
ORACLE®

Points-To Analysis: Provenance Generation

Paddy Krishnan
paddy.krishnan@oracle.com

Stepan Sindelar, Paddy Krishnan, Bernhard Scholz, K. R. Raghavendra, Yi Lu
Oracle Labs Australia
2016

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Points-To Analysis

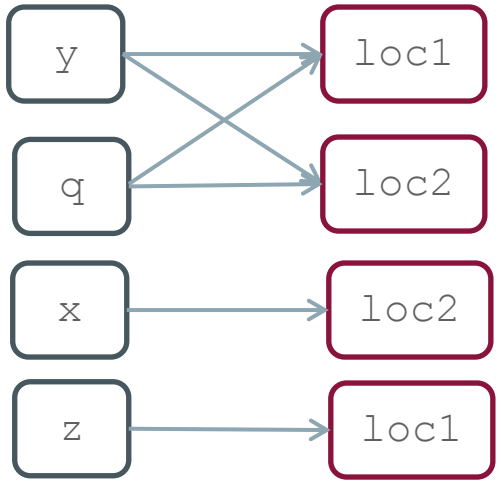
Object-Oriented Languages

- Which variables “may point-to” which objects
  - Andersen: Flow-Insensitive, subset based
  - Steensgaard: Flow-Insensitive, equality based
  - Allocation site abstraction
  - Access-path abstraction
- Basis of other analyses
  - Taint, Escape
- Tutorial by Yannis Smaragdakis

Andersen's Points-to

Example

- Allocation site abstraction




```
public void bar() {
    z = new T();    // loc1
    foo(z);
}

public void baz() {
   x = new T();    // loc2
   foo(x);
}

private void foo (T q) {
     T y = q;
}
```

- Scales for large code bases
  - JDK: 2M variables/500K allocation sites

Points-To Analysis

Implementation

- Declarative specification: Datalog

```
VarPointsTo(x, obj) :- VarPoints(y, obj), Assign(x, y).
```

- Taint Analysis

```
Tainted(x) :- VarPointsTo(x, obj), TaintedObject(obj).
```

- Escape Analysis:

```
Escapes(obj) :- VarPointsTo(x, obj), ReturnVar(x, method),
                Public(method).
```

- DOOP framework from Yannis Smaragdakis
  - Soufflé engine

Points-To Analysis

Implementation

- Propagates/Uses points-to information
  - Reason/Provenance not explicit stored
  - ✓ `VarPointsTo(x,obj)`
  - ✗ `Assign(x,y), Assign(y,z), "z = new obj()"`

- Infeasible to explicitly store all provenance information
  - Preliminary Investigation: Each tuple requires about 50 more tuples for provenance

