Declarative Static Program Analysis

Yannis Smaragdakis,
University of Athens
U. Massachusetts, Amherst
LogicBlox Inc.
These Lectures: Program Analysis with a twist

- We will see concepts you may know (e.g., dataflow analysis), expressed in a computable logic framework
  - declarative program analysis using the Datalog language
- Datalog = simple logic programming language
  - captures polynomial computation ($P$)
- Datalog: not quite logic (if you want efficiency) but much closer to pure logic than to a program
- Captures complex recursive/fixpoint reasoning elegantly
Outline

- Introduction: points-to analysis: a fundamental inter-procedural analysis
- Datalog for points-to analysis, techniques
- Generalizing: flow-sensitive program analysis and dataflow ideas
- Advanced optimization ideas, points-to analysis concepts, client analyses
Points-To Analysis and Datalog

An introduction
Program Analysis: Run Faster
Program Analysis: Software Understanding
Program Analysis: Find Bugs

Yannis Smaragdakis
Pointer Analysis

- What objects can a variable point to?

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

void bar() {
    a = new A2();
    b = id(a);
}
// A1 <: A, A2 <: A
A id(A a) {
    return a;
}
```

points-to

- objects represented by allocation sites

```java
foo:a new A1()
bar:a new A2()
```
Pointer Analysis

- What objects can a variable point to?

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

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

// A1 <: A, A2 <: A
A id(A a) {
    return a;
}
points-to
foo:a new A1()
bar:a new A2()
id:a new A1(), new A2()
```
## Pointer Analysis

- What objects can a variable point to?

