Object-sensitive points-to analysis for Java

K. V. Raghavan

Indian Institute of Science, Bangalore

Contributions of the paper [Milanova and Ryder, ISSTA '02 & TOSEM '05]

- A new kind of context sensitive points-to analysis: Object sensitivity.
- Object-sensitive inter-procedural analysis, as well as object-sensitive object names.
- Two sample object-sensitive client analyses – side-effect analysis, reaching definitions analysis.
- Implementation, and empirical evidence.
- [Lhotak and Hendren, CC 2006]: Object-sensitivity is both more scalable and more precise than other techniques such as call-strings and BDD based approaches.
- Has had high impact: For instance, available in both Wala and Soot.

Baseline analysis: Flow- and context- insensitive Andersen's analysis

- Set R contains all reference variables (including static variables).
  - Variables across all methods are assumed to be unambiguously named. `this` is also a variable.
- Set O contains symbolic object names. For each allocation site s_i, there is a separate object name $o_i \in O$.
- Set F contains all instance fields across all classes.
- The analysis constructs points-to graphs containing two kinds of edges.
  - Edge $(r, o_i) \in R \times O$ indicates that reference variable r points to object o_i.
  - Edge $(o_i, f, o_j) \in (O \times F) \times O$ indicates that field f of object o_i points to object o_j.
- Fig. 1 in the paper shows a sample program, and its (final) points-to graph.

Transfer functions of baseline analysis

$$f(G, s_i\colon l = new\ C) = G \cup \{(l, o_i)\}$$

$$f(G, l = r) = G \cup \{(l, o_i) \mid o_i \in Pt(G, r)\}$$

$$f(G, l.f = r) = G \cup \{(\langle o_i, f\rangle, o_j) \mid o_i \in Pt(G, l) \wedge o_j \in Pt(G, r)\}$$

$$f(G, l = r.f) = G \cup \{(l, o_i) \mid o_j \in Pt(G, r) \wedge o_i \in Pt(G, \langle o_j, f\rangle)\}$$

$$f(G, l = r_0.m(r_1, \ldots, r_n)) = G \cup \{resolve(G, m, o_i, r_1, \ldots, r_n, l) \mid o_i \in Pt(G, r_0)\}$$

$$\begin{array}{l} resolve(G, m, o_i, r_1, \ldots, r_n, l) = \\ \quad \texttt{let}\ \ m_j(p_0, p_1, \ldots, p_n, ret_j) = dispatch(o_i, m)\ \ \texttt{in} \\ \qquad \{(p_0, o_i)\} \cup f(G, p_1 = r_1) \cup \ldots \cup f(G, l = ret_j) \end{array}$$

Fig. 2. Points-to effects of program statements for Andersen's analysis.

Transfer functions of baseline analysis

$$f(G, s_i\colon l = new\ C) = G \cup \{(l, o_i)\}$$

$$f(G, l = r) = G \cup \{(l, o_i) \mid o_i \in Pt(G, r)\}$$

$$f(G, l.f = r) = G \cup \{(\langle o_i, f\rangle, o_j) \mid o_i \in Pt(G, l) \wedge o_j \in Pt(G, r)\}$$

$$f(G, l = r.f) = G \cup \{(l, o_i) \mid o_j \in Pt(G, r) \wedge o_i \in Pt(G, \langle o_j, f\rangle)\}$$

$$f(G, l = r_0.m(r_1, \ldots, r_n)) = G \cup \{resolve(G, m, o_i, r_1, \ldots, r_n, l) \mid o_i \in Pt(G, r_0)\}$$

$$resolve(G, m, o_i, r_1, \ldots, r_n, l) =$$
$$\texttt{let}\ \ m_j(p_0, p_1, \ldots, p_n, ret_j) = dispatch(o_i, m)\ \ \texttt{in}$$
$$\{(p_0, o_i)\} \cup f(G, p_1 = r_1) \cup \ldots \cup f(G, l = ret_j)$$

Fig. 2. Points-to effects of program statements for Andersen's analysis.

Notes:

- G is the points-to graph. $Pt(G,r)$ retrieves the points-to set of a reference r in G, while $Pt(G, < o_i, f >)$ retrieves the points-to set of edge f from object o_i.
- f is the transfer function, defined for each kind of statement.
- Statements that don't deal with reference variables can be ignored.

