Parameterized Object Sensitivity for Points-to Analysis for Java

Authors: Ana Milanova, Atanas Rountev, Barbara G. Ryder
Presenter: Zheng Song
Outlines

• Introduction
• Existing Method and its limitation
• Object Sensitive analysis
• Parameterized Object Sensitivity
• Implementation
• Evaluation
• Questions
Outlines

• Introduction
• Existing Method and its limitation
• Object Sensitive analysis
• Parameterized Object Sensitivity
• Implementation
• Evaluation
• Questions
Introduction

• One sentence to conclude this paper: analyze a method separately for each of the objects on which this method is invoked

• For: Points-to Analysis: Method in Java to determine the set of objects pointed to by a reference variable or a reference object field
Sample points-to graph

class Y {...}
class X {
    Y f;
    void set(Y r) {
        this.f = r;
    }
    static void main() {
        s1: X p = new X();
        s2: Y q = new Y();
        p.set(q);
    }
}
Outlines

• Introduction
• Existing Method and its limitation
• Object Sensitive analysis
• Parameterized Object Sensitivity
• Implementation
• Evaluation
• Questions
Existing Methods

• Andersen’s algorithm: flow insensitive & context insensitive
• Semantics (why called semantics?)
  – R – set of all reference variables
  – O – set of all objects created at object allocation sites
  – F – contains all instance fields in program class
  – Edge \((r,o) \in R \times O\)
  – \((<o,f>, o) \in (O \times F) \times O\)
  – Transfer functions
Example

All statements are divided into:

- Direct assignment: \( l = r \)
- Instance field write: \( l.f = r \)
- Instance field read: \( l = r.f \)
- Object creation: \( l = \text{new } C \)
- Virtual invocation: \( l = r_0.m(r_1, \ldots, r_k) \)

Go through each statement and conduct the graph following:

\[
\begin{align*}
 f(G, s_i : l = \text{new } C) &= G \cup \{(l, o_i)\} \\
 f(G, l = r) &= G \cup \{(l, o_i) \mid o_i \in \text{Pt}(G, r)\} \\
 f(G, l.f = r) &= \quad G \cup \{(o_i, f), o_j \mid o_i \in \text{Pt}(G, l) \land o_j \in \text{Pt}(G, r)\} \\
 f(G, l = r.f) &= \quad G \cup \{(l, o_i) \mid o_j \in \text{Pt}(G, r) \land o_i \in \text{Pt}(G, (o_j, f))\} \\
 f(G, l = r_0.m(r_1, \ldots, r_n)) &= \quad G \cup \{\text{resolve}(G, m, o_i, r_1, \ldots, r_n, l) \mid o_i \in \text{Pt}(G, r_0)\} \\
 \text{resolve}(G, m, o_i, r_1, \ldots, r_n, l) &= \quad \text{let } m_j(p_0, p_1, \ldots, p_n, \text{ret}_j) = \text{dispatch}(o_i, m) \text{ in} \\
 &\{(p_0, o_i)\} \cup f(G, p_1 = r_1) \cup \ldots \cup f(G, l = \text{ret}_j) \\
\end{align*}
\]

Problem:
1. what’s the difference between OOPSLA’01?
Existing Methods

- **Flow insensitive V.S. Flow sensitive:**

```
int x;
int *y, *z;
x = &y;
x = &z;
```

**Flow-sensitive analysis**
- Computes one answer for every program point
- Requires iterative data-flow analysis or similar technique

**Flow-insensitive analysis**
- Ignores control flow
- Computes one answer for every procedure
- Can compute in linear time
- Less accurate than flow-sensitive

```
void f(int x)
{
    x = 4;
    ...
    x = 5;
}
```

**Flow-sensitive analysis**
- Computes an answer at every program point:
  - \( x \) is 4 after the first assignment
  - \( x \) is 5 after the second assignment

**Flow-insensitive analysis**
- Computes one answer for the entire procedure:
  - \( x \) is not constant
Existing Methods

• Context insensitive V.S. Context sensitive:

**Context Sensitivity Example**

**Is x constant?**

```
a = id(4);  b = id(5);
4          5
id(x) { return x; }
```

**Context-sensitive analysis**

- Computes an answer for every callsite:
  - `x` is 4 in the first call
  - `x` is 5 in the second call

**Context-insensitive analysis**

- Computes one answer for all callsites:
  - `x` is not constant

- Suffers from unrealizable paths:
  - Can mistakenly conclude that `id(4)` can return 5 because we merge (smear) information from all callsites
Its limitation in Object Oriented Programming

- Encapsulation
- Inheritance
- Collection (Containers)...

