** is the analysis time as a factor to context insensitive analysis
We have presented our flow-sensitive Points-to analysis

New context sensitive approach to Points-to analysis: *this-sensitivity*

Exponential analysis cost in theory

Almost as fast as context insensitive analysis *in practice*

Experiments show:
- Almost as precise as object-sensitivity
- But much, much faster in practice