
ORACLE®

An Efficient Tunable Selective Points-to Analysis for Large Codebases

Behnaz Hassanshahi, Raghavendra Kagalavadi Ramesh, Padmanabhan Krishnan, Bernhard Scholz and Yi Lu

Oracle Labs, Australia

SOAP 2017

Safe Harbor Statement

The following is intended to provide some insight into a line of research in Oracle Labs. 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. Oracle reserves the right to alter its development plans and practices at any time, and the development, release, and timing of any features or functionality described in connection with any Oracle product or service remains at the sole discretion of Oracle. Any views expressed in this presentation are my own and do not necessarily reflect the views of Oracle.

Objective

- Scaling object-sensitive points-to analysis with high precision
- A tuneable analysis framework
  - Utilizing the knowledge of the codebase
- Client Independent

Background

Points-to Analysis

- Variables → Objects
- Prerequisite to many analyses
- Andersen points-to implemented in Datalog
  - Flow-insensitive, inclusion-based
  - Heap-allocated objects
    - Creation-site as an abstraction for dynamically created objects
    - Heap-allocated object have fields

Background

Context-sensitivity

- Context insensitive (CI) points-to is imprecise
- Different flavors: type, callsite, objects
- 2O1H

2O1H

```
public class A{
    void foo(){                          //CalleeCtx =[o1,o2]
            o3: B b = new B();           // HeapCtx = [o2]
            b.bar();
    }
}
public class B{
    void bar(){                          //CalleeCtx = [o2,o3]
            o4: C c = new C();           // HeapCtx = [o3]
    }
}
```

Motivation

OpenJDK-b147

- 2O1H improves precision vs CI points-to
  - removes 97% of FP in PointsTo
  - removes 49% of FP call-graph edges
- High resource consumption
  - 6.8 hours execution time
- Only particular parts of the codebase are problematic
  - OpenJDK7 minus javax.swing
    - 8.8x faster
    - 45% memory reduction

Problem Statement

- Is it possible to selectively generate contexts?
- Which heap allocation sites need different context sensitivity?
- How deep should the contexts be?

State-of-Art in client-independent Points-to for Java

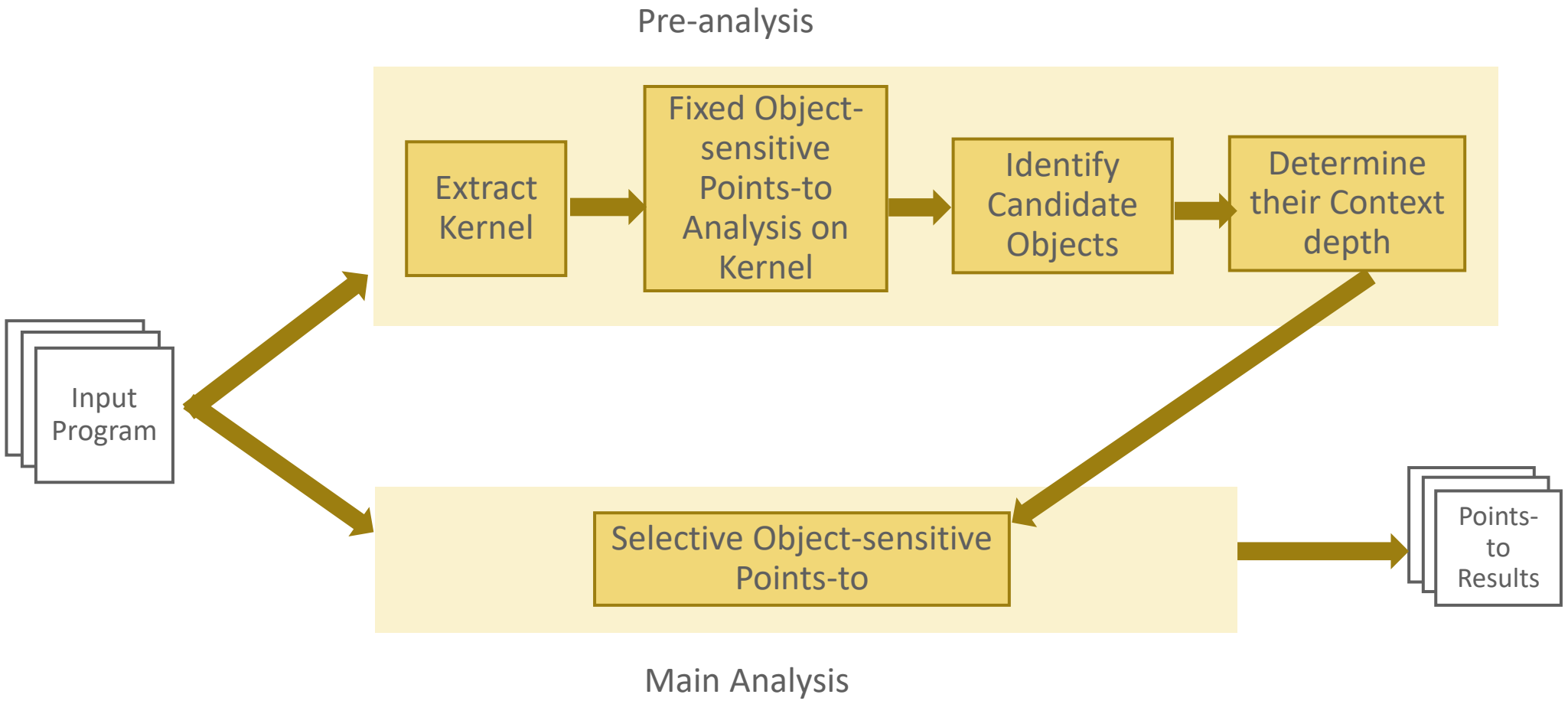
- Introspective context-sensitive points-to (Yannis et el., PLDI 2014)
  - Proposes two heuristics: HA, HB
  - Imprecise w.r.t. 2O1H
  - Case study: OpenJDK