Let's try to analyze these features using flow insensitive and context insensitive analysis
class X {...}
class Y {
    X f;
    1    void set(X x) { this.f = x; } }
2     s_1: X x_1 = new X();
3     s_2: X x_2 = new X();
4     s_3: Y y_1 = new Y();
5     s_4: Y y_2 = new Y();
6     y_1.set(x_1);
7     y_2.set(x_2);
Inheritance

class X { void n() {...} }
class Y extends X { void n() {...} }
class Z extends X { void n() {...} }

class A {
    X f;
    1 A(X xa) { this.f = xa; } }

class B extends A {
    2 B(X xb) { super(xb); ... }
    void m() {
        X xb = this.f;
        4 xb.n(); } }

class C extends A {
    5 C(X xc) { super(xc); ... }
    void m() {
        X xc = this.f;
        7 xc.n(); } }

8 s_1: Y y = new Y();
9 s_2: Z z = new Z();
10 s_3: B b = new B(y);
11 s_4: C c = new C(z);
12 b.m();
13 c.m();
class Container {
    Object[] data;
    Container(int size) {
        s1: Object[] data_tmp = new Object[size];
        this.data = data_tmp;
    }
    void put(Object e, int at) {
        Object[] data_tmp = this.data;
        data_tmp[at] = e;
    }
    Object get(int at) {
        Object[] data_tmp = this.data;
        return data_tmp[at];
    }
    s2: Container c1 = new Container(100);
    s3: Container c2 = new Container(200);
    s4: X x = new X();
    c1.put(x, 0);
    s5: X y = new Y();
    c2.put(y, 1);
Imprecision

• Encapsulation
• Inheritance
• Container
  – are all strong concepts of OOP
  – But not captured properly with old techniques
  – Solution is Object sensitivity
Outlines

• Introduction
• Existing Method and its limitation
• Object Sensitive analysis
• Parameterized Object Sensitivity
• Implementation
• Evaluation
• Questions
Object Sensitivity

• With object sensitivity, each instance method and each constructor is analyzed separately for each object on which this method/constructor may be invoked.

• How? Revised semantics
  – $O`$ - set of all object names
  – $R`$ - set of replicas of reference variable
  – Relation $\alpha(C,m) => \alpha(o,m)$: $C$, or $D$ which is a superclass of $C$
  – Set of new transfer functions
Object Sensitivity

Object Names

\( O_{ij...pq} \): the sequence of allocation sites \((si, sj, \ldots, sp, sq)\).

A particular name \( oij...pq \) represents all run-time objects that were created by \( sq \) when the enclosing instance method or constructor was invoked on an object represented by name \( oij...p \) which was created at allocation site \( sp \) (recursive).

\( S1: \text{object } O1 \)

\( S2: \text{object } O2 \)

\( S3: \text{object } O3 \)

\( O_1 \Rightarrow O_{21} \& O_{31} \)

class Container {
    Object[] data;
    Container(int size) {
        s1: Object[] data_tmp = new Object[size];
        this.data = data_tmp;
    }
    void put(Object e, int at) {
        Object[] data_tmp = this.data;
        data_tmp[at] = e;
    }
    Object get(int at) {
        Object[] data_tmp = this.data;
        return data_tmp[at];
    }
}

s2: Container c1 = new Container(100);
s3: Container c2 = new Container(200);
s4: X x = new X();
c1.put(x,0);
s5: X y = new Y();
c2.put(y,1);
Object Sensitivity

Context Sensitivity

With more objects, next step we make more references pointing to these objects:

\[
C = O' \cup \{\epsilon\} \\
R \times C \rightarrow R'.
\]

[\epsilon is to deal with static calls]

If \( r \) is a local variable or a formal parameter of an instance method or a constructor \( m \), the pair \((r, o)\) is mapped to a “fresh” variable \( r_o \) for every context \( o \in O' \) for which \( \alpha(o, m) \) holds.