Transfer functions of baseline analysis

$$f(G, s_i\colon l = new\ C) = G \cup \{(l, o_i)\}$$

$$f(G, l = r) = G \cup \{(l, o_i) \mid o_i \in Pt(G, r)\}$$

$$f(G, l.f = r) = G \cup \{(\langle o_i, f\rangle, o_j) \mid o_i \in Pt(G, l) \wedge o_j \in Pt(G, r)\}$$

$$f(G, l = r.f) = G \cup \{(l, o_i) \mid o_j \in Pt(G, r) \wedge o_i \in Pt(G, \langle o_j, f\rangle)\}$$

$$f(G, l = r_0.m(r_1, \ldots, r_n)) = G \cup \{resolve(G, m, o_i, r_1, \ldots, r_n, l) \mid o_i \in Pt(G, r_0)\}$$

$$\begin{array}{l} resolve(G, m, o_i, r_1, \ldots, r_n, l) = \\ \quad \texttt{let}\ \ m_j(p_0, p_1, \ldots, p_n, ret_j) = dispatch(o_i, m)\ \ \texttt{in} \\ \qquad \{(p_0, o_i)\} \cup f(G, p_1 = r_1) \cup \ldots \cup f(G, l = ret_j) \end{array}$$

Fig. 2. Points-to effects of program statements for Andersen's analysis.

Notes:

- $dispatch(o_i, m)$ is a utility routine, where m is a call-site. Say C is the allocated type of o_i. This routine walks up the inheritance hierarchy starting at C and returns the closest declared method m_j whose signature matches m. The p_i's are the formal parameters of m_j, with $p_0 \equiv$ `this`. ret_j is the local variable of m_j whose value is returned by m_j.

Transfer functions of baseline analysis

$$f(G, s_i : l = new\ C) = G \cup \{(l, o_i)\}$$

$$f(G, l = r) = G \cup \{(l, o_i) \mid o_i \in Pt(G, r)\}$$

$$f(G, l.f = r) = G \cup \{(\langle o_i, f\rangle, o_j) \mid o_i \in Pt(G, l) \wedge o_j \in Pt(G, r)\}$$

$$f(G, l = r.f) = G \cup \{(l, o_i) \mid o_j \in Pt(G, r) \wedge o_i \in Pt(G, \langle o_j, f\rangle)\}$$

$$f(G, l = r_0.m(r_1, \ldots, r_n)) = G \cup \{resolve(G, m, o_i, r_1, \ldots, r_n, l) \mid o_i \in Pt(G, r_0)\}$$

$$resolve(G, m, o_i, r_1, \ldots, r_n, l) =$$
$$\texttt{let}\ \ m_j(p_0, p_1, \ldots, p_n, ret_j) = dispatch(o_i, m)\ \ \texttt{in}$$
$$\{(p_0, o_i)\} \cup f(G, p_1 = r_1) \cup \ldots \cup f(G, l = ret_j)$$

Fig. 2. Points-to effects of program statements for Andersen's analysis.

Notes:

- Note that a call is modeled as a set of assignments that (a) copy the actual parameters to the formal parameters, and (b) copy the return variable to the return-value variable in the caller.

Example 1

```
class Y {
  X f;
  void set(X x) {this.f = x;}
}

class Z {
  X newX() {s1: return new X();}
  Y newY() {s2: return new Y();}
}
```

```
main() {
  s3: Z z1 = new Z();
      Y y1 = z1.newY();
      X x1 = z1.newX();
      y1.set(x1);

  s4: Z z2 = new Z();
      Y y2 = z2.newY();
      X x2 = z2.newX();
      y2.set(x2);
}
```

Example 1

```
class Y {
  X f;
  void set(X x) {this.f = x;}
}

class Z {
  X newX() {s1: return new X();}
  Y newY() {s2: return new Y();}
}
```

```
main() {
  s3: Z z1 = new Z();
      Y y1 = z1.newY();
      X x1 = z1.newX();
      y1.set(x1);

  s4: Z z2 = new Z();
      Y y2 = z2.newY();
      X x2 = z2.newX();
      y2.set(x2);
}
```

Ideal result: Two instances of `Y`, each pointing to a different instance of `X`.

Example 1

```
class Y {
  X f;
  void set(X x) {this.f = x;}
}

class Z {
  X newX() {s1: return new X();}
  Y newY() {s2: return new Y();}
}
```

```
main() {
  s3: Z z1 = new Z();
      Y y1 = z1.newY();
      X x1 = z1.newX();
      y1.set(x1);

  s4: Z z2 = new Z();
      Y y2 = z2.newY();
      X x2 = z2.newX();
      y2.set(x2);
}
```

With no sensitivity (i.e., baseline analysis): $\texttt{z1} \rightarrow o_3$, $\texttt{y1} \rightarrow o_2$, $\texttt{x1} \rightarrow o_1$. $\texttt{z2} \rightarrow o_4$, $\texttt{y2} \rightarrow o_2$, $\texttt{x2} \rightarrow o_1$.

