IC637 Program Analysis

Lecture 8: Selective Context Sensitivity

Minseok Jeon

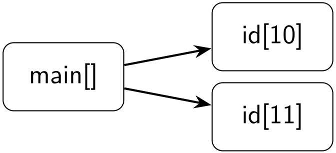
2025 Fall

Review: 1-Call-Site Sensitivity

- 1-**call-site sensitive** analysis can prove the castings are always safe.
- call-site sensitivity uses the call sites as context elements.

```
class A{}
class B{}
class C{
  static Object id(Object v) {
    return v;
  }
  void main(String[] args) {
    A a = new A();//l1
    B b = new B();//l2
    A a2 = (A)id(a);//query1
    B b2 = (B)id(b);//query2
  }
}
```

1-**call-site** sensitive analysis



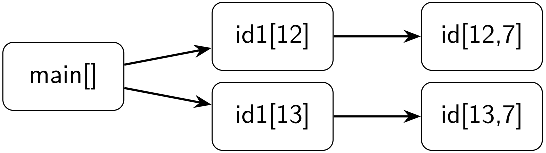
$$a[] \rightarrow \{l_1[]\} \; b[] \rightarrow \{l_2[]\}$$
$$v[10] \rightarrow \{l_1[]\} \; v[11] \rightarrow \{l_2[]\}$$
$$a2[] \rightarrow \{l_1[]\} \; b2[] \rightarrow \{l_2[]\}$$

Review: 2-Call-Site Sensitive Analysis

- conventional 2-**call-site sensitive** uses the caller methods' contexts.

```
class A{} class B{}
class C{
  static Object id(Object v) {
    return v;
  }
  Object id1(Object v) {
    return this.id(v);
  }
  void main(String[] args) {
    A a = new A();//l1
    B b = new B();//l2
    A a2 = (A)id1(a);//query1
    B b2 = (B)id1(b);//query2
  }
}
```

2-**call-site** sensitive analysis



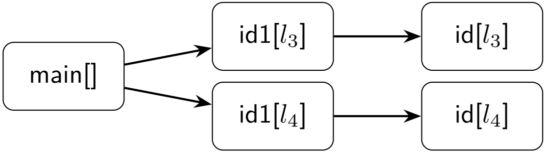
$$a[] \rightarrow \{l_1[]\}\ b[] \rightarrow \{l_2[]\}$$
$$v[10] \rightarrow \{l_1[]\}\ v[11] \rightarrow \{l_2[]\}$$
$$a2[] \rightarrow \{l_1[]\}\ b2[] \rightarrow \{l_2[]\}$$

Review: Object Sensitivity

- Object sensitivity uses the receiver objects as context elements.

```
class A{} class B{}
class C{
  Object id(Object v) {
    return v;
  }
  Object id1(Object v) {
    return this.id(v);
  }
}
class D{
  void main(String[] args) {
    A a = new A();//l1
    B b = new B();//l2
    C c1 = new C();//l3
    C c2 = new C();//l4
    A a3 = (A)c1.id1(a);//query3
    B b3 = (B)c2.id1(b);//query4
  }
}
```

1-**object** sensitive analysis



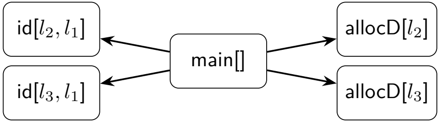
$$a[] \rightarrow \{l_1[]\} \; b[] \rightarrow \{l_2[]\}$$
$$id1.v[l_3] \rightarrow \{l_1[]\} \; id1.v[l_4] \rightarrow \{l_2[]\}$$
$$id.v[l_3] \rightarrow \{l_1[]\} \; id.v[l_4] \rightarrow \{l_2[]\}$$
$$a2[] \rightarrow \{l_1[]\} \; b2[] \rightarrow \{l_2[]\}$$

Review: 2-Object Sensitivity

- 2-**object sensitive** analysis uses the heap contexts of the receiver objects.

