Precision-Preserving Yet Fast Object-Sensitive Pointer Analysis with Partial Context Sensitivity

OOPSLA

Athens 2019

Jingbo Lu and Jingling Xue

UNSW SYDNEY

Object-Sensitive Pointer Analysis

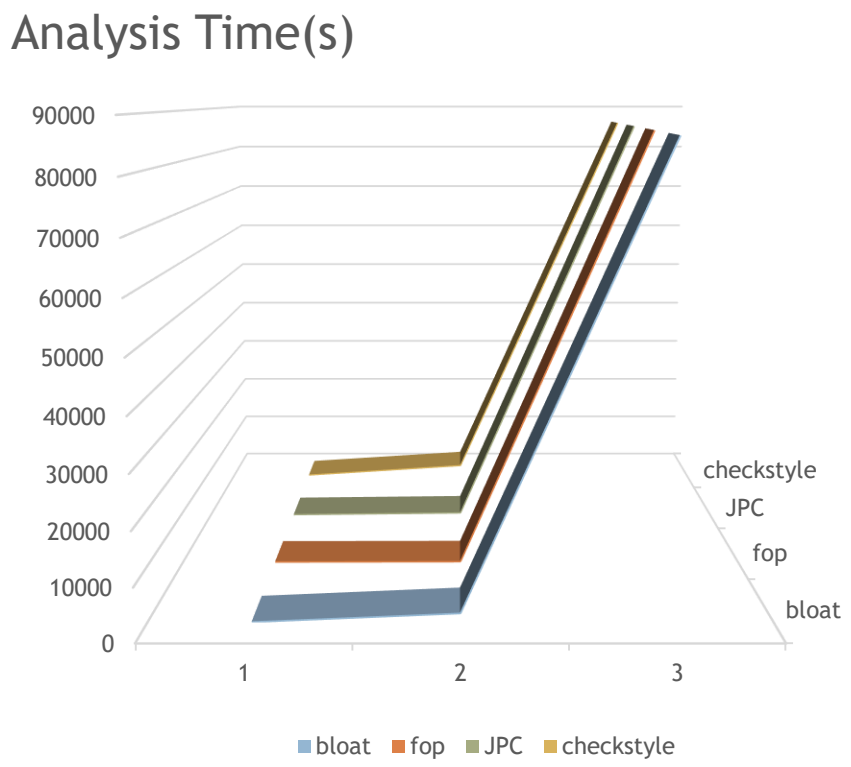
- Static Program Analysis
  - Taint Analysis
  - TypeState Analysis
  - Bug Detection
  - …

Object-Sensitive Pointer Analysis

- Context Sensitivity
  - Call-Site Sensitivity
  - Object Sensitivity ✓ OO Languages
  - …

Related Work: Milanova et al. TOSEM'05, Tan et al. SAS'16, Tan et al. PLDI'17,Hassanshahi et al. SOAP'17, Jeong et al. OOPSLA'17, Jeon et al. OOPSLA'18, Li et al. OOPSLA'18, …

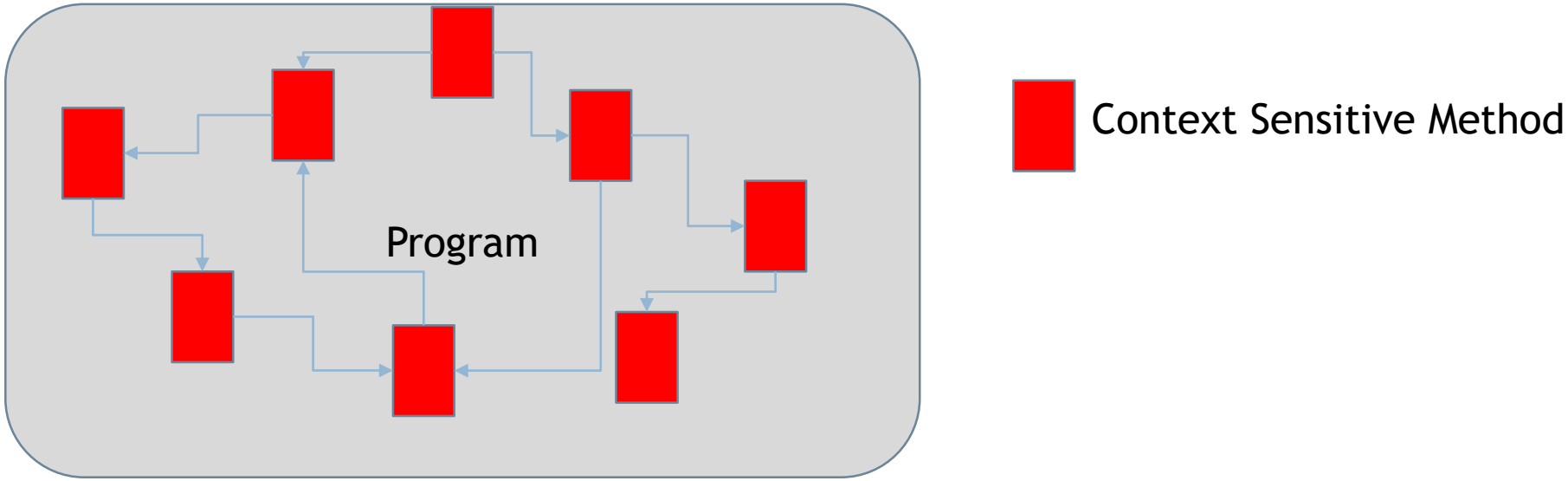
K-Limiting
Object Sensitive Pointer Analysis



Related Work: Milanova et al. TOSEM'05, Tan et al. SAS'16, Tan et al. PLDI'17,Hassanshahi et al. SOAP'17, Jeong et al. OOPSLA'17, Jeon et al. OOPSLA'18, Li et al. OOPSLA'18, …

