Dynamic Analysis of Inefficiently-Used Containers

Shengqian (Victor) Yang
Dacong Yan
Guoqing Xu
Atanas Rountev

1 Ohio State University
2 University of California, Irvine

PRESTO: Program Analyses and Software Tools Research Group, Ohio State University
Dynamic Analysis of Containers

- Collections are used in many Java applications

- Performance problems related to containers
  - Low space utilization
  - Inappropriate implementation
  - Unnecessary work

- Existing solutions: look at each container separately

- Our solution: analyze containers in relationship with other containers and the elements that flow between them
Example

ArrayList elements = ...; //C1
ArrayList result = ...; //C2
ArrayList factors = ...; //C3

void filter(){ // filter all elements divisible by at least one factor

    for(int i = 0; i < elements.size(); i++){
        Integer element = elements.get(i);
        boolean divisible = false;
        for(int j = 0; j < factors.size(); j++){
            Integer factor = factors.get(j);
            if(element % factor == 0){
                divisible = true;break;
            }
        }
        if (!divisible) result.add(element);
    }
}
Example

ArrayList elements = ...; //C1
ArrayList result = ...; //C2
ArrayList factors = ...; //C3

void filter(){
    // filter all elements divisible by at least one factor
    for(int i = 0; i < elements.size(); i++){
        Integer element = elements.get(i);
        boolean divisible = false;
        for(int j = 0; j < factors.size(); j++){
            Integer factor = factors.get(j);
            if(element % factor == 0){
                divisible = true; break;
            }
        }
        if (!divisible)
            result.add(element);
    }
}

What if factors is empty?
Example

ArrayList elements = ...; //C1
ArrayList result = ...;     //C2
ArrayList factors = ...;   //C3

void filter(){
    // filter all elements divisible by at least one factor
    for(int i = 0; i < elements.size(); i++) {
        Integer element = elements.get(i);
        boolean divisible = false;
        for(int j = 0; j < factors.size(); j++) {
            Integer factor = factors.get(j);
            if(element % factor == 0) {
                divisible = true; break;
            }
        }
        if (!divisible) result.add(element);
    }
}

What if factors is empty?
Our Approach

• Tracking elements as well as collection API calls
  – API tracking: reveals how elements interact with containers
  – Flow tracking: reveals how elements flow when they are outside of containers

• CEFG (container-element flow graph)
  – Specialized representation showing the flow of elements
  – Goal: Integrating API and flow tracking, CEFG provides basic foundation for analyses related to containers
Element Flow Tracking

- Shadows for every memory location to record the source of element being propagated

```
1 a = new E;
   a.shadow = NewNode(1)
2 b = a;
3 c = new Container;
4 d = new Container;
5 c.add(b);
6 e = c.get(0);
7 f = e;
8 g = f.fld
9 d.add(f);
```
## Element Flow Tracking

- Shadows for every memory location to record the source of element being propagated

<table>
<thead>
<tr>
<th>Code</th>
<th>Shadow</th>
<th>Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a = new E; a.shadow = NewNode(1)</td>
<td><img src="chart.png" alt="" /></td>
</tr>
<tr>
<td>2</td>
<td>b = a; b.shadow = a.shadow</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>c = new Container;</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>d = new Container;</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>c.add(b);</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>e = c.get(0);</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>f = e;</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>g = f.fld</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>d.add(f);</td>
<td></td>
</tr>
</tbody>
</table>
Element Flow Tracking

- Shadows for every memory location to record the source of element being propagated

<table>
<thead>
<tr>
<th>Code</th>
<th>Shadow</th>
<th>Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 a = new E;</td>
<td>a.shadow = NewNode(1)</td>
<td>1</td>
</tr>
<tr>
<td>2 b = a;</td>
<td>b.shadow = a.shadow</td>
<td></td>
</tr>
<tr>
<td>3 c = new Container;</td>
<td>c.shadow = NewNode(3)</td>
<td>3</td>
</tr>
<tr>
<td>4 d = new Container;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 c.add(b);</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 e = c.get(0);</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7 f = e;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8 g = f.fld</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9 d.add(f);</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Element Flow Tracking

- Shadows for every memory location to record the source of element being propagated

