Context Debloating for Object-Sensitive Pointer Analysis

Dongjie He, Jingbo Lu, and Jingling Xue

UNSW
SYDNEY

A new
Pointer Analysis Technique
for
Object-Oriented Programs

Pointer Analysis

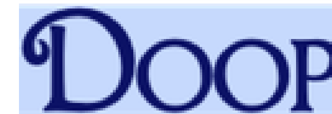
- Statically determines

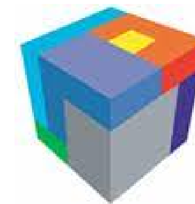
“possible runtime values of a variable?”

Uses of Pointer Analysis

- Foundation of many clients
  - Call-graph construction
  - Security analysis
  - Bug detection
  - Compiler optimization
  - Program understanding
  - …

- Many tools available





A precise and efficient pointer analysis benefits all above clients & tools.

Context Sensitivity

- One of the most successful techniques in developing highly precise pointer analysis for OO programs

- Distinguish variables/objects in a method by different calling contexts

Context Sensitivity

- Call-site Sensitivity (*k*CFA)
- Object Sensitivity (kOBJ)
- Type Sensitivity (kType)
- …

Arguably the best context abstraction for OO programs

Motivating Example

A JDK class, i.e. HashSet

```
1. class Set {
2.     Object f;
3.     void add(Object o) {
4.         this.f = o;
5.     }
6.     Object get() {
7.         return this.f;
8.     }
9. }
```

A third-party Library class

```
10. class Lib {
11.     Set g;
12.     Lib { this.g = new Set(); // S }
13.     void API1() {
14.         Object o1 = new Object(); // O1
15.         Set s1 = this.g;
16.         s1.add(o1);
17.         Object v1 = s1.get();
18.     }
19.     void API2() {
20.         Object o2 = new Object(); // O2
21.         Set s2 = this.g;
22.         s2.add(o2);
23.         Object v2 = s2.get();
24.     }
25. }
```

A unit test

```
26. void testLib() {
27.     Lib l1 = new Lib(); // L1
28.     Lib l2 = new Lib(); // L2
29.     l1.API1();
30.     l2.API2();
31. }
```

Motivating Example: Andersen's Analysis

A JDK class, i.e. HashSet

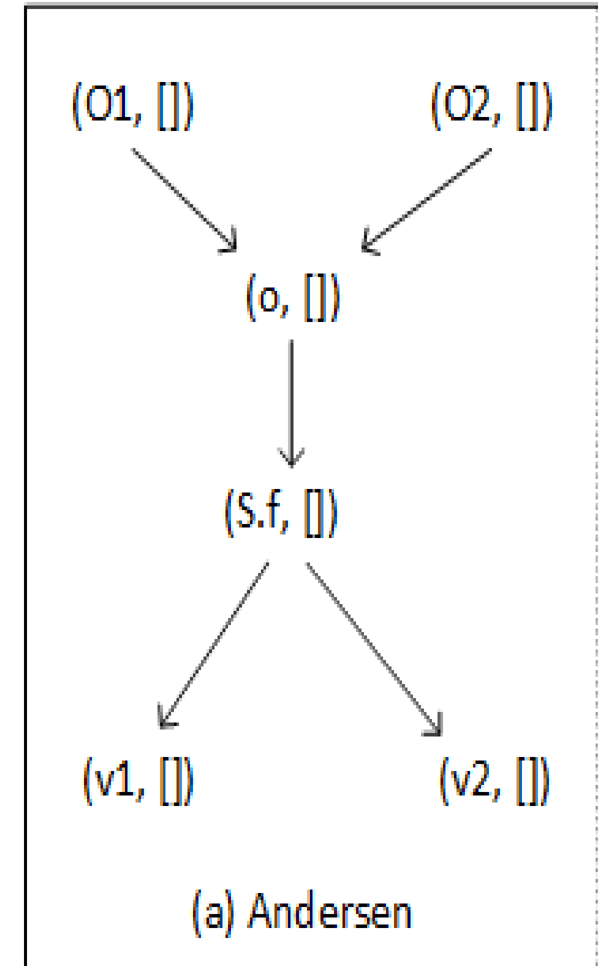
```
1. class Set {
2.     Object f;
3.     void add(Object o) {
4.         this.f = o;
5.     }
6.     Object get() {
7.         return this.f;
8.     }
9. }
```

A unit test

```
26. void testLib() {
27.     Lib l1 = new Lib(); // L1
28.     Lib l2 = new Lib(); // L2
29.     l1.API1();
30.     l2.API2();
31. }
```

A third-party Library class

```
10. class Lib {
11.     Set g;
12.     Lib { this.g = new Set(); // S }
13.     void API1() {
14.         Object o1 = new Object(); // O1
15.         Set s1 = this.g;
16.         s1.add(o1);
17.         Object v1 = s1.get();
18.     }
19.     void API2() {
20.         Object o2 = new Object(); // O2
21.         Set s2 = this.g;
22.         s2.add(o2);
23.         Object v2 = s2.get();
24.     }
25. }
```