K-Limiting
Object Sensitive Pointer Analysis

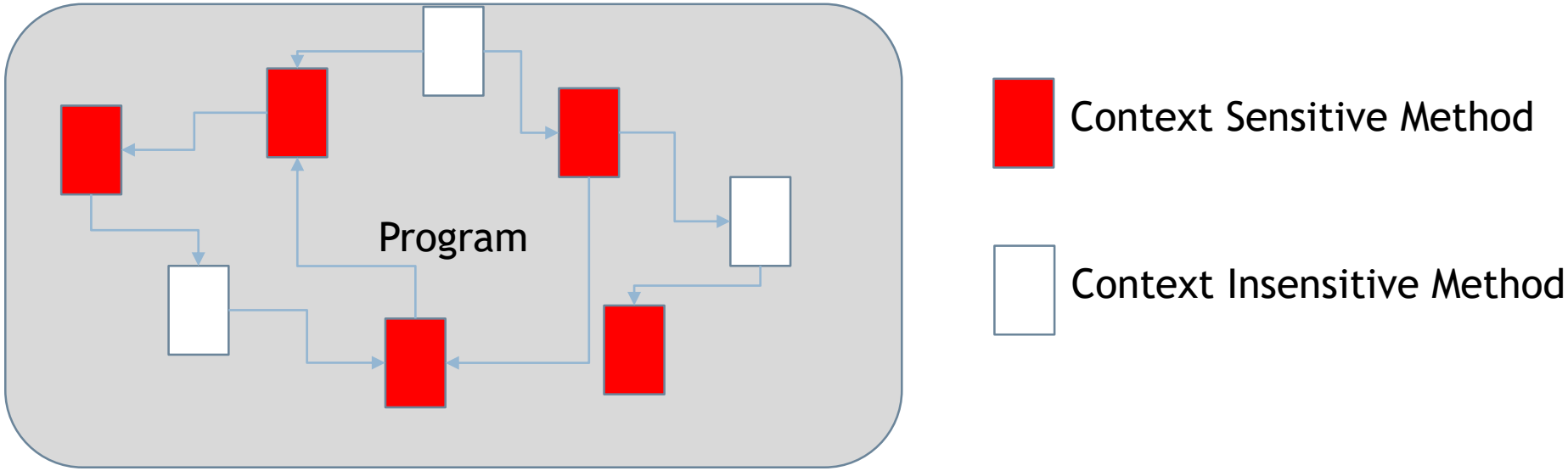
Conventional: apply Object-Sensitivity to all methods



Related Work: Milanova et al. TOSEM'05, Tan et al. SAS'16, Tan et al. PLDI'17,Hassanshahi et al. SOAP'17, Jeong et al. OOPSLA'17, Jeon et al. OOPSLA'18, Li et al. OOPSLA'18, …

Selective
Object Sensitive Pointer Analysis

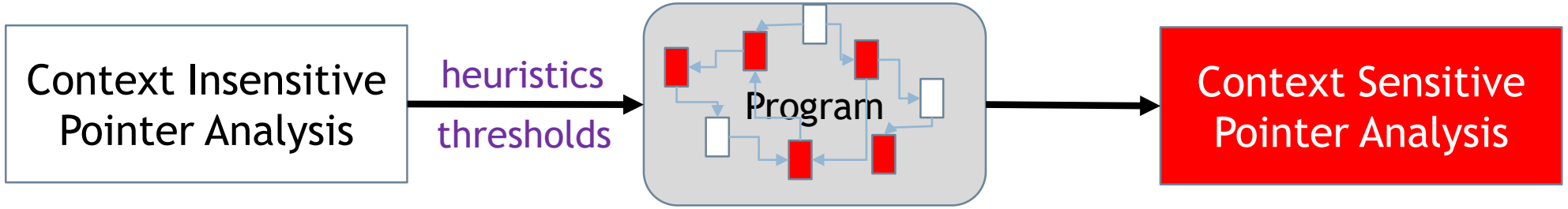
Existing Efforts: apply Object-Sensitivity to
Selected methods



Related Work: Smaragdakis et al. PLDI'14, Hassanshahi et al. SOAP'17, Jeong et al. OOPSLA'17, Li et al. OOPSLA'18, …

Selective
Object Sensitive Pointer Analysis

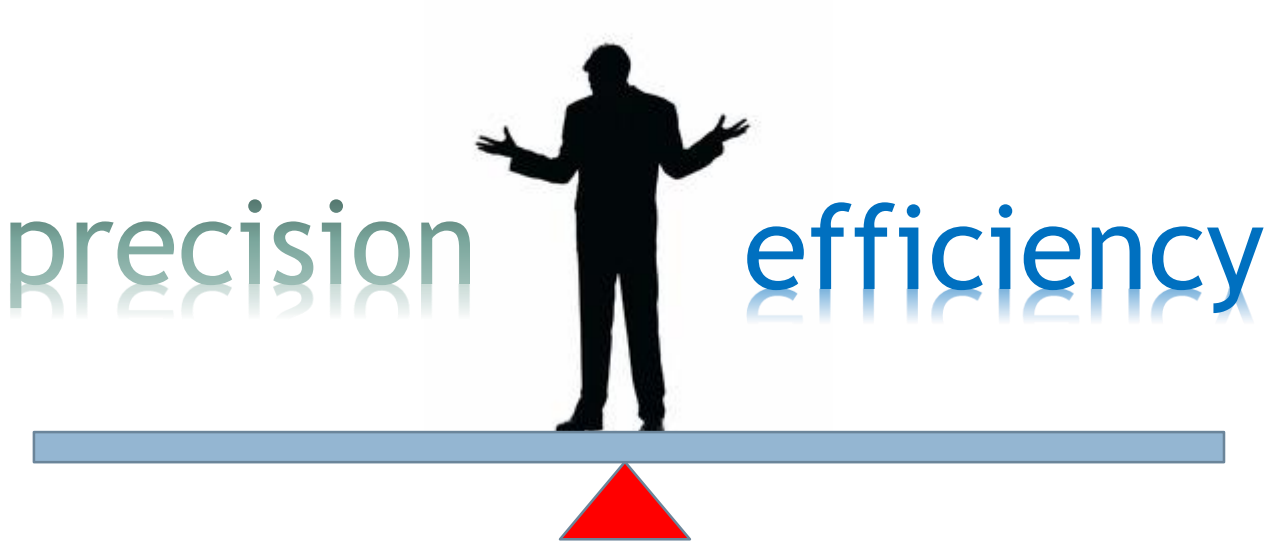
Existing Efforts: apply Object-Sensitivity to
Selected methods



Related Work: Smaragdakis et al. PLDI'14, Hassanshahi et al. SOAP'17, Jeong et al. OOPSLA'17, Li et al. OOPSLA'18, …

Challenge

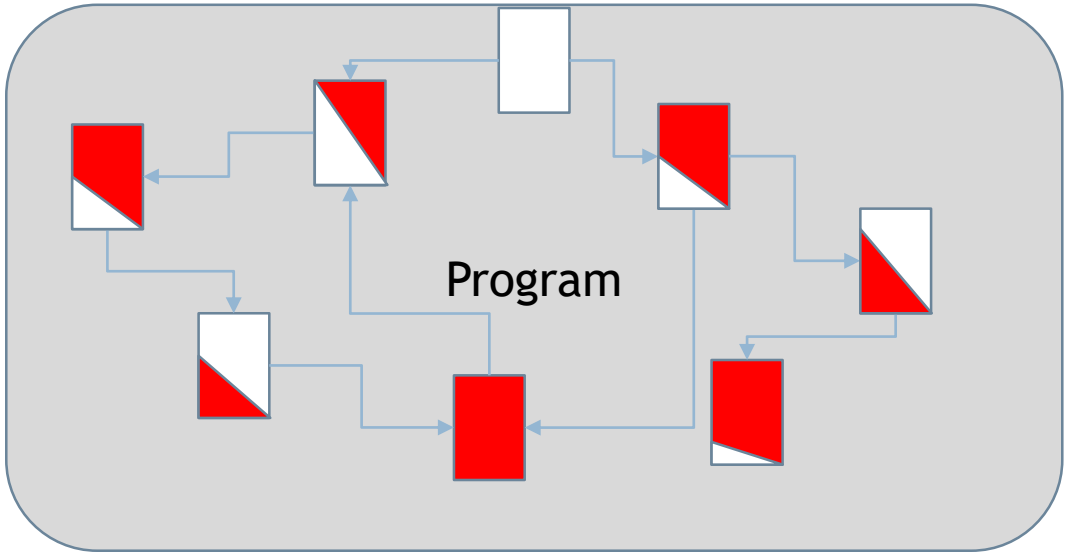
Existing Efforts: a trade-off between precision and efficiency



