# Points-to Analysis for Java Using Annotated Constraints

~ Atanas Rountev, Ana Milanova and Barbara Ryder

OOPSLA 2001

# Who, When and Where?

- All three are researchers whose primary area of interest is static/dynamic analysis of programming languages and compilers
- Rountev and Milanova were graduate students of Dr. Ryder in 2001.
- OOPSLA is one of the premier conferences for advances related the OOP languages. Falls under ACM SIGPLAN. Acceptance rate < 20%.

# What is … Points-to Analysis?

Analysis to determine the set of objects pointed to by reference object fields or variables

# Why … Points-to Analysis?

- Enables downstream analyses & optimizations because it is general-purpose
- Examples: Call graph construction, synchronization removal, read-write cognizance and virtual call resolution
- These improve the compiled code and better the run-time performance of Java code

# Why …. these three?

- Recent profusion of object-oriented programming languages.
- Based on Andersen's points-to analysis for C
  - Need to adapt it for object-oriented programming languages
  - Tackle the disadvantages of Andersen's analysis
- Recent work that is closely related to this paper has problems

# How … Points-to Analysis?

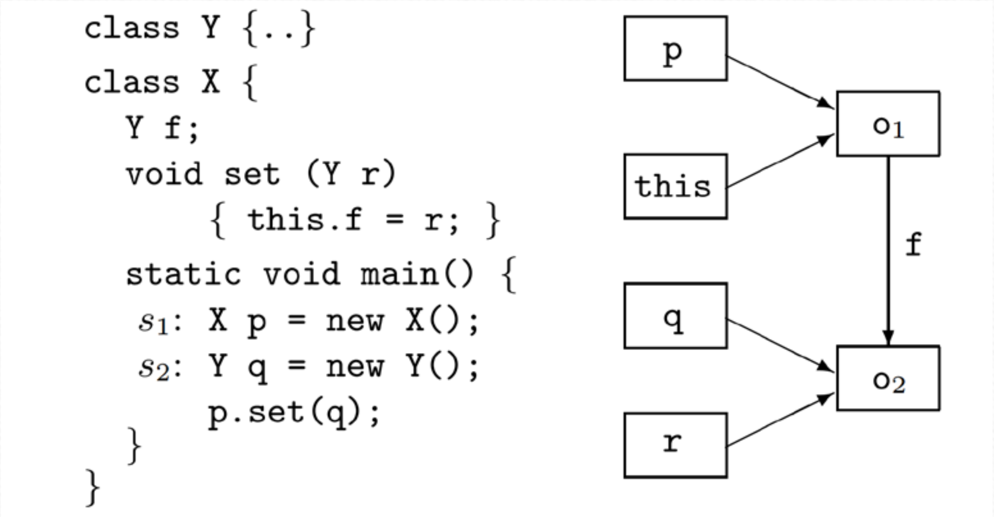
- Done with a constraint-based approach
- More specifically done using constraint annotations
- Two types of annotations:
  - Field annotations
  - Method annotations

# How… semantics of the analysis ?

- Analysis is done through sets
  - F contains all object fields
  - O contains unique names for all the objects
  - R contains all reference variables
- The five kind of statements in focus

# How … Points-to Graph?

- These statements generate constraints, which add edges to the points-to graph.
- For virtual call sites, resolution is performed for every receiver object pointed to by the calling object.



# What is … Annotated Inclusion Constraint?

- Authors introduced the concept of annotated constraints.
- Used to model both virtual calls and separate tracking of object fields.
- Expressed using constraint language:

$$L, R \rightarrow v \mid c(v_1, \ldots, v_n) \mid proj(c, i, v) \mid 0 \mid 1$$

$$L \subseteq_a R$$

# What is… Annotated Constraint Graph?

- Graph built from statements/rules given in the program
- Directed-edge between L to R.
- The edge is labeled with annotation *a*.
- Is a multi-graph

# What is… Annotated Constraint Graph?

- Graph contains source, variable and sink nodes.
- Use constraint graphs based on the inductive form.
- Graph is construct use two adjacency lists: *pred(n)* and *succ(n)*

# How to … solve annotated constraints?

$$\left.\begin{array}{l}\langle L, a\rangle \in \mathit{pred}(v) \\ \langle R, b\rangle \in \mathit{succ}(v) \\ \mathit{Match}(a, b)\end{array}\right\} \Rightarrow L \subseteq_{a \circ b} R \qquad (\textsc{Trans})$$

$$\mathit{Match}(a,b) = \begin{cases} \text{true} & \text{if } a \text{ or } b \text{ is the empty annotation } \epsilon \\ \text{true} & \text{if } a = b \\ \text{false} & \text{otherwise} \end{cases}$$

The annotation of the new constraint is

$$a \circ b = \begin{cases} a & \text{if } b = \epsilon \\ b & \text{if } a = \epsilon \\ \epsilon & \text{otherwise} \end{cases}$$

# How to … implement this?

- The abstract memory location representation of run-time memory locations is given by R *U* O
- $v_x$ represents the set of abstract locations pointed to by x.
- Representation of abstract locations is ref(x, $v_x$ , `$v_x$)