(a) Andersen

Motivating Example: Object Sensitivity

A JDK class, i.e. HashSet

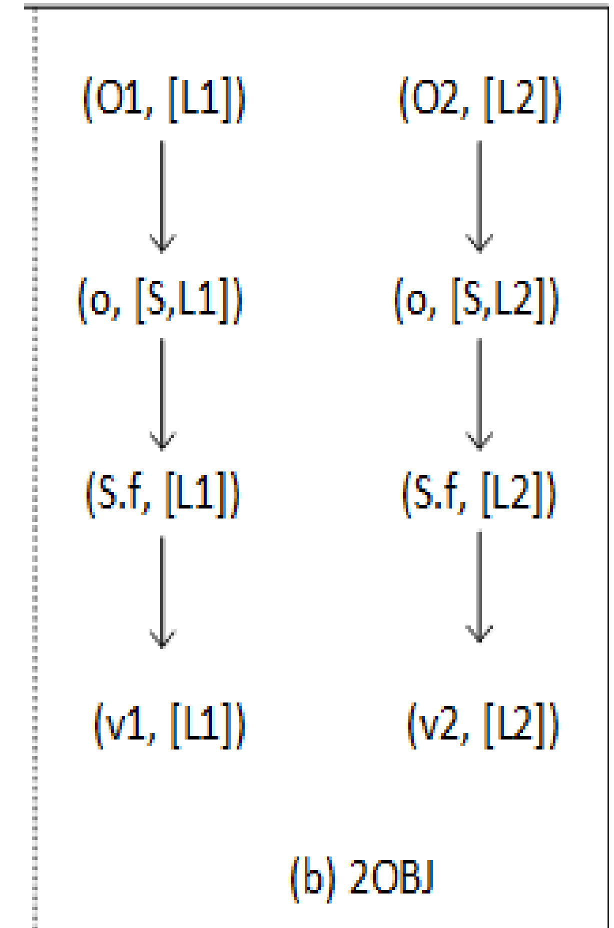
```
1. class Set {
2.     Object f;
3.     void add(Object o) {
4.         this.f = o;
5.     }
6.     Object get() {
7.         return this.f;
8.     }
9. }
```

A third-party Library class

```
10. class Lib {
11.     Set g;
12.     Lib { this.g = new Set(); // S }
13.     void API1() {
14.         Object o1 = new Object(); // O1
15.         Set s1 = this.g;
16.         s1.add(o1);
17.         Object v1 = s1.get();
18.     }
19.     void API2() {
20.         Object o2 = new Object(); // O2
21.         Set s2 = this.g;
22.         s2.add(o2);
23.         Object v2 = s2.get();
24.     }
25. }
```

A unit test

```
26. void testLib() {
27.     Lib l1 = new Lib(); // L1
28.     Lib l2 = new Lib(); // L2
29.     l1.API1();
30.     l2.API2();
31. }
```



