Strictly Declarative Specification of Sophisticated Points-to Analysis

DOOP: NERG FLEEBLE GLORP BARUMPADUM

Shiri / Evan
3 / 8 / 10
Overview

- Efficient points-to analysis in Java
- Purely declarative analysis spec
- Most precise context-sensitive analysis
Context Sensitivity

• Call-site

```java
void f()
{
    Obj i1 = null;
    h(i1);
}

void g()
{
    Obj i2 = new Obj;
    h(i2);
}

void h(Obj i)
{
    Obj j = i;
    j.foo();
}
```
Context Sensitivity

• Call-site

```java
void f()
{
    Obj i1 = null;
    h(i1);
}

void g()
{
    Obj i2 = new Obj;
    h(i2);
}

void h(Obj i)
{
    Obj j = i;
    j.foo();
}
```

• Object

```java
Obj a(null), b(new Int), c(new Int);
a.f();
b.f();
c.f();

class Obj {
    Int i;
    Obj(Int j) {i = j;}
    void f() {print i.intValue();}
}
```
Datalog = Database + Prolog

• logic programming
• implemented with database operations

StudentOwesAssignment(S, X) ← StudentEnrolled(S, C), Assignment(C, X)

<table>
<thead>
<tr>
<th>S</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jack</td>
<td>cs101</td>
</tr>
<tr>
<td>Jill</td>
<td>cs546</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>cs101</td>
<td>tree</td>
</tr>
<tr>
<td>cs321</td>
<td>rbtree</td>
</tr>
<tr>
<td>cs546</td>
<td>dtree</td>
</tr>
</tbody>
</table>
Datalog = Database + Prolog

• logic programming
• implemented with database operations

StudentOwesAssignment(S, X) ← StudentEnrolled(S, C), Assignment(C, X)
Points-to in Datalog

1. VarPointsTo(?var, ?heap) <-
3. VarPointsTo(?to, ?heap) <-
4. Assign(?from, ?to), VarPointsTo(?from, ?heap).

1. T a = new T();
2. T b = new T();
3. T c = a;
4. T d;
5. c = d;
6. d = b;
Points-to in Datalog

1. \texttt{VarPointsTo(?var, ?heap) <-}
2. \texttt{AssignHeapAllocation(?var, ?heap).}
3. \texttt{VarPointsTo(?to, ?heap) <-}
4. \texttt{Assign(?from, ?to), VarPointsTo(?from, ?heap).}

From program

1. \texttt{T a = new T();}
2. \texttt{T b = new T();}
3. \texttt{T c = a;}
4. \texttt{T d;}
5. \texttt{c = d;}
6. \texttt{d = b;}

AssignHeapAllocation(a, alloc#1).
AssignHeapAllocation(b, alloc#2).
Assign(a, c).
Assign(d, c).
Assign(b, d).
Points-to in Datalog

1. `VarPointsTo(\text{?var}, \text{?heap}) \leftarrow`  
2. `AssignHeapAllocation(\text{?var}, \text{?heap}).`  
3. `VarPointsTo(\text{?to}, \text{?heap}) \leftarrow`  
4. `Assign(\text{?from}, \text{?to}), VarPointsTo(\text{?from}, \text{?heap}).`

From program

- `AssignHeapAllocation(a, alloc#1).`  
- `AssignHeapAllocation(b, alloc#2).`  
- `Assign(a, c).`  
- `Assign(d, c).`  
- `Assign(b, d).`

From inference

- `VarPointsTo(a, alloc#1).`  
- `VarPointsTo(b, alloc#2).`
Points-to in Datalog

1. `VarPointsTo(?var, ?heap) <-
   AssignHeapAllocation(?var, ?heap).
2. `VarPointsTo(?to, ?heap) <-
   Assign(?from, ?to), VarPointsTo(?from, ?heap).

From program
- `AssignHeapAllocation(a, alloc#1).
- `AssignHeapAllocation(b, alloc#2).
- `Assign(a, c).
- `Assign(d, c).
- `Assign(b, d).