Method `set`: $\texttt{set\_this} \rightarrow o_2$, $\texttt{x} \rightarrow o_1$.

Method `newX`: $\texttt{newX\_this} \rightarrow \{o_3, o_4\}$, $\texttt{newX\_}ret \rightarrow o_1$.

Method `newY`: $\texttt{newY\_this} \rightarrow \{o_3, o_4\}$, $\texttt{newY\_}ret \rightarrow o_2$.

$o_2.f \rightarrow o_1$.

Example 1

```
class Y {
  X f;
  void set(X x) {this.f = x;}
}

class Z {
  X newX() {s1: return new X();}
  Y newY() {s2: return new Y();}
}
```

```
main() {
  s3: Z z1 = new Z();
      Y y1 = z1.newY();
      X x1 = z1.newX();
      y1.set(x1);

  s4: Z z2 = new Z();
      Y y2 = z2.newY();
      X x2 = z2.newX();
      y2.set(x2);
}
```

Only context sensitivity: $\texttt{z1} \rightarrow o_3$, $\texttt{y1} \rightarrow o_2$, $\texttt{x1} \rightarrow o_1$, $\texttt{z2} \rightarrow o_4$, $\texttt{y2} \rightarrow o_2$, $\texttt{x2} \rightarrow o_1$.

First call to set: $\texttt{set\_this} \rightarrow o_2$, $\texttt{x} \rightarrow o_1$.

Second call to set: $\texttt{set\_this} \rightarrow o_2$, $\texttt{x} \rightarrow o_1$.

First call to newX: $\texttt{newX\_this} \rightarrow o_3$, $\texttt{newX\_}\mathit{ret} \rightarrow o_1$.

First call to newY: $\texttt{newY\_this} \rightarrow o_3$, $\texttt{newY\_}\mathit{ret} \rightarrow o_2$.

Second call to newX: $\texttt{newX\_this} \rightarrow o_4$, $\texttt{newX\_}\mathit{ret} \rightarrow o_1$.

Second call to newY: $\texttt{newY\_this} \rightarrow o_4$, $\texttt{newY\_}\mathit{ret} \rightarrow o_2$.

$o_2.f \rightarrow o_1$.

Example 1

```
class Y {
  X f;
  void set(X x) {this.f = x;}
}

class Z {
  X newX() {s1: return new X();}
  Y newY() {s2: return new Y();}
}
```

```
main() {
  s3: Z z1 = new Z();
      Y y1 = z1.newY();
      X x1 = z1.newX();
      y1.set(x1);

  s4: Z z2 = new Z();
      Y y2 = z2.newY();
      X x2 = z2.newX();
      y2.set(x2);
}
```

Only name sensitivity: $\texttt{z1} \rightarrow o_3$, $\texttt{y1} \rightarrow o_{32}$, $\texttt{x1} \rightarrow o_{31}$, $\texttt{z2} \rightarrow o_4$, $\texttt{y2} \rightarrow o_{42}$, $\texttt{x2} \rightarrow o_{41}$.

Method set: $\texttt{set\_this} \rightarrow \{o_{32}, o_{42}\}$, $\texttt{x} \rightarrow \{o_{31}, o_{41}\}$.

Method newX: $\texttt{newX\_this} \rightarrow \{o_3, o_4\}$, $\texttt{newX\_}\mathit{ret} \rightarrow \{o_{31}, o_{41}\}$.

Method newY: $\texttt{newY\_this} \rightarrow \{o_3, o_4\}$, $\texttt{newY\_}\mathit{ret} \rightarrow \{o_{32}, o_{42}\}$.

$o_{32}.f \rightarrow o_{31}$, $o_{32}.f \rightarrow o_{41}$, $o_{42}.f \rightarrow o_{31}$, $o_{42}.f \rightarrow o_{41}$.

Example 1

```
class Y {
  X f;
  void set(X x) {this.f = x;}
}

class Z {
  X newX() {s1: return new X();}
  Y newY() {s2: return new Y();}
}
```

```
main() {
  s3: Z z1 = new Z();
      Y y1 = z1.newY();
      X x1 = z1.newX();
      y1.set(x1);

  s4: Z z2 = new Z();
      Y y2 = z2.newY();
      X x2 = z2.newX();
      y2.set(x2);
}
```

Both context and name sensitivity: $\texttt{z1} \rightarrow o_3$, $\texttt{y1} \rightarrow o_{32}$, $\texttt{x1} \rightarrow o_{31}$, $\texttt{z2} \rightarrow o_4$, $\texttt{y2} \rightarrow o_{42}$, $\texttt{x2} \rightarrow o_{41}$.

