Points-to Analysis for Java Using Annotated Constraints*

Dr. Barbara G. Ryder
Rutgers University

http://www.cs.rutgers.edu/~ryder
http://prolangs.rutgers.edu/

*Joint research with Atanas Rountev and Ana Milanova, supported by NSF-CCR 9900988.
Outline

- PROLANGS research projects
- Points-to analysis for Java
- Initial constraint-based implementation, OOPSLA’01
  - Empirical results
- Object-sensitive analysis, ISSTA’02
  - Empirical results
- Related work
- Summary
PROLANGS

• Research projects at boundary of Programming Languages/Compilers and Software Engineering
  - Algorithm design and prototyping
• Mature research projects
  - Pointer analysis of C programs
  - Side-effect analysis of C systems
  - Semantic software change analysis
  - Studies of Java exception usages
  - PROLANGS Analysis Framework (PAF), version 1.1 released June 1999
Ongoing Research

- **Object-oriented systems (C++/Java)**
  - Analysis
    - Points-to, side-effect analyses POPL'99, OOPSLA'01, ISSTA'02, ICSM'02
    - Change impact analysis (with Frank Tip, IBM) PASTE'01
  - Optimization
    - Profiling framework for feedback-directed optimization PLDI'01
    - Experience with feedback-directed optimization OOPSLA'02

- **Static/dynamic analyses of application resource usage**
  (with Rich Martin and Thu Nguyen, Rutgers)

- **Analysis of program fragments** FSE'99, CC'01, ICSE'03
Points-to Analysis for Java

• Which objects may reference variable \( x \) point to?

• Builds a points-to graph

\[
x = \text{new } A(); \\
y = \text{new } B(); \\
x.f = y;
\]
Uses of Points-to Information in Compilers

- **Object read-write information**
  - Side-effect analysis, dependence analysis

- **Call graph construction**
  - Devirtualization & inlining

- **Synchronization removal**

- **Stack-based object allocation**
Uses of Points-to Information in Software Engineering Tools

• Object read-write information
  - Semantic browsers
  - Program slicers
  - Debuggers

• Change impact analysis tools

• Testing
  - Object relationships (Object relation diagrams)
  - Program-based coverage metrics
Contributions

Points-to Analyses for Java using annotated constraints

- Initial analysis based on Andersen’s analysis for C, OOPSLA’01
  - Annotations embody OO notions needed
  - Maintained efficient constraint-based implementation
  - Empirical evaluation of cost and precision
Contributions

• Object-sensitive analysis, ISSTA’02
  – Adding context sensitivity
    • Parameterization framework
  – Empirical evaluation demonstrates more precision for same cost
Points-to Analysis, OOPSLA’01

- Handles virtual calls
  - Simulates the run-time method lookup
- Models the fields of objects
- Analyzes executable code
  - Ignores unreachable code from libraries
Points-to Analysis in Action

A.m() not analyzed because it’s unreachable.

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);
Efficient Implementation

• Constraint-based approach
  - Extends previous work for C pointer analysis using BANE (UC Berkeley)
    • Extended constraint system and resolution rules
• Define and solve a system of annotated set-inclusion constraints to obtain points-to sets
Annotated Constraints

- Form: \( L \subseteq_{a} R \)
  - \( L \) and \( R \) denote set expressions
  - Annotation \( a \): additional information (e.g., object fields)

- Kinds of set expressions \( L \) and \( R \)
  - **Set variables**: represent points-to sets
  - **ref terms**: represent objects
  - **Other kinds of expressions**
Set variables and \textit{ref} terms

- Set variables represent points-to sets
  - For each reference variable \( p \): \( \mathcal{V}_p \)
  - For each object \( o \): \( \mathcal{V}_o \)

- Object \( o \) is denoted by term \( \text{ref}(o,\mathcal{V}_o) \)
Example: Accessing Fields

\[
p = \text{new } A();
\]
\[
q = \text{new } B();
\]
\[
p.f = q;
\]
\[
\text{ref}(o_1, V_{o_1}) \subseteq V_p
\]
\[
\text{ref}(o_2, V_{o_2}) \subseteq V_q
\]
\[
V_p \subseteq \text{proj} \left( \text{ref}, W \right)
\]
\[
V_q \subseteq_f W
\]