From inference
- `VarPointsTo(a, alloc#1).
- `VarPointsTo(b, alloc#2).
- `VarPointsTo(c, alloc#1).
Points-to in Datalog

1. \texttt{VarPointsTo(?var, ?heap) \leftarrow}
2. \texttt{AssignHeapAllocation(?var, ?heap).}
3. \texttt{VarPointsTo(?to, ?heap) \leftarrow}
4. \texttt{Assign(?from, ?to), VarPointsTo(?from, ?heap).}

From program
\begin{align*}
1 & \texttt{T a = new T();} \\
2 & \texttt{T b = new T();} \\
3 & \texttt{T c = a;} \\
4 & \texttt{T d;} \\
5 & \texttt{c = d;} \\
6 & \texttt{d = b;} \\
\end{align*}

From inference
\begin{align*}
1 & \texttt{VarPointsTo(a, alloc#1).} \\
2 & \texttt{VarPointsTo(b, alloc#2).} \\
3 & \texttt{VarPointsTo(c, alloc#1).} \\
4 & \texttt{VarPointsTo(d, alloc#2).} \\
\end{align*}
Points-to in Datalog

1. VarPointsTo(?var, ?heap) <-
3. VarPointsTo(?to, ?heap) <-
4. Assign(?from, ?to), VarPointsTo(?from, ?heap).

From program
AssignHeapAllocation(a, alloc#1).
AssignHeapAllocation(b, alloc#2).
Assign(a, c).
Assign(d, c).
Assign(b, d).

From inference
VarPointsTo(a, alloc#1).
VarPointsTo(b, alloc#2).
VarPointsTo(c, alloc#1).
VarPointsTo(d, alloc#2).
VarPointsTo(c, alloc#2).
Points-to details

VarPointsTo(\textit{\texttt{?ctx}}, \textit{\texttt{?var}}, \textit{\texttt{?heap}}) \leftarrow
AssignHeapAllocation(\textit{\texttt{?var}}, \textit{\texttt{?heap}}, \textit{\texttt{?inmethod}}),
CallGraphEdge(\_, \_, \textit{\texttt{?ctx}}, \textit{\texttt{?inmethod}}).

VarPointsTo(\textit{\texttt{?toCtx}}, \textit{\texttt{?to}}, \textit{\texttt{?heap}}) \leftarrow
Assign(\textit{\texttt{?fromCtx}}, \textit{\texttt{?from}}, \textit{\texttt{?toCtx}}, \textit{\texttt{?to}}, \textit{\texttt{?type}}),
VarPointsTo(\textit{\texttt{?fromCtx}}, \textit{\texttt{?from}}, \textit{\texttt{?heap}}),
HeapAllocation::Type[\textit{\texttt{?heap}}] = \textit{\texttt{?heaptype}},
AssignCompatible(\textit{\texttt{?type}}, \textit{\texttt{?heaptype}}).

\begin{itemize}
\item Context sensitivity
\end{itemize}
Points-to details

VarPointsTo(?ctx, ?var, ?heap) <-
   AssignHeapAllocation(?var, ?heap, ?inmethod),
   CallGraphEdge(_, _, ?ctx, ?inmethod).

VarPointsTo(?toCtx, ?to, ?heap) <-
   Assign(?fromCtx, ?from, ?toCtx, ?to, ?type),
   VarPointsTo(?fromCtx, ?from, ?heap),
   HeapAllocation:Type[?heap] = ?heapType,
   AssignCompatible(?type, ?heapType).

• Context sensitivity
• Call graph reachability
Points-to details

VarPointsTo(?ctx, ?var, ?heap) <-
  AssignHeapAllocation(?var, ?heap, ?inmethod),
  CallGraphEdge(_, _, ?ctx, ?inmethod).

VarPointsTo(?toCtx, ?to, ?heap) <-
  Assign(?fromCtx, ?from, ?toCtx, ?to, ?type),
  VarPointsTo(?fromCtx, ?from, ?heap),
  HeapAllocation:Type[?heap] = ?heaptype,
  AssignCompatible(?type, ?heaptype).

• Context sensitivity
• Call graph reachability
• Type checking
void f()
{
    Obj a = new Obj;
    a.g();
}

callCtx = {main}
call = <line 4>
calleeCtx = {f}
callee = Obj::g
base = a
name = 'g'
descriptor = 'void Obj::g()' or some such
heap = alloc#3
heaptype = Obj

main()
{
    f();
}
Semi-Naïve Evaluation

• Standard optimization for Datalog engines
• At each iteration, only run over last step’s changes

Ancestor(A, B) <- Parent(A, C), Ancestor(C, B)
Ancestor(A, B) <- Parent(A, B)

