Complexity-Guided Container Replacement Synthesis

Chengpeng Wang  Prism Group, HKUST
Peisen Yao      Prism Group, HKUST
Wensheng Tang  Prism Group, HKUST
Qingkai Shi    Ant Group -> Purdue University
Charles Zhang  Prism Group, HKUST
Container

• General-purpose abstract data type
  • Inserting, retrieving, removing and iterating over elements
  • E.g., ArrayList, HashMap, HashSet, etc

• A variety of implementations
Performance Profile

• Resource consumption differs [Hasan, ICSE 16]
Container Selection

- Programmers are often
  - Unaware of how container objects are manipulated
    - Focus on specific modules of applications
  - Unaware of performance difference of container method calls
    - Unfamiliar with new implementations provided by libraries

Assist programmers in finding proper container types
Our Aim

• Synthesize container replacements automatically to reduce the resource consumption
  • Container replacement
    • Container types in allocation statements
    • Container method calls
  • Resource consumption
    • Time, memory, CPU usage, energy
    • Focus on time cost but can be generalized
Existing Studies

- Solving an optimization problem [Basios, FSE 18] [Manotas, ICSE 14]
  - Enumerate container types
  - Monitor resource consumption when executing test cases
  - Find the optimal replacement to minimize the resource consumption
Existing Studies

• Solving a prediction problem [Jung, PLDI 11] [Vechev, PLDI 09]
  • Profile the program to obtain runtime info
  • Apply pre-trained model or pre-defined rules
  • Infer the container replacement
Limitations of Existing Approaches

• Huge overhead
  • Execute programs with test suites to profile dynamically

• Overfitting
  • Optimal replacements for specific inputs rather than general inputs

• Unsoundness
  • Unable to preserve behavioral equivalence, e.g., replace TreeSet with HashSet
Problem Formulation

• Replace container types and container methods in the program P and obtain a new program P’, such that
  • (Behavioral equivalence) P and P’ are behavioral equivalent
  • (Complexity superiority) P’ consumes less time than P for a sufficiently large input

*Behavioral Equivalence:*
*For any given input, P and P’ always return the same value*
Two Critical Goals

• Which container types are exchangeable to ensure behavioral equivalence?

• How to measure the performance of a container-manipulating program to check complexity superiority?
Goal I: Behavioral Equivalence

• Which container types are exchangeable to ensure behavioral equivalence?

```java
public boolean foo1(String area) {
    ArrayList<String> l = new ArrayList<>();
    l.add("PL"); l.add("SE");
    boolean b1 = l.contains(area);
    return b1;
}
```

{LinkedList, HashSet, TreeSet, …} whether a value is in the list

• Exchangeable container types achieve the original container usage intention.
Two Classes of Container Usage Intention

• Container-property queries
• Container-property modifiers
Class I: Container Property Queries

• Value ownership
  • ArrayList.contains(O), HashSet.contains(O)
• Index ownership
  • HashMap.containsKey(O)
• Index-value correlation
  • ArrayList.get(I), HashMap.get(O)
• Size
  • ArrayList.size(), HashSet.size()
• Insertion order
  • LinkedHashMap.iterator()
• Key order
  • TreeMap.firstKey(), TreeMap.lastKey()
Class II: Container Property Modifiers

• A container method can update container properties
  • Support querying container properties in other program locations

```java
public boolean foo1(String area) {
    ArrayList<String> l = new ArrayList<>();
    l.add("PL"); l.add("SE");
    boolean b1 = l.contains(area);
    return b1;
}
```

ArrayList.add(O):

- increase the size by 1
- add a new value
- add a new value at the end
- Modify

size
value ownership
index-value correlation
Method Semantic Specification

- Decompose method semantics into container-property queries and container-property modifiers

<table>
<thead>
<tr>
<th>Method</th>
<th>Queries</th>
<th>Modifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArrayList.contains(O)</td>
<td>{ isVal }</td>
<td>{ }</td>
</tr>
<tr>
<td>HashSet.contains(O)</td>
<td>{ isVal }</td>
<td>{ }</td>
</tr>
<tr>
<td>ArrayList.get(I)</td>
<td>{ isCor }</td>
<td>{ }</td>
</tr>
</tbody>
</table>

isVal: query value ownership
isCor: query index-value correlation
Method Semantic Specification

- Decompose method semantics into container-property queries and container-property modifiers