Related Work: Smaragdakis et al. PLDI'14, Hassanshahi et al. SOAP'17, Jeong et al. OOPSLA'17, Li et al. OOPSLA'18, …

EAGLE : A New Points-to Analysis Technique

apply Object-Sensitivity to selected pointers

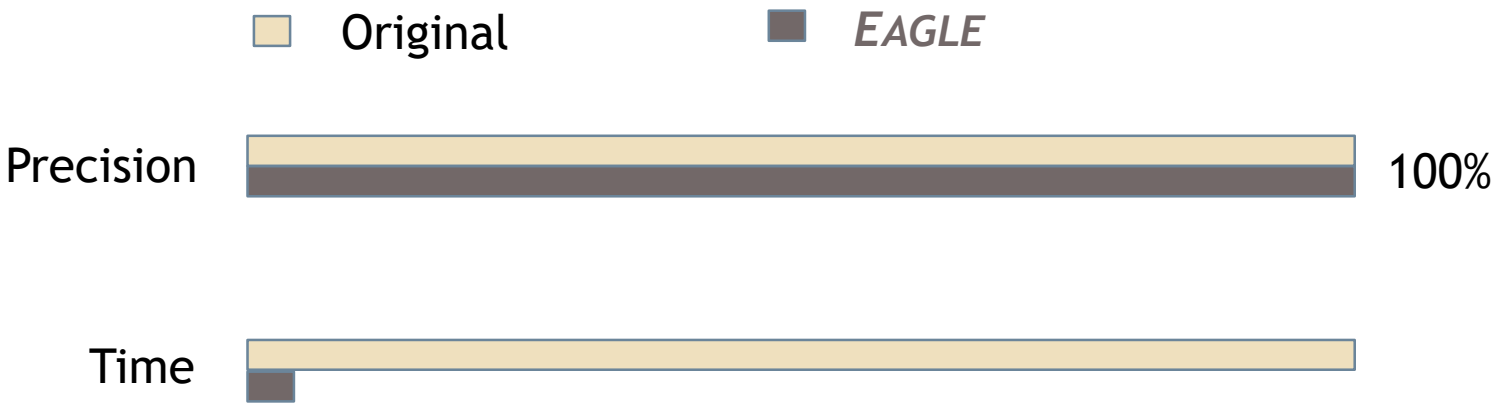


Context Sensitive

Context Insensitive

EAGLE :
A New Points-to Analysis Technique

faster while 100% precision-preserving



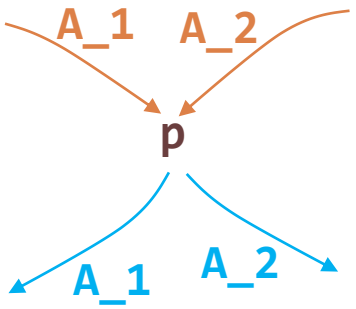
How?

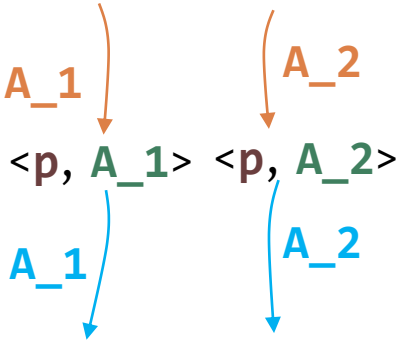
Context-Sensitivity and Precision

```
class A{
    Object id(Object p){
        return p;
    }
}

Object o1 = new Object(); // O_1
Object o2 = new Object(); // O_2
A a1 = new A(); // A_1
A a2 = new A(); // A_2
Object v1 = a1.id(o1);
Object v2 = a2.id(o2);
```

A_2 A_1 A_1 A_2





Entry Flows

<o1, ϵ> → <p, A_1>

<o1, ϵ> → <p, A_2>

Exit Flows

<p, A_1> → <v1, ϵ>

<p, A_2> → <v2, ϵ>

Context-Sensitivity and Precision

```
class A{
    void print(Object s){
        //Print s
    }
}

Object o1 = new Object(); // O_1
Object o2 = new Object(); // O_2
A a1 = new A(); // A_1
A a2 = new A(); // A_2
a1.print(o1);
a2.print(o2);
```

A_1 A_2

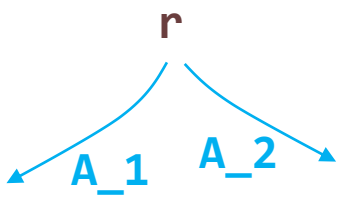
Entry Flows

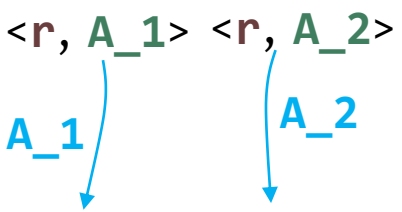
<o1, ϵ> → <s, A_1>

<o1, ϵ> → <s, A_2>

Context-Sensitivity and Precision

```
class A{
    Object create(){
        r = new Object(); // O_1
        return r;
    }
}
A_2   A_1
A a1 = new A(); // A_1
A a2 = new A(); // A_2
Object v1 = a1.create();
Object v2 = a2.create();
```