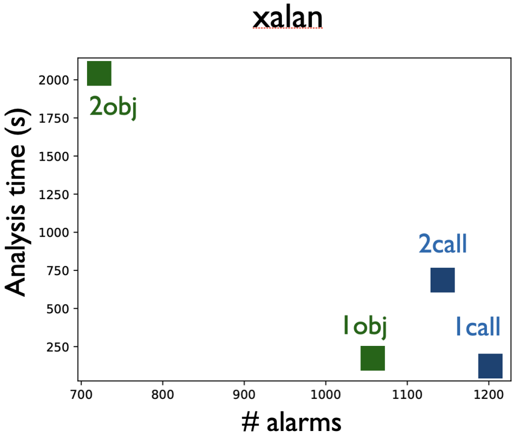
```
class C{
  D allocD() {
    return new D();}//l1
}
class D{
  Object id(Object v) {
    return v;}
}
class E{
  void main(String[] args) {
    C c1 = new C();//l2
    C c2 = new C();//l3
    D d1 = c1.allocD();
    D d2 = c2.allocD();
    A a = (A)d1.id(new A());//l4
    B b = (B)d2.id(new B());//l5
}}
```

2-**object** sensitive analysis



$$c1[] \rightarrow \{l_2[]\} \; c2[] \rightarrow \{l_3[]\}$$
$$d1[] \rightarrow \{l_1[l_2]\} \; d2[] \rightarrow \{l_1[l_3]\}$$
$$v[l_2, l_1] \rightarrow \{l_4[]\} \; v[l_3, l_1] \rightarrow \{l_5[]\}$$
$$a[] \rightarrow \{l_4[]\} \; b[] \rightarrow \{l_5[]\}$$

Performance of Context Sensitivities in Practice



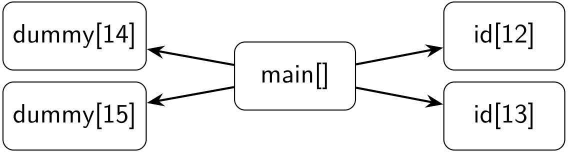
- **Problem:** k-context sensitivity is precise but very expensive.

Necessity of Selective Context Sensitivity

- In the following example, dummy does not need to be analyzed context sensitively.

```
class A{} class B{}
class C{
  static Object id(Object v) {
    return v;
  }
  static void dummy(){
    return;
  }
  void main(String[] args) {
    A a1 = new A();//l1
    B b1 = new B();//l2
    A a2 = (A)id(a1);//query1
    B b2 = (B)id(b1);//query2
    dummy();
    dummy();
  }
}
```

1-call-site sensitivity



$$a1[] \rightarrow \{l_1[]\} \; b1[] \rightarrow \{l_2[]\}$$
$$v[12] \rightarrow \{l_1[]\} \; v[13] \rightarrow \{l_2[]\}$$
$$a2[] \rightarrow \{l_1[]\} \; b2[] \rightarrow \{l_2[]\}$$

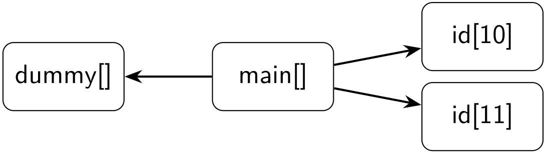
Necessity of Selective Context Sensitivity

- Applying 1-call-site sensitivity to the method `id` and context insensitive analysis to `dummy` still proves the queries.

```
1  class A{} class B{}
2  class C{
3    static Object id(Object v) {
4      return v;
5    }
6    static void dummy(){return;}
7    void main(String[] args) {
8      A a1 = new A();
9      B b1 = new B();
10     A a2 = (A)id(a1);//query1
11     B b2 = (B)id(b1);//query2
12     dummy();
13     dummy();
14 }}
```

`id`: 1-call-site sensitivity

`dummy`: context insensitive

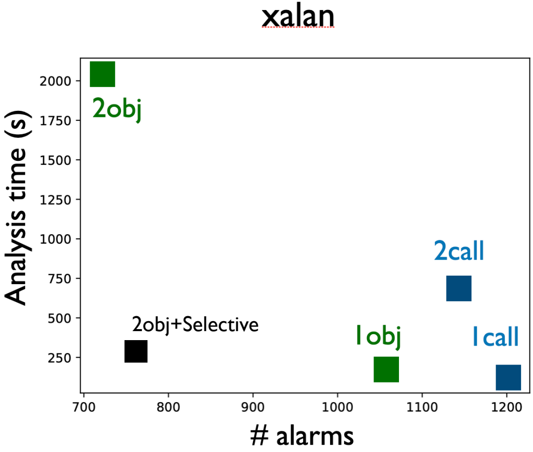


$$a[] \rightarrow \{l_1[]\} \; b[] \rightarrow \{l_2[]\}$$
$$v[10] \rightarrow \{l_1[]\} \; v[11] \rightarrow \{l_2[]\}$$
$$a2[] \rightarrow \{l_1[]\} \; b2[] \rightarrow \{l_2[]\}$$

- The goal of selective context sensitivity is to identify suitable context sensitivity to each method call before analysis.