**Program**

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

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

// A1 <: A, A2 <: A
A id(A a) {
    return a;
}
```

**Points-to**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Points-to</th>
</tr>
</thead>
<tbody>
<tr>
<td>foo:a</td>
<td>new A1()</td>
</tr>
<tr>
<td>bar:a</td>
<td>new A2()</td>
</tr>
<tr>
<td>id:a</td>
<td>new A1(), new A2()</td>
</tr>
<tr>
<td>foo:b</td>
<td>new A1(), new A2()</td>
</tr>
<tr>
<td>bar:b</td>
<td>new A1(), new A2()</td>
</tr>
</tbody>
</table>

**Context-sensitive points-to**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Points-to</th>
</tr>
</thead>
<tbody>
<tr>
<td>foo:a</td>
<td>new A1()</td>
</tr>
<tr>
<td>bar:a</td>
<td>new A2()</td>
</tr>
<tr>
<td>id:a (foo)</td>
<td>new A1()</td>
</tr>
<tr>
<td>id:a (bar)</td>
<td>new A2()</td>
</tr>
<tr>
<td>foo:b</td>
<td>new A1()</td>
</tr>
<tr>
<td>bar:b</td>
<td>new A2()</td>
</tr>
</tbody>
</table>

Yannis Smaragdakis
Pointer Analysis: A Complex Domain

- flow-sensitive
- field-sensitive
- heap cloning
- context-sensitive
- binary decision diagrams
- inclusion-based
- unification-based
- on-the-fly call graph
- k-cfa
- object sensitive
- field-based
- demand-driven

Keywords: alias analysis, pointer analysis

Results 1 - 20 of 2,343
Sort by relevance in expanded form

1. Semi-sparse flow-sensitive pointer analysis
   Publisher: ACM
   Full text available: PDF (246.08 KB)
   Additional Information: full citation, abstract, references, index terms
   Bibliometrics: Downloads (6 Weeks): 34, Downloads (12 Months): 34, Citation Count: 0
   Pointer analysis is a prerequisite for many program analyses, and the effectiveness of these analyses depends on the precision of the pointer information they receive. Two major axes of pointer analysis precision are flow-sensitivity and context-sensitivity, ...
   Keywords: alias analysis, pointer analysis

2. Efficient field-sensitive pointer analysis of C
   David J. Pearce, Paul D. Kelly, Chris Hankin
   November 2007, Transactions on Programming Languages and Systems (TOPLAS), Volume 30 Issue 1
   Publisher: ACM
   Full text available: PDF (924.64 KB)
   Additional Information: full citation, abstract, references, index terms
   Bibliometrics: Downloads (6 Weeks): 31, Downloads (12 Months): 282, Citation Count: 1
   The subject of this article is flow- and context-insensitive pointer analysis. We present a novel approach for precisely modelling struct variables and indirect function calls. Our method emphasises efficiency and simplicity and is based on a simple ...
   Keywords: Set-constraints, pointer analysis

3. Cloning-based context-sensitive pointer alias analysis using binary decision diagrams
   Paul Dehnert, Martin Rinard
   June 2004, PLDI '04: Proceedings of the ACM SIGPLAN 2004 conference on Programming language design and implementation
   Publisher: ACM
   Full text available: PDF (301.86 KB)
   Additional Information: full citation, abstract, references, index terms
   Bibliometrics: Downloads (6 Weeks): 8, Downloads (12 Months): 8, Citation Count: 1
   Yannis Smaragdakis
From Algorithms To Specification And Back

Yannis Smaragdakis
From Algorithms To Specification
What Does It Mean To Be Declarative?

“denoting high-level programming languages which can be used to solve problems without requiring the programmer to specify an exact procedure to be followed.”

- high-level
- what, not how
- no control-flow
- no side-effects
- specifications, not programs, not algorithms
Pointer Analysis: Current Approaches

Context-sensitive pointer analysis for Java

- **paddle**
  - java + relational algebra + binary decision diagrams (BDD)

- **wala**
  - Java, conventional approach

- **bddbddd** (pioneered Datalog for realistic points to analysis)
  - Datalog + Java + BDD

Not a single purely declarative specification

Coupling of specification and algorithm
Our Pointer Analysis Framework

- Datalog-based pointer analysis framework for Java
- Declarative: what, not how
  - easier to express sophisticated analyses
  - correctness more clear
  - clear variation points
  - eases exploration of approximations
  - eases performance and precision comparison
  - enables aggressive optimization
- 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
Key Contributions

- Expressed complete, complex pointer analyses in Datalog
  - core specification: ~250 lines of logic
  - full specification: ~2500 lines of logic
- Synthesized efficient algorithms from specification
  - order of magnitude performance improvement
  - support all analyses considered interesting
- Approach: heuristics for searching algorithm space
  - targeted at recursive problem domains
- Demonstrated scalability with explicit representation
  - no BDDs
These Contributions Are Surprising

- Expressed complete, complex pointer analyses in Datalog
  Lhotak: “[E]ncoding all the details of a complicated program analysis problem [on-the-fly call graph construction, handling of Java features] purely in terms of subset constraints may be difficult or impossible.”

- Scalability and Efficiency
  Lhotak: “Efficiently implementing a 1H-object-sensitive analysis without BDDs will require new improvements in data structures and algorithms”
  Whaley: “Owing to the power of the BDD data structure, bddbddb can even solve analysis problems that were previously intractable”
  Lhotak: “I’ve never managed to get Paddle to run in available memory with these settings [2-cfa context-heap], at least not on real benchmarks complete with the standard library.”
From Algorithms To Specification

Pointer Analysis and Datalog Background
Program Analysis: a Domain of Mutual Recursion

- var points-to
- call graph
- reachable methods
- exceptions
- field points-to

Yannis Smaragdakis
Program Analysis: a Domain of Mutual Recursion

\[ x = y \]

\texttt{var points-to}
Program Analysis: a Domain of Mutual Recursion

\[ x = f() \]

var points-to

call graph
Program Analysis: a Domain of Mutual Recursion

\[ x = y.f() \]