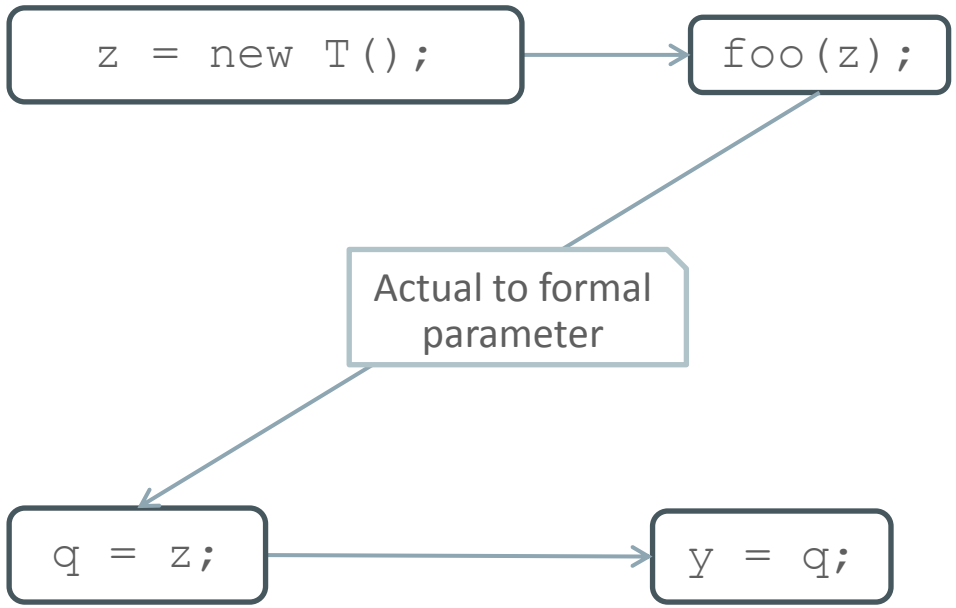
Points-To Analysis

Usability Issues/Challenges

- Debugging: Classify results as True Positives or False positives
  - Why does variable x point to object o?
- Visualisation
  - Show me a taint/escape trace?
- Requires *provenance information*
- Challenge: Provenance generation for large code-bases
  - At least 1 million variables and 300 thousand allocation sites
  - Motivated by various subsets of the JDK

Provenance

Example




```
public void bar() {
    z = new T();    // loc1
    foo(z);
}

public void baz() {
   x = new T();    // loc2
   foo(x);
}

private void foo (T q) {
     T y = q;
}
```

Problem Statement

- Given
  - Results of context sensitive points-to analysis
  - **Client** specified queries: Variable `var` points to location `loc`
- Find "*all provenance traces*" for each of the queries *using pre-computed results*
  - May points-to: All traces must be infeasible for report to be a False Positive
- Resource limits
  - Typical clients: Provenance generation less than 10% of points-to analysis

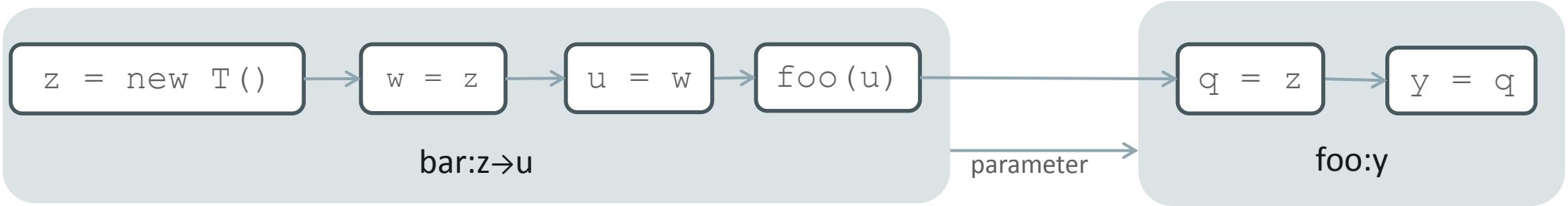
Solution: Abstract Actual Trace

Interprocedural Flow

- Track only flows between method boundaries

```
public void bar() {
   z = new T();
   w = z;
   u = w;
   foo(u);
}
```

```
private void foo(T q)
{
   T y = q;
}
```



Solution: Abstract Actual Trace

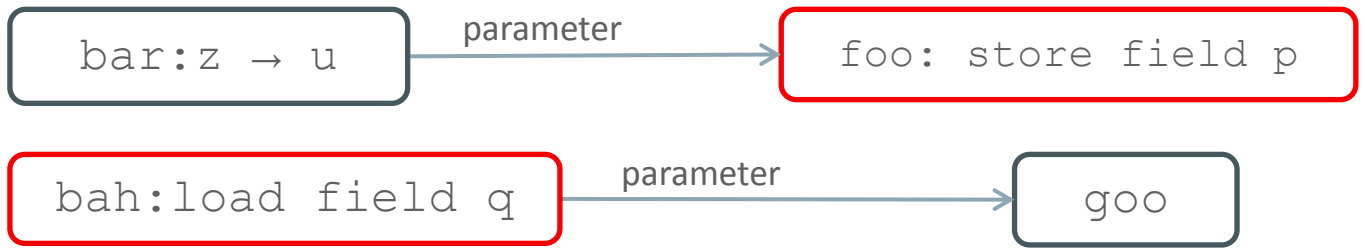
Dataflow via Fields: Alias of base objects