<table>
<thead>
<tr>
<th>Code</th>
<th>Shadow</th>
<th>Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 a = new E;</td>
<td>a.shadow = NewNode(1)</td>
<td>1</td>
</tr>
<tr>
<td>2 b = a;</td>
<td>b.shadow = a.shadow</td>
<td>3</td>
</tr>
<tr>
<td>3 c = new Container;</td>
<td>c.shadow = NewNode(3)</td>
<td>3</td>
</tr>
<tr>
<td>4 d = new Container;</td>
<td>d.shadow = NewNode(4)</td>
<td>4</td>
</tr>
<tr>
<td>5 c.add(b);</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 e = c.get(0);</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7 f = e;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8 g = f.fld</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9 d.add(f);</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Element Flow Tracking

- Shadows for every memory location to record the source of element being propagated

<table>
<thead>
<tr>
<th>Code</th>
<th>Shadow</th>
<th>Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 a = new E;</td>
<td>a.shadow = NewNode(1)</td>
<td>1</td>
</tr>
<tr>
<td>2 b = a;</td>
<td>b.shadow = a.shadow</td>
<td></td>
</tr>
<tr>
<td>3 c = new Container;</td>
<td>c.shadow = NewNode(3)</td>
<td>3</td>
</tr>
<tr>
<td>4 d = new Container;</td>
<td>d.shadow = NewNode(4)</td>
<td></td>
</tr>
<tr>
<td>5 c.add(b);</td>
<td>FlowIn(b.shadow, c.shadow)</td>
<td></td>
</tr>
<tr>
<td>6 e = c.get(0);</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7 f = e;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8 g = f.fld</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9 d.add(f);</td>
<td></td>
<td>4</td>
</tr>
</tbody>
</table>
Element Flow Tracking

- Shadows for every memory location to record the source of element being propagated

<table>
<thead>
<tr>
<th>Code</th>
<th>Shadow</th>
<th>Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 a = new E;</td>
<td>a.shadow = NewNode(1)</td>
<td>1</td>
</tr>
<tr>
<td>2 b = a;</td>
<td>b.shadow = a.shadow</td>
<td></td>
</tr>
<tr>
<td>3 c = new Container;</td>
<td>c.shadow = NewNode(3)</td>
<td>3</td>
</tr>
<tr>
<td>4 d = new Container;</td>
<td>d.shadow = NewNode(4)</td>
<td></td>
</tr>
<tr>
<td>5 c.add(b);</td>
<td>FlowIn(b.shadow, c.shadow)</td>
<td>1 → 3</td>
</tr>
<tr>
<td>6 e = c.get(0);</td>
<td>FlowOut(e.shadow, c.shadow)</td>
<td></td>
</tr>
<tr>
<td>7 f = e;</td>
<td>f.shadow = e.shadow</td>
<td></td>
</tr>
<tr>
<td>8 g = f.fld</td>
<td>Use(f.shadow)</td>
<td></td>
</tr>
<tr>
<td>9 d.add(f);</td>
<td></td>
<td>3 → 4</td>
</tr>
</tbody>
</table>
Element Flow Tracking

- Shadows for every memory location to record the source of element being propagated

<table>
<thead>
<tr>
<th>Code</th>
<th>Shadow</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 a = new E;</td>
<td>a.shadow = NewNode(1)</td>
</tr>
<tr>
<td>2 b = a;</td>
<td>b.shadow = a.shadow</td>
</tr>
<tr>
<td>3 c = new Container;</td>
<td>c.shadow = NewNode(3)</td>
</tr>
<tr>
<td>4 d = new Container;</td>
<td>d.shadow = NewNode(4)</td>
</tr>
<tr>
<td>5 c.add(b);</td>
<td>FlowIn(b.shadow, c.shadow)</td>
</tr>
<tr>
<td>6 e = c.get(0);</td>
<td>FlowOut(e.shadow, c.shadow)</td>
</tr>
<tr>
<td>7 f = e;</td>
<td>f.shadow = e.shadow</td>
</tr>
<tr>
<td>8 g = f.fld</td>
<td>Use(f.shadow)</td>
</tr>
<tr>
<td>9 d.add(f);</td>
<td>FlowIn(f.shadow, d.shadow)</td>
</tr>
</tbody>
</table>
Element Flow Tracking

- Shadows for every memory location to record the source of element being propagated