<table>
<thead>
<tr>
<th>Method</th>
<th>Queries</th>
<th>Modifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArrayList.add(O)</td>
<td>{ }</td>
<td>{ t-size_{=1} \ t-val_{+} \ t-cor_{e} }</td>
</tr>
<tr>
<td>HashSet.add(O)</td>
<td>{ }</td>
<td>{ t-size_{\leq 1} \ t-val_{+} }</td>
</tr>
</tbody>
</table>

- $t-size_{=1}$: increase the size by 1
- $t-size_{\leq 1}$: increase the size by at most 1
- $t-val_{+}$: add a new value
- $t-cor_{e}$: add a new value at the end
Key Idea: Achieve Original Usage Intention

• Support the original container-property queries
• Modify the queried container properties as the original ones

Original program:

```java
public boolean foo1(String area) {
    ArrayList<String> l = new ArrayList<>();
    l.add("PL"); l.add("SE");
    boolean b1 = l.contains(area);
    return b1;
}
```

{t-size$_1^\uparrow$, t-val$_\uparrow$, t-cor$_\uparrow$}

∅

{t-size$_1^\uparrow$, t-val$_\uparrow$, t-cor$_\uparrow$}
Key Idea: Achieve Original Usage Intention

- Support the original container-property queries
- Modify the queried container properties as the original ones
Key Idea: Achieve Original Usage Intention

• Support the original container-property queries
• Modify the queried container properties as the original ones

Guarantee behavioral equivalence
Ensuring Behavioral Equivalence

• Achieve the original usage intention with exchangeable container types
  • Support the original container-property queries
  • Modify the queried container properties as the original ones
Goal II: Complexity Superiority

• How to measure the performance of a container-manipulating program to check complexity superiority?

```java
public boolean foo1(String area) {
    ArrayList<String> l = new ArrayList<>();
    l.add(“PL”);  l.add(“SE”);
    boolean b1 = l.contains(area);
    return b1;
}
```

```java
public boolean foo2(String area) {
    HashSet<String> s = new HashSet<>();
    s.add(“PL”);  s.add(“SE”);
    boolean b2 = s.contains(area);
    return b2;
}
```

• Only measure the time costs of container method calls.
Method Complexity Specification

- Cost model CS
  - Complexity classes
    - Constant
    - Amortized constant
    - Logarithmic
    - Amortized logarithmic
    - Linear
    - Amortized linear
    - Super linear
  - Complexity functions

<table>
<thead>
<tr>
<th></th>
<th>constant</th>
<th>amortized constant</th>
<th>logarithmic</th>
<th>amortized logarithmic</th>
<th>linear</th>
<th>amortized linear</th>
<th>super linear</th>
</tr>
</thead>
<tbody>
<tr>
<td>( tc_1(n) = 1 )</td>
<td>( tc_2(n) )</td>
<td>( tc_3(n) = \log n )</td>
<td>( tc_4(n) )</td>
<td>( tc_5(n) = n )</td>
<td>( tc_6(n) )</td>
<td>( tc_7(n) )</td>
<td></td>
</tr>
</tbody>
</table>
Method Complexity Specification

• Cost model CS
  • Constant factor

<table>
<thead>
<tr>
<th>Container</th>
<th>Method</th>
<th>Complexity Function</th>
<th>$\theta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArrayList</td>
<td>add(O)</td>
<td>$tc_2(n)$</td>
<td>1</td>
</tr>
<tr>
<td>ArrayList</td>
<td>add(I, O)</td>
<td>$tc_2(n)$</td>
<td>2</td>
</tr>
<tr>
<td>ArrayList</td>
<td>contains(O)</td>
<td>$tc_5(n)$</td>
<td>1</td>
</tr>
<tr>
<td>ArrayList</td>
<td>get(I)</td>
<td>$tc_1(n)$</td>
<td>1</td>
</tr>
<tr>
<td>ArrayList</td>
<td>iterator()</td>
<td>$tc_1(n)$</td>
<td>1</td>
</tr>
<tr>
<td>ArrayList</td>
<td>remove(I)</td>
<td>$tc_1(n)$</td>
<td>2</td>
</tr>
<tr>
<td>ArrayList</td>
<td>size()</td>
<td>$tc_1(n)$</td>
<td>1</td>
</tr>
</tbody>
</table>

$tc_1(n)$: constant

$tc_2(n)$: amortized constant

$tc_5(n)$: linear

<table>
<thead>
<tr>
<th>Container</th>
<th>Method</th>
<th>Complexity Function</th>
<th>$\theta$</th>
<th>CS(ArrayList.add(O)) = 1 \cdot tc_2(n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HashSet</td>
<td>add(O)</td>
<td>$tc_2(n)$</td>
<td>2</td>
<td>CS(HashSet.add(O)) = 2 \cdot tc_2(n)</td>
</tr>
<tr>
<td>HashSet</td>
<td>contains(O)</td>
<td>$tc_1(n)$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>HashSet</td>
<td>iterator()</td>
<td>$tc_1(n)$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>HashSet</td>
<td>remove(O)</td>
<td>$tc_2(n)$</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>HashSet</td>
<td>size()</td>
<td>$tc_1(n)$</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Checking Complexity Superiority

• How to measure the performance of a container-manipulating program to check complexity superiority?