(b) 2OBJ

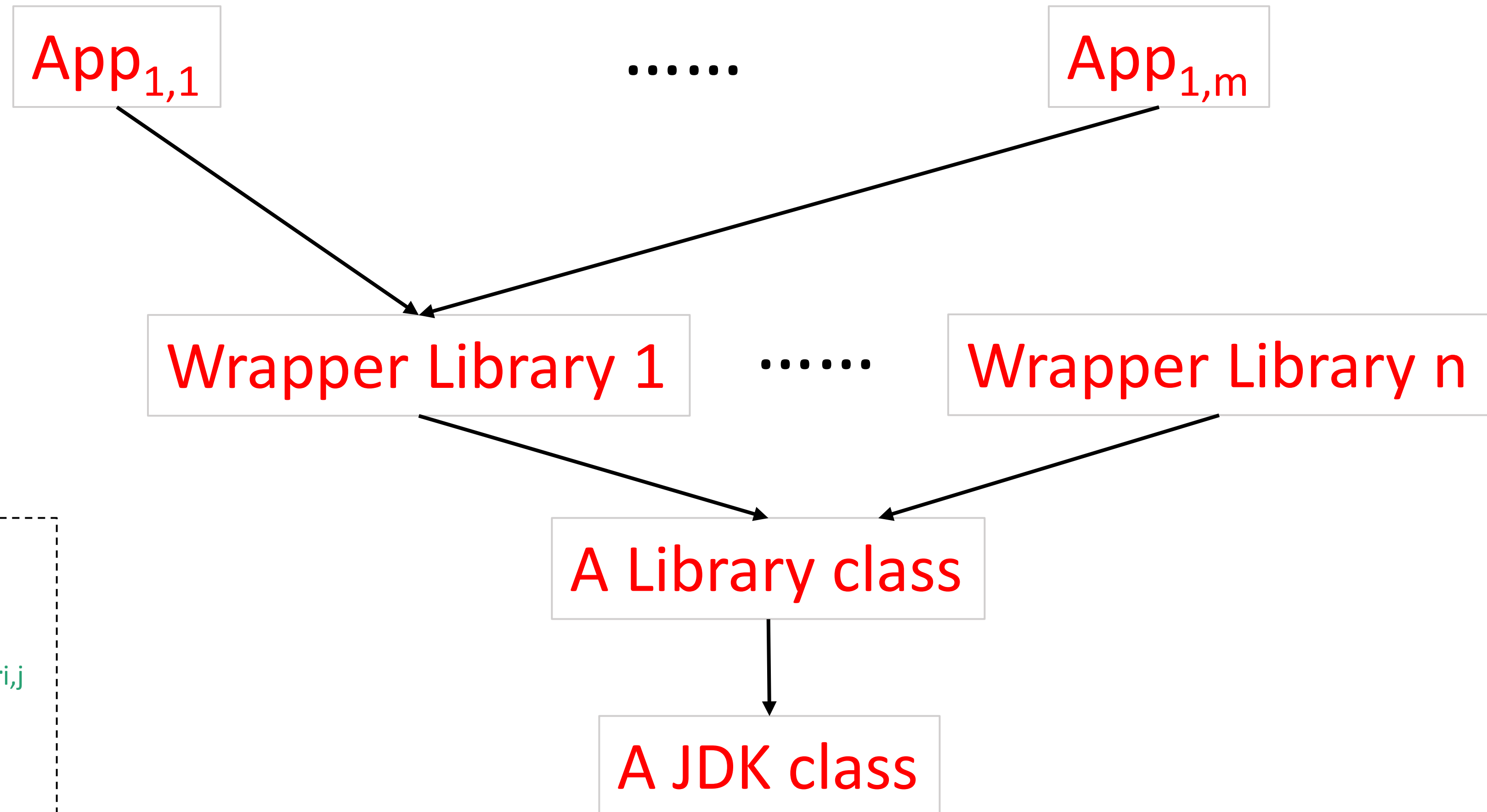
Context Explosion

Wrapper Library i:

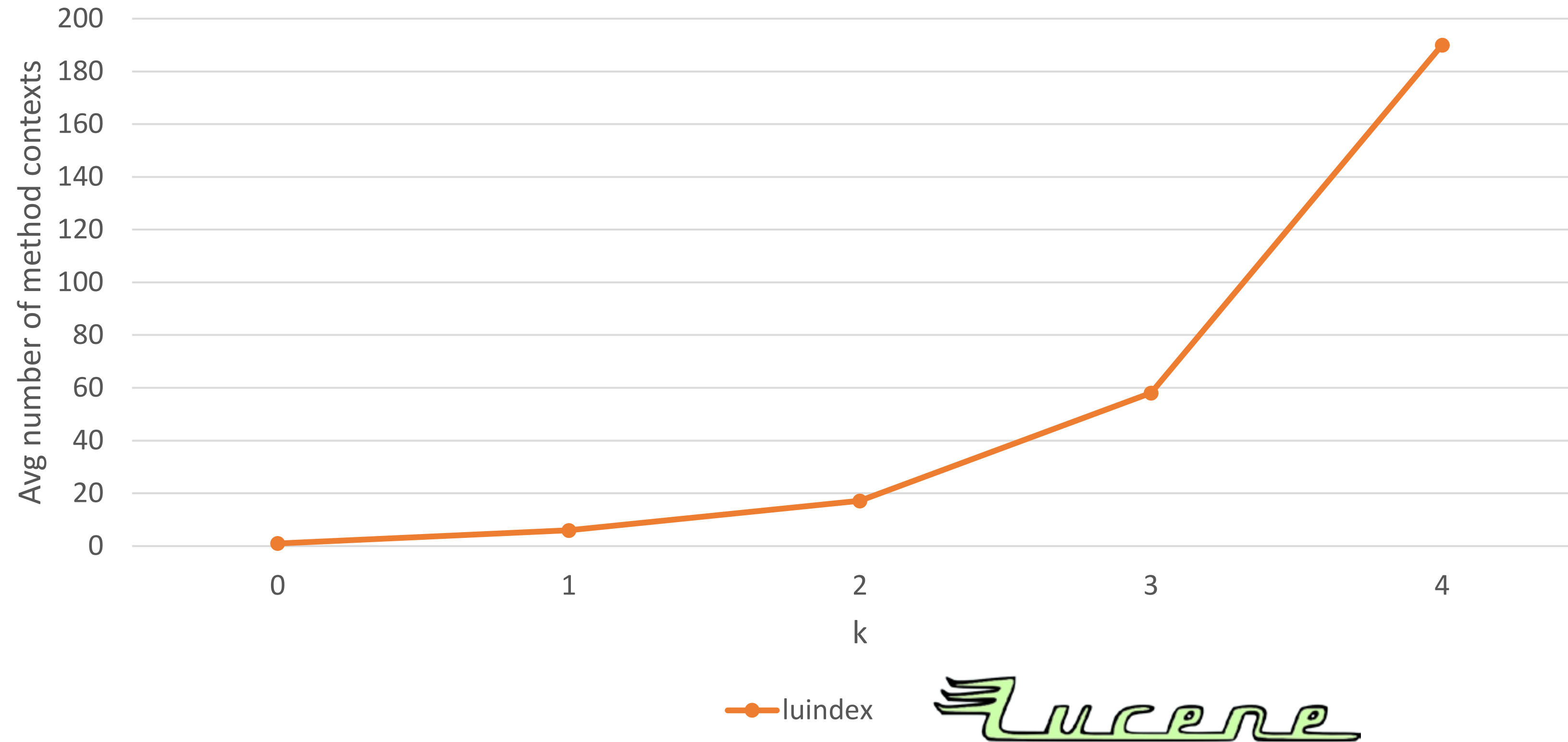
```
class WrapLib_i {
   void API_WL_i() {
      Lib l_i = new Lib(); // L_i
      l_i.API1();
   }
}
```