Fig. 4

\( \alpha(o_3, A.A) \) \( \alpha(o_4, A.A) \), two copies of A.this corresponding to contexts \( o_3 \) and \( o_4 \)

Fig. 5

\( \alpha(o_2, \text{Container.put}) \) and \( \alpha(o_3, \text{Container.put}) \), therefore there are context copies of put.this, put.data tmp, and put.e corresponding to contexts \( o_2 \) and \( o_3 \)

Q: how to formalize each element in \( R' \) ?
Breaking News...

- Introduction
- Existing Method and its limitation
- Object Sensitive analysis
- Parameterized Object Sensitivity
- Implementation
- Evaluation
- Questions
Parameterized Object Sensitivity

The formal definition of $O'$ is as follows:

- $o_q \in O'$ for each $s_q \in S$ located in a static method
- if $o_{ij \ldots p} \in O'$ and $s_q \in S$ is located in an instance method or a constructor $m$ such that $\omega(o_{ij \ldots p}, m)$, then

1. if $|ij \ldots p| < k$, then $o_{ij \ldots pq} \in O'$
2. if $|ij \ldots p| = k$, then $o_{j \ldots pq} \in O'$

Notes:
1. The parameterized framework only apply to the set of objects ($O'$), by affecting ($O'$), it further affects $R'$ and the transfer functions;
2. If $k=1$, it is actually Andersen’s algorithm
3. $K$ can be different to different statements

Q: why we need $k$?
Object Sensitivity

• **Transfer Functions**

\[
F(G, s_q: l = \text{new } C) = G \cup \bigcup_{c \in C_m} \{(l^c, c \oplus_k s_q)\}
\]

\[
F(G, l = r) = G \cup \bigcup_{c \in C_m} f(G, l^c = r^c)
\]

\[
F(G, l . f = r) = G \cup \bigcup_{c \in C_m} f(G, l^c . f = r^c)
\]

\[
F(G, l = r . f) = G \cup \bigcup_{c \in C_m} f(G, l^c = r^c . f)
\]

\[
F(G, l = r_0 . m(r_1, \ldots, r_n)) = G \cup \bigcup_{c \in C_m} \{\text{resolve}(G, m, o, r_1^c, \ldots, r_n^c, l^c) \mid o \in Pt(G, r_0^c)\}
\]

\[
\text{resolve}(G, m, o, r_1^c, \ldots, r_n^c, l^c) =
\]

\[
\text{let } c' = o
\]

\[
m_j(p_0, p_1, \ldots, p_n, \text{ret}_j) = \text{dispatch}(o, m) \text{ in }
\]

\[
\{(p_0^{c'}, o)\} \cup f(G, p_1^{c'} = r_1^c) \cup \ldots \cup f(G, l^c = \text{ret}^{c'}_j)
\]
Context sensitivity included

- $B.\text{this}^o ^3, B.\text{xb}^o ^3,$
  $A.\text{xa}^o ^3$

- $C.\text{this}^o ^4, C.\text{xc}^o ^4,$
  $A.\text{xa}^o ^4$
Example 3.1.4

3.1.4 Example. Consider the set of statements in Figure 4. Since \( \alpha(B, B.B) \) and \( \alpha(B, A.A) \), we have

\[
\{B.b_3^{o_3}, B.xb^{o_3}, A.this^{o_3}, A.xa^{o_3}\} \subseteq R'
\]

Similarly, we have

\[
\{C.this^{o_4}, C.xc^{o_4}, A.this^{o_4}, A.xa^{o_4}\} \subseteq R'
\]

At line 2, \( B.this^{o_3} \) points to \( o_3 \) and \( B.xb^{o_3} \) points to \( o_1 \). When the analysis processes the call to \( A.A \) at line 2, \( A.this \) and \( A.xa \) are mapped to the context copies corresponding to \( o_3 \), and points-to edges \( A.this^{o_3}, o_3 \) and \( (A.xa^{o_3}, o_1) \) are added to the graph. Similarly, because of line 3, \( A.this^{o_4} \) points to \( o_4 \) and \( A.xa^{o_4} \) points to \( o_2 \). Statement \( this.f=xa \) at line 1 occurs in the context of \( o_3 \) and \( o_4 \). Thus,

\[
A.this^{o_3} = A.xa^{o_3} \quad A.this^{o_4} = A.xa^{o_4}
\]

which produces edges \((o_3, f), o_1\) and \((o_4, f), o_2\). Since \( \alpha(B, m.m) \) and \( \alpha(C, m.m) \), we have

\[
\{B.m.this^{o_3}, B.m.xb^{o_3}, C.m.this^{o_4}, C.m.xc^{o_4}\} \subseteq R'
\]