<table>
<thead>
<tr>
<th>Initial facts</th>
<th>Parent</th>
<th>Ancestor</th>
</tr>
</thead>
<tbody>
<tr>
<td>x y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x z</td>
<td></td>
<td></td>
</tr>
<tr>
<td>y w</td>
<td></td>
<td></td>
</tr>
<tr>
<td>w f</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Semi-Naïve Evaluation

- Standard optimization for Datalog engines
- At each iteration, only run over last step’s changes

Ancestor(A, B) <- Parent(A, C), Ancestor(C, B)
Ancestor(A, B) <- Parent(A, B)

<table>
<thead>
<tr>
<th>Parent</th>
<th>Ancestor</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
</tr>
<tr>
<td>x</td>
<td>z</td>
</tr>
<tr>
<td>y</td>
<td>w</td>
</tr>
<tr>
<td>w</td>
<td>f</td>
</tr>
<tr>
<td>x</td>
<td>y</td>
</tr>
<tr>
<td>x</td>
<td>z</td>
</tr>
<tr>
<td>y</td>
<td>w</td>
</tr>
<tr>
<td>w</td>
<td>f</td>
</tr>
</tbody>
</table>
Semi-Naïve Evaluation

• Standard optimization for Datalog engines
• At each iteration, only run over last step's changes

Ancestor(A, B) <- Parent(A, C), Ancestor(C, B)
Ancestor(A, B) <- Parent(A, B)
Semi-Naïve Evaluation

- Standard optimization for Datalog engines
- At each iteration, only run over last step’s changes

Ancestor(A, B) ← Parent(A, C), Ancestor(C, B)
Ancestor(A, B) ← Parent(A, B)
Inference Optimizations

• tuples are indexed based on storage format
• we have the ability to set the storage format
→ we can choose indexes for fast joins
Inference Optimizations

• tuples are indexed based on storage format
• we have the ability to set the storage format
→ we can choose indexes for fast joins

• existing software only allows one index per relation; to add indexes we have to add relations
Join Folding

• standard database technique
• here, gets around software limits on indexes

Before folding

VarPointsTo(?heap, ?var) <-
  AssignHeapAllocation(?heap, ?var).
VarPointsTo(?heap, ?to) <-
  Assign(?to, ?from), VarPointsTo(?heap, ?from).
VarPointsTo(?heap, ?to) <-
  LoadInstanceField(?to, ?signature, ?base),
  VarPointsTo(?baseheap, ?base),
  InstanceFieldPointsTo(?heap, ?signature, ?baseheap).
InstanceFieldPointsTo(?heap, ?signature, ?baseheap) <-
  StoreInstanceField(?from, ?signature, ?base),
  VarPointsTo(?baseheap, ?base),
  VarPointsTo(?heap, ?from).

After folding

VarPointsTo(?heap, ?var) <-
  AssignHeapAllocation(?heap, ?var).
VarPointsTo(?heap, ?to) <-
  Assign(?to, ?from), VarPointsTo(?heap, ?from).
VarPointsTo(?heap, ?to) <-
  LoadInstanceField(?to, ?signature, ?base),
  VarPointsTo(?baseheap, ?base),
  InstanceFieldPointsTo(?heap, ?signature, ?baseheap).
InstanceFieldPointsTo(?heap, ?signature, ?baseheap) <-
  StoreHeapInstanceField(?baseheap, ?signature, ?from),
  VarPointsTo(?heap, ?from).
StoreHeapInstanceField(?baseheap, ?signature, ?from) <-
  StoreInstanceField(?from, ?signature, ?base),
  VarPointsTo(?baseheap, ?base).
Binary Decision Diagrams

(eg PADDLE, bddbddb)

Relation $R = \{<a, b, c, d> \mid (a \land c) \lor (b \land d)\}$

as a relational DB table

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Binary Decision Diagrams

(eg PADDLE, bddbddb)

Relation $R = \{<a, b, c, d> \mid (a \land c) \lor (b \land d)\}$

as a relational DB table

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

as a BDD
Binary Decision Diagrams

(eg PADDLE, bddbbdb)

Relation \( R = \{<a, b, c, d> \mid (a \land c) \lor (b \land d)\} \)

as a relational DB table

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

as a BDD

as a minimal BDD
Summary

- “most precise context-sensitive analyses ever evaluated”
- “order-of-magnitude performance improvements”
- Make use of existing work on relational databases/datalog
Questions

• claim they can separate spec from implementation; what about their optimizations?
• how do their optimizations interact with context sensitivity?
• how much do they gain from using Java?