Appi,j:

```
class App_i,j {
   void main() {
      WrapLib_i wl_i,j = new WrapLib_i(); // WL_i,j
      wl_i,j. API_WL_i();
   }
}
```

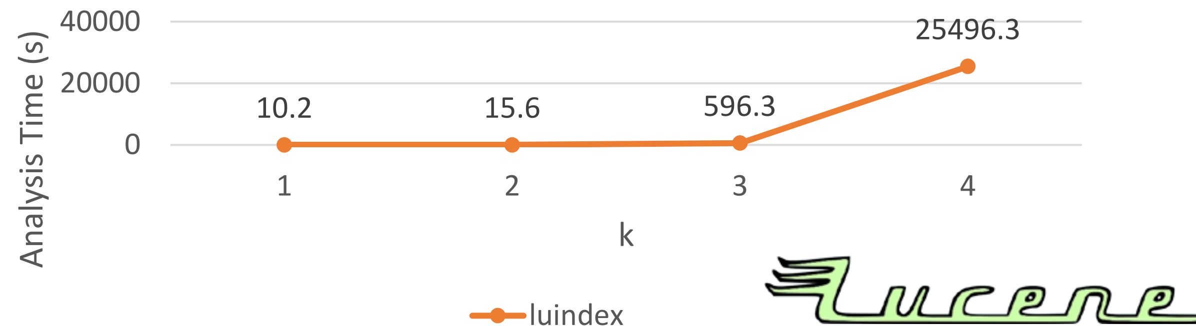


Contexts Explosion

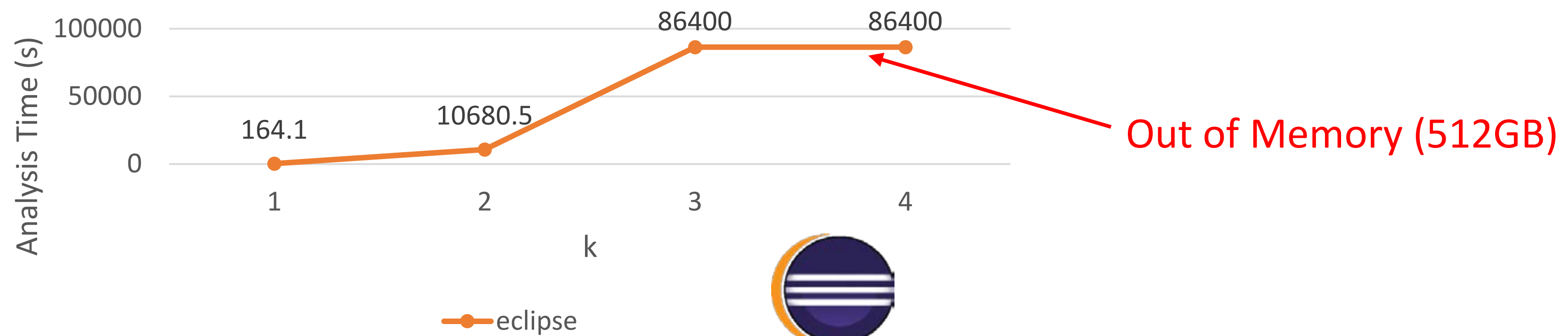


Problem: Inefficient & Unscalable

- Object Sensitivity (*k*OBJ) becomes inefficient as *k* increase



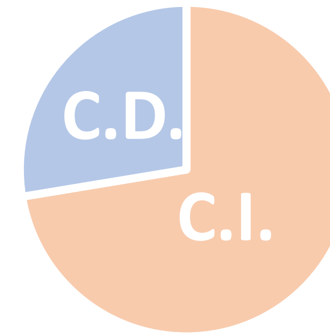
- *k*OBJ is often unscalable for reasonable large programs.



Context Debloating

Key Insights

- ❑ 70% + objects are context-independent (C.I.).
- ❑ Only S is context-dependent (C.D.).
  - ✓ n contexts is enough for Set:add()



