More on Points-To Analysis

Parameterized Object Sensitivity for Points-to Analysis for Java, A. Milanova, A. Rountev, and B. G. Ryder, *ACM Transactions on Software Engineering and Methodology (TOSEM)*, January 2005 (available at my research web page)

Scaling Java Points-to Analysis using Spark, O. Lhotak and L. Hendren, *Int. Conf. Compiler Construction (CC),* 2003

Earlier Discussion of Points-To Analysis

- Question (oversimplified): can variable **x** contain the address of variable **y** at program point *p*?
- Instructions of interest in C
  - **x = &y** (note that we do not consider taking the address of a function, an array element, or a struct field)
  - **x = y**
  - **x = *y**
  - ***x = y**
  - **x = null**
  - **x = malloc(…)**: think of it as **x = &heap**$_i$
  - **x = (*y).fld**
  - **(*x).fld = y**
  - **a[x] = y**
  - **x = a[y]**

Basic Ideas

- IN[n] $\subseteq$ (Vars × Vars) $\cup$ (Vars × Fields × Vars)
  - That is, a set of pairs (**x**,**y**) or triples (**x,fld,y**)
- Often defined as "points-to graph"
  - an edge **x** $\rightarrow$ **y** shows that **x** may point to **y**
  - an edge **x** $\xrightarrow{\textbf{fld}}$ **y** shows that field **fld** of struct **x** may point to **y**
- Merging two points-to graphs: just the union of their edge sets

Flow-Insensitive Formulation

- One graph G instead of per-node graphs IN[n]
- Switch-in-a-loop artificial structure
- No "kills": for a node n, $\mathbf{f_n}$(G) only adds to G, but does not remove any edges
  - E.g., for **x = y**: instead of $(G - \{x\}\times\ldots) \cup \{\,\mathbf{(x,z)} \mid (y,z)\in G\}$, we will use $G \cup \{\,\mathbf{(x,z)} \mid (y,z)\in G\}$
  - Can ignore **x = null** statements
- Conceptual fixed-point computation
  1. G := $\varnothing$
  2. for each n in some arbitrary order, **G:= $\mathbf{f_n}$(G)**
  3. If G changed in step 2, repeat step 2
- In reality, it would be implemented w/ worklist

Transfer Functions for C

- **x = &y**: $G \cup \{$ **(x,y)** $\}$
- **x = y**: $G \cup \{$ **(x,z)** $\mid (y,z) \in G\}$
- **x = *y**: $G \cup \{$ **(x,z)** $\mid (y,w) \in G \wedge (w,z) \in G \}$
- ***x = y**: $G \cup \{$ **(w,z)** $\mid (x,w) \in G \wedge (y,z) \in G \}$
- **x = malloc(…)** : $G \cup \{$ **(x,heap$_i$)** $\}$
- **x = a[y]**: $G \cup \{$ **(x,z)** $\mid (a,z) \in G \}$
- **a[x] = y**: $G \cup \{$ **(a,z)** $\mid (y,z) \in G \}$
- **x = (*y).fld**: $G \cup \{$ **(x,z)** $\mid (y,w) \in G \wedge (w,fld,z) \in G \}$
- **(*x).fld = y**: $G \cup \{$ **(w,fld,z)** $\mid (x,w) \in G \wedge (y,z) \in G \}$

Transfer Functions for Java

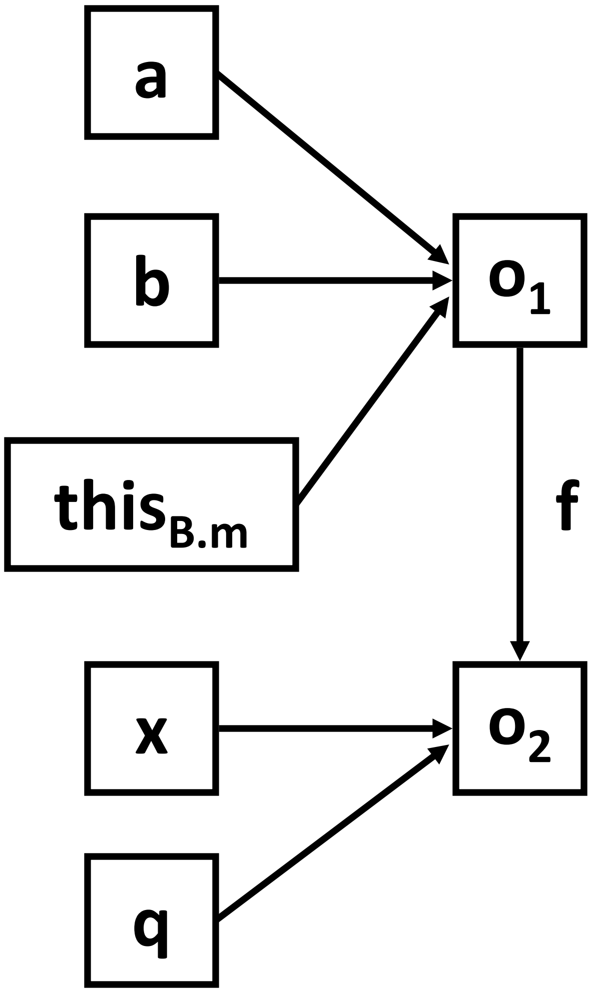
- **x = y**: same
- **x = new X**: same as malloc(…) calls
- **x = y.fld**: same as x = (*y).fld in C
- **x.fld = y**: same as (*x).fld = y in C
- **x = a[y]**: same as x = a.**any** (artificial field **any**)
- **a[x] = y**: same as a.**any** = y
- How about calls?
  - Option 1: pre-compute the call graph w/ CHA or RTA or …; treat parameter passing as x=y
  - Option 2: on-the-fly call graph (next slide)

