Evaluating the Impact of Context-Sensitivity on Andersen’s Algorithm for Java Programs

Donglin Liang
University of Minnesota

Collaborate with Maikel Pennings and Mary Jean Harrold

Andersen’s Algorithm
- One points-to graph for the whole program
- Flow-insensitive and context-insensitive
  - flow-insensitive
    - process statements in an arbitrary order
  - context-insensitive
    - name objects by allocation sites
    - simulate parameter passing and method return using assignments

An Example
```java
1 class A {
2     Object f;
3     Object alloc() {
4         return new Object();
5     }
6     A() {
7         this.f = this.alloc();
8     }
9     static main() {
10        A a1 = new A();
11        A a2 = new A();
12        A p = a2.alloc();
13     }
14 }
```

Overview
Andersen’s algorithm
- How it works
- Why it is imprecise

Implementation and empirical studies
- Comparing the information computed by different algorithms with that collected during the execution

A context-sensitive extension to Andersen’s algorithm
- Call-string contexts
- Receiver contexts

An Example
```
alloc:ret= o4;
alloc:this = A:this;
A:this.f = alloc:ret;
A:this=o10; a1=o10;
A:this=o11; a2=o11;
alloc:this=a2; p = alloc:ret;
```
class A {
    Object f;
    Object alloc() { return new Object(); }
    A() {
        this.f = this.alloc();
    }
    static main() {
        A a1 = new A();
        A a2 = new A();
        A p = a2.alloc();
    }
}

An Example

```
alloc:ret = o4;
alloc:this = A:this;
A:this.f = alloc:ret;
A:this=o10; a1=o10;
A:this=o11; a2=o11;
alloc:this=a2; p = alloc:ret;
```
A Reason for Imprecision: Context-Insensitivity

```java
class A {
    Object f;
    Object alloc() {
        return new Object();
    }
    A() {
        this.f = this.alloc();
    }
    static main() {
        A a1 = new A();
        A a2 = new A();
        A p = a2.alloc();
    }
}
```

A Context-sensitive Extension to Andersen's Algorithm

- Start the analysis from "main()"
- During the analysis, iteratively
  - Discover the contexts under which a method may be invoked in the program
  - Analyze each method specifically under each context
    - Clone the local variables and parameters
    - Clone the assignments
    - Name the instances allocated in the method by annotating the allocation site with the context

Challenge: There may be intractable (or even infinite) number of possible contexts.

Abstracter the Contexts: a traditional approach

- Bounded call-strings
  - Using the string of statement numbers of the k top-most callsites
  - Level-2 call-string contexts for M3()
    - "cg", "eg", "df", "fr"

Growing Object Names and Receiver Contexts

- Receiver contexts come from object names
- Object names are created by annotating the allocation sites with receiver contexts

```java
class A {
    Object f;
    Object alloc() {
        return new Object();
    }
    A() {
        this.f = this.alloc();
    }
    static main() {
        A a1 = new A();
        A a2 = new A();
        A p = a2.alloc();
    }
}
```
### Overview

Andersen's algorithm
- How it works
- Why it is imprecise

A context-sensitive extension to Andersen's algorithm
- Call-string contexts
- Receiver contexts

Implementation and empirical studies
- Comparing the information computed by different algorithms with that collected during the execution

### Questions to Answer about the Context-Sensitive Andersen's Algorithms
- How does a context-sensitive version of the algorithm compare with the context-insensitive version?
- How does the calling-string context approach compare with the receiver context approach?
- How would increasing the allowing size of contexts affect the performance?
- How would the client analyses benefit from the context-sensitive reference information?
- How to design a reference analysis algorithm for Java that achieves the best trade-off between precision and efficiency?
Challenges in Evaluating Reference Analysis Algorithms

- No unique metrics
- No absolutely precise information to compare with
- No well-accepted benchmarks

An additional challenge for our studies

Results of different algorithms not directly comparable:
- The instances are identified with different naming schemes.

Our Evaluation Approach

- Compare the static results with the dynamic information
  - Static results: computed by each algorithm
    - Provide names for each possible instance
    - Estimate how a potential instance identified by a name may be used at the call sites
  - Dynamic info: collected by a profiler
    - Provide info for mapping an instance to a static name
    - Record how a real instance is actually used at the call sites
- Measure the accuracy in estimating how each instance will be used
  - Compute a precision reference value (PRV) for each instance

Computing PRVs

\[ C[i] = \{ \text{callsites recorded for instance } i \text{ by profiler} \} \]

\[ C_A[n] = \{ \text{callsites recorded for name } n \text{ according to the info computed by algorithm } A \} \]

\[ Cover = \{ \text{callsites hit at least once during some execution} \} \]

If \( n \) is the name for \( i \) in \( A \), then the PRV for \( i \) according to \( A \) is

\[ r_A[i] = \frac{|C[i]|}{|C_A[n] \cap Cover|} \]

An Example

```
10 foo(ret flag) {
11  A p = new A(); // o11
12  if (flag > 0)
13     p.m1();
14     p.m2();
17 }
```

Static info:

- \( C(11) = \{11,13,15\} \)
- Assuming that instance 100 is created by 11 when flag is >0 and instance 200 is created by 11 when flag is =0
- \( C(100) = \{11,13,15\} \)
- \( C(200) = \{11,15\} \)
- \( r[100] = 3/3 = 1.0 \)
- \( r[200] = 2/3 = 0.66 \)