- Load/Store also between method boundaries

```
public void bar() {
   z = new T();
   w = z;
   u = w;
   foo(u);
}
```

```
private void foo(T p){
   field = p;
}
```

```
public void bah(){
   T q = field;
   goo(q);
}
```



Dataflow via Fields: Alias

Depth Limitation

- Alias of base variables can depend on alias of other variables
- Cascading alias
  - Expensive to compute
  - Hard to visualise
  - Increases FPs
- Solution:
  - Limit depth to 1 or 2
  - Sufficient for most traces

Reusing Information

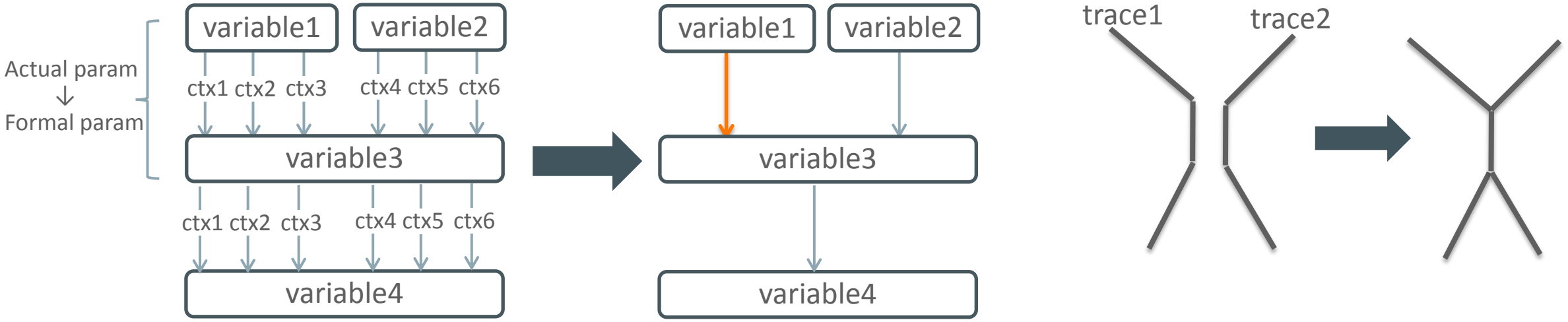
- Method boundaries: Identified using computed call-graph edge
- Load/Store pairs: Alias computed using points-to information
- Provenance: Identifies only such methods/variables/fields
- Optimisation: Remove irrelevant methods
  - Methods that return parameter

```
public T check(T v) {
      if (v != null) return v;
      throw new Exception("error");
}
```

Context Sensitivity

- Context in traces: Not scalable
- Solution: Contexts not part of the trace; but used to construct trace

$\exists ctx, ctx', hctx: (hctx, loc, ctx, variable1) \in VP \;\&\; (hctx, loc, ctx', variable2) \in VP$ & CallGraphEdge(ctx, …, ctx', …)



Results

Subsets of Various Versions of the JDK

Code-Base	VarPointsTo Size (tuples)	Client	#Client Queries	Provenance Graph			Time Overhead	Memory Overhead
				Nodes	Edges	Max Degree		
Subset Version-1	539Million	Taint	836	4287	8538	112	25%	3.4%
		Escape	10	13	14	2	1%	9.0%
Subset Version-2	871Million	Taint	900	4563	6774	105	23%	2.2%
		Escape	445	1115	6663	33	3.0%	8.8%