<table>
<thead>
<tr>
<th>Code</th>
<th>Shadow</th>
<th>Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 a = new E;</td>
<td>a.shadow = NewNode(1)</td>
<td><img src="image" alt="Graph Diagram" /></td>
</tr>
<tr>
<td>2 b = a;</td>
<td>b.shadow = a.shadow</td>
<td><img src="image" alt="Graph Diagram" /></td>
</tr>
<tr>
<td>3 c = new Container;</td>
<td>c.shadow = NewNode(3)</td>
<td><img src="image" alt="Graph Diagram" /></td>
</tr>
<tr>
<td>4 d = new Container;</td>
<td>d.shadow = NewNode(4)</td>
<td><img src="image" alt="Graph Diagram" /></td>
</tr>
<tr>
<td>5 c.add(b);</td>
<td>FlowIn(b.shadow, c.shadow)</td>
<td><img src="image" alt="Graph Diagram" /></td>
</tr>
<tr>
<td>6 e = c.get(0);</td>
<td>FlowOut(e.shadow, c.shadow)</td>
<td><img src="image" alt="Graph Diagram" /></td>
</tr>
<tr>
<td>7 f = e;</td>
<td>f.shadow = e.shadow</td>
<td><img src="image" alt="Graph Diagram" /></td>
</tr>
<tr>
<td>8 g = f.fld</td>
<td>Use(f.shadow)</td>
<td><img src="image" alt="Graph Diagram" /></td>
</tr>
<tr>
<td>9 d.add(f);</td>
<td>FlowIn(f.shadow, d.shadow)</td>
<td><img src="image" alt="Graph Diagram" /></td>
</tr>
</tbody>
</table>
CEFG Nodes

• Types of nodes
  - Container node (represents allocation site of containers)
  - Element node (shows allocation site of elements)
  - Phantom node associated with container node

• Counters at a node
  - Number of objects created by an allocation site
  - For a container node: number of retrieve and add operations
CEFG Edges

• An edge shows that elements flow out of or into containers
  - Source node: container or element node
  - Target node: container or phantom node

• Counters at an edge
  - Number of elements that flow through this edge
  - Number of elements which are used when they flow through this edge
Example

```java
ArrayList elements = ...; //C1
ArrayList result = ...;    //C2
ArrayList factors = ...;   //C3

void filter(){
    for(int i = 0; i < elements.size(); i++){
        Integer element = elements.get(i);
        boolean divisible = false;
        for(int j = 0; j < factors.size(); j++){
            Integer factor = factors.get(j);
            if(element % factor == 0){
                divisible = true;break;
            }
        }
        if (!divisible) result.add(element);
    }
}
```
Implementation

• Element flow tracking
  – Implemented in Jikes RVM
  – Modify the runtime compiler for code instrumentation
  – Create *shadow locations* to track data dependences
  – Instrument method calls for tracking of inter-procedural flow

• Overhead
  – Time: 3-35x
  – Space: 1.2-4x
Client Analyses

• Intermediate containers
  − Identify containers not serving a useful purpose

• Under-utilized containers
  − Identify containers that have a small number of elements

• Over-populated containers
  − Identify containers that maintain many never-retrieved elements
Inefficiency Detection

• Intermediate container
  – Symptom: elements flow out of C1, but they are never used

• Under-utilized container
  – Symptom: #adds is very small

• Over-populated container
  – Symptom: #retrieves << #adds (not in this example)
Tuning

ArrayList elements = ...;  //C1

ArrayList result = null;

ArrayList factors = ...;  //C3

void filter(){
    result = ...;  //C2
    for(int i = 0; i < elements.size(); i++){
        Integer element = elements.get(i);
        boolean divisible = false;
        for(int j = 0; j < factors.size(); j++){
            Integer factor = factors.get(j);
            if(element % factor == 0){
                divisible = true; break;
            }
        }
        if (!divisible) result.add(element);
    }
}

if(factors.size() == 0){  
    result = elements;  
    return;
}
Case Studies

• Evaluation on DaCapo benchmarks and additional programs

  - reductions for pmd (reason: intermediate + over-populated container): 13.6% in running time and 12.5% in #allocated objects

  empty set of filters; redundant wrapper objects

  - bloat (intermediate + over-populated container): 15.6% and 14.0%

  inefficiently computing the size of the union of two sets; redundant wrappers
Case Studies

- **ps** (intermediate container): 13.5% and 56.0%
  inappropriate use of stack APIs
- **soot** (intermediate container): 4.0% and 3.1%
  multiple representations of the same information
- **lusearch** (under-utilized container): 4.8% and 3.4%
  eager vs on-demand creation of containers;
  redundant work for a frequent special case
Conclusion

• CEFG provides an overview of the movements of elements among containers

• Analyses based on the CEFG can detect certain categories of inefficiencies related to containers

• Future work
  – CEFG visualization and understanding
  – More detector analyses
  – Finer-grain abstractions (e.g. context-sensitive nodes)
Thank you