Settings

- Algorithms to evaluate
  - Without using contexts (Cl)
  - Levels 1, 2, 3 call-string (CS1,CS2,CS3) contexts
  - Level 3 call-string contexts but without contexts for naming objects (SemIC3S)
  - Levels 1, 2 name-bounded (NB1,NB2) or context-bounded (CB1,CB2) receiver contexts
- Implementation: focus attention on application code
  - Use models to avoid analyzing the library code
- Subjects: 12 Java programs collected from various sources

<table>
<thead>
<tr>
<th>program</th>
<th>Loc</th>
<th>Cls</th>
<th>Meth</th>
<th>And</th>
<th>Dyn</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>antlr</td>
<td>32030</td>
<td>148</td>
<td>1858</td>
<td>950</td>
<td>682</td>
<td>72%</td>
</tr>
<tr>
<td>jar</td>
<td>1839</td>
<td>8</td>
<td>89</td>
<td>36</td>
<td>24</td>
<td>67%</td>
</tr>
<tr>
<td>jas</td>
<td>4020</td>
<td>116</td>
<td>413</td>
<td>295</td>
<td>264</td>
<td>89%</td>
</tr>
<tr>
<td>java_cup</td>
<td>10155</td>
<td>35</td>
<td>372</td>
<td>260</td>
<td>206</td>
<td>77%</td>
</tr>
<tr>
<td>jb</td>
<td>6025</td>
<td>45</td>
<td>548</td>
<td>184</td>
<td>100</td>
<td>54%</td>
</tr>
<tr>
<td>jess</td>
<td>19017</td>
<td>207</td>
<td>1132</td>
<td>974</td>
<td>757</td>
<td>78%</td>
</tr>
<tr>
<td>jfe</td>
<td>31636</td>
<td>310</td>
<td>1837</td>
<td>830</td>
<td>726</td>
<td>87%</td>
</tr>
<tr>
<td>jflex</td>
<td>7351</td>
<td>20</td>
<td>134</td>
<td>105</td>
<td>92</td>
<td>88%</td>
</tr>
<tr>
<td>jtr</td>
<td>11904</td>
<td>40</td>
<td>202</td>
<td>147</td>
<td>97</td>
<td>66%</td>
</tr>
<tr>
<td>jraja</td>
<td>6369</td>
<td>65</td>
<td>391</td>
<td>37</td>
<td>32</td>
<td>86%</td>
</tr>
<tr>
<td>sablecc</td>
<td>44521</td>
<td>295</td>
<td>2025</td>
<td>1508</td>
<td>1376</td>
<td>86%</td>
</tr>
<tr>
<td>toba</td>
<td>6417</td>
<td>26</td>
<td>196</td>
<td>150</td>
<td>105</td>
<td>70%</td>
</tr>
</tbody>
</table>
Comparing the PRVs for Instances

Percentage of instances whose PRVs are below a certain value

Compare Average PRVs for Allocation Sites

$I_a = \{\text{Instances allocated at allocation site } a\}$

Average PRV for $a$: $\bar{P}_A[a] = \frac{\sum_{i \in I_a} P_A[i]}{|I_a|}$

Compare two algorithms $A$ and $B$: $Diff_{A/B}[a] = \bar{P}_A[a] - \bar{P}_B[a]$

Observations

- For some subjects, considering contexts have insignificant effects on the precision of the computed information
- For some other subjects, considering contexts can improve the precision of the computed information
  - The two different types of contexts seem to offer their unique improvements
  - More context information does not always mean significant better precision
Observations

- Distinguishing the instances allocated at the same statement with contexts is crucial for getting more precise reference information.

Deriving Context-Insensitive Information

- Many (traditional) analyses compute information that will be true for all contexts
- Slicing, modification side-effect
- Two approaches

<table>
<thead>
<tr>
<th>Andersen: analyze methods under each context</th>
<th>Client: analyze methods under each context</th>
<th>Summarize: merge info computed for all contexts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Andersen: analyze methods under each context</td>
<td>Summarize: merge info computed for all contexts</td>
<td>Client: analyze methods without distinguishing context</td>
</tr>
</tbody>
</table>

Comparing with Info Computed by the Context-Insensitive Algorithm

\[ C_a[a] = \{ \text{callsites recorded for allocation site } a \text{ according to the info computed by algorithm } A \} \]

\[ Diff_{C_{\text{A}}, C_{\text{A}}}[a] = \frac{\|C_a[a] - C_p[a]\|}{|C_a[a]|} \times 100 \]

<table>
<thead>
<tr>
<th>The Distribution of Diff[%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>program</td>
</tr>
<tr>
<td>-----------------------------</td>
</tr>
<tr>
<td>Jb</td>
</tr>
<tr>
<td>Jb</td>
</tr>
<tr>
<td>Jb</td>
</tr>
<tr>
<td>Jb</td>
</tr>
<tr>
<td>Jb</td>
</tr>
<tr>
<td>Jb</td>
</tr>
<tr>
<td>Jb</td>
</tr>
<tr>
<td>Jb</td>
</tr>
<tr>
<td>Jb</td>
</tr>
<tr>
<td>Jb</td>
</tr>
<tr>
<td>Jb</td>
</tr>
<tr>
<td>Jb</td>
</tr>
<tr>
<td>Jb</td>
</tr>
<tr>
<td>Jb</td>
</tr>
</tbody>
</table>