Points-to Analysis for Java in Action

```
class A { void m(X p) {..} }

class B extends A {
    X f;
    void m(X q) {  this.f=q; }
}

B b = new B();
X x = new X();
A a = b;
a.m(x);
```



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

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

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

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

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

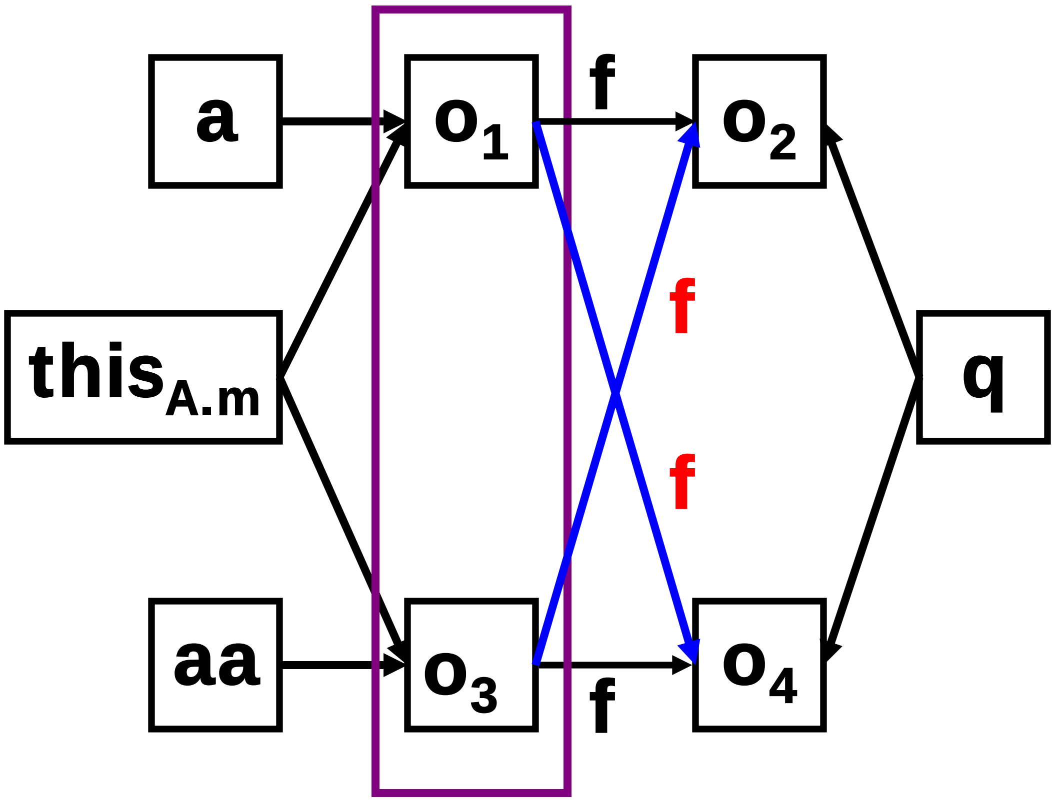
$$resolve(G, m, o_i, r_1, \ldots, r_n, l) =$$
$$\texttt{let} \ \ m_j(p_0, p_1, \ldots, p_n, ret_j) = dispatch(o_i, m) \ \ \texttt{in}$$
$$\{(p_0, o_i)\} \cup f(G, p_1 = r_1) \cup \ldots \cup f(G, l = ret_j)$$

Example: Imprecision

```
class Y extends X { … }

class A {
  X f;
  void m(X q) {
      this.f=q ; }
  }

A a = new A() ;
a.m(new X()) ;
A aa = new A() ;
aa.m(new Y()) ;
```