Approach	Runtime	# VarPointsTo	#CallGraphEdge
2Obj+1H	6.8h	10M	261K
Context-insensitive	6.8m	318M	513K
HA	14m	206M	496K
HB	timeout	-	-

Our Contributions

- Identifying the *kernel*: the core problematic subset of input program
  - Based on CI points-to results
- Designing metrics on *kernel* for:
  - Evaluating effectiveness of generated contexts
  - Identifying candidate heap allocation sites
  - Identifying depth of the context needed
- A selective object-sensitive points-to
  - Going beyond 2O1H
- A general client independent framework

Tunable Selective Object-Sensitive Points-to Workflow



Automatically Extracting Kernel

Parameterized by P1, P2

1. A cheap context insensitive points-to
2. Kernel = Input program - RHA
3. RHA has heap allocation site o
   - where # PointedByVar(o) < P1 AND
   - which is NOT allocated in a method of class C where
     - o' has type C with #PointedByVar(o') >= P1 OR
     - C has method m with #MethodPointsTo(m) > P2

Analyzing Kernel

- 2O1H points-to on kernel
- Applying metrics to evaluate the effectiveness of contexts
- Finding candidate heap allocation sites
- Choosing context depth

Context Evaluation

Key insights

- False positives in 2O1H can be more expensive than in CI points-to
- Sources of imprecision
  - Certain program elements are handled by *smashing techniques* [1]
    - E.g., arrays, static fields, open world, exceptions, etc.
    - Due to array smashing, elements of the array are not distinguished
- Inappropriate contexts for these elements compounds *smashing effect*
  - Arrays cause 88% of VarPointsTo tuples in CS Points-to
  - Static fields cause 50% of VarPointsTo tuples in CS Points-to

1. B. Blanchet et al., Design and implementation of a special-purpose static program analyzer for safety-critical real-time embedded software. In The Essence of Computation: Complexity, Analysis, Transformation., pages 85–108. Springer-Verlag, 2002.

Context Evaluation

Metrics

- Given a program element *o* with smashing effect (e.g., array) in context *oc*

$$ContextValue(o, oc) = \min_{v} \frac{|InFlow(o)| \cdot |CtxInOutFlow(v, o, oc)|}{|OutFlow(v, o, oc)|}$$

- **InFlow:** objects of all contexts that are stored in *o* qualified by any context
- **OutFlow:** objects that v points to where v is loaded from *o*, with context *oc*
- **CtxInOutFlow**: contexts in which v is loaded from *o*, with context *oc*