• Introduce *container complexity superiority*
  • For each container object o, S and S' are the sets of methods manipulating o in P and P', we need to ensure that

\[ \sum_{f' \in S'} CS(f') \leq \sum_{f \in S} CS(f) \]
Key Idea: Container Complexity Superiority

• For each container object o, S and S’ are the sets of methods manipulating o in P and P’, we need to ensure that

\[
\sum_{f \in S'} CS(f') \leq \sum_{f \in S} CS(f)
\]

public boolean foo1(String area) {
    ArrayList<String> l = new ArrayList<>();
    l.add("PL");  l.add("SE");
    boolean b1 = l.contains(area);
    return b1;
}

\(2 \cdot tn_2(n) + tn_5(n)\)

public boolean foo2(String area) {
    HashSet<String> s = new HashSet<>();
    s.add("PL");  s.add("SE");
    boolean b2 = s.contains(area);
    return b2;
}

\(4 \cdot tn_2(n) + tn_1(n)\)

\(tc_1(n): \text{constant} \quad tc_2(n): \text{amortized constant} \quad tc_5(n): \text{linear} \)
Cres: Synthesizing Container Replacement

• Achieve the original intention of container usage to ensure behavioral equivalence
  • Support the original container-property queries
  • Modify the queried container properties as the original ones

• Achieve container complexity superiority to improve program efficiency
  • For each container object o, S and S’ are the sets of methods manipulating o in P and P’, we need to ensure that

\[
\sum_{f' \in S'} CS(f') \leq \sum_{f \in S} CS(f)
\]
Workflow of Cres

- Method semantic specification (queries, modifiers)
- Method complexity specification (complexity function, constant factor)
Stage I: Container Property Analysis

• Collect queried container properties for each container object via a sound points-to analysis

```java
1  public List getUniqueResource(String curdir) {
2      List uniqueResources = new ArrayList<IResource>();
3      for (int i = 0; i < RESOURCE_NUM; i++) {
4          IResource s = getResourceInCurrentDir(curdir, i);
5          if (!uniqueResources.contains(s))
6              uniqueResources.add(s);
7      }
8      return uniqueResources;
9  }
10     public List computeAllDirs() {
11        List dirs = new ArrayList<Dir>();
12        for (int i = 0; i < DIR_NUM; i++)
13            dirs.add(getSubDirInWorkingDir(i));
14        return dirs;
15    }
16    public void main() {
17        List resources = getUniqueResource("/home/OOPSLA");
18        List dirs = computeAllDirs();
19        for (Dir dir : dirs) {
20            if (resources.contains(dir))
21                System.out.println("Accessible Resource");
22        }
23    }
```
Stage II: Method Candidate Identification