First call to set: $\texttt{set\_this} \rightarrow o_{32}$, $\texttt{x} \rightarrow o_{31}$.

First call to set: $\texttt{set\_this} \rightarrow o_{42}$, $\texttt{x} \rightarrow o_{41}$.

First call to newX: $\texttt{newX\_this} \rightarrow o_3$, $\texttt{newX\_}ret \rightarrow o_{31}$.

First call to newY: $\texttt{newY\_this} \rightarrow o_3$, $\texttt{newY\_}ret \rightarrow o_{32}$.

Second call to newX: $\texttt{newX\_this} \rightarrow o_4$, $\texttt{newX\_}ret \rightarrow o_{41}$.

Second call to newY: $\texttt{newY\_this} \rightarrow o_4$, $\texttt{newY\_}ret \rightarrow o_{42}$.

$o_{32}.f \rightarrow o_{31}$, $o_{42}.f \rightarrow o_{41}$.

More examples of imprecision of base analysis

- Fig. 3: erroneous edges $o_3.f \rightarrow o_2$ and $o_4.f \rightarrow o_1$.
  - Setter methods are very common in Java, and cause this kind of imprecision.
- Fig. 4:
  - "In the presence of inheritance, instance fields are often located in superclasses and are written through invocations of superclass constructors or methods." This causes loss of context sensitivity.
  - In the example, line 4 is incorrectly resolved to both `Y.n` and `Z.n` (only `Y.n` is correct). Similarly, line 7 resolves erroneously to both targets (only `Z.n` is correct).
- Fig. 5:
  - Single array object o_1 appears to store values from both containers.

Basic concepts

- S is the set of allocation sites.
- O' is the domain of object names: $S \cup S^2 \cup \ldots \cup S^k$, where $k \geq 1$ is a parameter of the analysis.
- Name sensitivity: A particular name $o_{ij\ldots pq}$ represents all run-time objects that were created by site s_q when the enclosing instance method or constructor was invoked on a receiver object represented by name $o_{ij\ldots p}$ which was created at allocation site s_p.
  - In case an object's name needs to exceed k in length, its k-suffix is retained.
- Context sensitivity: For each local reference variable r of an instance method or constructor m, and for each context $c \in O'$ under which method m is called (i.e., for each receiver object c on which m is called), create a new local variable r^c. Let R' represent this replicated set of variables.

Transfer functions

$$F(G, s_q : l = new\ C) = G \cup \bigcup_{c \in \mathcal{C}_m} \{(l^c, c \oplus_k s_q)\}$$

$$F(G, l = r) = G \cup \bigcup_{c \in \mathcal{C}_m} f(G, l^c = r^c)$$

$$F(G, l.f = r) = G \cup \bigcup_{c \in \mathcal{C}_m} f(G, l^c.f = r^c)$$

$$F(G, l = r.f) = G \cup \bigcup_{c \in \mathcal{C}_m} f(G, l^c = r^c.f)$$

$$F(G, l = r_0.m(r_1, \ldots, r_n)) = G \cup \bigcup_{c \in \mathcal{C}_m} \{resolve(G, m, o, r_1^c, \ldots, r_n^c, l^c) \mid o \in Pt(G, r_0^c)\}$$

$$\begin{aligned} &resolve(G, m, o, r_1^c, \ldots, r_n^c, l^c) = \\ &\quad \texttt{let}\ \ c' = o \\ &\qquad m_j(p_0, p_1, \ldots, p_n, ret_j) = dispatch(o, m)\ \ \texttt{in} \\ &\qquad\quad \{(p_0^{c'}, o)\} \cup f(G, p_1^{c'} = r_1^c) \cup \ldots \cup f(G, l^c = ret_j^{c'}) \end{aligned}$$

Fig. 6. Object-sensitive points-to effects of statements in instance methods and constructors. $\mathcal{C}_m$ is the set of possible contexts for the enclosing method.