Context Evaluation

Metrics

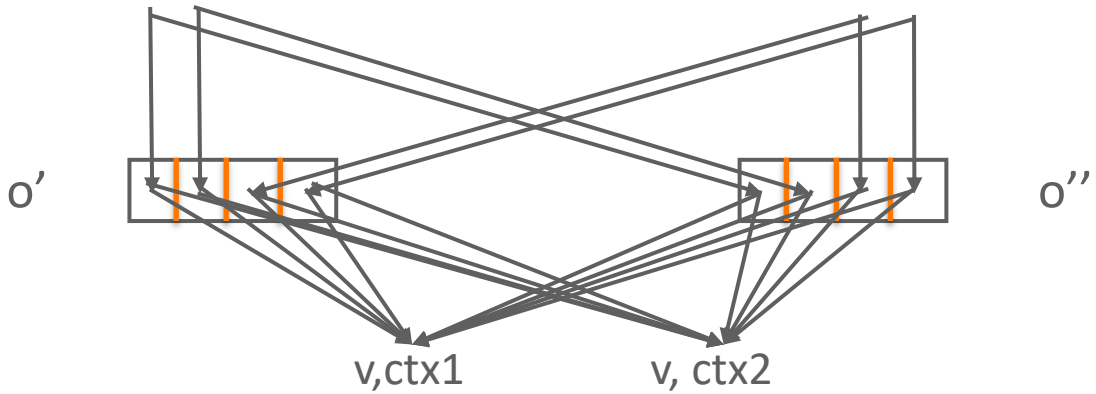
- Given a program element *o* with smashing effect (e.g., array) in context *oc*

$$ContextValue(o, oc) = \min_{v} \frac{|InFlow(o)| \cdot |CtxInOutFlow(v, o, oc)|}{|OutFlow(v, o, oc)|}$$

o: a = new … //oc = [o’]

a[i] = x

v = a[i]

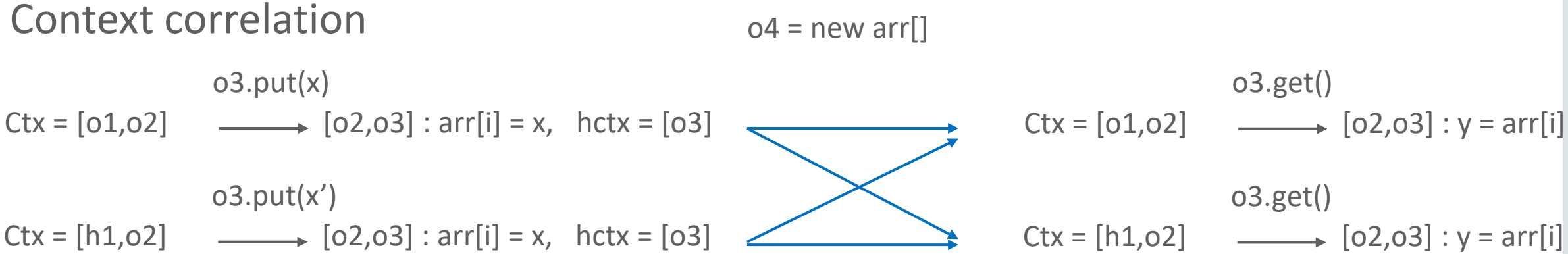


InFlow(o) = 4
CtxInOutFlow(v,o,o’) = 2
OutFlow(v,o,o’) = 8
=>
ContextValue(o,o’) = 1

Selective Context generation

General Algorithm

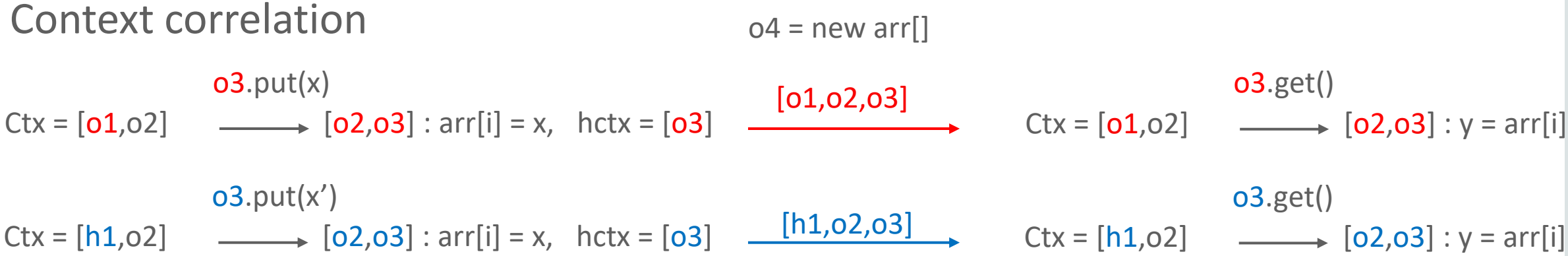
- Combining a subset of the metrics to:
  - Identifying candidates
  - Choosing context depth based on context correlation
- Context correlation



