On Abstraction Refinement for Static Analyses in Datalog

Xin Zhang Ravi Mangal Radu Grigore
Mayur Naik Hongseok Yang

October 2014

Parameterized Analysis Example: Must Alias for Type State

Abstraction: track variables x and y.

```
                                   // x⊥, y⊥
x = new FileReader(/*...*/);       // x open, y⊥
y = x;                             // xy open
y.close();                         // xy closed
// assert x is closed              // ✓
// assert x is opened              // ✗
```

Parameterized Analysis Example: Must Alias for Type State

Abstraction: track variable x.

```
                                    // x⊥
x = new FileReader(/*...*/);        // x open
y = x;                              // x* open
y.close();                          // x* ⊥
// assert x is closed               // ✗
// assert x is opened               // ✗
```

Parameterized Analysis Example: Alias with Call Context

```
void a() {
  c(new Object(), new Object());
}
void b() {
  Object x = new Object(); c(x, x);
}
void c(Object x, Object y) {
  assert (x != y);
}
```

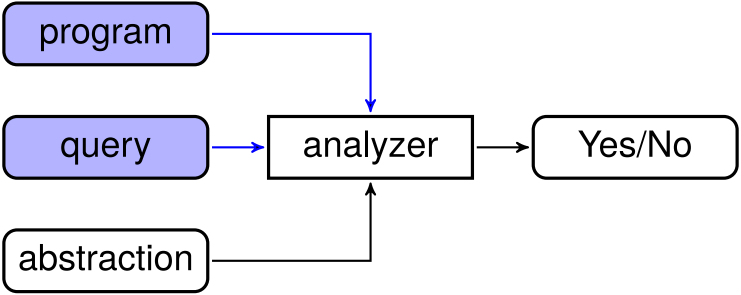
OK in contexts $[c, a, *]$, and NOK in $[c, b, *]$

Parameterized Analysis Example: Alias with Call Context

```
void a() {
  c(new Object(), new Object());
}
void b() {
  Object x = new Object(); c(x, x);
}
void c(Object x, Object y) {
  assert (x != y);
}
```

Unknown in context $[c, *]$

The Problem



Query: Could this bad thing happen?

Problem: Is the answer Yes for all abstractions?

The Model

- $i, j : \{0, 1, \ldots, n-1\}$ are *parameters*
- x, y, z are *abstractions*
  they assign integer values to parameters
- u, v are are *parameter masks*
  they assign boolean values to parameters
- f is the *analysis*
- $g(x)$ is the *local provenance at abstraction* x
- h is the *global provenance*

The Model

$$f : (n \to \omega) \to 2$$
$$g : (n \to \omega) \to (n \to 2) \to 2$$
$$h : ((n \times \omega) \to 2) \to 2$$

$$f(x) \stackrel{\Delta}{=} h(\lambda(i,k).\ x(i) = k)$$
$$g(x)(u) \stackrel{\Delta}{=} h(\lambda(i,k).\ x(i) = k \wedge u(i))$$
$$x \leq y \Rightarrow f(y) \leq f(x)$$
$$\mathbf{x} \leq \mathbf{y} \Rightarrow h(\mathbf{x}) \leq h(\mathbf{y})$$

Simple Properties

For all x, the boolean function $g(x)$ contains more information than the value $f(x)$.

$$
\begin{aligned}
& g(x)(\lambda i.\, 1) \\
= \quad & h(\lambda(i,k).\, x(i) = k \wedge (\lambda i.\, 1)(i)) \\
= \quad & h(\lambda(i,k).\, x(i) = k) \\
= \quad & f(x)
\end{aligned}
$$

Simple Properties

For all x, the boolean function $g(x)$ is monotonic.

Hypothesis: $u \leq v$

$$
\begin{array}{ll}
 & g(x)(u) \\
= & h(\lambda(i,k).\ x(i) = k \wedge u(i)) \\
\leq & h(\lambda(i,k).\ x(i) = k \wedge v(i)) \quad \text{by hypothesis} \\
= & g(x)(v)
\end{array}
$$

Simple Properties — Details

$$
\begin{aligned}
& h(\lambda(i,k).\, B) \le h(\lambda(i,k).\, C) \\
\Leftarrow\quad & (\lambda(i,k).\, B) \le (\lambda(i,k).\, C) \\
=\quad & \forall(i,k)\ (B \le C) \\
=\quad & (x(i) = k \wedge u(i)) \le (x(i) = k \wedge v(i)) \\
\Leftarrow\quad & u(i) \le v(i) \\
=\quad & \forall i\ (u(i) \le v(i)) \\
=\quad & (\lambda i.\, u(i)) \le (\lambda i.\, v(i)) \\
=\quad & u \le v
\end{aligned}
$$

Prediction Lemma

Local provenance lets us predict the result for other abstractions.

Hypothesis: If $u(i)$, then $x(i) = y(i)$.

$$
\begin{aligned}
& g(y)(u) \\
= \quad & h(\lambda(i,k).\ y(i) = k \wedge u(i)) \\
\leq \quad & h(\lambda(i,k).\ x(i) = k) \\
= \quad & f(x)
\end{aligned}
$$

Prediction Lemma

Local provenance lets us predict the result for other abstractions.

Hypothesis: If $u(i)$, then $x(i) = y(i)$.

$$\begin{array}{rl} & g(y)(u) \\ = & h(\lambda(i,k).\ y(i) = k \wedge u(i)) \\ \leq & h(\lambda(i,k).\ x(i) = k) \\ = & f(x) \end{array}$$

Combined with the anti-monotonicity of f it predicts even more!

Impossible Queries

Local provenance sometimes lets us decide that the analyzer always answers Yes. Lingo: the query is *impossible*.