Transfer functions

$$F(G, s_q : l = new\ C) = G \cup \bigcup_{c \in \mathcal{C}_m} \{(l^c, c \oplus_k s_q)\}$$

$$F(G, l = r) = G \cup \bigcup_{c \in \mathcal{C}_m} f(G, l^c = r^c)$$

$$F(G, l.f = r) = G \cup \bigcup_{c \in \mathcal{C}_m} f(G, l^c.f = r^c)$$

$$F(G, l = r.f) = G \cup \bigcup_{c \in \mathcal{C}_m} f(G, l^c = r^c.f)$$

$$F(G, l = r_0.m(r_1, \ldots, r_n)) = G \cup \bigcup_{c \in \mathcal{C}_m} \{resolve(G, m, o, r_1^c, \ldots, r_n^c, l^c) \mid o \in Pt(G, r_0^c)\}$$

$$resolve(G, m, o, r_1^c, \ldots, r_n^c, l^c) =$$
$$\texttt{let}\ \ c' = o$$
$$m_j(p_0, p_1, \ldots, p_n, ret_j) = dispatch(o, m)\ \ \texttt{in}$$
$$\{(p_0^{c'}, o)\} \cup f(G, p_1^{c'} = r_1^c) \cup \ldots \cup f(G, l^c = ret_j^{c'})$$

Fig. 6. Object-sensitive points-to effects of statements in instance methods and constructors. $\mathcal{C}_m$ is the set of possible contexts for the enclosing method.

Notes:

- F is the transfer function. Note that it makes use of the 'f' transfer function of the baseline analysis.
- $c \oplus_k s_q$ is a string concatenation operation, followed by k-suffixing.

Transfer functions

$$F(G, s_q : l = new\ C) = G \cup \bigcup_{c \in \mathcal{C}_m} \{(l^c, c \oplus_k s_q)\}$$

$$F(G, l = r) = G \cup \bigcup_{c \in \mathcal{C}_m} f(G, l^c = r^c)$$

$$F(G, l.f = r) = G \cup \bigcup_{c \in \mathcal{C}_m} f(G, l^c.f = r^c)$$

$$F(G, l = r.f) = G \cup \bigcup_{c \in \mathcal{C}_m} f(G, l^c = r^c.f)$$

$$F(G, l = r_0.m(r_1, \ldots, r_n)) = G \cup \bigcup_{c \in \mathcal{C}_m} \{resolve(G, m, o, r_1^c, \ldots, r_n^c, l^c) \mid o \in Pt(G, r_0^c)\}$$

$$resolve(G, m, o, r_1^c, \ldots, r_n^c, l^c) =$$
$$\texttt{let}\ \ c' = o$$
$$m_j(p_0, p_1, \ldots, p_n, ret_j) = dispatch(o, m)\ \ \texttt{in}$$
$$\{(p_0^{c'}, o)\} \cup f(G, p_1^{c'} = r_1^c) \cup \ldots \cup f(G, l^c = ret_j^{c'})$$

Fig. 6. Object-sensitive points-to effects of statements in instance methods and constructors. $\mathcal{C}_m$ is the set of possible contexts for the enclosing method.

Notes:

- For now assume that $\mathcal{C}_m$, which is the set of all contexts under which method m is invoked, is somehow given apriori.

Illustration

Example 1:

```
class Y {
  X f;
  void set(X x) {this.f = x;}
}

class Z {
  X newX() {s1: return new X();}
  Y newY() {s2: return new Y();}
}
```

```
main() {
  s3: Z z1 = new Z();
      Y y1 = z1.newY();
      X x1 = z1.newX();
      y1.set(x1);

  s4: Z z2 = new Z();
      Y y2 = z2.newY();
      X x2 = z2.newX();
      y2.set(x2);
}
```

Assume for now that an Oracle tells us that `set` is called from contexts o_{32} and o_{42}, and that `newX` and `newY` each is called from context o_3 and from context o_4. Then, the following instructions are generated by replication:

Illustration

Example 1:

```
class Y {
  X f;
  void set(X x) {this.f = x;}
}

class Z {
  X newX() {s1: return new X();}
  Y newY() {s2: return new Y();}
}
```

```
main() {
  s3: Z z1 = new Z();
      Y y1 = z1.newY();
      X x1 = z1.newX();
      y1.set(x1);

  s4: Z z2 = new Z();
      Y y2 = z2.newY();
      X x2 = z2.newX();
      y2.set(x2);
}
```

Assume for now that an Oracle tells us that `set` is called from contexts o_{32} and o_{42}, and that `newX` and `newY` each is called from context o_3 and from context o_4. Then, the following instructions are generated by replication:

$$\texttt{set\_this}^{32}.f = \texttt{x}^{32}$$
$$\texttt{set\_this}^{42}.f = \texttt{x}^{42}$$

$$\texttt{newX\_ret}^{3} = o_{31}$$
$$\texttt{newX\_ret}^{4} = o_{41}$$

$$\texttt{newY\_ret}^{3} = o_{32}$$
$$\texttt{newY\_ret}^{4} = o_{42}$$

Illustration

Example 1:

```
class Y {
  X f;
  void set(X x) {this.f = x;}
}

class Z {
  X newX() {s1: return new X();}
  Y newY() {s2: return new Y();}
}
```

```
main() {
  s3: Z z1 = new Z();
      Y y1 = z1.newY();
      X x1 = z1.newX();
      y1.set(x1);

  s4: Z z2 = new Z();
      Y y2 = z2.newY();
      X x2 = z2.newX();
      y2.set(x2);
}
```

