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

Jingbo Lu and Jingling Xue
Object-Sensitive Pointer Analysis

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

Object-Sensitive Pointer Analysis

- Context Sensitivity
  - Call-Site Sensitivity
  - Object Sensitivity
  - ...

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

Context Insensitive Pointer Analysis → heuristics → Program → Context Sensitive Pointer Analysis

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

Program

Context Sensitive

Context Insensitive
EAGLE: A New Points-to Analysis Technique

faster while 100% precision-preserving

- Precision: 100%
- Time: how?
Context-Sensitivity and Precision

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

Object o1 = new Object(); // 0_1
Object o2 = new Object(); // 0_2
A a1 = new A(); // A_1
A a2 = new A(); // A_2
Object v1 = a1.id(o1);
Object v2 = a2.id(o2);

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(); // 0_1
Object o2 = new Object(); // 0_2
A a1 = new A(); // A_1
A a2 = new A(); // A_2
a1.print(o1);
a2.print(o2);

Entry Flows
<o1, ε> → <s, A_1>
<o1, ε> → <s, A_2>
class A{
    Object create(){
        r = new Object(); // 0_1
        return r;
    }
}

A a1 = new A(); // A_1
A a2 = new A(); // A_2
Object v1 = a1.create();
Object v2 = a2.create();
Context-Sensitivity and Precision

Context-Sensitivity  =>  Entry Flows $\land$ Exit Flows
class A{
    B create(){
        r = new B(); // B_1
        return r;
    }
}

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

\[ L_{FC} = L_F \land L_C \]

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

\( L_F \)

- \( \text{flowsto} \rightarrow \text{new flows}^* \)
- \( \text{flowsto} \rightarrow \text{flows}^* \text{ new} \)
- \( \text{flows} \rightarrow \text{assign} \mid \text{store}[f] \mid \text{alias load}[f] \)
- \( \text{flows} \rightarrow \text{assign} \mid \text{load}[f] \mid \text{alias store}[f] \)
- \( \text{alias} \rightarrow \text{flowsto} \text{ flowsto} \)

\( L_C \)

- \( \text{realizable} \rightarrow \text{exit entry} \)
- \( \text{exit} \rightarrow \text{exit balanced} \mid \text{exit } ĉ \mid \epsilon \)
- \( \text{entry} \rightarrow \text{entry balanced} \mid \text{entry } ĉ \mid \epsilon \)
- \( \text{balanced} \rightarrow \text{balanced balanced} \mid \text{balanced } ĉ \mid \epsilon \)

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:  
x = new C() // o

Edges:  
o \xrightarrow{\text{new}} x

\[ x = y \]

\[ y \xrightarrow{\text{assign}} x \]

\[ x.f = y \]

\[ y \xrightarrow{\text{store}[f]} x \]

\[ o \in \overline{p}(x) \]
\[ o \xrightarrow{\text{load}[f]} \hat{o} \]
\[ \hat{o} \xrightarrow{\text{f}} f \]

\[ x = y.f \]

\[ y \xrightarrow{\text{load}[f]} x \]

\[ o \in \overline{p}(y) \]
\[ f \xrightarrow{\text{htore}[f]} \hat{o} \]
\[ \hat{o} \xrightarrow{\text{o}} 0 \]

\[ x = y.m(..., a_i, ...) \]

o \in \overline{p}(y), \ m' = dispatch(m, o)

\[ y \xrightarrow{\text{store}[this[m']]} y \]
\[ a_i \xrightarrow{\text{store}[p_i]} y \]
\[ y \xrightarrow{\text{load}[ret[m']]} \]
\[ X \]

\[ o \xrightarrow{\text{load}[this[m']]} \hat{o} \]
\[ \hat{o} \xrightarrow{\text{o}} \text{this}^{m'} \]
\[ o \xrightarrow{\text{load}[p_i]} \hat{o} \]
\[ \hat{o} \xrightarrow{\text{p}_i^{m'}} \]
\[ ret^{m'} \]
\[ \text{histore}[ret[m']] \]
\[ \hat{o} \xrightarrow{\text{o}} 0 \]
Object-Sensitive
Context Free Language Reachability

\[ L_F \]

\[
\begin{align*}
\text{flowsto} & \rightarrow \text{new flows}^* \\
\text{flowsto} & \rightarrow \text{flows}^* \text{ new} \\
\text{flows} & \rightarrow \text{assign} \quad \text{store}[f] \text{ flowsto } \text{hload}[f] \quad \text{hstore}[f] \text{ flowsto } \text{load}[f] \\
\text{flows} & \rightarrow \text{assign} \quad \text{hload}[f] \text{ flowsto } \text{store}[f] \quad \text{load}[f] \text{ flowsto } \text{hstore}[f]
\end{align*}
\]
Object-Sensitive
Context Free Language Reachability

Call-Site Sensitive

Object Sensitive
Context-Sensitivity and Precision

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

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;

\[ \text{Exit Flows} \]
\[ <r, A_1> \rightarrow <v1, \varepsilon> \]
\[ <r, A_2> \rightarrow <v2, \varepsilon> \]
Selection Process is Undecidable

\[ L_{FC} = L_F \land L_C \]

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

\[ L_{RC} = L_R \land L_C \]

Stack of Alloc-Sites
Where is the Over-Approximation?

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

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

\[ L_{RC} = L_R \land L_C \]
Selection Process is Polynomial

\[ L_{RC} = L_R \land L_C \]

Stack of Alloc-Sites
Selection Process is Polynomial

Stack of Alloc-Sites
Selection Process is Linear

Stack of Alloc-Sites
Selection Process is Linear

\[ n \rightarrow n' \in G^R_{\text{pag}} \hspace{1cm} n'.cs = \text{true} \hspace{1cm} \text{[ENTRYCTX]} \]

\[ n \rightarrow n' \in G^R_{\text{pag}} \hspace{1cm} n.cs = \text{true} \hspace{1cm} \text{[ExitCTX]} \]

\[ n \rightarrow n' \in G^R_{\text{pag}} \hspace{1cm} n.cs = \text{true} \hspace{1cm} \text{[PROP]} \]
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**
- antlr: 48/32
- bloat: 6759/40
- chart: 8756/35
- eclipse: 5442/65
- fop: 3668/87

**EAGLE**
- antlr: 3513/38
- bloat: 5029/42
- chart: 3768/31
- eclipse: 4878/33
- fop: 5809/40

**ZIPPER**
- antlr: 3552/44
- bloat: 6729/45
- chart: 5641/56
- eclipse: 9109/46
- fop: 5581/41

**EAGLE**
- antlr: 3311/41
- bloat: 5036/46
- chart: 4267/55
- eclipse: 5426/53
- fop: 5893/48

**APPLICATIONS**
- antlr
- bloat
- chart
- eclipse
- fop
- lusearch
- pmdd
- xalan
- checkstyle
- findbugs
- JPC
Evaluation

Context-Insensitive

Context-Sensitive

ZIPPER
34613 952622
52622 39558

EAGLE
55695 56061
49 50

antlr  bloat  chart  eclipse  fop  luindex

ZIPPER
32214 34657
54299 97355

EAGLE
46435 76291
57 51

lusearch  pmd  xalan  checkstyle  findbugs  JPC
Conclusion

EAGLE

“Linear pre-analysis”