Selective Context generation

General Algorithm

- Combining a subset of the metrics to:
  - Identifying candidates
  - Choosing context depth based on context correlation
- Context correlation



Selective Context Generation

Metrics Parametrized by P3 and P4

- Given *o* with context *oc* = [*o*']
- If ContextValue(*o*,*oc*) < P3 AND InFlow > P4
  - If extending oc results in context correlation
  - Else remove the context for *o*'.

Implementation

- Implementing
  - Datalog specification for DOOP points-to analysis
- Use Souffle as the Datalog engine
- SQLite relational database to compute the pre-processing metrics

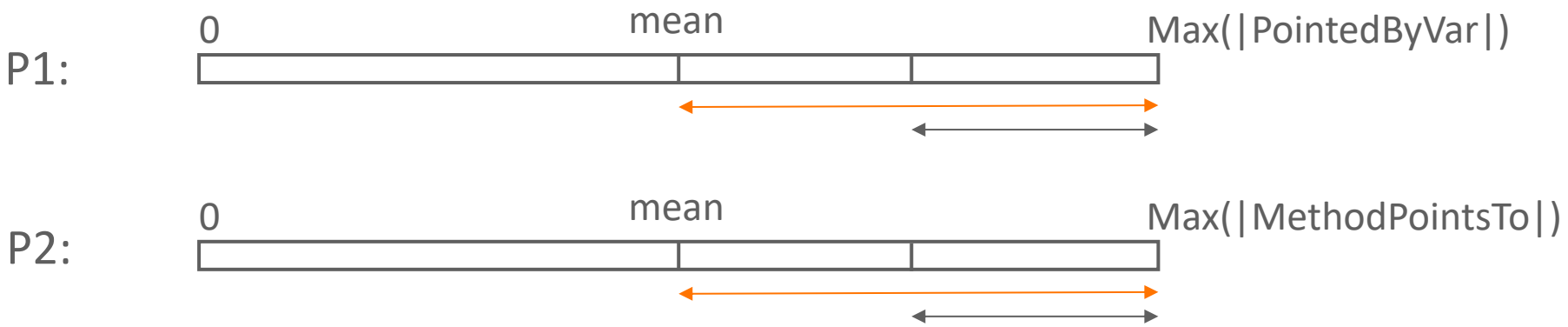
Evaluation

- Experimental setup:
  - Xeon E5-2699 2.30GHz, 8 threads running at a time, 396G RAM
  - P3 and P4 parameters are both 200
  - OpenJDK7-b147 and DaCapo 2006-10-MR2
    - No context refinement needed for objects of programs in DaCapo benchmarks, except for Jython

Benchmarks	#Variables	#Call-sites	#Heap-allocations
OpenJDK7	1440875	591262	185352
Jython	142641	59379	25608

Identifying the Kernel

Computing P1 and P2



Program	P1	P2
OpenJDK-b147	20K	50K
Jython	2149	1452

Evaluation

Performance

Benchmarks	Runtime (seconds)			Memory (GB)		
	2O1H	Ours	Imp.	2O1H	Ours	Imp.
OpenJDK	16200	11880	27%	186	153	18%
Jython	1280	109	91%	2.8	2.8	0%

Evaluation

Precision

Benchmarks	#VarPointsTo			#Alias			#ReachableMethods		
	2O1H	Ours	Loss	2O1H	Ours	Loss.	2O1H	Ours	Loss
OpenJDK	10100000	10400000	3%	845518	854600	1%	39909	39880	-0.1%
Jython	68519	68519	0%	28294	28294	0%	2876	2876	0%

Conclusion

- Designed a novel framework for scaling object-sensitive points-to analysis for large object-oriented programs
  - Involves identifying and experimenting on the kernel of program
  - Selective object-sensitive points-to analysis on the input program
- Client-independent
- Reduced the runtime of CS points-to with negligible precision loss

Questions/Suggestions

E-mail: behnaz.hassanshahi@oracle.com

Integrated Cloud

Applications & Platform Services