Results: Alias Graph

Context-Sensitive Points-To Analysis: Depth 1

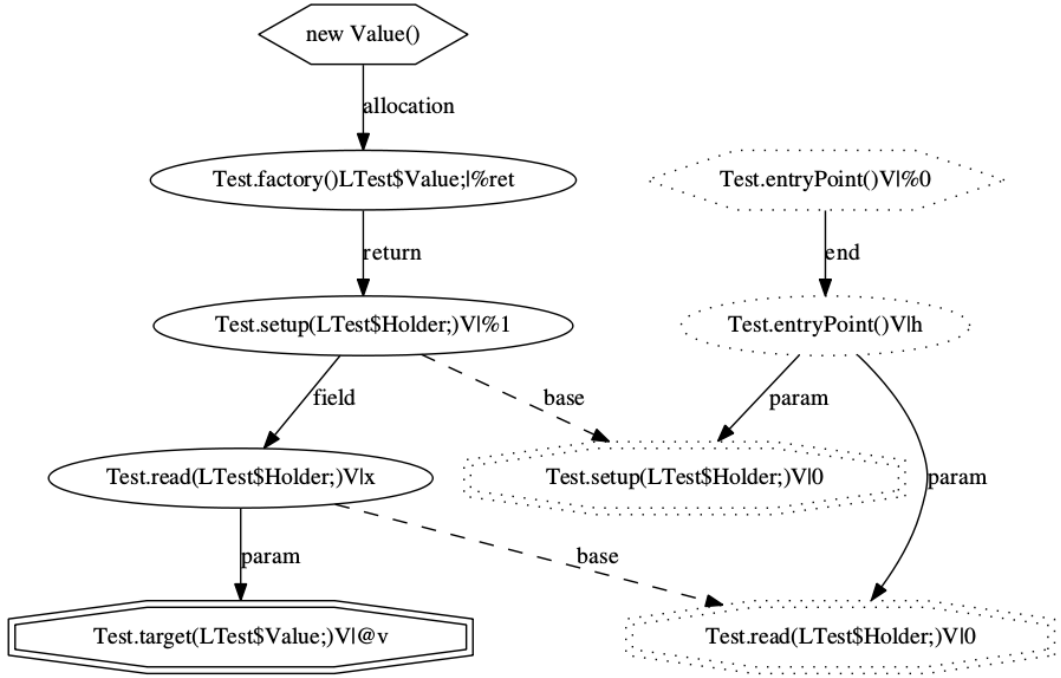
Code-Base	Client	Alias Graph		
		Nodes	Edges	Max Degree
Subset Version-1	Taint	23,791	98,245	136
	Escape	0	0	0
Subset Version-2	Taint	12,484	16,443	35
	Escape	7,588	28,180	37

Presentation

- Graphviz
- Parfait: Internal format
- Shortest path
- Acyclic path
- Time/Output: Depends on size provenance graph
  - Taint analysis: Computing acyclic paths times-out

Graphviz: Example

```
public class Test {
  public static class Value {}
  public static class Holder {
    public Value v;
  }

  public static void entryPoint() {
    Holder h = new Holder();
    setup(h);
    read(h);
  }

  private static Value factory() {
    return new Value(); }

  private static void setup(Holder h) {
    h.v = factory(); }

  private static void read(Holder h) {
    Value x = forward(h.v);
    target(x); }

  private static void target(Value v) { ... }

  private static Value forward(Value v) {
    return v; }
}
```



Conclusion

- Provenance using sequence of method invocations
  - Limited control flow paths
- Reuse points-to for call-graph and alias analysis
- Scales for JDK
  - Time threshold of 10% not met: Further optimisations required
- Challenges
  - Support flow-sensitive analysis
  - Dynamic Traces: Querying mechanism of provenance information
  - Better UI for trace representation

Questions

Points-To Analysis: Provenance Generation

Contact paddy.krishnan@oracle.com for more details

Integrated Cloud

Applications & Platform Services


ORACLE®