WL$_{1,1}$ …… WL$_{1,m}$ ……

L$_1$ …… L$_n$

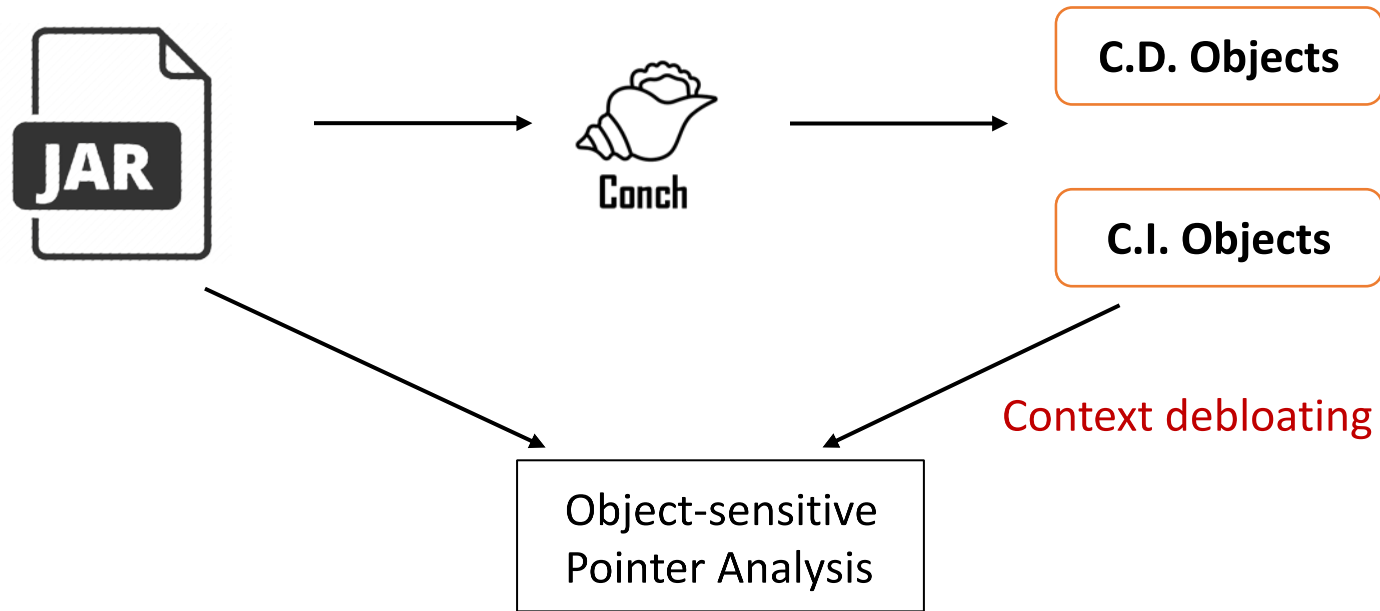
S

WL$_{1,1}$ …… WL$_{1,m}$ ……

L$_1$ …… L$_n$

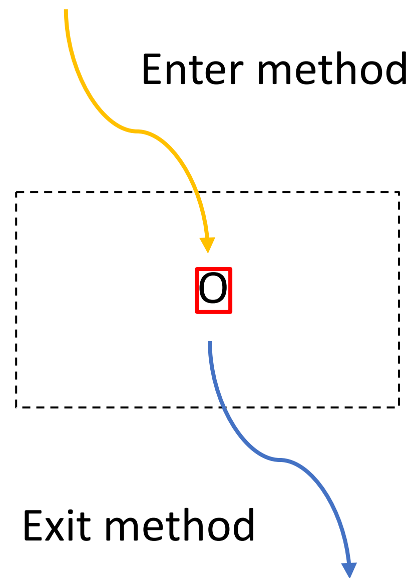
S

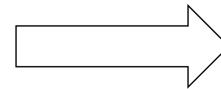
Workflow of Context Debloating



Challenge

- How to verify an object is C.D. or C.I. ?
  - ✓ Precise verification is undecidable





Undecidable problem

$$L_{FC} = L_F \bigcap L_C$$ [1][2][3]

L_F: field sensitivity

L_C: context sensitivity

[1] Manu Sridharan and Rastislav Bodík. Refinement-based context-sensitive points-to analysis for Java. In PLDI 2006.
[2] Jingbo Lu and Jingling Xue. Precision-Preserving Yet Fast Object-Sensitive Pointer Analysis with Partial Context Sensitivity. In OOPSLA 2019.
[3] Jingbo Lu, Dongjie He and Jingling Xue. Eagle: CFL-Reachability-based Precision-Preserving Acceleration of Object-Sensitive Pointer Analysis with Partial Context Sensitivity. In TOSEM 2021.

Our Solution: Conch

- Based on three key observations governing how objects are used in real code.

Our Solution: Conch

- ❑ Observation 1:

A context-dependent object O often has at least one instance field O.f that is both written into (x.f = …) and read from (… = x.f), where O ∈ pts(x).

```
class Set {
   Object f;
   void add(Object o) {
      this.f = o; // x.f = …
   }
   Object get() {
      return this.f; // … = x.f
   }
}
```

```
class Lib {
   Set g;
   Lib { this.g = new Set(); // O }
   …
}
```

Our Solution: Conch

- Observation 2:

A context-dependent object O, pointed by a variable or a field of some object, usually flows out of its containing method (i.e. the method where O is allocated).

```
Vector (int size) {
   this.elems = new Object[size]; // O
}
```

Case 1: from Vector

```
Iterator iterator() {
   return new KeyIterator(); // O
}
```

Case 2: from HashMap

```
void SunJCE_e_a(…) {
   BufferedReader br = new BufferedReader(); // O
   this.f = new StreamTokenizer(br);
}
```

Case 3: from SunJCE_e

Our Solution: Conch

❑ Observation 3:

A context-dependent object O tends to have a store statement x.f = y in a method m_1, where O $\in$pts(x). Let m be the method where O is allocated if m_1 is an O's constructor method and m_1 otherwise. Then y (a) is data-dependent on a parameter of m or (b) points to a context-dependent object.

```
ArrayList () {
    this.elems = new Object[5]; // O
}
void set(int idx, E e) {
    this.elems[idx] = e; // x.f = y
}
```

Case 1: m is set()

```
void addEntry(int idx, K k, V v) {
    this.table[idx] = new Entry(k, v); // O
}
Entry (K k, V v) {
    this.key = k; this.value = v; // x.f = y
}
```

Case 2: m is addEntry()

```
HashSet () {
    this.map = new HashMap(); // O
}
HashMap() {
    this.table = new Entry[10]; // x.f = y
}
```

Case 3: m is HashSet()