When the analysis processes the statement at line 3, \( B.m.xb \) and \( B.m.this \) will be mapped to the context copies corresponding to \( o_3 \). Since \( B.m.this^{o_3} \) points to \( o_3 \) and \( (o_3, f) \) points only to \( o_1 \), the statement at line 3 produces edge \((B.m.xb, o_1)\). Similarly, the statement at line 6 produces edge \((C.m.xc, o_2)\).
Advantages

• Models OOP features
• Distinguishes between different receiver objects
• Static methods and variables can be handled with insensitivity
• Can be parameterized
Outlines

• Introduction
• Existing Method and its limitation
• Object Sensitive analysis
• Parameterized Object Sensitivity
• Implementation
• Evaluation
• Questions
Side-effect Analysis (MOD)

input  Stmt: set of statements  map: \( R \times C \rightarrow R' \)
Methods: set of methods  Pt: \( R' \rightarrow \mathcal{P}(O') \)
output  Mod: Stmt \times C \rightarrow \mathcal{P}(O')
initialized to \( \emptyset \) for all \((s, c) \in Stmt \times C\)
declare MMod: Methods \times C \rightarrow \mathcal{P}(O')
initialized to \( \emptyset \) for all \((m, c) \in Methods \times C\)

1. foreach instance field write \( s: p.f = q \in Stmt \) do
2. foreach context \( c \) in which \( s \) appears do
3. \( \text{Mod}(s, c) := \text{Pt}(\text{map}(p, c)) \)
4. add Mod\((s, c)\) to MMod\((\text{EnclMethod}(s), c)\)
5. while changes occur in Mod or MMod do
6. foreach virtual call \( s: l = r.m(...) \in Stmt \) do
7. foreach context \( c \) in which \( s \) appears do
8. foreach object \( o \in \text{Pt}(\text{map}(r, c)) \) do
9. add MMod\((\text{target}(o, m), o)\) to Mod\((s, c)\)
10. add Mod\((s, c)\) to MMod\((\text{EnclMethod}(s), c)\)
11. foreach static call \( s: l = C.m(...) \in Stmt \) do
12. foreach context \( c \) in which \( s \) appears do
13. \( \text{Mod}(s, c) := \text{Mod}(s, c) \cup \text{MMod}(m, c) \)
14. add Mod\((s, c)\) to MMod\((\text{EnclMethod}(s), c)\)

Typo: should be \( c \)
Outlines

• Introduction
• Existing Method and its limitation
• Object Sensitive analysis
• Parameterized Object Sensitivity
• Implementation
• Evaluation
• Questions
Implementations

• Parameterized object-sensitive points-to analysis (context depth = 1):
  – *ObjSens1*: keeps context-sensitive information for implicit parameters `this` and formal parameters of instance methods and constructors.
  – *ObjSens2*: the same as *ObjSens1*, but it also keeps track of return variables.
Implementations

• Context-sensitive analysis based on the call string approach to context sensitivity, for a call string $k = 1 \, (CallSite)$.

• Distinguishes context per call site.

• To allow for comparison, the context replication is performed for $\textit{this}$, formal parameters and return variables in instance methods and constructors.
### Characteristics of Programs

Table I. Characteristics of the Data Programs. First Two Columns Show the Number and Bytecode Size of User Classes. Last Three Columns Include Library Classes