# How to… translate to constraints?

$$\langle l = new\ o_i \rangle \Rightarrow \{ref(o_i, v_{o_i}, \overline{v_{o_i}}) \subseteq v_l\}$$

$$\langle l = r \rangle \Rightarrow \{v_r \subseteq v_l\}$$

$$\langle l.f = r \rangle \Rightarrow \{v_l \subseteq proj(ref, 3, u),\ v_r \subseteq_f u\},\ \ u \text{ fresh}$$

$$\langle l = r.f \rangle \Rightarrow \{v_r \subseteq proj(ref, 2, u),\ u \subseteq_f v_l\},\ \ u \text{ fresh}$$

# What the ??????

```
s1:  p = new X();
s2:  q = new Y();
     p.f = q;
     r = p.f;
```

$$ref(o_1, v_{o_1}, \overline{v_{o_1}}) \subseteq v_p \qquad ref(o_2, v_{o_2}, \overline{v_{o_2}}) \subseteq v_q$$

$$v_p \subseteq proj(ref, 3, u) \qquad v_q \subseteq_f u$$

$$v_p \subseteq proj(ref, 2, w) \qquad w \subseteq_f v_r$$

# How do you… work it out?

# How to … handle virtual calls?

- Generate constraints from statements using the rule:

$$\langle l = r_0.m(r_1, \ldots, r_k) \rangle \Rightarrow \{ v_{r_0} \subseteq_m lam(\overline{0}, \overline{v_{r_1}}, \ldots, \overline{v_{r_k}}, v_l) \}$$

- This rule is based on lambda constructor

# What is … lamba and lookup table?

- Lambda constructor is a term that encapsulates actual arguments and left-hand side of the call
- Lookup table determines run-time target method of object at virtual call site
- Separately perform virtual dispatch for every receiver object.
- Lambda terms are created for all methods and stored in lookup table

# How do you… work it out(again)?

```
class A { X n() { ...  return rA; } }
class B extends A
        { X n() { ...  return rB; } }
s1:  A a = new A();
s2:  B b = new B();
     A c = b;
c1:  X x = b.n();
c2:  X y = c.n();
     if (...)  a = b;
c3:  X z = a.n();
```

(1) $ref(o_2, v_{o_2}, \overline{v_{o_2}}) \subseteq v_b \subseteq_{B.n} lam(\overline{0}, v_x) \Rightarrow$

$\{ref(o_2, v_{o_2}, \overline{v_{o_2}}) \subseteq v_{B.n.this},\ v_{rB} \subseteq v_x\}$

(2) $ref(o_2, v_{o_2}, \overline{v_{o_2}}) \subseteq v_b \subseteq_{A.n} lam(\overline{0}, v_y) \Rightarrow$

$\{ref(o_2, v_{o_2}, \overline{v_{o_2}}) \subseteq v_{B.n.this},\ v_{rB} \subseteq v_y\}$

(3) $ref(o_1, v_{o_1}, \overline{v_{o_1}}) \subseteq v_a \subseteq_{A.n} lam(\overline{0}, v_z) \Rightarrow$

$\{ref(o_1, v_{o_1}, \overline{v_{o_1}}) \subseteq v_{A.n.this},\ v_{rA} \subseteq v_z\}$

(4) $ref(o_2, v_{o_2}, \overline{v_{o_2}}) \subseteq v_a \subseteq_{A.n} lam(\overline{0}, v_z) \Rightarrow$

$\{ref(o_2, v_{o_2}, \overline{v_{o_2}}) \subseteq v_{B.n.this},\ v_{rB} \subseteq v_z\}$

# What is … cycle elimination and projection merging?

- Cycle detection: Detect a set of variables in a constraint graph that form a cycle.

$$v_1 \subseteq v_2 \subseteq \ldots \subseteq v_k \subseteq v_1$$

- Projection merging: Technique to reduce redundant edge additions in constraint systems.

$$v \subseteq proj(c, i, w) \qquad w \subseteq u_1 \qquad w \subseteq u_2$$

# How does it … better Andersen's method?

- A severe limitation of Andersen's method is the assumption that all code in the program is executable.
- Dealt with by taking into account only those methods that are reachable from the start points.
- Update the list of reachable methods as the analysis proceeds

# Does it work?

- Soot 1.0.0 used to generate intermediate representation from Java bytecode.
- BANE toolkit is modified to represent and solve annotated constraints.
- Tested on 23 large data programs
- Uses CHA to produce initial call graph.

# Does it work??

- Analysis runs under a minute for most of the programs
- Upper bound of less than six minutes
- Without field annotations, it takes substantially longer to run
- This analysis produces a more compact call graph out of CHA than RTA.
- This analysis resolves more virtual call sites than RTA does

# Does it work???

- Analysis detects about half of the thread-local allocations.
- Analysis detects about 29% of the sites for the method-local objects
- The combination of this has shown to substantially improves the run-time performance

# Questions?