```
var points-to
```

call graph
Program Analysis: a Domain of Mutual Recursion

```java
x = new A()
```

```
var points-to
```

```
call graph
```

```
reachable methods
```
Program Analysis: a Domain of Mutual Recursion

\[ x.f = y \]

- call graph
- var points-to
- field points-to
- reachable methods
Program Analysis: a Domain of Mutual Recursion

x = y.f

var points-to

call graph

field points-to

reachable methods
Program Analysis: a Domain of Mutual Recursion

throw e

var points-to

call graph

exceptions

field points-to

reachable methods

Yannis Smaragdakis
Program Analysis: a Domain of Mutual Recursion

catch(E e)

var points-to

call graph

reachable methods

exceptions

field points-to
Program Analysis: a Domain of Mutual Recursion

- var points-to
- call graph
- reachable methods
- exceptions
- field points-to

Yannis Smaragdakis
Datalog: Declarative Mutual Recursion

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

Yannis Smaragdakis
Datalog: Declarative Mutual Recursion

source

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

AssignObjectAllocation

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

Assign(from, to)

```
b | a
a | b
b | c
```

rules

```
VarPointsTo(?var, ?obj) <-
   AssignObjectAllocation(?var, ?obj).

VarPointsTo(?to, ?obj) <-
   Assign(?from, ?to),
   VarPointsTo(?from, ?obj).
```
# Datalog: Declarative Mutual Recursion

```plaintext
<table>
<thead>
<tr>
<th>source</th>
</tr>
</thead>
</table>
| a = new A();
| b = new B();
| c = new C();
| a = b;
| b = a;
| c = b; |

<table>
<thead>
<tr>
<th>AssignObjectAllocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
</tr>
<tr>
<td>b</td>
</tr>
<tr>
<td>c</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Assign(from, to)</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
</tr>
<tr>
<td>a</td>
</tr>
<tr>
<td>b</td>
</tr>
</tbody>
</table>
```

```
\[ \text{VarPointsTo(?var, ?obj) } \leftarrow \]
\[ \text{AssignObjectAllocation(?var, ?obj).} \]

\[ \text{VarPointsTo(?to, ?obj) } \leftarrow \]
\[ \text{Assign(?from, ?to),} \]
\[ \text{VarPointsTo(?from, ?obj).} \]
```

Yannis Smaragdakis
Datalog: Declarative Mutual Recursion

source

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

AssignObjectAllocation

AssignObjectAllocation(?var, ?obj) <-
  AssignObjectAllocation(?var, ?obj).

VarPointsTo

VarPointsTo(?to, ?obj) <-
  AssignObjectAllocation(?from, ?to),
  VarPointsTo(?from, ?obj).
Datalog: Declarative Mutual Recursion

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

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

VarPointsTo

assign(from, to)
b | a
a | b
b | c

VarPointsTo(?var, ?obj) <-
AssignObjectAllocation(?var, ?obj).

VarPointsTo(?to, ?obj) <-
Assign(?from, ?to),
VarPointsTo(?from, ?obj).
Datalog: Declarative Mutual Recursion

source

\[
\begin{align*}
a &= \text{new A()}; \\
b &= \text{new B()}; \\
c &= \text{new C()}; \\
a &= b; \\
b &= a; \\
c &= b;
\end{align*}
\]

AssignObjectAllocation

\[
\begin{align*}
a &\quad \text{new A()} \\
b &\quad \text{new B()} \\
c &\quad \text{new C()}
\end{align*}
\]

VarPointsTo

\[
\begin{align*}
\text{VarPointsTo(}\ ?\text{var, } ?\text{obj}) &\leftarrow \text{AssignObjectAllocation(}\ ?\text{var, } ?\text{obj}). \\
\text{VarPointsTo(}\ ?\text{to, } ?\text{obj}) &\leftarrow \text{Assign(}\ ?\text{from, } ?\text{to}), \text{VarPointsTo(}\ ?\text{from, } ?\text{obj}).
\end{align*}
\]

Assign(from, to)

\[
\begin{align*}
b &\quad a \\
a &\quad b \\
b &\quad c
\end{align*}
\]

body relations
**Datalog: Declarative Mutual Recursion**

### AssignObjectAllocation

<table>
<thead>
<tr>
<th>Source</th>
<th>AssignObjectAllocation</th>
<th>VarPointsTo</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>a = new A();</code>&lt;br&gt;<code>b = new B();</code>&lt;br&gt;<code>c = new C();</code>&lt;br&gt;<code>a = b;</code>&lt;br&gt;<code>b = a;</code>&lt;br&gt;<code>c = b;</code></td>
<td>`a</td>
<td>new A()<code>&lt;br&gt;</code>b</td>
</tr>
</tbody>
</table>

### VarPointsTo