Exit Flows

<r, A_1> → <v1, ϵ>

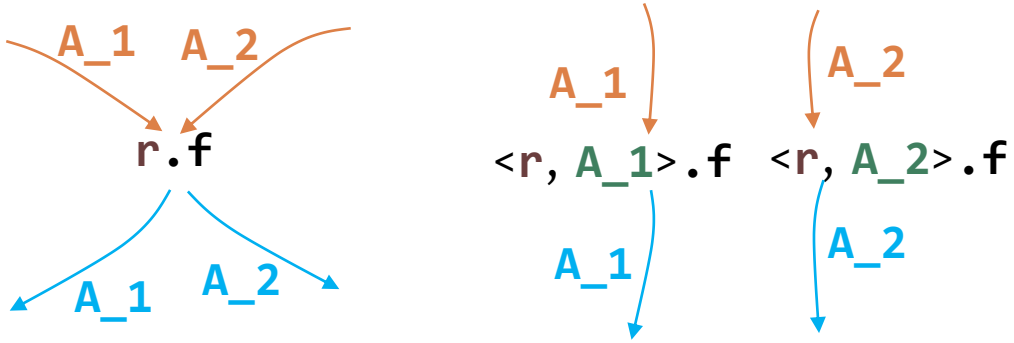
<r, A_2> → <v2, ϵ>

Context-Sensitivity and Precision

Context-Sensitivity => *Entry Flows* $\wedge$ *Exit Flows*

Context-Sensitivity and Precision

```
class A{
    B create(){
        r = new B(); // B_1
        return r;
    }
}
A_2        A_1
A a1 = new A(); // A_1
A a2 = new A(); // A_2
B b1 = a1.create();
B b2 = a2.create();
Object o1 = new Object(); // O_1
Object o2 = new Object(); // O_2
b1.f = o1;
b2.f = o2;
Object v1 = b1.f;
Object v2 = b2.f;
```



Exit Flows

<r, A_1> → <v1, ϵ>

<r, A_2> → <v2, ϵ>

Object-Sensitive
Context Free Language Reachability

Statement	Edges
$x = \text{new } C() \; //o$	$o \xrightarrow{\text{new}} x$
$x = y$	$y \xrightarrow{\text{assign}} x$
$x.f = y$	$y \xrightarrow{\text{store}[f]} x$
$x = y.f$	$y \xrightarrow{\text{load}[f]} x$
$x = y.m(\ldots, a_i, \ldots)$ //c	…

$$L_{FC} = L_F \wedge L_C$$

L_F

$$\begin{aligned}
\underline{flowsto} &\rightarrow \underline{new}\ flows^* \\
\overline{flowsto} &\rightarrow \overline{flows}^*\ \overline{new} \\
\underline{flows} &\rightarrow \underline{assign} \mid \underline{store[f]}\ alias\ \underline{load[f]} \\
\overline{flows} &\rightarrow \overline{assign} \mid \overline{load[f]}\ alias\ \overline{store[f]} \\
alias &\rightarrow \overline{flowsto}\ flowsto
\end{aligned}$$

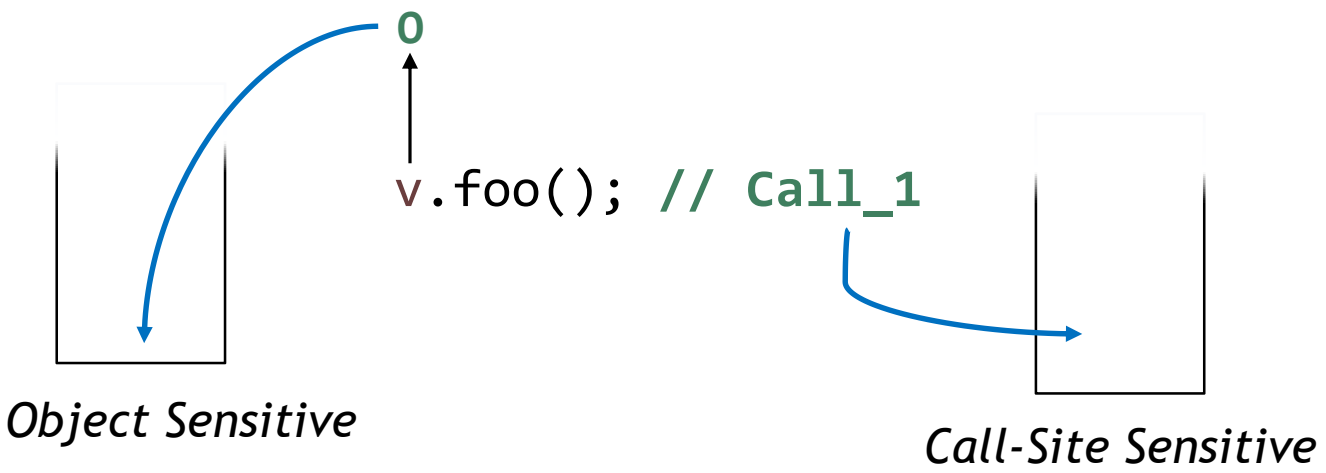
L_C

$$\begin{aligned}
\text{realizable} &\rightarrow \text{exit entry} \\
\text{exit} &\rightarrow \text{exit balanced} \mid \text{exit } \check{c} \mid \epsilon \\
\text{entry} &\rightarrow \text{entry balanced} \mid \text{entry } \hat{c} \mid \epsilon \\
\text{balanced} &\rightarrow \text{balanced balanced} \mid \hat{c}\ \text{balanced}\ \check{c} \mid \epsilon
\end{aligned}$$

Related Work: Reps IST'98, Sridharan et al. PLDI'06, Xu et al. 09, Shang et al. CGO'12, Thiessen et al. PDLI'17, Cai et al. PLDI'18, …