Necessity of Selective Context Sensitivity

- Selective context sensitivity can improve the performance of static analysis.



Analysis Rule

- Rule for method calls (e.g., $VCall$) needs to be updated as follows:

$$\textbf{Merge}(\mathbf{depth}, heap, hctx, invo, callerCtx) = calleeCtx,$$
$$VarPointsTo(this, calleeCtx, heap, hctx),$$
$$CallGraph(invo, callerCtx, toMeth, calleeCtx) \leftarrow$$
$$VCall(base, sig, invo, inMeth), CallGraph(\_, \_, inMeth, callerCtx),$$
$$VarPointsTo(base, callerCtx, heap, hctx),$$
$$HeapType(heap, heapT), LookUp(heapT, sig, toMeth),$$
$$ThisVar(toMeth, this), \mathbf{ApplyDepth}(\mathbf{toMeth}, \mathbf{depth}).$$

- Key difference 1: $\mathbf{ApplyDepth}(\mathbf{toMeth}, \mathbf{depth})$
- Key difference 2: **Merge** takes another parameter $\mathbf{depth}$

- Define **Merge** for selective call-site sensitivity:
- Define **Merge** for selective object sensitivity:

Analysis Rule

- **Merge** for selective 2-call-site sensitivity:

$$\textbf{Merge}(depth, heap, hctx, invo, ctx) = \lceil ctx \mathbin{+\!\!+} invo \rceil_{depth}$$

- **Merge** for selective object sensitivity:

$$\textbf{Merge}(depth, heap, hctx, invo, ctx) = \lceil hctx \mathbin{+\!\!+} heap \rceil_{depth}$$

Modeling Static Analyzer

- Given a program P, a parametric static analyzer F_P is modeled as follows.

$$F_P : \mathcal{A}_P \to \mathcal{P}(\mathbb{Q}_P) \times \mathbb{N}.$$

where $\mathcal{A}_P : \mathbb{M}_P \to \{0, 1, 2\}$ denotes mappings from methods to depths, $\mathbb{Q}_P$ denotes sets of proven queries, and $\mathbb{N}$ denotes analysis costs.

- Given a mapping $\mathbf{a} \in \mathcal{A}$, so-called abstraction, we can generate input facts

$$ApplyDepth(Meth : M, Depth : Int)$$[1]

For example, if a method $meth$ is mapped to depth $depth$ in the abstraction $\mathbf{a}$, we generate an input fact $ApplyDepth(meth, depth)$.

[1] M: the set of method identifiers, Int: the set of integers.

Modeling Static Analyzer

- Given a program P, a parametric static analyzer F_P is modeled as follows.

$$F_P : \mathcal{A}_P \rightarrow \mathcal{P}(\mathbb{Q}_P) \times \mathbb{N}.$$

where $\mathcal{A}_P : \mathbb{M}_P \rightarrow \{0, 1, 2\}$ denotes mappings from methods to depths, $\mathbb{Q}_P$ denotes sets of proven queries, and $\mathbb{N}$ denotes analysis costs.

- For convenience, we define two projection functions: $proved(F_P(a))$ returns the set of proven queries, and $cost(F_P(a))$ returns the analysis cost $(a \in \mathcal{A}_P)$.

- Property (monotonicity): assigning a deeper depth to a method call-site does not degrade analysis precision.

Monotonicity of Analysis

- Two mappings $\mathbf{a}, \mathbf{a}' \in \mathcal{A}_P$ can be ordered as follows:

$$\forall m \in \mathbb{M}_P . \mathbf{a}(m) \leq \mathbf{a}'(m) \iff \mathbf{a} \sqsubseteq \mathbf{a}'$$

Using a bigger mapping does not degrade analysis precision:

$$\mathbf{a} \sqsubseteq \mathbf{a}' \implies \mathit{proved}(F_P(\mathbf{a})) \subseteq \mathit{proved}(F_P(\mathbf{a}'))$$

Open Challenge

- How can we find the right context sensitivity for each method call?