Corollary

If $g(y)(\lambda i.\, 0)$ *for some* $y : n \to \omega$*, then* $f(x)$ *for all* $x : n \to \omega$*.*

Representation of Local Provenance

For which β is α monotonic?

$$\alpha(u) \stackrel{\Delta}{=} \forall w \forall q \ (\beta(u, w, q) \rightarrow q)$$

Types:

$$\alpha : (n \rightarrow 2) \rightarrow 2$$
$$\beta : ((n \rightarrow 2) \times (m \rightarrow 2) \times 2) \rightarrow 2$$

Representation of Local Provenance

For which β is α monotonic?

$$\alpha(u) \stackrel{\Delta}{=} \forall w \forall q \ (\beta(u,w,q) \rightarrow q)$$
$$\neg\alpha(u) = \exists w \exists q \ (\beta(u,w,q) \wedge \neg q)$$

Types:

$$\alpha : (n \rightarrow 2) \rightarrow 2$$
$$\beta : ((n \rightarrow 2) \times (m \rightarrow 2) \times 2) \rightarrow 2$$

Representation of Local Provenance

For which β is α monotonic?

$$\alpha(u) \stackrel{\Delta}{=} \forall w \forall q \ (\beta(u, w, q) \to q)$$
$$\neg\alpha(u) = \exists w \exists q \ (\beta(u, w, q) \land \neg q)$$

Types:

$$\alpha : (n \to 2) \to 2$$
$$\beta : ((n \to 2) \times (m \to 2) \times 2) \to 2$$

Representation of Local Provenance

For which β is α monotonic?

$$\alpha(u) \stackrel{\Delta}{=} \forall w \forall q \ (\beta(u, w, q) \rightarrow q)$$
$$\neg\alpha(u) = \exists w \exists q \ (\beta(u, w, q) \wedge \neg q)$$
$$\neg\alpha(u) = \exists w \ \beta(u, w, 0)$$

Types:

$$\alpha : (n \rightarrow 2) \rightarrow 2$$
$$\beta : ((n \rightarrow 2) \times (m \rightarrow 2) \times 2) \rightarrow 2$$

Representation of Local Provenance

For which β is α monotonic?

$$\alpha(u) \stackrel{\Delta}{=} \forall w \forall q \ (\beta(u, w, q) \rightarrow q)$$

$$\neg\alpha(u) = \exists w \exists q \ (\beta(u, w, q) \wedge \neg q)$$

$$\neg\alpha(u) = \exists w \ \beta(u, w, 0)$$

$$u(0)u(1)u(2) \ldots u(n-1) = 1011100111001$$

Types:

$$\alpha : (n \rightarrow 2) \rightarrow 2$$

$$\beta : ((n \rightarrow 2) \times (m \rightarrow 2) \times 2) \rightarrow 2$$

Representation of Local Provenance

For which β is α monotonic?

$$\alpha(u) \stackrel{\Delta}{=} \forall w \forall q \ (\beta(u,w,q) \to q)$$
$$\neg\alpha(u) = \exists w \exists q \ (\beta(u,w,q) \wedge \neg q)$$
$$\neg\alpha(u) = \exists w \ \beta(u,w,0)$$
$$u(0)u(1)u(2)\ldots u(n-1) = 1011100011001$$

Types:

$$\alpha : (n \to 2) \to 2$$
$$\beta : ((n \to 2) \times (m \to 2) \times 2) \to 2$$

Representation of Local Provenance

For which β is α monotonic?

$$\begin{aligned} \alpha(u) &\stackrel{\Delta}{=} \forall w \forall q \ (\beta(u,w,q) \rightarrow q) \\ \neg\alpha(u) &= \exists w \exists q \ (\beta(u,w,q) \wedge \neg q) \\ \neg\alpha(u) &= \exists w \ \beta(u,w,0) \end{aligned}$$

Types:

$$\begin{aligned} \alpha &: (n \rightarrow 2) \rightarrow 2 \\ \beta &: ((n \rightarrow 2) \times (m \rightarrow 2) \times 2) \rightarrow 2 \end{aligned}$$

If β is anti-monotonic in u, then α is monotonic.

Example

Let $\beta(u, w, q)$ be the conjunction of

$$\begin{aligned} u(0) &\to w(0) \\ u(1) &\to w(1) \\ w(0) &\to w(1) \\ w(1) &\to w(0) \\ w(0) \wedge w(1) &\to q \end{aligned}$$

Example

Let $\beta(u, w, q)$ be the conjunction of

$$\begin{aligned} u(0) &\to w(0) \\ u(1) &\to w(1) \\ w(0) &\to w(1) \\ w(1) &\to w(0) \\ w(0) \land w(1) &\to q \end{aligned}$$

The models of α are 01, 10, 11. Indeed monotonic.

Refinement

Given constraint sets $\beta_1, \ldots, \beta_k$, representing local provenances $g(x_1) = \alpha_1, \ldots, g(x_k) = \alpha_k$, find an abstraction x for which the prediction lemma together with the anti-monotonicity of f *do not* imply $f(x)$.

Related Work

- **2005** Using Datalog with Binary Decision Diagrams for Program Analysis
- **2009** Strictly Declarative Specification of Sophisticated Points-to Analyses
- **2011** Learning Minimal Abstractions
- **2012** Abstractions from Tests
- **2013** Finding Optimum Abstractions in Parametric Dataflow Analyses
- **2013** MiFuMaX – a Literate MaxSAT Solver
- **2013** Minimal Sets over Monotone Predicates in Boolean Formulae

Where Next?

- Details and experimental results in
  On Abstraction Refinement for Static Analyses in Datalog, PLDI 2014
- Implementation in jChord

It has often been said that a person does not really understand something until after teaching it to someone else. Actually a person does not really understand something until after teaching it to a computer.

Donald Knuth