Program Analysis in Datalog

Majority of the slides compiled from John Whaley's
Outline

- Challenges
- Essential Background
- Using the Tools
- Developing Advanced Analyses
Challenges

Most DF analyses are intraprocedural
- We need whole program analysis
- Recall our efforts of making reaching definition analysis interprocedural

Including context sensitivity is prohibitively expensive

The pain of points-to analysis

Conclusion
- What you have learned only handles toy programs.
With Datalog

56 pages!

vs.

2x faster
Fewer bugs
Extensible
Commercial

People who use it swear by it and could “never go back”
Essential Background
Datalog

- Declarative language for deductive databases [Ullman 1989]
- Like Prolog, but no function symbols, no predefined evaluation strategy
**Datalog Basics**

**Atom** = \text{Reach}(d,x,i)

**Literal** = Atom or NOT Atom

**Rule** = Atom : - Literal \& ... \& Literal

**Predicate**

**Arguments:** variables or constants

Make this atom true (the *head*).

The *body*:
For each assignment of values to variables that makes all these true ...
Datalog Example

\[
\text{parent}(x,y) :\text{child}(y,x).
\]

\[
\text{grandparent}(x,z) :\text{parent}(x,y), \text{parent}(y,z).
\]

\[
\text{ancestor}(x,y) :\text{parent}(x,y).
\]

\[
\text{ancestor}(x,z) :\text{parent}(x,y), \text{ancestor}(y,z).
\]
Datalog

- **Intuition:** subgoals in the body are combined by “and” (strictly speaking: “join”).
- **Intuition:** Multiple rules for a predicate (head) are combined by “or.”
Another Datalog Example

```
hasChild(x) :- child(_,x).
hasNoChild(x) :- !child(_,x).

hasSibling(x) :- child(x,y), child(z,y), z!=x.
onlyChild(x) :- child(x,_), !hasSibling(x).

_ means “Dont-care” (at least one)
! means “Not”
```
Reaching Defs in Datalog

Reach(d,x,j) :- Reach(d,x,i),
    StatementAt(i,s),
    !Assign(s,x),
    Follows(i,j).

Reach(s,x,j) :- StatementAt(i,s),
    Assign(s,x),
    Follows(i,j).
Definition: EDB Vs. IDB Predicates

- Some predicates come from the program, and their tuples are computed by inspection.
  - Called EDB, or extensional database predicates.
- Others are defined by the rules only.
  - Called IDB, or intensional database predicates.
Iterative Algorithm for Datalog

- Start with the EDB predicates = “whatever the code dictates,” and with all IDB predicates empty.
- Repeatedly examine the bodies of the rules, and see what new IDB facts can be discovered from the EDB and existing IDB facts.
Datalog evaluation strategy

“Semi-naïve” evaluation

- Remember that a new fact can be inferred by a rule in a given round only if it uses in the body some fact discovered on the previous round.

Evaluation strategy

- Top-down (goal-directed) [Ullman 1985]
- Bottom-up (infer from base facts) [Ullman 1989]
Negation makes things tricky.

Semantics of negation
- No negation allowed [Ullman 1988]
- Stratified Datalog [Chandra 1985]
- Well-founded semantics [Van Gelder 1991]
Problematic Rules

\[ P(x) :- E(x), \neg P(x). \]

- If \( E(1) \) is true, is \( P(1) \) true?
- It is after the first round.
- But not after the second.
- True after the third, not after the fourth,...
Stratification

A risk occurs if there are negated literals involved in a recursive predicate.

- Leads to oscillation in the result.

Requirement for *stratification*:

- Must be able to order the IDB predicates so that if a rule with $P$ in the head has NOT $Q$ in the body, then $Q$ is either EDB or earlier in the order than $P$. 
Why Datalog?

- Developed a tool to translate inference rules to BDD implementation
- Later, discovered Datalog (Ullman, Reps)
- Semantics of BDDs match Datalog exactly
  - Obvious implementation of relations
  - Operations occur a set-at-a-time
  - Fast set compare, set difference
  - Wealth of literature about semantics, optimization, etc.
Inference Rules

Datalog rules directly correspond to inference rules.
Inference Rules

Datalog rules directly correspond to inference rules.

\[ \text{Assign}(v_1, v_2), \text{vPointsTo}(v_2, o) \]

\[ \text{vPointsTo}(v_1, o) \]

\[ \text{vPointsTo}(v_1, o) :\neg \text{Assign}(v_1, v_2), \text{vPointsTo}(v_2, o). \]
Flow-Insensitive Pointer Analysis

\[
\begin{align*}
&: p = \text{new Object();} \\
&: q = \text{new Object();} \\
&: p.f = q; \\
&: r = p.f;
\end{align*}
\]

**Input Tuples**
- \(vPointsTo(p, o_1)\)
- \(vPointsTo(q, o_2)\)
- \(\text{Store}(p, f, q)\)
- \(\text{Load}(p, f, r)\)

**Output Relations**
- \(hPointsTo(o_1, f, o_2)\)
- \(vPointsTo(r, o_2)\)
Inference Rule in Datalog

Assignments:

\[ \text{vPointsTo}(v_1, o) :\text{- Assign}(v_1, v_2), \]
\[ \text{vPointsTo}(v_2, o). \]

\[ v_1 = v_2; \]
Inference Rule in Datalog

Stores:

\[ h\text{PointsTo}(o_1, f, o_2) \quad \text{:-} \quad \text{Store}(v_1, f, v_2), \]
\[ \quad \text{vPointsTo}(v_1, o_1), \]
\[ \quad \text{vPointsTo}(v_2, o_2). \]

\[ v_1.f = v_2; \]
vPointsTo(v₂, o₂) :- Load(v₁, f, v₂), vPointsTo(v₁, o₁), hPointsTo(o₁, f, o₂).

v₂ = v₁.f;

Loads:
The Whole Algorithm

\[ \text{vPointsTo}(v, o) \quad ::= \quad \text{vPointsTo}_0(v, o). \]

\[ \text{vPointsTo}(v_1, o) \quad ::= \quad \text{Assign}(v_1, v_2), \]
\[ \quad \quad \text{vPointsTo}(v_2, o). \]

\[ \text{hPointsTo}(o_1, f, o_2) \quad ::= \quad \text{Store}(v_1, f, v_2), \]
\[ \quad \quad \text{vPointsTo}(v_1, o_1), \]
\[ \quad \quad \text{vPointsTo}(v_2, o_2). \]

\[ \text{vPointsTo}(v_2, o_2) \quad ::= \quad \text{Load}(v_1, f, v_2), \]
\[ \quad \quad \text{vPointsTo}(v_1, o_1), \]
\[ \quad \quad \text{hPointsTo}(o_1, f, o_2). \]

\( \text{vP}_0(v, h) \) means there is an invocation site \( h \) that assigns a newly allocated object to variable \( v \)