Machine Learning-based Selective Context Sensitivity

- Machine learning-based approach can be used:

Data-Driven Context-Sensitivity for Points-to Analysis

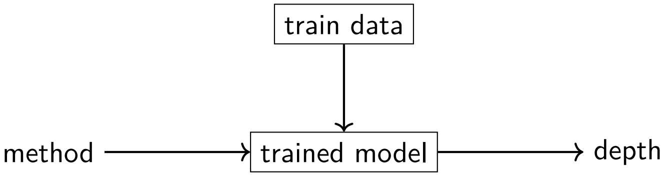
SEHUN JEONG, Korea University, Republic of Korea
MINSEOK JEON*, Korea University, Republic of Korea
SUNGDEOK CHA, Korea University, Republic of Korea
HAKJOO OH†, Korea University, Republic of Korea

We present a new data-driven approach to achieve highly cost-effective context-sensitive points-to analysis for Java. While context-sensitivity has greater impact on the analysis precision and performance than any other precision-improving techniques, it is difficult to accurately identify the methods that would benefit the most from context-sensitivity and decide how much context-sensitivity should be used for them. Manually

- Idea: learn a machine learning model that maps each method call to the right depth.

Machine Learning-based Selective Context Sensitivity

- Goal: learn a machine learning model that maps each method call to the right depth from a set of training data (e.g., a set of small programs).



- Assumption, if a model is effective for the training data, it is also effective for the test data.

Disjunctive Model

- Disjunctive model: The disjunctive model describes selective context sensitivity using boolean formulas.

Let $\mathbb{A} = \{a_1, a_2, \ldots, a_m\}$ be a set of atomic features, where each atomic feature is a predicate over methods (e.g., methods take more than 1 parameter):

$$a_i(P) : \mathbb{M}_P \to \{\mathit{true}, \mathit{false}\}.$$

Given a program P, a formula f represents a set of program components as follows:

$$\begin{array}{rcl} [\![\mathit{true}]\!]_P & = & \mathbb{M}_P \\ [\![\mathit{false}]\!]_P & = & \emptyset \\ [\![a_i]\!]_P & = & \{c \in [\![\mathit{true}]\!]_P \mid a_i(P)(c) = \mathit{true}\} \end{array} \qquad \begin{array}{rcl} [\![\neg f]\!]_P & = & [\![\mathit{true}]\!]_P \setminus [\![f]\!]_P \\ [\![f_1 \wedge f_2]\!]_P & = & [\![f_1]\!]_P \cap [\![f_2]\!]_P \\ [\![f_1 \vee f_2]\!]_P & = & [\![f_1]\!]_P \cup [\![f_2]\!]_P \end{array}$$

Disjunctive Model

- Let (f_1, f_2) be a pair of mutually exclusive selective formulas. Then, the model $\mathcal{M}^{(f_1,f_2)}$ assigns a depth to each formula as follows:

$$\mathcal{M}^{(f_1,f_2)}(P) = \lambda m \in \mathbb{M}_P. \begin{cases} 2 & \text{if } m \in [\![f_2]\!]_P \\ 1 & \text{if } m \in [\![f_1]\!]_P \\ 0 & \text{otherwise} \end{cases}$$

Learning Problem

Given a training set $\mathbf{P} = \{P_1, P_2, \ldots, P_n\}$, the learning problem is as follows:

Given a codebase $\mathbf{P}$, find (f_1, f_2) that maximizes

$$\sum_{P \in \mathbf{P}} proved(F_P(\mathcal{M}^{(f_1,f_2)}(P))) \text{ while minimizing } cost(F_P(\mathcal{M}^{(f_1,f_2)}(P))). \quad (1)$$

- Question: what does an ideal solution of the learning problem look like?