Assume for now that an Oracle tells us that `set` is called from contexts o_{32} and o_{42}, and that `newX` and `newY` each is called from context o_3 and from context o_4. Then, the following instructions are generated by replication:

$$\texttt{set\_this}^{32}.f = \texttt{x}^{32}$$
$$\texttt{set\_this}^{42}.f = \texttt{x}^{42}$$

$$\texttt{newX\_}ret^{3} = o_{31}$$
$$\texttt{newX\_}ret^{4} = o_{41}$$

$$\texttt{newY\_}ret^{3} = o_{32}$$
$$\texttt{newY\_}ret^{4} = o_{42}$$

$$\texttt{z1}^{\epsilon} = o_3 \qquad \texttt{z2}^{\epsilon} = o_4$$
$$\texttt{newY\_this}^{3} = \texttt{z1}^{\epsilon} \qquad \texttt{newY\_this}^{4} = \texttt{z2}^{\epsilon}$$
$$\texttt{y1}^{\epsilon} = \texttt{newY\_}ret^{3} \qquad \texttt{y2}^{\epsilon} = \texttt{newY\_}ret^{4}$$
$$\texttt{newX\_this}^{3} = \texttt{z1}^{\epsilon} \qquad \texttt{newX\_this}^{4} = \texttt{z2}^{\epsilon}$$
$$\texttt{x1}^{\epsilon} = \texttt{newX\_}ret^{3} \qquad \texttt{x2}^{\epsilon} = \texttt{newX\_}ret^{4}$$
$$\texttt{set\_this}^{32} = \texttt{y1}^{\epsilon} \qquad \texttt{set\_this}^{42} = \texttt{y2}^{\epsilon}$$
$$\texttt{x}^{32} = \texttt{x1}^{\epsilon} \qquad \texttt{x}^{42} = \texttt{x2}^{\epsilon}$$

Illustration

$$\texttt{set\_this}^{32}.f = \texttt{x}^{32}$$
$$\texttt{set\_this}^{42}.f = \texttt{x}^{42}$$

$$\texttt{newX\_}\mathit{ret}^{3} = o_{31}$$
$$\texttt{newX\_}\mathit{ret}^{4} = o_{41}$$

$$\texttt{newY\_}\mathit{ret}^{3} = o_{32}$$
$$\texttt{newY\_}\mathit{ret}^{4} = o_{42}$$

$$\texttt{z1}^{\epsilon} = o_3$$
$$\texttt{newY\_this}^{3} = \texttt{z1}^{\epsilon}$$
$$\texttt{y1}^{\epsilon} = \texttt{newY\_}\mathit{ret}^{3}$$
$$\texttt{newX\_this}^{3} = \texttt{z1}^{\epsilon}$$
$$\texttt{x1}^{\epsilon} = \texttt{newX\_}\mathit{ret}^{3}$$
$$\texttt{set\_this}^{32} = \texttt{y1}^{\epsilon}$$
$$\texttt{x}^{32} = \texttt{x1}^{\epsilon}$$

$$\texttt{z2}^{\epsilon} = o_4$$
$$\texttt{newY\_this}^{4} = \texttt{z2}^{\epsilon}$$
$$\texttt{y2}^{\epsilon} = \texttt{newY\_}\mathit{ret}^{4}$$
$$\texttt{newX\_this}^{4} = \texttt{z2}^{\epsilon}$$
$$\texttt{x2}^{\epsilon} = \texttt{newX\_}\mathit{ret}^{4}$$
$$\texttt{set\_this}^{42} = \texttt{y2}^{\epsilon}$$
$$\texttt{x}^{42} = \texttt{x2}^{\epsilon}$$

Illustration

$\texttt{set\_this}^{32}.f = \texttt{x}^{32}$
$\texttt{set\_this}^{42}.f = \texttt{x}^{42}$