<table>
<thead>
<tr>
<th>Program</th>
<th>User Class</th>
<th>Size (Kb)</th>
<th>Whole-program</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Class</td>
</tr>
<tr>
<td>proxy</td>
<td>18</td>
<td>56.6</td>
<td>565</td>
</tr>
<tr>
<td>compress</td>
<td>22</td>
<td>76.7</td>
<td>568</td>
</tr>
<tr>
<td>db</td>
<td>14</td>
<td>70.7</td>
<td>565</td>
</tr>
<tr>
<td>jb-6.1</td>
<td>21</td>
<td>55.6</td>
<td>574</td>
</tr>
<tr>
<td>echo</td>
<td>17</td>
<td>66.7</td>
<td>577</td>
</tr>
<tr>
<td>raytrace</td>
<td>35</td>
<td>115.9</td>
<td>582</td>
</tr>
<tr>
<td>mtrt</td>
<td>35</td>
<td>115.9</td>
<td>582</td>
</tr>
<tr>
<td>jtar-1.21</td>
<td>64</td>
<td>185.2</td>
<td>618</td>
</tr>
<tr>
<td>jlex-1.2.5</td>
<td>25</td>
<td>95.1</td>
<td>578</td>
</tr>
<tr>
<td>javacup-0.10</td>
<td>33</td>
<td>127.3</td>
<td>581</td>
</tr>
<tr>
<td>rabbit-2</td>
<td>52</td>
<td>157.4</td>
<td>615</td>
</tr>
<tr>
<td>jack</td>
<td>67</td>
<td>191.5</td>
<td>613</td>
</tr>
<tr>
<td>jflex-1.2.2</td>
<td>54</td>
<td>198.2</td>
<td>608</td>
</tr>
<tr>
<td>jess</td>
<td>160</td>
<td>454.2</td>
<td>715</td>
</tr>
<tr>
<td>mpegaudio</td>
<td>62</td>
<td>176.8</td>
<td>608</td>
</tr>
<tr>
<td>jtree-1.0</td>
<td>72</td>
<td>272.0</td>
<td>620</td>
</tr>
<tr>
<td>sablecc-2.9</td>
<td>312</td>
<td>532.4</td>
<td>864</td>
</tr>
<tr>
<td>javac</td>
<td>182</td>
<td>614.7</td>
<td>730</td>
</tr>
<tr>
<td>creature</td>
<td>65</td>
<td>259.7</td>
<td>626</td>
</tr>
<tr>
<td>mindterm1.1.5</td>
<td>120</td>
<td>461.1</td>
<td>686</td>
</tr>
<tr>
<td>soot-1.beta.4</td>
<td>677</td>
<td>1070.4</td>
<td>1214</td>
</tr>
<tr>
<td>muffin-0.9.2</td>
<td>245</td>
<td>655.2</td>
<td>824</td>
</tr>
<tr>
<td>javacc-1.0</td>
<td>63</td>
<td>502.6</td>
<td>615</td>
</tr>
</tbody>
</table>
## Analysis Cost