- A method f' is a candidate for the method call v=c.f(u) iff
  - f and f' support the same container-property queries
  - f and f' have the same modifiers on the queried container properties

```java
public List getUniqueResource(String curdir) {
    List uniqueResources = new ArrayList<IResource>();
    for (int i = 0; i < RESOURCE_NUM; i++) {
        IResource s = getResourceInCurrentDir(curdir, i);
        if (!uniqueResources.contains(s))
            uniqueResources.add(s);
    }
    return uniqueResources;
}
```

```java
for (int i = 0; i < DIR_NUM; i++)
    dirs.add(getSubDirInWorkingDir(i));
return dirs;
```

```java
public void main() {
    List resources = getUniqueResource("/home/OOPSLA");
    List dirs = computeAllDirs();
    for (Dir dir : dirs) {
        if (resources.contains(dir))
            System.out.println("Accessible Resource");
    }
}
Stage III: Container Replacement Synthesis

• For each container object \( o \), \( S \) and \( S' \) are the sets of methods manipulating \( o \) in \( P \) and \( P' \), we need to ensure that

\[
\sum_{f' \in S'} CS(f') \leq \sum_{f \in S} CS(f)
\]

2 * \( CS(\text{HashSet.contains}) + CS(\text{HashSet.add}) \) is minimal
Theoretical Results

• Theorem 1: The new program $P'$ is the behavioral equivalent to the original program $P$.

• Theorem 2: The new program $P'$ has container complexity superiority over the original program $P$.

• Theorem 3: The time complexity of the algorithm is $O(|S_a| \cdot |S_c|)$ for given container types.
  • $S_a$ and $S_c$ contain container allocation statements and container method calls, respectively.
Implementation of Cres

- Implement Cres based on Pinpoint [Shi, PLDI 18]
  - Flow-sensitive points-to analysis for container property analysis

- Analyze containers in Java Collection Framework
  - List: ArrayList, LinkedList
  - Set: HashSet, LinkedHashSet, TreeSet
  - Map: HashMap, LinkedHashSetMap, TreeMap
Research Questions

• RQ1: Effectiveness
  • Performance improvement brought by Cres

• RQ2: Replacement patterns
  • Kinds and numbers of replacements Cres synthesize

• RQ3: Overhead
  • The time and space costs of Cres
Evaluation: Effectiveness

• What is the improvement Cres achieves for real-world programs?

<table>
<thead>
<tr>
<th>Project</th>
<th>Description</th>
<th>Size (KLoC)</th>
<th>Medium (%)</th>
<th>95% CI (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>bootique</td>
<td>Microservice platform</td>
<td>18.6</td>
<td>4.5</td>
<td>[4.4, 4.6]</td>
</tr>
<tr>
<td>mapper</td>
<td>Server application</td>
<td>22.4</td>
<td>7.3</td>
<td>[7.0, 7.6]</td>
</tr>
<tr>
<td>incubator-eventmesh</td>
<td>Eventing infrastructure</td>
<td>24.9</td>
<td>4.1</td>
<td>[3.9, 4.3]</td>
</tr>
<tr>
<td>google-http-java-client</td>
<td>Web client</td>
<td>25.2</td>
<td>27.1</td>
<td>[25.9, 28.3]</td>
</tr>
<tr>
<td>light-4j</td>
<td>Microservice platform</td>
<td>44.3</td>
<td>5.2</td>
<td>[5.0, 5.4]</td>
</tr>
<tr>
<td>roller</td>
<td>Server application</td>
<td>54.4</td>
<td>9.5</td>
<td>[9.2, 9.8]</td>
</tr>
<tr>
<td>IginX</td>
<td>Data management system</td>
<td>68.1</td>
<td>3.5</td>
<td>[3.4, 3.6]</td>
</tr>
<tr>
<td>sofa-rpc</td>
<td>RPC framework</td>
<td>76.4</td>
<td>3.7</td>
<td>[3.4, 4.0]</td>
</tr>
<tr>
<td>Glowstone</td>
<td>Server application</td>
<td>85.6</td>
<td>13.1</td>
<td>[12.9, 13.3]</td>
</tr>
<tr>
<td>dolphinscheduler</td>
<td>Eventing infrastructure</td>
<td>89.5</td>
<td>5.3</td>
<td>[5.1, 5.5]</td>
</tr>
<tr>
<td>dubbo</td>
<td>RPC framework</td>
<td>196.5</td>
<td>7.5</td>
<td>[7.2, 7.8]</td>
</tr>
<tr>
<td>iotdb</td>
<td>Data management system</td>
<td>384.2</td>
<td>6.3</td>
<td>[6.2, 6.4]</td>
</tr>
</tbody>
</table>

Speedup: On average 8.1%, up to 27.1%
Evaluation: Replacement Pattern

• Which kinds of container replacements does Cres synthesize?

<table>
<thead>
<tr>
<th>Project</th>
<th>#Conf/#Total</th>
<th>#R1</th>
<th>#R2</th>
<th>#R3</th>
<th>#R4</th>
<th>#R5</th>
<th>#R6</th>
</tr>
</thead>
<tbody>
<tr>
<td>bootique</td>
<td>0/4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>mapper</td>
<td>0/6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>incubator-eventmesh</td>
<td>19/19</td>
<td>1</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>google-http-java-client</td>