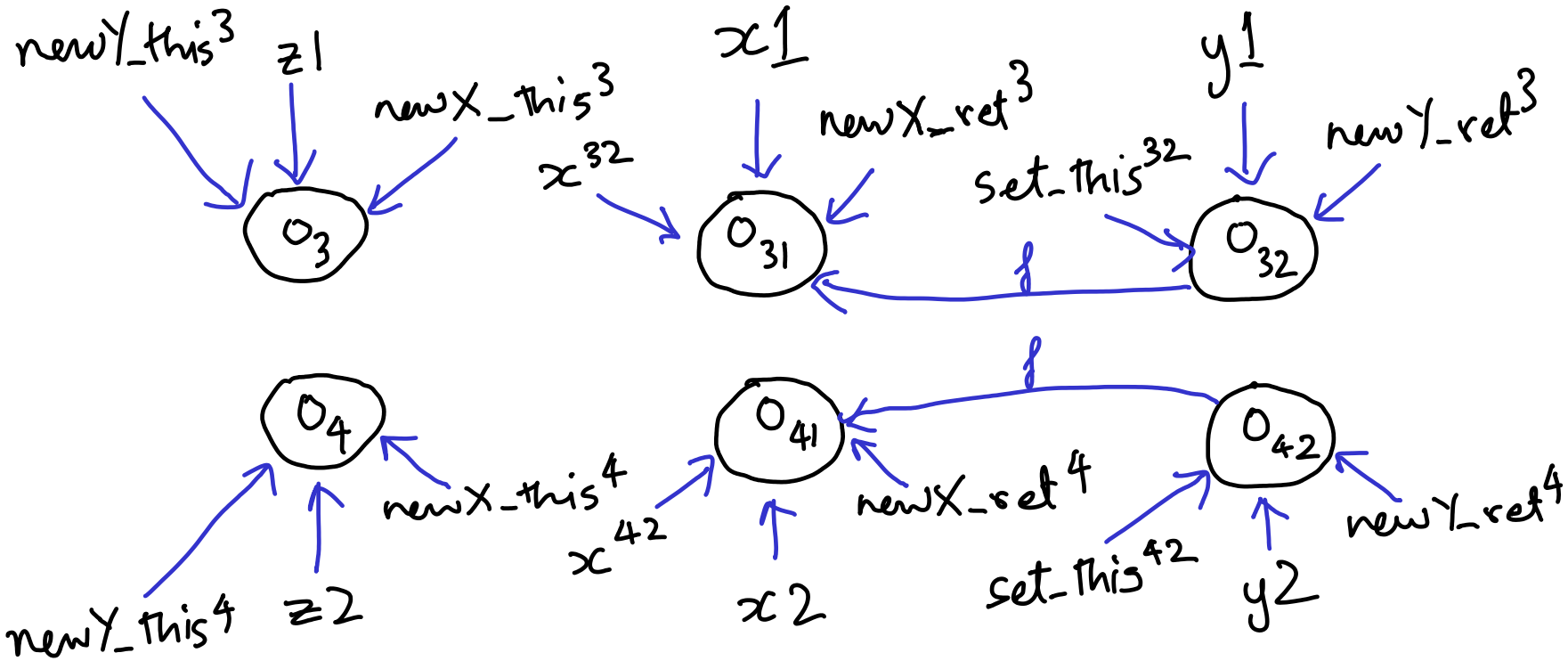
$\texttt{newX\_}\mathit{ret}^{3} = o_{31}$
$\texttt{newX\_}\mathit{ret}^{4} = o_{41}$

$\texttt{newY\_}\mathit{ret}^{3} = o_{32}$
$\texttt{newY\_}\mathit{ret}^{4} = o_{42}$

$\texttt{z1}^{\epsilon} = o_3$
$\texttt{newY\_this}^{3} = \texttt{z1}^{\epsilon}$
$\texttt{y1}^{\epsilon} = \texttt{newY\_}\mathit{ret}^{3}$
$\texttt{newX\_this}^{3} = \texttt{z1}^{\epsilon}$
$\texttt{x1}^{\epsilon} = \texttt{newX\_}\mathit{ret}^{3}$
$\texttt{set\_this}^{32} = \texttt{y1}^{\epsilon}$
$\texttt{x}^{32} = \texttt{x1}^{\epsilon}$

$\texttt{z2}^{\epsilon} = o_4$
$\texttt{newY\_this}^{4} = \texttt{z2}^{\epsilon}$
$\texttt{y2}^{\epsilon} = \texttt{newY\_}\mathit{ret}^{4}$
$\texttt{newX\_this}^{4} = \texttt{z2}^{\epsilon}$
$\texttt{x2}^{\epsilon} = \texttt{newX\_}\mathit{ret}^{4}$
$\texttt{set\_this}^{42} = \texttt{y2}^{\epsilon}$
$\texttt{x}^{42} = \texttt{x2}^{\epsilon}$