Solution of the Learning Problem (Minimal Solution)

Definition

Let $\mathbf{P}$ be a codebase and (f_1, f_2) be a parameter. We say (f_1, f_2) is a minimal solution of the learning problem (1) if

1. (f_1, f_2) is precise enough: $\frac{\sum_{P\in\mathbf{P}} |proved(F_P(\mathcal{M}^{(f_1,f_2)}(P)))|}{\sum_{P\in\mathbf{P}} |proved(F_P(\lambda m.2))|} \geq \gamma$, and
2. there exists no solution smaller than (f_1, f_2): if (f_1', f_2') meets the precision constraint, i.e., $\frac{\sum_{P\in\mathbf{P}} |proved(F_P(\mathcal{M}^{(f_1',f_2')}(P)))|}{\sum_{P\in\mathbf{P}} |proved(F_P(\lambda m.2(P)))|} \geq \gamma$, and (f_1', f_2') is smaller than (f_1, f_2), i.e., $f_1' \sqsubseteq f_1$ and $f_2' \sqsubseteq f_2$, then (f_1', f_2') is equivalent to (f_1, f_2):

$$\forall P \in \mathbf{P}.\forall m \in \mathbb{M}_P.\mathcal{M}^{(f_1,f_2)}(P)(m) = \mathcal{M}^{(f_1',f_2')}(P)(m)$$

where $f \sqsubseteq f' \implies \forall P.f(P) \subseteq f'(P)$.

Challenge

- Learning the two formulas (f_1, f_2) at once is difficult.
- Suppose the search space for a formula is S, then the search space for (f_1, f_2) is $S \times S$.
- ***Question.*** How can we reduce the search space?

Our Approach to Reduce the Search Space

- To reduce the search space, we decompose the learning problem (1) into the following two subproblems.

The first subproblem is as follows:

$$\text{Find } f_2 \text{ that minimizes } \sum_{P \in \mathbf{P}} cost(F_P(\mathcal{M}^{(true, f_2)}(P)))$$

$$\text{while satisfying } \frac{\sum_{P \in \mathbf{P}} |proved(F_P(\mathcal{M}^{(true, f_2)}(P)))|}{\sum_{P \in \mathbf{P}} |proved(F_P(\lambda m.2(P)))|} \geq \gamma. \quad (2)$$

Solution to the first subproblem (2)

Definition

Let $\mathbf{P}$ be a codebase. We say f_2 is a minimal solution of the subproblem (2) if

1. f_2 is precise enough: $\frac{\sum_{P \in \mathbf{P}} |proved(F_P(\mathcal{M}^{(true, f_2)}(P)))|}{\sum_{P \in \mathbf{P}} |proved(F_P(\lambda m.2))|} \geq \gamma$, and
2. there exists no solution smaller than f_2: if f_2' meets the precision constraint, i.e., $\frac{\sum_{P \in \mathbf{P}} |proved(F_P(\mathcal{M}^{(true, f_2')}(P)))|}{\sum_{P \in \mathbf{P}} |proved(F_P(\lambda m.2(P)))|} \geq \gamma$, and f_2' is smaller than f_2, i.e., $f_2' \sqsubseteq f_2$, then f_2' is equivalent to f_2:

$$\forall P \in \mathbf{P}. \forall m \in \mathbb{M}_P. \mathcal{M}^{(true, f_2)}(P)(m) = \mathcal{M}^{(true, f_2')}(P)(m).$$

Our Approach to Reduce the Search Space

- Suppose we have a solution f_2 to the first subproblem. Then, we learn f_1.

The second subproblem is as follows:

$$\text{Find } f_1 \text{ that minimizes } \sum_{P \in \mathbf{P}} cost(F_P(\mathcal{M}^{(f_1,f_2)}(P)))$$

$$\text{while satisfying } \frac{\sum_{P \in \mathbf{P}} |proved(F_P(\mathcal{M}^{(f_1,f_2)}(P)))|}{\sum_{P \in \mathbf{P}} |proved(F_P(\lambda m.2(P)))|} \geq \gamma. \quad (3)$$

Solution to the second subproblem (3)

Definition

Let $\mathbf{P}$ be a codebase. We say f_1 is a minimal solution of the subproblem (3) if