### Table II. Running Time and Memory Usage of the Analyses

<table>
<thead>
<tr>
<th>Program</th>
<th>And</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>proxy</td>
<td>4.4</td>
<td>40</td>
<td>6.9</td>
<td>39</td>
<td>5.9</td>
<td>40</td>
<td>6.1</td>
</tr>
<tr>
<td>compress</td>
<td>12.0</td>
<td>46</td>
<td>12.0</td>
<td>47</td>
<td>8.8</td>
<td>46</td>
<td>13.2</td>
</tr>
<tr>
<td>db</td>
<td>12.2</td>
<td>47</td>
<td>11.4</td>
<td>47</td>
<td>11.7</td>
<td>48</td>
<td>12.2</td>
</tr>
<tr>
<td>jb</td>
<td>7.5</td>
<td>43</td>
<td>7.3</td>
<td>43</td>
<td>7.1</td>
<td>43</td>
<td>7.0</td>
</tr>
<tr>
<td>echo</td>
<td>27.3</td>
<td>60</td>
<td>24.8</td>
<td>59</td>
<td>24.3</td>
<td>59</td>
<td>27.2</td>
</tr>
<tr>
<td>raytrace</td>
<td>13.9</td>
<td>60</td>
<td>12.6</td>
<td>51</td>
<td>13.6</td>
<td>60</td>
<td>13.8</td>
</tr>
<tr>
<td>mtrt</td>
<td>15.4</td>
<td>50</td>
<td>12.9</td>
<td>51</td>
<td>15.2</td>
<td>50</td>
<td>15.6</td>
</tr>
<tr>
<td>jtar</td>
<td>23.3</td>
<td>58</td>
<td>19.9</td>
<td>56</td>
<td>21.4</td>
<td>56</td>
<td>20.6</td>
</tr>
<tr>
<td>jlex</td>
<td>8.5</td>
<td>46</td>
<td>9.0</td>
<td>46</td>
<td>8.8</td>
<td>46</td>
<td>8.7</td>
</tr>
<tr>
<td>javacup</td>
<td>13.1</td>
<td>57</td>
<td>17.3</td>
<td>56</td>
<td>16.8</td>
<td>53</td>
<td>15.1</td>
</tr>
<tr>
<td>rabbit</td>
<td>16.1</td>
<td>52</td>
<td>14.5</td>
<td>52</td>
<td>13.9</td>
<td>52</td>
<td>13.9</td>
</tr>
<tr>
<td>jack</td>
<td>38.4</td>
<td>62</td>
<td>38.1</td>
<td>62</td>
<td>37.5</td>
<td>62</td>
<td>37.6</td>
</tr>
<tr>
<td>jflex</td>
<td>20.2</td>
<td>71</td>
<td>20.1</td>
<td>70</td>
<td>19.7</td>
<td>70</td>
<td>19.7</td>
</tr>
<tr>
<td>jess</td>
<td>24.6</td>
<td>67</td>
<td>26.1</td>
<td>67</td>
<td>24.3</td>
<td>66</td>
<td>26.9</td>
</tr>
<tr>
<td>mpegaudio</td>
<td>15.4</td>
<td>52</td>
<td>14.1</td>
<td>54</td>
<td>16.2</td>
<td>52</td>
<td>13.2</td>
</tr>
<tr>
<td>jtree</td>
<td>12.4</td>
<td>53</td>
<td>11.8</td>
<td>53</td>
<td>11.2</td>
<td>53</td>
<td>8.7</td>
</tr>
<tr>
<td>sablecc</td>
<td>68.7</td>
<td>115</td>
<td>40.1</td>
<td>94</td>
<td>62.3</td>
<td>113</td>
<td>35.9</td>
</tr>
<tr>
<td>javac</td>
<td>427.6</td>
<td>121</td>
<td>430.7</td>
<td>123</td>
<td>430.2</td>
<td>120</td>
<td>416.9</td>
</tr>
<tr>
<td>creature</td>
<td>85.2</td>
<td>100</td>
<td>61.1</td>
<td>86</td>
<td>55.7</td>
<td>88</td>
<td>57.6</td>
</tr>
<tr>
<td>mindterm</td>
<td>44.5</td>
<td>91</td>
<td>44.9</td>
<td>89</td>
<td>49.9</td>
<td>88</td>
<td>42.6</td>
</tr>
<tr>
<td>soot</td>
<td>80.8</td>
<td>130</td>
<td>92.6</td>
<td>132</td>
<td>69.8</td>
<td>128</td>
<td>89.7</td>
</tr>
<tr>
<td>muffin</td>
<td>110.0</td>
<td>144</td>
<td>108.4</td>
<td>135</td>
<td>99.6</td>
<td>132</td>
<td>101.1</td>
</tr>
<tr>
<td>javacc</td>
<td>76.2</td>
<td>112</td>
<td>82.3</td>
<td>112</td>
<td>81.4</td>
<td>116</td>
<td>80.1</td>
</tr>
</tbody>
</table>
MOD Analysis Precision

Table III. Number of Modified Objects for Program Statements. Each Column Shows the Percentage of Statements Whose Number of Modified Objects is in the Corresponding Range