Our Solution: Conch

- ❑ Observations are linear verifiable.
  - ✓ Efficient
  - ✓ Effective

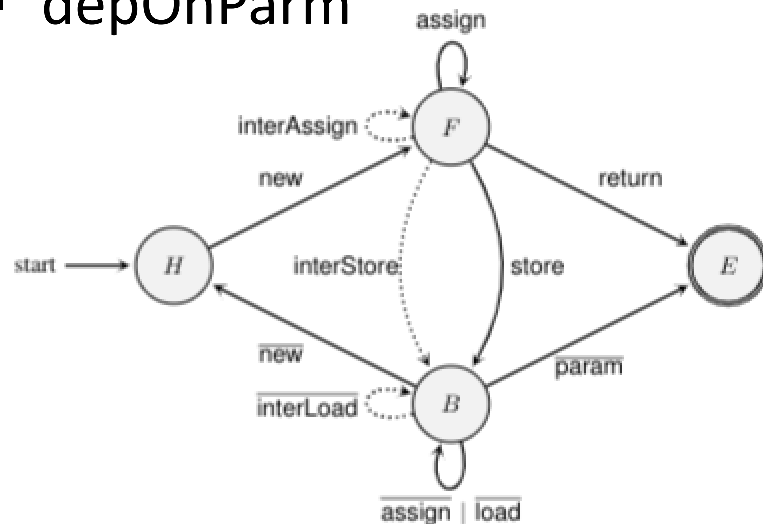
Algorithm 1: CONCH: context debloating.

Input: P // Input program
Output: $\mathcal{D}$. // Set of Context-Indep Objects

1. $\mathrm{CI} \leftarrow \mathrm{CD} \leftarrow \emptyset$
2. **for** $O_l \in \mathbb{H}$ **do**
3.   **if** $\nexists f \in \mathsf{fieldsOf}(O_l)$ s.t $\mathsf{hasLoad}(O_l, f) \wedge \mathsf{hasStore}(O_l, f)$ **then**
4. $\mathrm{CI} = \mathrm{CI} \cup \{O_l\}$ // Obs 1
5.   **else if** $O_l \notin \mathsf{leakObjects}$ **then**
6. $\mathrm{CI} = \mathrm{CI} \cup \{O_l\}$ // Obs 2
7.   **else**
8. $R(O_l) = \{l' : x.f = y \text{ in } P \mid O_l \in \overline{\mathsf{pts}}(x)\}$
9.   **for** $l' : x.f = y \in R(O_l)$ **do**
10.   **if** $\mathsf{methodOf}(l')$ is a constructor of O_l **then**
11. $m = \mathsf{methodOf}(l)$
12.   **else**
13. $m = \mathsf{methodOf}(l')$
14.   **if** $\mathsf{depOnParam}(y, m)$ **then**
15. $\mathrm{CD} = \mathrm{CD} \cup \{O_l\}$ // Obs 3(a)
16.   break
17. $\mathrm{UK} \leftarrow \mathbb{H} \setminus (\mathrm{CI} \cup \mathrm{CD})$, changed $\leftarrow$ true
18. **while** changed **do**
19.   changed $\leftarrow$ false
20.   **for** $O_l \in \mathrm{UK}$ **do**
21.   **if** $\exists\, l' : x.f = y \in R(O_l)$ $s.t.$ $\overline{\mathsf{pts}}(O_l.f) \cap \mathrm{CD} \neq \emptyset$ **then**
22. $\mathrm{CD} = \mathrm{CD} \cup \{O_l\}$ // Obs 3(b)
23.   changed $\leftarrow$ true
24. $\mathcal{D} = \mathrm{CI} \cup (\mathrm{UK} \setminus \mathrm{CD})$;
25. **return** $\mathcal{D}$

Our Solution: Conch

- Observations are linear verifiable.
  - ✓ Efficient
  - ✓ Effective
- IFDS-based algorithm for computing:
  - leakObjects
  - depOnParm



[SEEDS]

$$\frac{}{\langle O_l, H\rangle \to \langle O_l, H\rangle} \quad \frac{}{\langle p_i^m, F\rangle \to \langle p_i^m, F\rangle} \quad \frac{}{\langle ret^m, B\rangle \to \langle ret^m, B\rangle}$$

[PROPAGATE]

$$\frac{\langle n_1, S_1\rangle \to \langle O_l, H\rangle \quad l : n_2 = \text{new } T}{\langle n_1, S_1\rangle \to \langle n_2, F\rangle} \quad \frac{\langle n_1, S_1\rangle \to \langle n_2, F\rangle \quad l : n_3 = n_2}{\langle n_1, S_1\rangle \to \langle n_3, F\rangle}$$

$$\frac{\langle n_1, S_1\rangle \to \langle n_2, F\rangle \quad l : n_3.f = n_2}{\langle n_1, S_1\rangle \to \langle n_3, B\rangle} \quad \frac{\langle n_1, S_1\rangle \to \langle n_2, B\rangle \quad l : n_2 = n_3 \mid n_3.f}{\langle n_1, S_1\rangle \to \langle n_3, B\rangle}$$

$$\frac{\langle n_1, S_1\rangle \to \langle n_2, B\rangle \quad l : n_2 = \text{new } T \quad S_1 \neq B}{\langle n_1, S_1\rangle \to \langle O_l, H\rangle} \quad \frac{\langle n_1, S_1\rangle \to \langle n_2, S_2\rangle \quad \langle n_2, S_2\rangle \to \langle n_3, S_3\rangle \in \text{Sum}}{\langle n_1, S_1\rangle \to \langle n_3, S_3\rangle}$$

$$\frac{\langle n_1, S_1\rangle \to \langle ret^m, F\rangle}{\langle n_1, S_1\rangle \to \langle ret^m, E\rangle} \quad \frac{\langle n_1, S_1\rangle \to \langle p_i^m, B\rangle}{\langle n_1, S_1\rangle \to \langle p_i^m, E\rangle}$$