Object-Sensitive
Context Free Language Reachability

Object Sensitivity vs Call-Site Sensitivity



Related Work: Shivers 91, Milanova et al. TOSEM'05

Object-Sensitive
Context Free Language Reachability

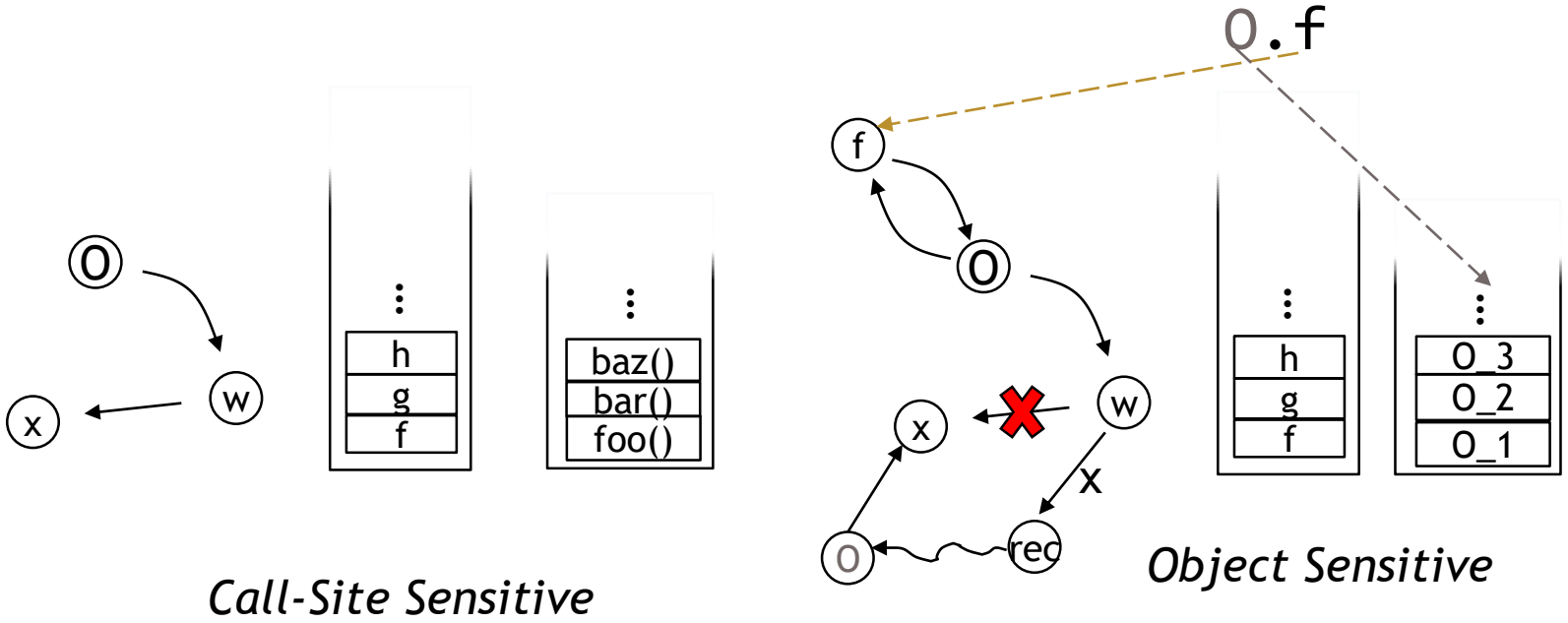
Statement	Edges	
$x = \text{new } C() \; //o$	$o \xrightarrow{\text{new}} x$	
$x = y$	$y \xrightarrow{\text{assign}} x$	
$x.f = y$	$y \xrightarrow{\text{store}[f]} x$	$\dfrac{o \in \overline{pt}(x)}{o \xrightarrow[\hat{o}]{\text{hload}[f]} f}$
$x = y.f$	$y \xrightarrow{\text{load}[f]} x$	$\dfrac{o \in \overline{pt}(y)}{f \xrightarrow[\check{o}]{\text{hstore}[f]} o}$
$x = y.m(\ldots, a_i, \ldots)$		$\dfrac{o \in \overline{pt}(y),\ m' = dispatch(m, o)}{\begin{array}{ccc} y \xrightarrow{\text{store}[this^{m'}]} y & a_i \xrightarrow{\text{store}[p_i^{m'}]} y & y \xrightarrow{\text{load}[ret^{m'}]} x \\ o \xrightarrow[\hat{o}]{\text{hload}[this^{m'}]} this^{m'} & o \xrightarrow[\hat{o}]{\text{hload}[p_i^{m'}]} p_i^{m'} & ret^{m'} \xrightarrow[\check{o}]{\text{hstore}[ret^{m'}]} o \end{array}}$

Object-Sensitive
Context Free Language Reachability

L_F

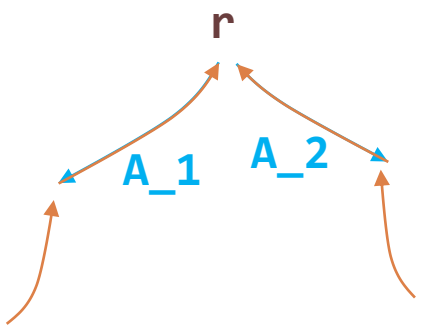
$$
\begin{array}{rcl}
\text{flowsto} & \rightarrow & \text{new flows}^* \\
\overline{\text{flowsto}} & \rightarrow & \overline{\text{flows}}^* \, \overline{\text{new}} \\
\text{flows} & \rightarrow & \text{assign} \quad | \quad \text{store}[f] \, \overline{\text{flowsto}} \, \text{hload}[f] \quad | \quad \text{hstore}[f] \, \text{flowsto} \, \text{load}[f] \\
\overline{\text{flows}} & \rightarrow & \overline{\text{assign}} \quad | \quad \overline{\text{hload}[f]} \, \text{flowsto} \, \overline{\text{store}[f]} \quad | \quad \overline{\text{load}[f]} \, \overline{\text{flowsto}} \, \overline{\text{hstore}[f]}
\end{array}
$$

Object-Sensitive
Context Free Language Reachability



Context-Sensitivity and Precision

```
class A{
    B create(){
        r = new B(); // B_1
        return r;
    }
}
A_2          A_1
A a1 = new A(); // A_1
A a2 = new A(); // A_2
B b1 = a1.create();
B b2 = a2.create();
Object o1 = new Object(); // O_1
Object o2 = new Object(); // O_2
b1.f = o1;
b2.f = o2;
Object v1 = b1.f;
Object v2 = b2.f;
```