Object-Sensitive Analysis

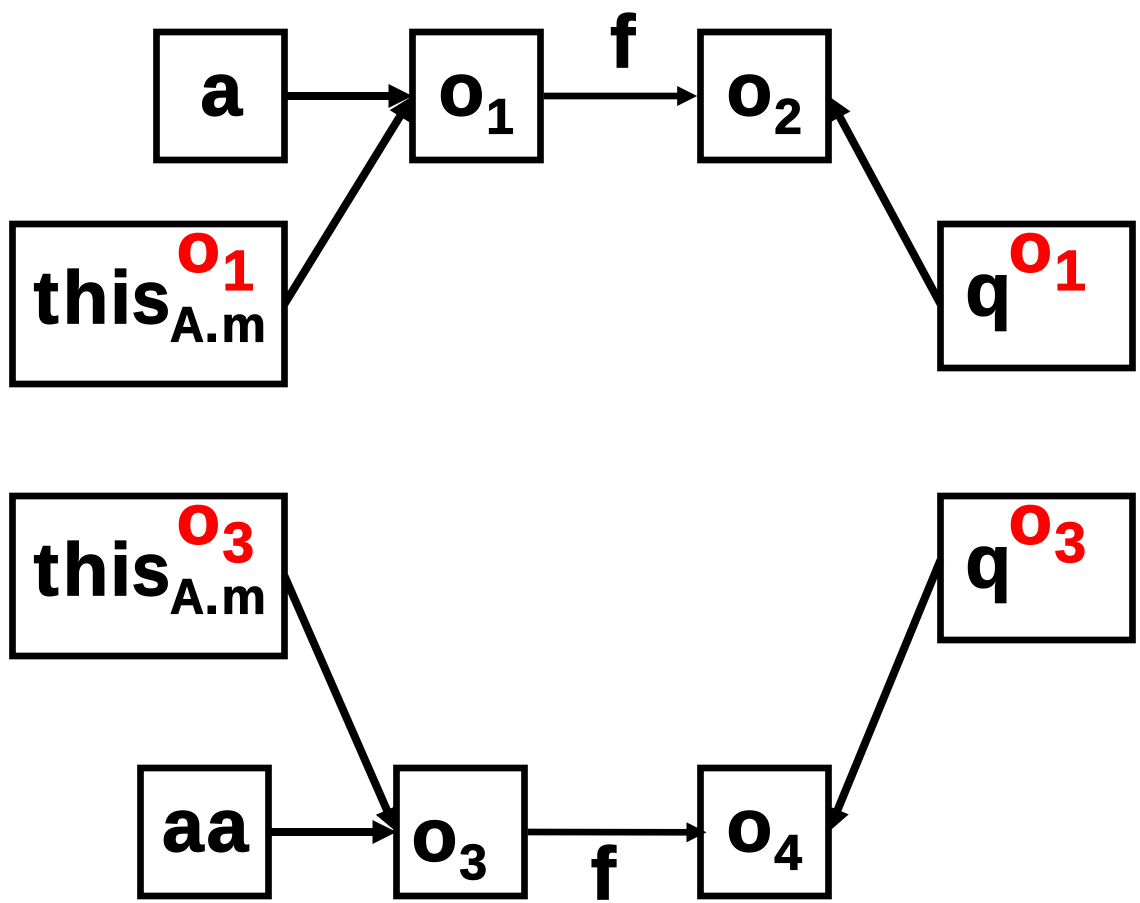
- Form of **calling-context-sensitive** analysis
- Instance methods and constructors analyzed for different contexts
- **Receiver objects used as contexts**
- Multiple copies of reference variables

$$\mathbf{this.f=q} \xrightarrow{o_1} \boxed{\mathbf{this}^{o_1}_{A.m}\mathbf{.f=q}^{o_1}}$$

Example: Object-sensitive Analysis

class **A** {
  X f;
  void m(X q) {
$\text{this}^{o3}_{A.m}\text{.f=q}^{o3}$; }
  }

A a = new A() ;
a.m(new X()) ;
A aa = new A() ;
aa.m(new Y()) ;



$$F(G, s_i : l = new\ C) = G \cup \bigcup_{o_{jk} \in \mathcal{C}_m} \{(l^{o_{jk}}, o_{ij})\}$$

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

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

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

$$F(G, l = r_0.m(r_1, \ldots, r_n)) =$$

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

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

Context-Insensitive Base-Object-Insensitive

- **x = y**: same; **x = new X**: same
- **x = y.fld**: same as **x = fld**
- **x.fld = y**: same as **fld = y**
- **x = a[y]**: same as **x = any**
- **a[x] = y**: same as **any = y**
  - One single field **any** for all array types, or a separate **any** for each array type (but beware of subtyping of array types, because variable **a** may have several array types)

Points-to graph has only **x** $\rightarrow$ **y**; i.e., no edges labeled with fields

Context-Insensitive Base-Object-Insensitive

- Specialized representation of the transfer functions through a **flow graph**
  - Not a points-to graph; not a control-flow graph
- **x = new X**: edge $\mathbf{o_i} \Rightarrow \mathbf{x}$ in the flow graph
- **x = y**: edge $\mathbf{y} \Rightarrow \mathbf{x}$ in the flow graph
- "**x** points-to $\mathbf{o_i}$" if and only if **x** is reachable from $\mathbf{o_i}$ in the flow graph
- Remember "for each n, **G:= f_n(G)**"? Equivalently
  1. For each $\mathbf{o_i} \Rightarrow \mathbf{x}$, **x** points to $\mathbf{o_i}$
  2. If **y** points to $\mathbf{o_i}$: for each $\mathbf{y} \Rightarrow \mathbf{x}$, **x** points to $\mathbf{o_i}$

  Easy to see that "points to" is same as "reachable from"