<table>
<thead>
<tr>
<th>Program</th>
<th>And</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1-3</td>
<td>4-9</td>
<td>≥10</td>
<td>1-3</td>
<td>4-9</td>
<td>≥10</td>
</tr>
<tr>
<td>proxy</td>
<td>19%</td>
<td>6%</td>
<td>75%</td>
<td>25%</td>
<td>7%</td>
<td>68%</td>
</tr>
<tr>
<td>compress</td>
<td>23%</td>
<td>4%</td>
<td>73%</td>
<td>27%</td>
<td>5%</td>
<td>67%</td>
</tr>
<tr>
<td>db</td>
<td>20%</td>
<td>4%</td>
<td>76%</td>
<td>24%</td>
<td>4%</td>
<td>72%</td>
</tr>
<tr>
<td>jb</td>
<td>15%</td>
<td>5%</td>
<td>80%</td>
<td>20%</td>
<td>5%</td>
<td>75%</td>
</tr>
<tr>
<td>echo</td>
<td>25%</td>
<td>6%</td>
<td>69%</td>
<td>30%</td>
<td>5%</td>
<td>65%</td>
</tr>
<tr>
<td>raytrace</td>
<td>23%</td>
<td>5%</td>
<td>72%</td>
<td>28%</td>
<td>6%</td>
<td>66%</td>
</tr>
<tr>
<td>mtrt</td>
<td>23%</td>
<td>5%</td>
<td>72%</td>
<td>28%</td>
<td>6%</td>
<td>66%</td>
</tr>
<tr>
<td>jtar</td>
<td>18%</td>
<td>8%</td>
<td>74%</td>
<td>24%</td>
<td>7%</td>
<td>69%</td>
</tr>
<tr>
<td>jflex</td>
<td>17%</td>
<td>4%</td>
<td>79%</td>
<td>20%</td>
<td>4%</td>
<td>76%</td>
</tr>
<tr>
<td>javacup</td>
<td>14%</td>
<td>3%</td>
<td>83%</td>
<td>21%</td>
<td>4%</td>
<td>75%</td>
</tr>
<tr>
<td>rabbit</td>
<td>18%</td>
<td>5%</td>
<td>77%</td>
<td>23%</td>
<td>6%</td>
<td>71%</td>
</tr>
<tr>
<td>jack</td>
<td>17%</td>
<td>3%</td>
<td>80%</td>
<td>20%</td>
<td>3%</td>
<td>77%</td>
</tr>
<tr>
<td>jjflex</td>
<td>19%</td>
<td>4%</td>
<td>77%</td>
<td>23%</td>
<td>5%</td>
<td>72%</td>
</tr>
<tr>
<td>jess</td>
<td>15%</td>
<td>5%</td>
<td>80%</td>
<td>25%</td>
<td>3%</td>
<td>72%</td>
</tr>
<tr>
<td>mpegaudio</td>
<td>23%</td>
<td>4%</td>
<td>73%</td>
<td>28%</td>
<td>4%</td>
<td>68%</td>
</tr>
<tr>
<td>jjtree</td>
<td>8%</td>
<td>2%</td>
<td>90%</td>
<td>10%</td>
<td>2%</td>
<td>88%</td>
</tr>
<tr>
<td>sablecc</td>
<td>20%</td>
<td>3%</td>
<td>77%</td>
<td>32%</td>
<td>4%</td>
<td>64%</td>
</tr>
<tr>
<td>javac</td>
<td>14%</td>
<td>4%</td>
<td>82%</td>
<td>18%</td>
<td>6%</td>
<td>76%</td>
</tr>
<tr>
<td>creature</td>
<td>18%</td>
<td>3%</td>
<td>79%</td>
<td>27%</td>
<td>3%</td>
<td>70%</td>
</tr>
<tr>
<td>mindterm</td>
<td>20%</td>
<td>8%</td>
<td>73%</td>
<td>25%</td>
<td>7%</td>
<td>68%</td>
</tr>
<tr>
<td>soot</td>
<td>16%</td>
<td>4%</td>
<td>80%</td>
<td>25%</td>
<td>8%</td>
<td>67%</td>
</tr>
<tr>
<td>muffin</td>
<td>16%</td>
<td>4%</td>
<td>80%</td>
<td>24%</td>
<td>4%</td>
<td>72%</td>
</tr>
<tr>
<td>javacc</td>
<td>10%</td>
<td>1%</td>
<td>89%</td>
<td>11%</td>
<td>1%</td>
<td>88%</td>
</tr>
<tr>
<td>Average</td>
<td>18%</td>
<td>4%</td>
<td>78%</td>
<td>23%</td>
<td>5%</td>
<td>72%</td>
</tr>
</tbody>
</table>

|           |          |          |          |          |          |          |
|           | 1-3      | 4-9      | ≥10      | 1-3      | 4-9      | ≥10      |
| Average   | 54%      | 18%      | 28%      | 54%      | 18%      | 28%      |
Conclusions

• Presented a framework for parameterized object-sensitive points-to analysis, and side-effect and def-use analyses based on it.

• Object-sensitive analysis achieves significantly better precision than context-insensitive analysis, while remaining efficient and practical.
Acknowledgement

• Besides the original paper and its journal version, some materials are derived from
  – UPENN CIS570 Lecture Notes
  – UMD CMSC737(Fundamentals of Software Testing), Student Presentation by Anand Bahety & Dan Bucatanschi