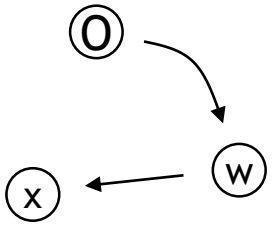
Exit Flows

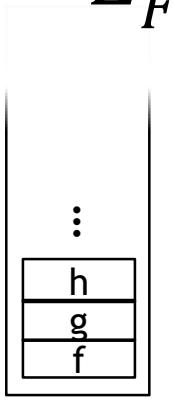
<r, A_1> → <v1, ϵ>

<r, A_2> → <v2, ϵ>

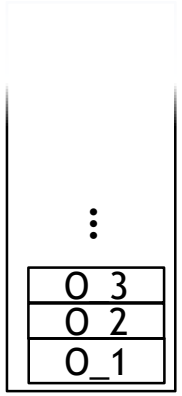
Selection Process is Undecidable

$$L_{FC} = L_F \wedge L_C$$





Stack of Fields



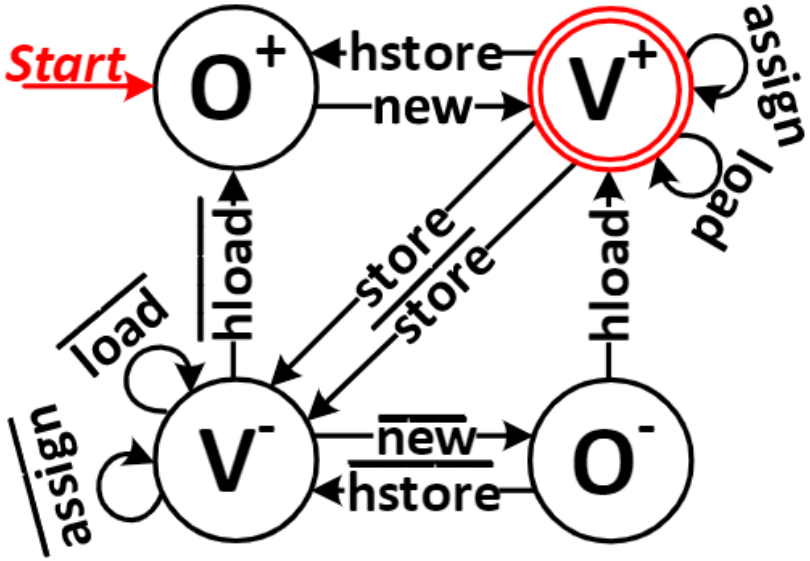
Stack of Alloc-Sites

Related Work: Reps TOPLAS'00

Selection Process is Undecidable

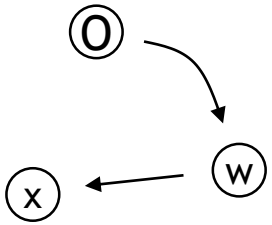
$$L_F \rightarrow L_R$$

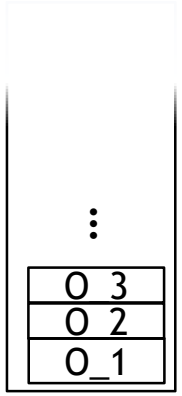
DFA for L_R



Selection Process is Polynomial

$$L_{RC} = L_R \wedge L_C$$





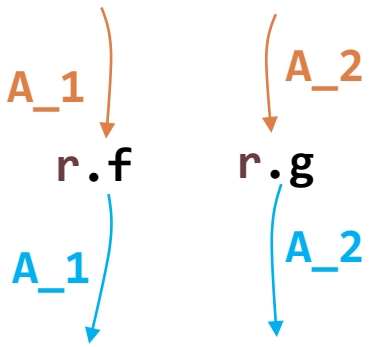
Stack of Alloc-Sites

Where is the Over-Approximation?

```
class A{
    B create(){
        r = new B(); // B_1
        return r;
    }
}
```

A_2 A_1

```
A a1 = new A(); // A_1
A a2 = new A(); // A_2
B b1 = a1.create();
B b2 = a2.create();
Object o1 = new Object(); // O_1
Object o2 = new Object(); // O_2
b1.f = o1;
b2.g = o2;
Object v1 = b1.f;
Object v2 = b2.g;
```



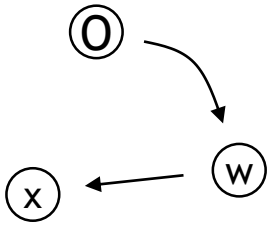
Exit Flows

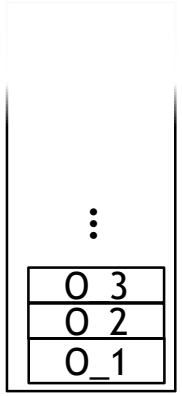
<r, A_1> → <v1, ϵ>

<r, A_2> → <v2, ϵ>

Selection Process is Polynomial

$$L_{RC} = L_R \wedge L_C$$

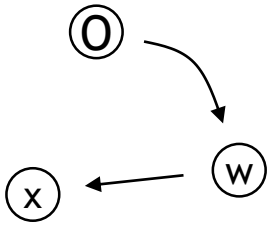


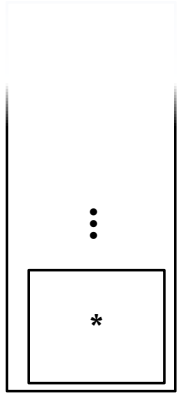


Stack of Alloc-Sites

Selection Process is Polynomial

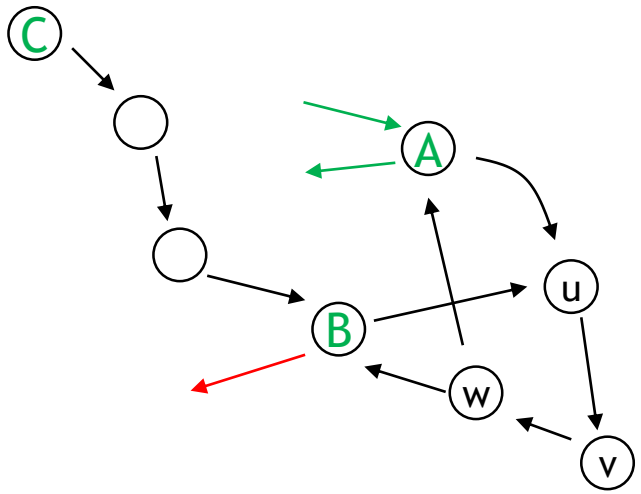
$$L_{RC} = L_R \wedge L_C$$

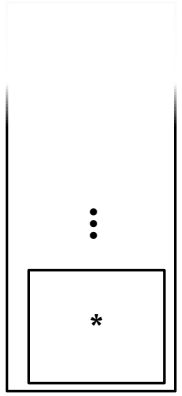




Stack of Alloc-Sites

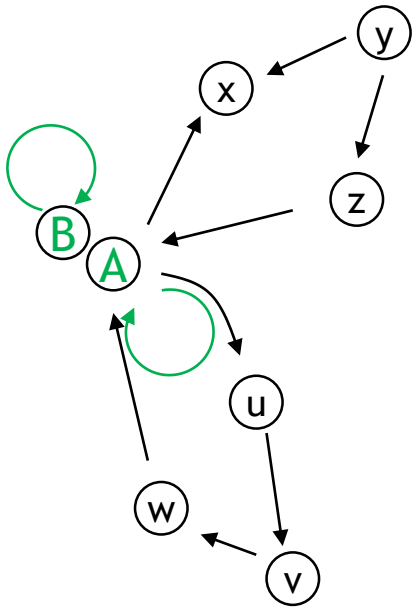
Selection Process is Polynomial

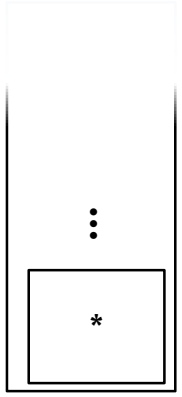




Stack of Alloc-Sites

Selection Process is Linear





Stack of Alloc-Sites

Selection Process is Linear

$$\frac{n \underset{\hat{o}}{\longrightarrow} n' \in G^R_{\text{pag}}}{n'.\text{cs} = \text{true}} \; [\textsc{EntryCTX}]$$

$$\frac{n \underset{\check{o}}{\longrightarrow} n' \in G^R_{\text{pag}} \qquad n.\text{cs} = \text{true}}{o^- \underset{\epsilon}{\longrightarrow} o^+ \in G^R_{\text{pag}}} \; [\textsc{ExitCTX}]$$

$$\frac{n \underset{\epsilon}{\longrightarrow} n' \in G^R_{\text{pag}} \qquad n.\text{cs} = \text{true}}{n'.\text{cs} = \text{true}} \; [\textsc{Prop}]$$

