Identifying Shareable Objects

Alejandro Infante
ainfante@dcc.uchile.cl
\begin{verbatim}
| b |
b := RTMondrianViewBuilder new.
b shape rectangle
    width: #numberOfVariables;
    height: #numberOfMethods;
    fillColor: [Color new r: 1 g: 0 b: 0].

b nodes: Collection withAllSubclasses.
b edges: Collection withAllSubclasses from: #superclass to: #yourself.
b treeLayout.
b build.
b view open.
\end{verbatim}
Related Work

• **Object Equality Profiling** *(Marinov and O’Callahan)*
  
  • Take a snapshot of the object graph and record last time an object has been written or read to identify potential shareable objects

• **Cachetor: Detecting Cacheable Data to Remove Bloat** *(Nguyen and Xu)*
  
  • *Use a novel technique called Value Abstracted Dependency Analysis at instruction level, then use the graph to detect cacheable values and structures*
Shareable Objects

\(o1 \sim o2\) if all objects referencing \(o1\) can change their reference to \(o2\) preserving the program behavior.
Shareable Objects

\[ o_1 \sim o_2 : \]

\[
\text{class}(o_1) = \text{class}(o_2) \land \\
(\forall f \in \text{instVars}(o_1). (o_1.f \sim o_2.f)) \land \\
\text{immutable}(o_1) \land \\
\text{immutable}(o_2)
\]
Defining Immutability

\[ \forall f \in \text{instVars}(o) \]
\[ \exists t_1, t_2 \land t_1 < t_2 \]
\[ o.f_{t_1} \neq \text{nil} \land o.f_{t_1} \neq o.f_{t_2} \]

Summarizing: Instance variables change only ONCE
Dynamic Analysis

Definition:
- Object
- Primitive Value
- External Classes
  - String*
  - Empty Arrays*

Events:
- Object Creation
- Object first time seen
- Instance Variable Writing
- Object Garbage Collection
Dynamic Analysis

Object Graph in time T

Condensed Graph

Reduce to acyclic graph (ignore the cycles)
## Results

<table>
<thead>
<tr>
<th></th>
<th>Objects</th>
<th>SCC</th>
<th>Avg SCC size</th>
<th>Shareable Sets</th>
<th>Savings</th>
<th>% Saving</th>
</tr>
</thead>
<tbody>
<tr>
<td>Roassal</td>
<td>8.710</td>
<td>722</td>
<td>2.00</td>
<td>4</td>
<td>723</td>
<td>8.3%</td>
</tr>
<tr>
<td>Nautilus</td>
<td>144.287</td>
<td>530</td>
<td>4.19</td>
<td>205</td>
<td>982</td>
<td>0.7%</td>
</tr>
<tr>
<td>Famix</td>
<td>181.450</td>
<td>22.055</td>
<td>3.5</td>
<td>83</td>
<td>135</td>
<td>0.1%</td>
</tr>
</tbody>
</table>
Partially Shareable Objects
For 1000 objects this reduce memory consumption by 29.5%
Partially Shareable Objects

$o_1 \sim o_2$ are partially shareable if 2 or more instance variables of $o_1$ are shareable with the instance variables of $o_2$

Computing this is too expensive!
My Approach

Compute all possible values for each instance variable:

✦ How many possible values instance variable #a has?

✦ How is it related to the number of instances of the class?

✦ How many instance variables have less than 3 possible values for class A?
## Results

<table>
<thead>
<tr>
<th></th>
<th>Instances</th>
<th>Inst Vars</th>
<th>Inst Vars &lt;= 7</th>
<th>Inst Vars &lt;= 3</th>
<th>Inst Vars = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTSVGPath</td>
<td>169</td>
<td>8</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>MorphExtension</td>
<td>3567</td>
<td>15</td>
<td>8</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>MorphTreeNode Morph</td>
<td>428</td>
<td>17</td>
<td>9</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>TableLayoutProperties</td>
<td>1.713</td>
<td>16</td>
<td>14</td>
<td>14</td>
<td>3</td>
</tr>
</tbody>
</table>
Conclusions

• Opportunities of optimization for fully shareable objects are not as frequent as we have thought

• Lots of arbitrary choices made in the research that are worth of discuss, or change to implement new heuristics

• Partially shareable objects promise interesting research opportunities

• The main problems about working with an object graph is Collections and Strongly Connected Components (cycles)
What have I learnt

• Try new possibilities when facing adverse results

• Need for a reliable Smalltalk benchmark for validating results

• Keeping track of objects history is an open problem, since it usually force us to choose between:
  • Efficiency
  • Quality of information (Level of detail)
  • Easiness to design and implement