Doop: The Latest

Yannis Smaragdakis
University of Athens

with George Balatsouras, Martin Bravenboer (LogicBlox Inc.),
Kostas Ferles, Neville Grech, George Kastrinis
and
Anastasis Antoniadis, Nikos Filippakis, Dimitris Mouris,
Nefeli Prokopaki, Kostas Saidis
Our Framework

- Datalog-based pointer analysis framework for Java
- Declarative: what, not how

- Sophisticated, very rich set of analyses
  - subset-based analysis, fully on-the-fly call graph discovery, field-sensitivity, context-sensitivity, call-site sensitive, object sensitive, thread sensitive, context-sensitive heap, abstraction, type filtering, precise exception analysis

- Support for full semantic complexity of Java
  - jvm initialization, reflection analysis, threads, reference queues, native methods, class initialization, finalization, cast checking, assignment compatibility

http://doop.program-analysis.org
Pointer (or *points-to*) Analysis

- What objects can a variable point to?

**Program**

```java
void foo() {
    Object a = new A1();
    Object b = id(a);
}

void bar() {
    Object a = new A2();
    Object b = id(a);
}

Object id(Object a) {
    return a;
}
```

**Points-to**

- `foo:a` - new A1()
- `bar:a` - new A2()

Objects represented by allocation sites
**Pointer Analysis**

- **What objects can a variable point to?**

<table>
<thead>
<tr>
<th>program</th>
<th>points-to</th>
</tr>
</thead>
</table>
| void foo() {  
  Object a = new A1();  
  Object b = id(a);  
} | foo:a | new A1()  
  bar:a | new A2()  
  id:a | new A1(), new A2() |

void bar() {  
  Object a = new A2();  
  Object b = id(a);  
}  

Object id(Object a) {  
  return a;  
}
Pointer Analysis

- What objects can a variable point to?

<table>
<thead>
<tr>
<th>program</th>
<th>points-to</th>
</tr>
</thead>
</table>
| void foo() {
  Object a = new A1();
  Object b = id(a);
} |

<table>
<thead>
<tr>
<th></th>
<th>foo:a</th>
<th>bar:a</th>
<th>id:a</th>
<th>foo:b</th>
<th>bar:b</th>
</tr>
</thead>
</table>

Object id(Object a) {
  return a;
}
Datalog: Properties

- Limited logic programming
  - SQL with recursion
  - Prolog without complex terms (constructors)
- Captures PTIME complexity class
- Strictly declarative
  - as opposed to Prolog
    - conjunction commutative
    - rules commutative

Less programming, more specification
Datalog: Declarative Mutual Recursion

```
source
a = new A();
b = new B();
c = new C();
a = b;
b = a;
c = b;
```
Datalog: Declarative Mutual Recursion

```
source
a = new A();
b = new B();
c = new C();
a = b;
b = a;
c = b;

Alloc
a | new A()
b | new B()
c | new C()

Move
a | b
b | a
c | b

VarPointsTo
a | new A()
b | new B()
c | new C()
a | new B()
b | new A()
c | new B()
c | new A()
```

```
VarPointsTo(var, obj) <- Alloc(var, obj).
VarPointsTo(to, obj) <- Move(to, from), VarPointsTo(from, obj).
```
Datalog: How Well Has It Worked?

- Our decision to write a full Datalog framework has worked extraordinarily well
  - ease of development, maintenance
  - ease of experimentation, communication
  - different engines, parallelization

- But not all is rosy
  - some analyses hard to express
    - e.g., Steensgaard-style points-to analysis
One Such Instance: Must-Alias Analysis (latest work)

- Flow-sensitive must-alias analysis on access paths:
  - “must-” : under-approximation
    - do two access paths definitely alias at a program point?
  - alias classes are equivalence classes
    - $a \sim b, b \sim c \Rightarrow a \sim c$
    - not true of may-alias analyses, unless grossly imprecise (Steensgaard)
  - classes need to be maintained compactly
    - much like the union-find trees of Steensgaard
      - though union operations do not arise here
  - access-paths maintained implicitly
Solution: Specialized Data Structure for Must-Alias
Benefits: Lots of Interesting, Fast Algorithms

- Intersection is the main one
- Often >20x speedup relative to Datalog implementation
  - different factors of 10 due to access path implicit representation, equivalence classes
- But: need to do this outside Datalog
Example Algorithm (Intersection)
Generalization: DeepDoop

- Extension of Datalog (a DSL) for staging Datalog analyses, importing/exporting to external analyses
- Kudos to Souffle for ideas!
- Also highly useful in other recent work
  - sound may-point-to analysis
- E.g.,
  - point-to $\rightarrow$ call-graph $\rightarrow$ point-to $\rightarrow$ escape $\rightarrow$ point-to $\rightarrow$ …
  - why non-monotonicity?
    - points-to maintained if !escape
    - points-to strong update if must-alias
Advertising Portion

- What else are we doing?
  - much faster Soot front-end (multi-threaded)
  - open-sourcing LogicBlox engine
  - experiment with Souffle
  - web-based program comprehension service
  - Android analysis
  - reflection improvements
  - information-flow

- Ask me!