```prolog
VarPointsTo(?var, ?obj) <-
    AssignObjectAllocation(?var, ?obj).

VarPointsTo(?to, ?obj) <-
    Assign(?from, ?to),
    VarPointsTo(?from, ?obj).
```

Yannis Smaragdakis
Datalog: Declarative Mutual Recursion

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

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

```
VarPointsTo
```

```
Assign(from, to)
b | a
a | b
b | c
```

```
VarPointsTo(?var, ?obj) <-
AssignObjectAllocation(?var, ?obj).

VarPointsTo(?to, ?obj) <-
Assign(?from, ?to),
VarPointsTo(?from, ?obj).
```
Datalog: Declarative Mutual Recursion

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

```
AssignObjectAllocation
a   new A()
b   new B()
c   new C()
```

```
Assign(from, to)
b   a
a   b
b   c
```

```
VarPointsTo(?var, ?obj) <-
  AssignObjectAllocation(?var, ?obj).

VarPointsTo(?to, ?obj) <-
  Assign(?from, ?to),
  VarPointsTo(?from, ?obj).
```

Yannis Smaragdakis
Datalog: Declarative Mutual Recursion

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

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

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

Assign(from, to)
b | a
a | b
b | c

2nd rule evaluation

VarPointsTo(?var, ?obj) <-
AssignObjectAllocation(?var, ?obj).

VarPointsTo(?to, ?obj) <-
Assign(?from, ?to),
VarPointsTo(?from, ?obj).

Yannis Smaragdakis
**Datalog: Declarative Mutual Recursion**

<table>
<thead>
<tr>
<th>source</th>
<th>AssignObjectAllocation</th>
<th>VarPointsTo</th>
</tr>
</thead>
<tbody>
<tr>
<td>a = new A();</td>
<td>a</td>
<td>new A()</td>
</tr>
<tr>
<td>b = new B();</td>
<td>b</td>
<td>new B()</td>
</tr>
<tr>
<td>c = new C();</td>
<td>c</td>
<td>new C()</td>
</tr>
<tr>
<td>a = b;</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>b = a;</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>c = b;</td>
<td>c</td>
<td></td>
</tr>
</tbody>
</table>

Assign(from, to):

- b  | a
- a  | b
- b  | c

VarPointsTo(?var, ?obj) <-
AssignObjectAllocation(?var, ?obj).

VarPointsTo(?to, ?obj) <-
Assign(?from, ?to),
VarPointsTo(?from, ?obj).

2\textsuperscript{nd} rule result
Datalog: Declarative Mutual Recursion

<table>
<thead>
<tr>
<th>source</th>
</tr>
</thead>
</table>
| `a = new A();`  
| `b = new B();`  
| `c = new C();`  
| `a = b;`  
| `b = a;`  
| `c = b;` |

<table>
<thead>
<tr>
<th>AssignObjectAllocation</th>
</tr>
</thead>
</table>
| `a | new A()`  
| `b | new B()`  
| `c | new C()` |

<table>
<thead>
<tr>
<th>Assign(from, to)</th>
</tr>
</thead>
</table>
| `b | a`  
| `a | b`  
| `b | c` |

<table>
<thead>
<tr>
<th>VarPointsTo</th>
</tr>
</thead>
</table>
| `a | new A()`  
| `b | new B()`  
| `c | new C()`  
| `a | new B()`  
| `b | new A()`  
| `c | new B()`  
| `c | new A()` |

Yannis Smaragdakis