[SUMMARY]

$$\frac{\langle p_i^m, F\rangle \to \langle p_j^m, E\rangle \quad p_i^m \neq p_j^m \quad l : x = a_0.f(a_1, \cdots, a_r) \quad O \in \overline{\text{pts}}(a_0) \quad m = \text{dispatch}(f, O)}{\langle a_i, F\rangle \to \langle a_j, B\rangle \in \text{Sum}}$$

$$\frac{\langle p_i^m, F\rangle \to \langle ret^m, E\rangle \quad l : x = a_0.f(a_1, \cdots, a_r) \quad O \in \overline{\text{pts}}(a_0) \quad m = \text{dispatch}(f, O)}{\langle a_i, F\rangle \to \langle x, F\rangle \in \text{Sum}}$$

$$\frac{\langle ret^m, B\rangle \to \langle p_i^m, E\rangle \quad l : x = a_0.f(a_1, \cdots, a_r) \quad O \in \overline{\text{pts}}(a_0) \quad m = \text{dispatch}(f, O)}{\langle x, B\rangle \to \langle a_i, B\rangle \in \text{Sum}}$$

$$\frac{\langle O, H\rangle \to \langle ret^m, F\rangle \quad l : x = a_0.f(a_1, \cdots, a_r) \quad O \in \overline{\text{pts}}(a_0) \quad m = \text{dispatch}(f, O)}{\langle x, B\rangle \to \langle Sym_l, H\rangle \in \text{Sum} \quad \langle Sym_l, H\rangle \to \langle x, F\rangle \in \text{Sum}}$$

[COLLECT]

$$\frac{\langle O_l, H\rangle \to \langle p_i^m, E\rangle}{O_l \in \text{leakObjects}} \quad \frac{\langle O_l, H\rangle \to \langle ret^m, E\rangle}{O_l \in \text{leakObjects}}$$

Rules for computing leakObjects.

Implementation

Conch

Available

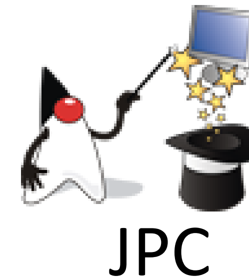
Reusable

- ❑Written in Java (~ 1500 LOC)
- ❑Artifact is deployed on Docker Hub: https://hub.docker.com/r/hdjay2013/conch-artifact
- ❑Open source: http://www.cse.unsw.edu.au/~corg/conch/

Evaluation

- ❑12 large Java Programs
  - 9 DaCapo benchmarks

  - 3 popular real-world applications



JPC

Evaluation

- 4 metrics
  - May-fail casting
  - De-virtualization
  - Call graph construction
  - Rechable methods

- Time budget: 12 hours

- Memory budget: 256 GB

Widely-used metrics to evaluate pointer analysis's precision

e.g., OOPSLA'19, OOPSLA'18, PLDI'17, OOPSLA'17, PLDI'14, PLDI'13, POPL'11

RQ1: Is Conch Precise (in identifying C.D.)?

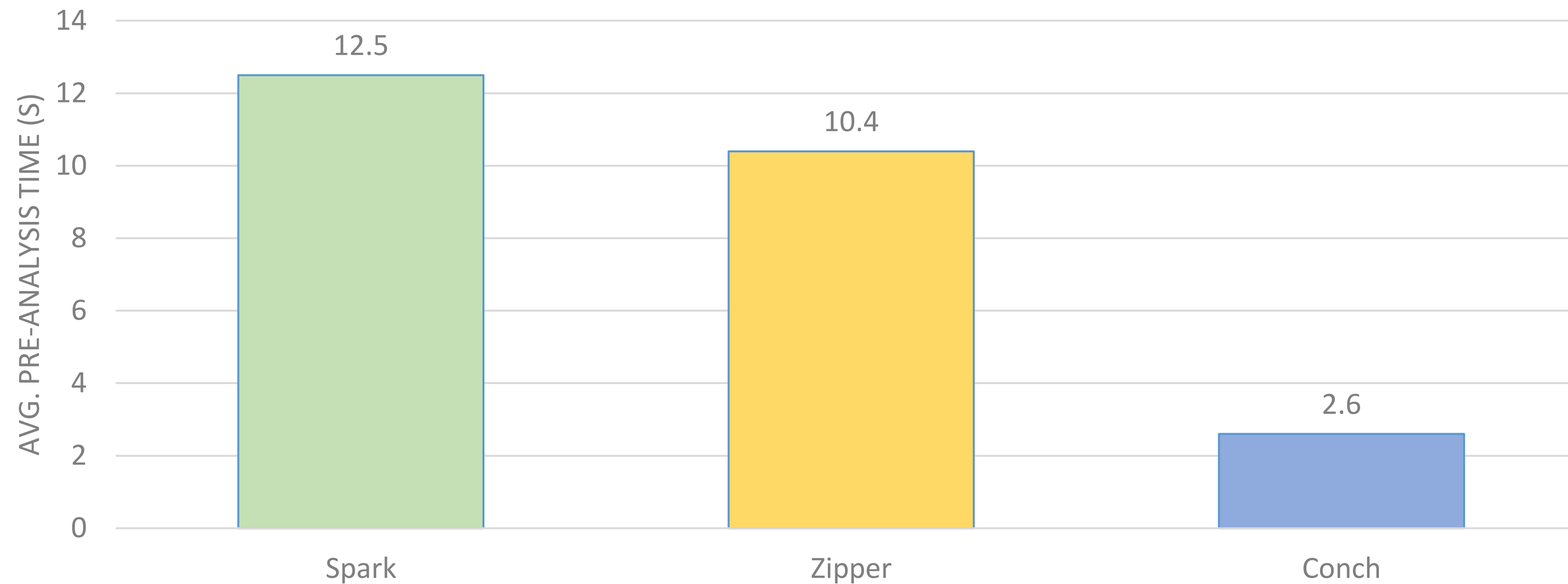
- Baselines (*k*OBJ & Z-*k*OBJ) vs. Debloated Baselines (*k*OBJ+D & Z-*k*OBJ+D)
  - Preserve precision for 9 DaCapo benchmarks and *findbugs*.
  - Less than 0.1% of precision loss in *checkstyle* and *JPC*.