$\Downarrow$ (apply Andersen's analysis)



Object sensitivity vs. call-string sensitivity

- Object sensitivity is an instance of the functional approach.
- In general, incomparable with the call-strings technique in terms of precision.
- Is typically more precise than call-strings technique in the presence of flow insensitivity.

```
C v = new();
if (..) {
  A t11 = new();
  v.setF(t11);
  A t12 = v.getF();
}
else {
  A t21 = new();
  v.setF(t21);
  A t22 = v.getF();
}
```

```
class C {
  A f;
  void setF(A x) {this.f = x;}
  A getF() {return this.f;}
}
```

Here, analyzing `setF` under two distinct contexts does not help, because the argument to each call to `setF` will anyway be returned by both calls to `getF` due to flow insensitivity.

Determining method contexts and replicated instructions

Generate instructions for `main`, under ϵ context, ignoring all call instructions in `main`. Put these instructions in a set I.

repeat

{Each iteration of this loop is called a *phase*.}

Apply Andersen's analysis on I, and obtain a points-to graph G.

for all call-site $n : v^c.x(\ldots)$ in method copy (m, c) in I such that v^c has a non-empty points-to set in G **do**

Resolve this call n, create appropriate cop(ies) of the target method(s), and place call instructions at n to the target method(s). (In each target-method copy created, omit all call instructions.) Add instructions generated here to I.

end for

until No more instructions get added to I

Note: *The points-to graph G in each phase is guaranteed to be a superset of the corresponding graph in the previous phase.*

Illustration of phases

Phase 1

$z1^{\epsilon} = o_3$ $z2^{\epsilon} = o_4$

Illustration of phases

Phase 1
Phase 2

$$\texttt{newX\_}ret^3 = o_{31}$$
$$\texttt{newX\_}ret^4 = o_{41}$$

$$\texttt{newY\_}ret^3 = o_{32}$$
$$\texttt{newY\_}ret^4 = o_{42}$$

$$\texttt{z1}^\epsilon = o_3$$
$$\texttt{newY\_this}^3 = \texttt{z1}^\epsilon$$
$$\texttt{y1}^\epsilon = \texttt{newY\_}ret^3$$
$$\texttt{newX\_this}^3 = \texttt{z1}^\epsilon$$
$$\texttt{x1}^\epsilon = \texttt{newX\_}ret^3$$

$$\texttt{z2}^\epsilon = o_4$$
$$\texttt{newY\_this}^4 = \texttt{z2}^\epsilon$$
$$\texttt{y2}^\epsilon = \texttt{newY\_}ret^4$$
$$\texttt{newX\_this}^4 = \texttt{z2}^\epsilon$$
$$\texttt{x2}^\epsilon = \texttt{newX\_}ret^4$$

Illustration of phases

Phase 1
Phase 2
Phase 3

$$\texttt{set\_this}^{32}.f = \texttt{x}^{32}$$
$$\texttt{set\_this}^{42}.f = \texttt{x}^{42}$$

$$\texttt{newX\_}\mathit{ret}^{3} = o_{31}$$
$$\texttt{newX\_}\mathit{ret}^{4} = o_{41}$$

$$\texttt{newY\_}\mathit{ret}^{3} = o_{32}$$
$$\texttt{newY\_}\mathit{ret}^{4} = o_{42}$$

$$\texttt{z1}^{\epsilon} = o_3$$
$$\texttt{newY\_this}^{3} = \texttt{z1}^{\epsilon}$$
$$\texttt{y1}^{\epsilon} = \texttt{newY\_}\mathit{ret}^{3}$$
$$\texttt{newX\_this}^{3} = \texttt{z1}^{\epsilon}$$
$$\texttt{x1}^{\epsilon} = \texttt{newX\_}\mathit{ret}^{3}$$
$$\texttt{set\_this}^{32} = \texttt{y1}^{\epsilon}$$
$$\texttt{x}^{32} = \texttt{x1}^{\epsilon}$$

$$\texttt{z2}^{\epsilon} = o_4$$
$$\texttt{newY\_this}^{4} = \texttt{z2}^{\epsilon}$$
$$\texttt{y2}^{\epsilon} = \texttt{newY\_}\mathit{ret}^{4}$$
$$\texttt{newX\_this}^{4} = \texttt{z2}^{\epsilon}$$
$$\texttt{x2}^{\epsilon} = \texttt{newX\_}\mathit{ret}^{4}$$
$$\texttt{set\_this}^{42} = \texttt{y2}^{\epsilon}$$
$$\texttt{x}^{42} = \texttt{x2}^{\epsilon}$$