Constraint generation
Example: Solving Constraints

\[
\begin{align*}
\text{ref}(o_1, V_{O_1}) & \subseteq V_p \\
\text{ref}(o_2, V_{O_2}) & \subseteq V_q \\
V_p & \subseteq \text{proj}(\text{ref}, W) \\
V_q & \subseteq f W \\
W & \subseteq V_{O_1} \\
V_q & \subseteq f V_{O_1} \\
\text{ref}(o_2, V_{O_2}) & \subseteq f V_{O_1}
\end{align*}
\]
Example: Virtual Calls

```
p.m(x);
```

```
V_p \subseteq_{m} \text{lam}(V_x)
```

receiver object \( o \)

```
\text{ref}(o, V_0) \subseteq V_p
```

Actual method called, \( A.m \)

```
V_x \subseteq V_z
```

```
\text{ref}(o, V_0) \subseteq V_{\text{this}(A.m)}
```
Experiments

- 23 Java programs: 14 – 677 user classes
  - Added the necessary library classes
  - Machine: 360 MHz, 512Mb SUN Ultra-60
- Cost measured in time and memory
- Precision (wrt usage in client analyses and transformations)
  - Object read-write information
  - Call graph construction
  - Synchronization removal and stack allocation
Resolution of Virtual Call Sites

% Resolved Call Sites

Points-to

RTA

Java Points-to, 12/02, BGR
Thread-local new sites

% Thread-local Sites

0% 10% 20% 30% 40% 50% 60% 70%

Java Points-to, 12/02, BGR
Practical Points-to Analyses for Java, ISSTA’02

- Existing analyses were flow- and context-insensitive extensions of C analyses
- Context insensitivity inherently compromises precision for object-oriented languages
- Goal: Introduce context sensitivity and remain practical
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());
Imprecision of Context-insensitive Analysis

• Does not distinguish contexts for instance methods and constructors
  - States of distinct objects are merged
• Common OO features and idioms
  - Encapsulation
  - Inheritance
  - Containers, maps and iterators
Object-sensitive Points-to Analysis

• Object sensitivity
  - Form of context sensitivity for flow-insensitive points-to analysis of OO languages

• Object-sensitive Andersen’s analysis
  - Object sensitivity applicable to other analyses

• Parameterization framework
  - Cost vs. precision tradeoff

• Empirical evaluation
  - Vs. context-insensitive OOPSLA’01 analysis
Details

- Instance methods and constructors analyzed for different contexts
- Receiver objects used as contexts
- Multiple copies of reference variables

\[ \text{this}.f=q \quad \Rightarrow \quad \text{this}_{A.m}.f=q_{01} \]
Example: Object-sensitive Analysis

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());
Implementation

- Implemented one instance of parameterization framework
  - `this`, formals and return variables (effectively) replicated
  - Optimized constraint-based analysis using previous technique
  - Comparison with OOPSLA’01 analysis
Empirical Results

• 23 Java programs: 14 – 677 user classes
  - Added the necessary library classes
  - Machine: 360 MHz, 512Mb, SUN Ultra-60

• Object Sensitive vs. OOPSLA’01 points-to

• Found comparable cost with better precision
  • Modification side-effect analysis
  • Virtual call resolution
Side-effect Analysis:
Modified Objects Per Statement

jb  jess  sablecc  raytrace  Average
One  Two or three  Four to nine  More than nine

0%  10%  20%  30%  40%  50%  60%  70%  80%  90%  100%
Improvement in Resolved Calls

Percent

proxy  compress  db  jb  echo  raytrace  mtrt  jtar  jflex  javacup  rabbit  jack  jflex  jess  mpegaudio  jitree  sablecc  javac  creature  mindterm  soot  muffin  javacc  Avg
Related Work

• Context-sensitive points-to analysis for OO languages
  - Grove et al. OOPSLA’97, Chatterjee et al. POPL’99, Ruf PLDI’00, Grove-Chambers TOPLAS’01

• Context-insensitive points-to analysis for OO languages
  - Liang et al. PASTE’01

• Context-sensitive class analysis
  - Oxhoj et al. ECOOP’92, Agesen SAS’94, Plevyak-Chien OOPSLA’94, Agesen ECOOP’95, Grove et al. OOPSLA’97, Grove-Chambers TOPLAS’01
Summary

• Defined two new points-to analyses for references in OOPLs using annotated constraints

• Context-insensitive analysis (OOPSLA'01)
  - Based on Andersen's points-to for C
  - Practical cost and good precision wrt client analyses and transformations
Summary

- Object-sensitive (context-sensitive) points-to analysis -
  - New kind of context sensitivity for flow-insensitive analysis
  - Parameterization framework allows tunable algorithm choice
  - Practical cost, comparable to OOPSLA'01 analysis
  - Better precision than OOPSLA'01 analysis
Number of new X() whose objects are accessed by p.f

One  Two or three  More than three
Parameterization

- **Goal:** tunable analysis
- **Multiple copies for a subset of variables**
  - For the other variables a single copy
- **Result:** reduces points-to graph size and analysis cost
  - At the expense of precision loss