<td>0/4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>light-4j</td>
<td>0/5</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>roller</td>
<td>0/6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>IginnX</td>
<td>11/11</td>
<td>9</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>sofa-rpc</td>
<td>12/12</td>
<td></td>
<td></td>
<td></td>
<td>5</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Glowstone</td>
<td>0/11</td>
<td></td>
<td></td>
<td>6</td>
<td></td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>dolphinscheduler</td>
<td>7/7</td>
<td></td>
<td></td>
<td></td>
<td>6</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>dubbo</td>
<td>12/12</td>
<td>1</td>
<td>3</td>
<td></td>
<td>1</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>iotdb</td>
<td>10/10</td>
<td>2</td>
<td>1</td>
<td></td>
<td>6</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>71/107</strong></td>
<td><strong>4</strong></td>
<td><strong>57</strong></td>
<td><strong>25</strong></td>
<td><strong>4</strong></td>
<td><strong>4</strong></td>
<td><strong>13</strong></td>
</tr>
</tbody>
</table>

R1: LinkedList → ArrayList
R2: ArrayList → LinkedList
R3: ArrayList → HashSet
R4: TreeMap → HashMap
R5: LinkedHashMap → HashMap
R6: LinkedHashMap → HashSet

71 confirmed replacements
Evaluation: Time and Memory Costs

• What are the time and memory costs of Cres?

 linear scalability
~ 14 minutes analyzing 384.2 KLoC
Interesting Findings

• Equipped with flow-insensitive pointer analysis, Cres synthesizes 74 container replacements out of 107 replacements.
  • Miss several optimization opportunities due to the imprecision of container property analysis
Interesting Findings

• Using randomly generated constant factors in the method complexity specification does not affect the result as long as they conform to a specific order.

\[
\begin{array}{|c|c|c|c|}
\hline
\text{Container} & \text{Method} & \text{Complexity Function} & \theta \\
\hline
\text{HashSet} & \text{add(O)} & \theta_2 \text{c}_{2}(n) & 2 \\
\text{HashSet} & \text{contains(O)} & \theta_1 \text{c}_{1}(n) & 1 \\
\text{HashSet} & \text{iterator()} & \theta_3 \text{c}_{1}(n) & 1 \\
\text{HashSet} & \text{remove(O)} & \theta_2 \text{c}_{2}(n) & 2 \\
\text{HashSet} & \text{size()} & \theta_1 \text{c}_{1}(n) & 1 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|c|c|}
\hline
\text{Container} & \text{Method} & \text{Complexity Function} & \theta \\
\hline
\text{LinkedHashSet} & \text{add(O)} & \theta_2 \text{c}_{2}(n) & 3 \\
\text{LinkedHashSet} & \text{contains(O)} & \theta_1 \text{c}_{1}(n) & 1 \\
\text{LinkedHashSet} & \text{iterator()} & \theta_3 \text{c}_{1}(n) & 1 \\
\text{LinkedHashSet} & \text{remove(O)} & \theta_2 \text{c}_{2}(n) & 3 \\
\text{LinkedHashSet} & \text{size()} & \theta_1 \text{c}_{1}(n) & 1 \\
\hline
\end{array}
\]

\[
\begin{align*}
\text{HashSet}.\text{add(O)} & < \text{LinkedHashSet}.\text{add(O)} & \theta_1 < \theta_2 \\
\text{HashSet}.\text{remove(O)} & < \text{LinkedHashSet}.\text{remove(O)} & \theta_3 < \theta_4
\end{align*}
\]
Drawbacks

• Container complexity superiority does not imply complexity superiority.

$$\sum_{f' \in S'} CS(f') \leq \sum_{f \in S} CS(f) \quad \Rightarrow \quad \text{Time complexity of } P' \leq \text{Time complexity of } P$$

• Complexity analysis/WCET analysis are impractical for real-world programs.
• Need a precise and computable complexity guidance
Drawbacks

- Unaware of usage intention of loops
  - Example from IoTDB: `pageReader` is a LinkedList object

```java
public Point retrieveValidLastPoint(int n) {
    List<IArchiveMetadata> seqDataList = new LinkedList<>();
    for (int i = 0; i < n; i++)
        seqDataList.add(getDataFromDevice());
    for (int i = seqDataList.size() - 1; i >= 0; i--)
        Point lastPoint = getChunkLastPoint(seqDataList.get(i));
    if (lastPoint.getValue() != null)
        return lastPoint;
    return null;
}
```

*Cres: LinkedList => ArrayList
Optimal solution: use iterators*
Conclusions

• A new program abstraction with
  • Container property & Method semantic specification
  • Cost model & Method complexity specification

• An efficient and sound synthesis algorithm Cres
  • Ensuring behavioral equivalence with container property analysis
  • Improving program efficiency with complexity-guided synthesis
Thank you for your listening!