1. f_1 is precise enough: $\frac{\sum_{P \in \mathbf{P}} |proved(F_P(\mathcal{M}^{(f_1,f_2)}(P)))|}{\sum_{P \in \mathbf{P}} |proved(F_P(\lambda m.2))|} \geq \gamma$, and
2. there exists no solution smaller than f_1: if f_1' meets the precision constraint, i.e., $\frac{\sum_{P \in \mathbf{P}} |proved(F_P(\mathcal{M}^{(f_1',f_2)}(P)))|}{\sum_{P \in \mathbf{P}} |proved(F_P(\lambda m.2(P)))|} \geq \gamma$, and f_1' is smaller than f_1, i.e., $f_1' \sqsubseteq f_1$, then f_1' is equivalent to f_1:

$$\forall P \in \mathbf{P}. \forall m \in \mathbb{M}_P. \mathcal{M}^{(f_1,f_2)}(P)(m) = \mathcal{M}^{(f_1',f_2)}(P)(m).$$

Our Decomposition is Safe

- If f_2 and f_1 are solutions of the subproblems (2) and (3), respectively, then (f_1, f_2) is a solution of the learning problem (1).

Theorem

Let f_1, f_2 be minimal solutions of the two problems (2) and (3), respectively. Then, (f_1, f_2) is a minimal solution of the learning problem (1).

Our Decomposition is Safe

Proof.

- (Precision) the precision constraint of the learning problem (1) is satisfied as f_1 is a solution of the subproblem (3).
- (Minimality) Suppose $k \in \{1, 2\}. f'_k \sqsubseteq f_k$ and (f'_1, f'_2) meets the precision contraint.
  - As (f'_1, f'_2) meets the precision contraint, $(true, f'_2)$ also meets the precision constraint and $f'_2 \sqsubseteq f_2$. As f_2 is a solution to the subproblem (2), f_2 and f'_2 are equivalent.
  - As f_2 and f'_2 are equivalent, (f'_1, f_2) meets the precision constraint and $f'_1 \sqsubseteq f_1$. As f_1 is a solution to the problem (3) when f_2 is used, f'_1 and f_1 are equivalent.

□

- Detailed proof is provided in the paper[2].

[2] https://dgistpl.github.io/papers/oopsla17a.pdf

Overall Learning Procedure

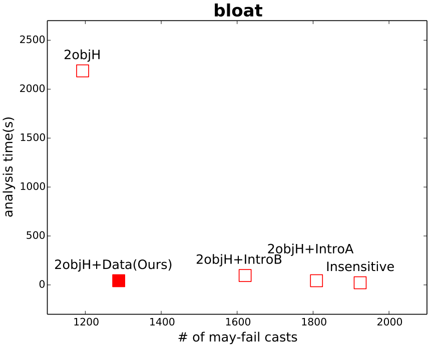
1. Learn f_2 that meets the precision constraint ($\frac{\sum_{P\in\mathbf{P}} |proved(F_P(\mathcal{M}^{(true,f_2)}(P)))|}{\sum_{P\in\mathbf{P}} |proved(F_P(\lambda m.2(P)))|} \geq \gamma$) of the subproblem (2) while minimizing the cost of $\sum_{P\in\mathbf{P}} cost(F_P(\mathcal{M}^{(true,f_2)}(P)))$.

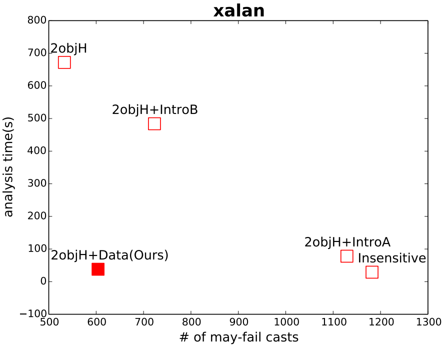
2. Learn f_1 that meets the precision constraint ($\frac{\sum_{P\in\mathbf{P}} |proved(F_P(\mathcal{M}^{(f_1,f_2)}(P)))|}{\sum_{P\in\mathbf{P}} |proved(F_P(\lambda m.2(P)))|} \geq \gamma$) of the subproblem (3) while minimizing the cost of $\sum_{P\in\mathbf{P}} cost(F_P(\mathcal{M}^{(f_1,f_2)}(P)))$.

- We can use standard learning algorithms to learn f_2 and f_1.

Evaluation Results

- Our selective context sensitivity enhances the balance between the precision and the cost.





Wrap-up

- Context sensitivity is essential for precise pointer analysis, but it can be expensive.
- Selective context sensitivity applies different depths to different methods to balance precision and cost.
- Machine learning can be used to learn the right context sensitivity for each method from training data.