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)

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)
  - \( x = y \)
  - \( x = *y \)
  - \( *x = y \)
  - \( x = \text{null} \)
  - \( x = \text{malloc}(...) \): think of it as \( x = \&\text{heap}_i \)
  - \( x = (*y).\text{fld} \)
  - \( (*x).\text{fld} = y \)
  - \( a[x] = y \)
  - \( x = a[y] \)
Basic Ideas

• $\text{IN}[n] \subseteq (\text{Vars} \times \text{Vars}) \cup (\text{Vars} \times \text{Fields} \times \text{Vars})$
  – That is, a set of pairs $(x, y)$ or triples $(x, \text{fld}, y)$

• Often defined as “points-to graph”
  – an edge $x \to y$ shows that $x$ may point to $y$
  – an edge $x^{\text{fld}} \to y$ shows that field $\text{fld}$ 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 $n$, $f_n(G)$ only adds to $G$, but does not remove any edges
  - E.g., for $x = y$: instead of $(G - \{x\} \times \ldots) \cup \{(x, z) \mid (y, z) \in G\}$, we will use $G \cup \{(x, z) \mid (y, z) \in G\}$
  - Can ignore $x = \text{null}$ statements

Conceptual fixed-point computation

1. $G := \emptyset$
2. for each $n$ in some arbitrary order, $G := f_n(G)$
3. If $G$ changed in step 2, repeat step 2

In reality, it would be implemented w/ worklist
Transfer Functions for C

- $x = \&y: \quad G \cup \{ (x,y) \}$
- $x = y: \quad G \cup \{ (x,z) \mid (y,z) \in G \}$
- $x = *y: \quad G \cup \{ (x,z) \mid (y,w) \in G \land (w,z) \in G \}$
- $*x = y: \quad G \cup \{ (w,z) \mid (x,w) \in G \land (y,z) \in G \}$
- $x = \text{malloc}(\ldots): \quad G \cup \{ (x,\text{heap}_i) \}$
- $x = a[y]: \quad G \cup \{ (x,z) \mid (a,z) \in G \}$
- $a[x] = y: \quad G \cup \{ (a,z) \mid (y,z) \in G \}$
- $x = (*y).\text{fld}: \quad G \cup \{ (x,z) \mid (y,w) \in G \land (w,\text{fld},z) \in G \}$
- $(*x).\text{fld} = y: \quad G \cup \{ (w,\text{fld},z) \mid (x,w) \in G \land (y,z) \in G \}$
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)
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);
\[ f(G, l = \text{new } C) = G \cup \{(l, o_i)\} \]

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

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

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

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

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

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

\[ \{(p_0, o_i)\} \cup f(G, p_1 = r_1) \cup \ldots \cup f(G, l = \text{ret}_j) \]
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

\[
\text{this}.f = q \\
\text{this}_{A.m}.f = q^{0_1}
\]
Example: Object-sensitive Analysis

```java
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());
```
\[ F(G, s_i : l = \text{new } C) = G \cup \bigcup_{o_{jk} \in C_m} \{(l^{o_{jk}}, o_{ij})\} \]

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

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

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

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

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

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

\[ \text{let } c' = o_{ij} \]

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

\[ \{(p_0^{c'}, o_{ij})\} \cup f(G, p_1^c = r_1^c) \cup \ldots \cup f(G, l^c = \text{ret}^{c'}_j) \]
Context-Insensitive Base-Object-Insensitive

- \( x = y \): same; \( x = \text{new } X \): same
- \( x = y.fld \): same as \( x = \text{fld} \)
- \( x.fld = y \): same as \( \text{fld} = y \)
- \( x = a[y] \): same as \( x = \text{any} \)
- \( a[x] = y \): same as \( \text{any} = y \)

- One single field \( \text{any} \) for all array types, or a separate \( \text{any} \) 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

• \( x = \text{new } X \): edge \( o_i \Rightarrow x \) in the flow graph

• \( x = y \): edge \( y \Rightarrow x \) in the flow graph

• “\( x \text{ points-to } o_i \)” if and only if \( x \) is reachable from \( o_i \) in the flow graph

• Remember “for each \( n, G := f_n(G) \)”? Equivalently
  1. For each \( o_i \Rightarrow x \), \( x \) points to \( o_i \)
  2. If \( y \) points to \( o_i \): for each \( y \Rightarrow x \), \( x \) points to \( o_i \)

Easy to see that “points to” is same as “reachable from”