Zipper[4]

Very precise as a context debloating technique.

[4] Li, Yue, Tian Tan, Anders Møller, and Yannis Smaragdakis. "Precision-guided context sensitivity for pointer analysis." Proceedings of the ACM on Programming Languages 2, no. OOPSLA (2018): 1-29.

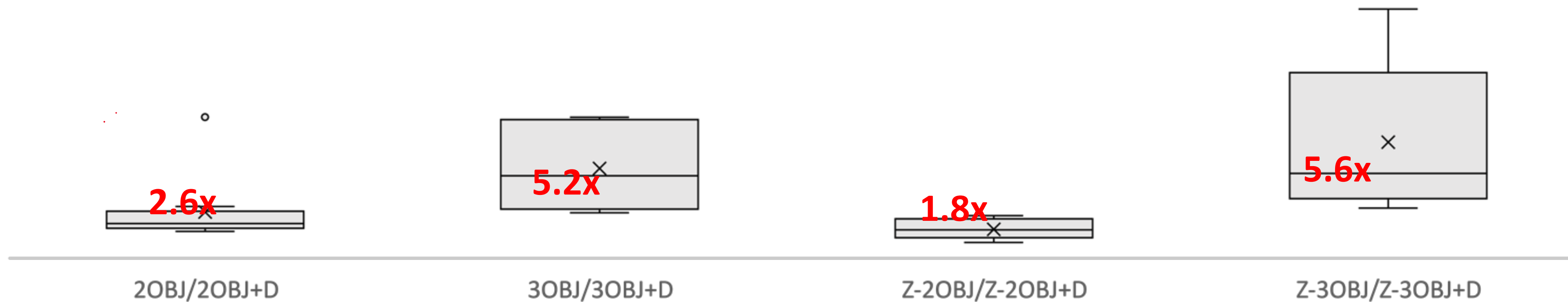
RQ2: Is Conch efficient (as a pre-analysis)?



Very efficient as a pre-analysis.

RQ3: Can Conch improve baselines?

- Average Speedups

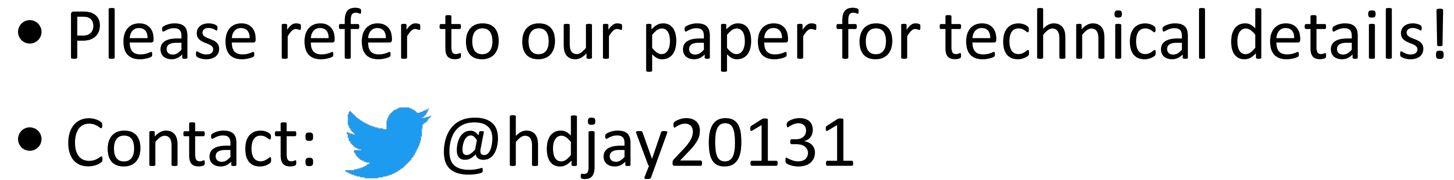


- Scalability
  - 2OBJ+D scales 1 more benchmark than 2OBJ: *eclipse*
  - 3OBJ+D scales 4 more benchmarks than 3OBJ: *bloat, chart, xalan, findbugs*
  - Z-3OBJ+D scales 2 more benchmarks than Z-3OBJ: *bloat, checkstyle*

Conclusion

- Context Debloating
  - Objects: Context Dependent + Context Independent
  - Conch: 3 Observations (linear verifiable)
- Implementation
  - open source: http://www.cse.unsw.edu.au/~corg/conch/
  - artifact: https://hub.docker.com/r/hdjay2013/conch-artifact
- Evaluation
  - very precise (preserves almost all the precision)
  - very efficient (not only in accelerating pointer analysis but also as a pre-analysis)

Q & A



This presentation and recording belong to the authors.
No distribution is allowed without the authors' permission.