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

```
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

```
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

```
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)

SOA

S	X

SE

S	C
Jack	cs101
Jill	cs546

A

C	Y
cs101	tree
cs321	rbtree
cs546	dtree

Datalog = Database + Prolog

- logic programming
- implemented with database operations

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

SOA

S	X
Jack	tree
Jill	dtree

SE

S	C
Jack	cs101
Jill	cs546

A

C	Y
cs101	tree
cs321	rbtree
cs546	dtree

Points-to in Datalog

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

```
T a = new T();
T b = new T();
T c = a;
T d;
c = d;
d = b;
```

Points-to in Datalog

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

```
T a = new T();
T b = new T();
T c = a;
T d;
c = d;
d = b;
```

From program

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

Points-to in Datalog

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

```
T a = new T();
T b = new T();
T c = a;
T d;
c = d;
d = b;
```

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

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

```
T a = new T();
T b = new T();
T c = a;
T d;
c = d;
d = b;
```

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

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

```
T a = new T();
T b = new T();
T c = a;
T d;
c = d;
d = b;
```

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).

Points-to in Datalog

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

```
T a = new T();
T b = new T();
T c = a;
T d;
c = d;
d = b;
```

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(?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

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

```
CallGraphEdge(?callerCtx, ?call, ?calleeCtx, ?callee) <-
  VirtualMethodCall:Base[?call] = ?base,
  VirtualMethodCall:SimpleName[?call] = ?name,
  VirtualMethodCall:Descriptor[?call] = ?descriptor,
  VarPointsTo(?callerCtx, ?base, ?heap),
  HeapAllocation:Type[?heap] = ?heaptype,
  MethodLookup[?name, ?descriptor, ?heaptype] = ?callee,
  ?calleeCtx = ?call.
```

```
1  void f()
2  {
3     Obj a = new Obj;
4     a.g();
5  }
6
7  class Obj
8  {
9     void g() {}
10 }
11
12 main()
13 {
14  f();
15 }
```

callerCtx = {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

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)

Parent

Initial facts	x	y
	x	z
	y	w
	w	f

Ancestor

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)

Parent

x	y
x	z
y	w
w	f

Ancestor

x	y
x	z
y	w
w	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)

Parent

x	y
x	z
y	w
w	f

Ancestor

x	y
x	z
y	w
w	f
y	f

Δ

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)

Parent

x	y
x	z
y	w
w	f

Ancestor

x	y	
x	z	
y	w	
w	f	
y	f	Δ
x	f	

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> | (a ∧ c) ∨ (b ∧ d)}

as a relational
DB table

a	b	c	d
1	0	1	0
1	0	1	1
1	1	1	0
1	1	1	1
0	1	0	1
1	1	0	1
0	1	1	1

Binary Decision Diagrams

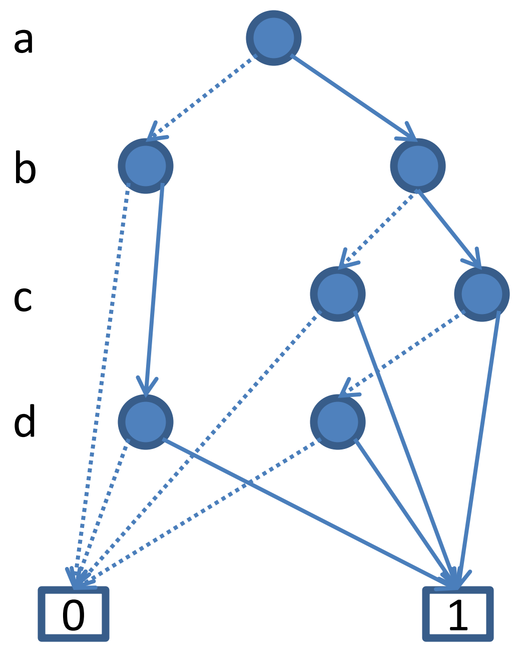
(eg PADDLE, bddbddb)

Relation R = {<a, b, c, d> | (a ∧ c) ∨ (b ∧ d)}

as a relational DB table

a	b	c	d
1	0	1	0
1	0	1	1
1	1	1	0
1	1	1	1
0	1	0	1
1	1	0	1
0	1	1	1

as a BDD



Binary Decision Diagrams

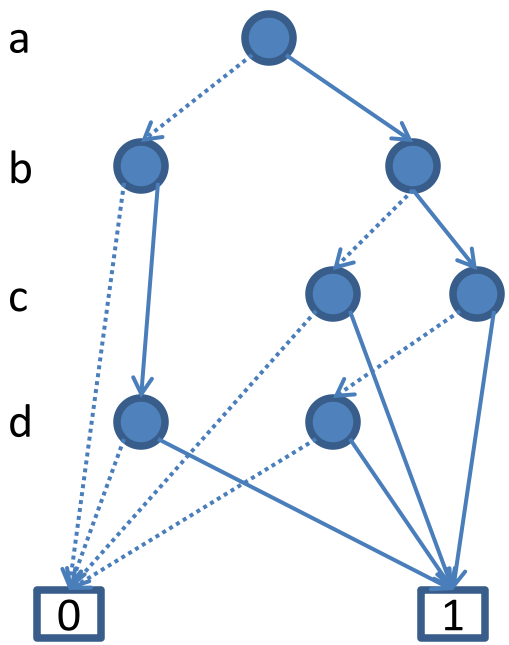
(eg PADDLE, bddbddb)

Relation R = {<a, b, c, d> | (a ∧ c) ∨ (b ∧ d)}

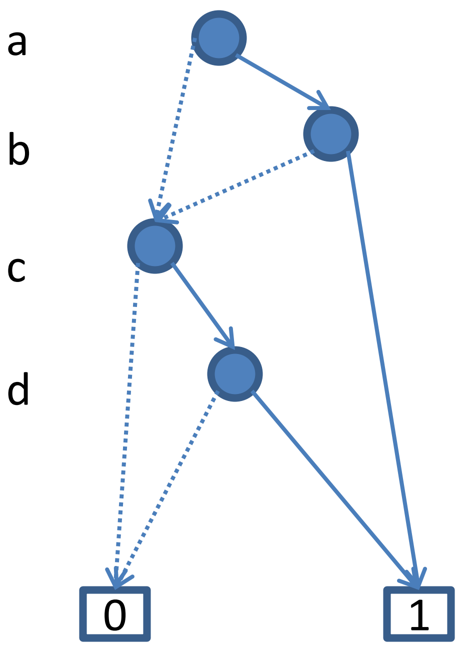
as a relational DB table

a	b	c	d
1	0	1	0
1	0	1	1
1	1	1	0
1	1	1	1
0	1	0	1
1	1	0	1
0	1	1	1

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

FLOW-INSENSITIVE
PATH-INSENSITIVE
CONTEXT-SENSITIVE
FIELD-SENSITIVE
ARRAY-ELEMENT-INSENSITIVE

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?