Evaluation

12 large Java programs

the DaCapo benchmark suite

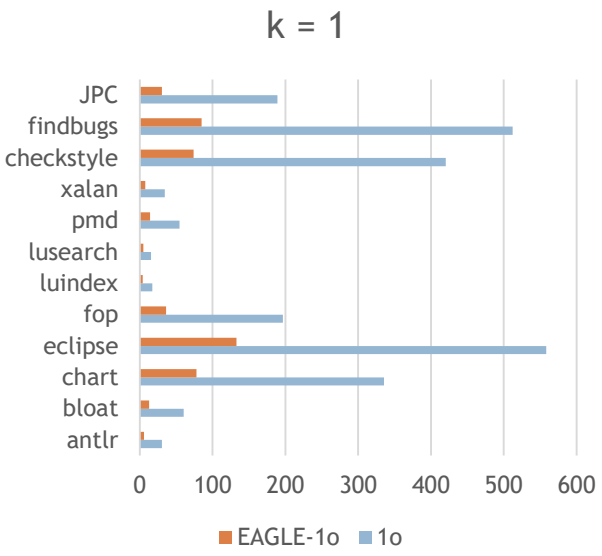
checkstyle

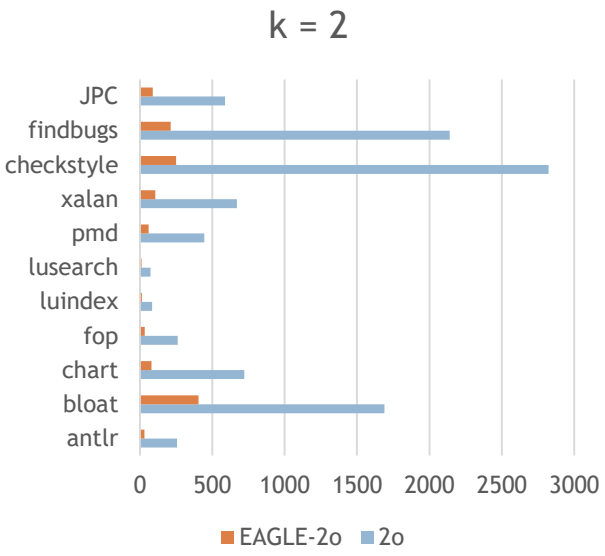
FindBugs™

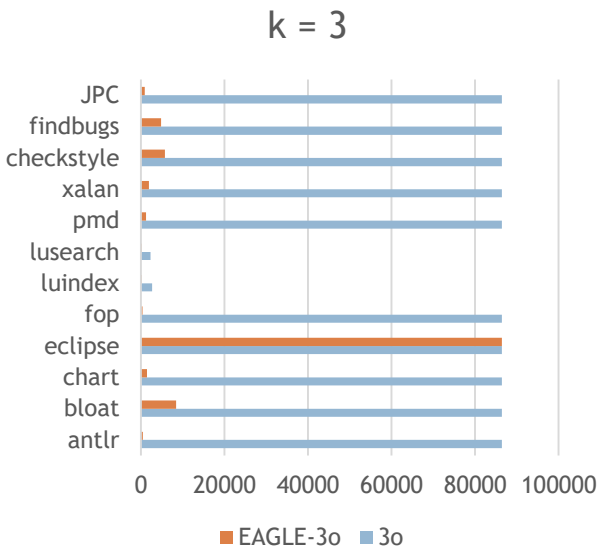
JPC

Evaluation

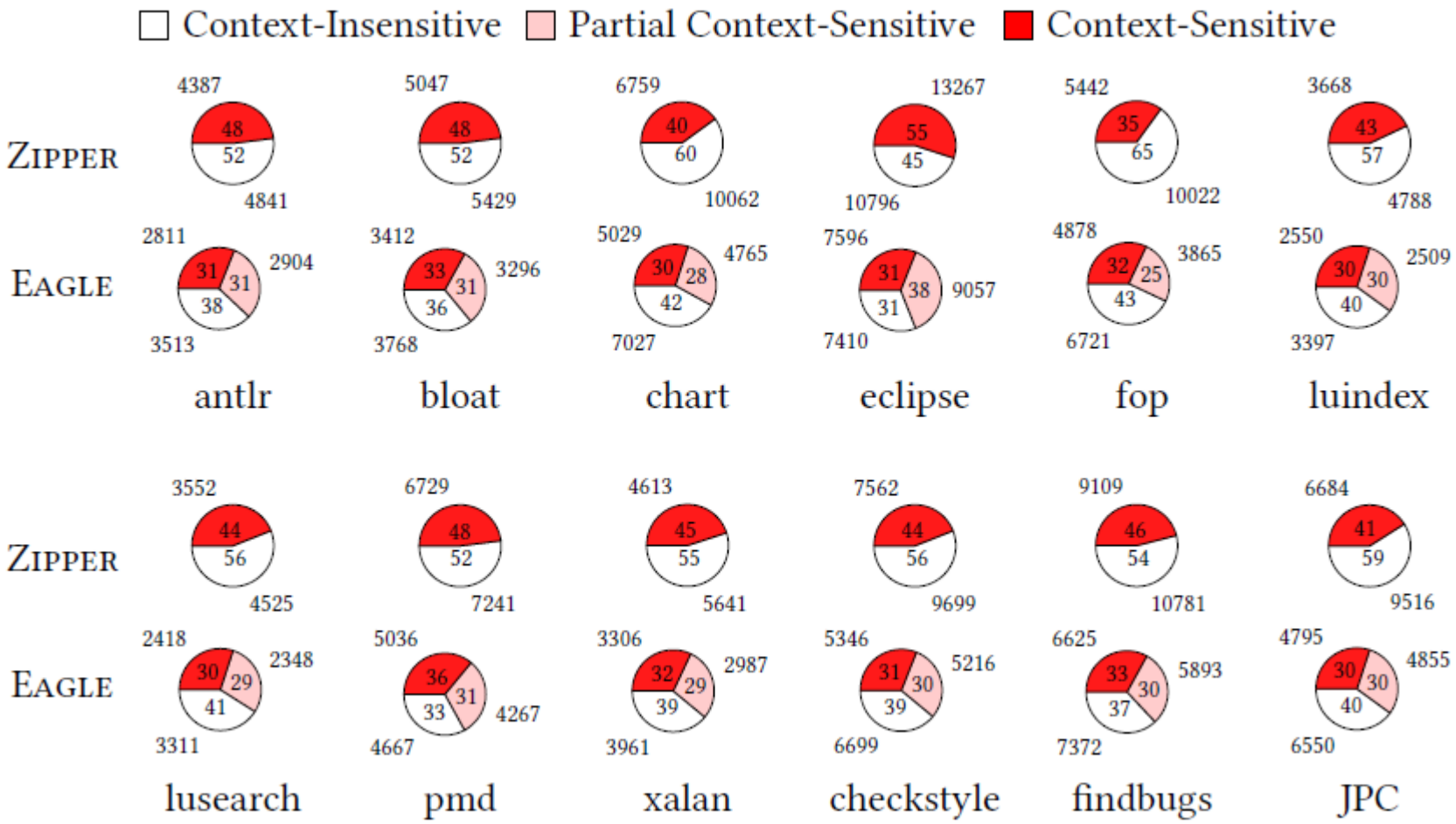
100% precise, i.e., identical points-to relation



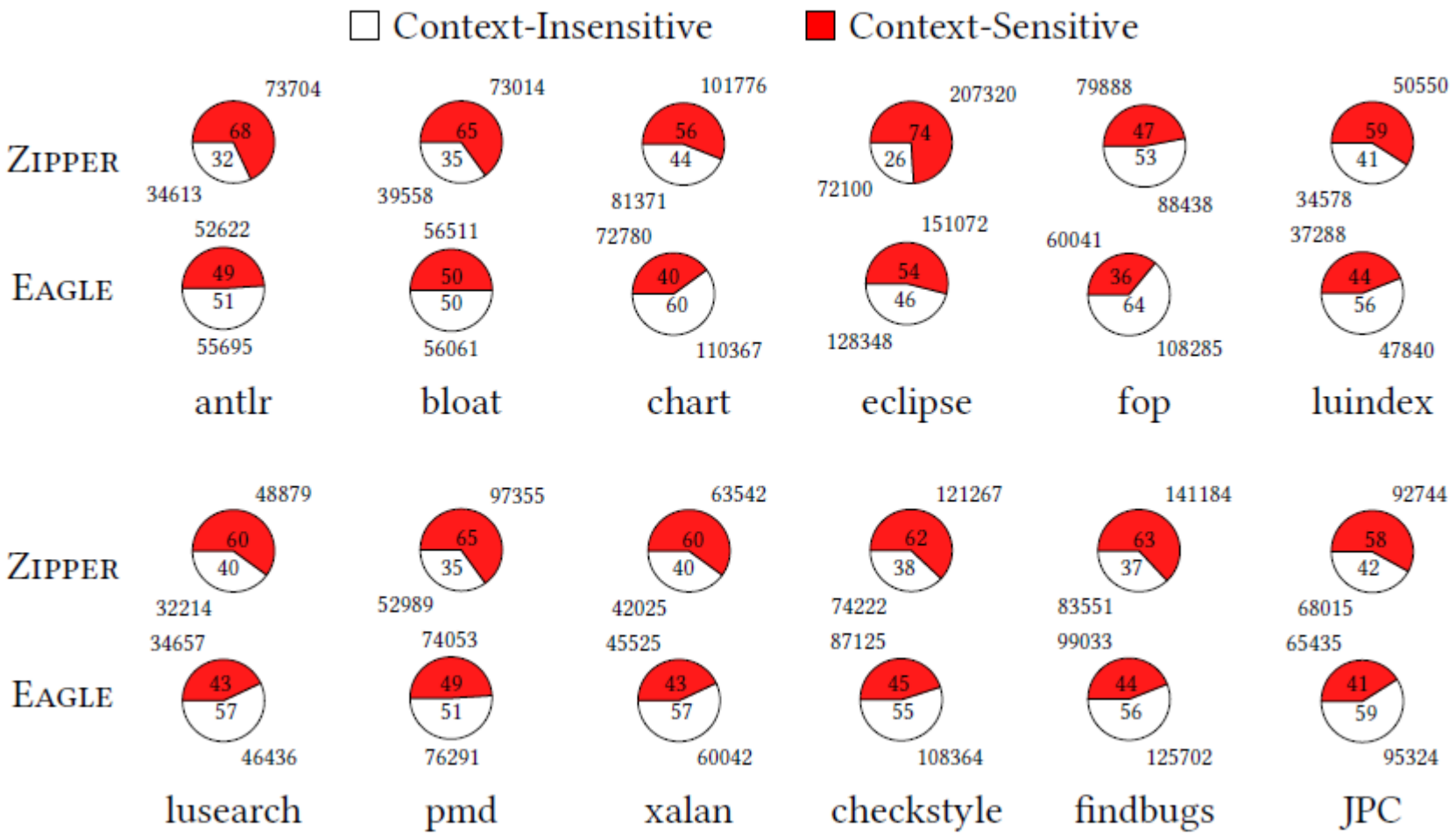




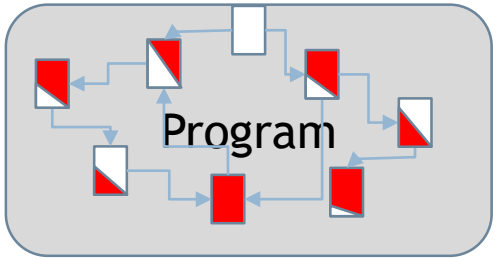
Evaluation

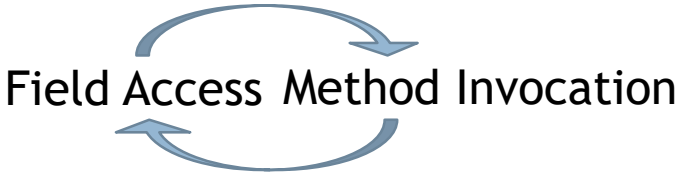

Context-Insensitive
Partial Context-Sensitive
Context-Sensitive
ZIPPER
EAGLE
4387 48 52 4841
2811 31 31 38 2904 3513
5047 48 52 5429
3412 33 31 36 3296 3768
6759 40 60 10062
5029 30 28 42 4765 7027
13267 55 45 10796
7596 31 38 31 9057 7410
5442 35 65 10022
4878 32 25 43 3865 6721
3668 43 57 4788
2550 30 30 40 2509 3397
antlr
bloat
chart
eclipse
fop
luindex
3552 44 56 4525
2418 30 29 41 2348 3311
6729 48 52 7241
5036 36 31 33 4267 4667
4613 45 55 5641
3306 32 29 39 2987 3961
7562 44 56 9699
5346 31 30 39 5216 6699
9109 46 54 10781
6625 33 30 37 5893 7372
6684 41 59 9516
4795 30 30 40 4855 6550
lusearch
pmd
xalan
checkstyle
findbugs
JPC

Evaluation


Context-Insensitive
Context-Sensitive
ZIPPER
EAGLE
73704
68
32
34613
73014
65
35
39558
101776
56
44
81371
207320
74
26
72100
79888
47
53
88438
50550
59
41
34578
52622
49
51
55695
56511
50
50
56061
72780
40
60
110367
151072
54
46
128348
60041
36
64
108285
37288
44
56
47840
antlr
bloat
chart
eclipse
fop
luindex
ZIPPER
EAGLE
48879
60
40
32214
97355
65
35
52989
63542
60
40
42025
121267
62
38
74222
141184
63
37
83551
92744
58
42
68015
34657
43
57
46436
74053
49
51
76291
45525
43
57
60042
87125
45
55
108364
99033
44
56
125702
65435
41
59
95324
lusearch
pmd
xalan
checkstyle
findbugs
JPC

Conclusion





EAGLE

"*Linear pre-analysis*"

100% precise