Precise and Scalable Points-to Analysis via Data-Driven Context Tunneling

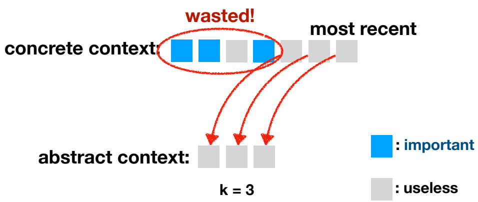
Minseok Jeon, Sehun Jeong, and Hakjoo Oh

SPLASH BOSTON 2018

KOREA UNIVERSITY

I. Problem of Most-recent-k Context Abstraction

- **Most-recent-*k*** policy often abandons important context elements



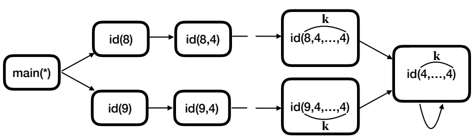
Motivating Example

- Two may-fail casting queries, which are safe
- `main` calls identity function `id` twice
- Value of `i` is unknown
- `id` calls itself `i+1` times recursively

```
1 class A{} class B{}
2 class C{
3   static Object id(Object v, int i){
4     return i >= 0 ? id(v, i-1): v;
5 }
6   public static void main(){
7     int i = input();
8     A a = (A) id(new A(),i);//query1
9     B b = (B) id(new B(),i);//query2
10  }
11 }
```

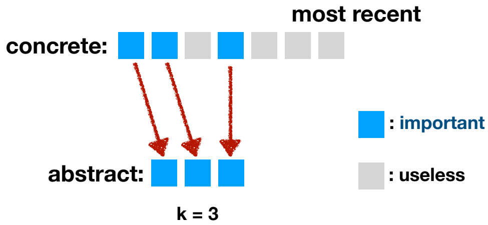
Conventional K-CFA

- k-call-site-sensitivity fails to prove the queries no matter what k value is used
- Since the i is unbounded, analysis eventually losses important contexts 8 and 9, becomes imprecise



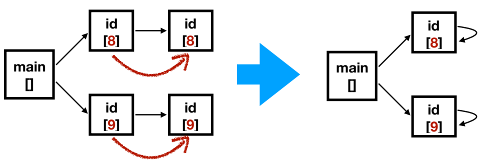
II. Our Approach: Context Tunneling

- Do not keep **most-recent-*k***
- Instead, keep **most-important-*k***



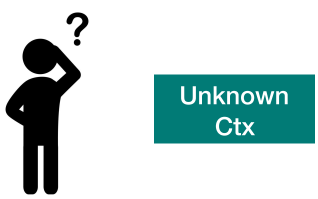
1-CFA with Context Tunneling

- Proves all the queries
- With context tunneling, method calls **selectively update callee context**
- When id calls id at line 4, callee method does not update context but inherit context from caller's (e.g., context tunneling is applied)

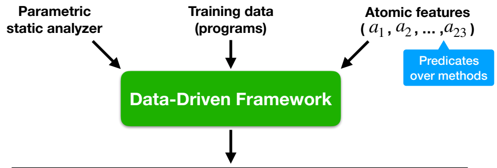


Challenge

- It is difficult to know right places for applying context tunneling (i.e., `id` called in `id`)
- Wrong choices of context tunneling may result imprecise *and* expensive analysis



III. Data-Driven Context Tunneling



- $\mathbf{f}_{\mathbf{caller}}$: Property of caller methods
  $(\neg a_6 \wedge a_8 \wedge \neg a_{10} \wedge \neg a_{11} \wedge a_{14} \wedge a_{15} \wedge \neg a_{16} \wedge \neg a_{17} \wedge \neg a_{18} \wedge \neg a_{19} \wedge \neg a_{20} \wedge \neg a_{22}) \vee$
  $(a_1 \wedge a_2 \wedge \neg a_3 \wedge \neg a_4 \wedge \neg a_6 \wedge a_8 \wedge \neg a_9 \wedge \neg a_{10} \wedge \neg a_{11} \wedge a_{12} \wedge a_{14} \wedge a_{15} \wedge \ldots) \vee$
  $(a_1 \wedge \neg a_2 \wedge \neg a_3 \wedge a_4 \wedge \neg a_6 \wedge \neg a_7 \wedge \wedge a_8 \wedge \neg a_9 \wedge \neg a_{10} \wedge \neg a_{11} \wedge \neg a_{12} \wedge \ldots)$
- $\mathbf{f}_{\mathbf{callee}}$: Property of callee methods
  $(a_1 \wedge \neg a_2 \wedge \neg a_3 \wedge \neg a_6 \wedge \neg a_9 \wedge a_{11} \wedge \wedge \neg a_{13} \wedge a_{14} \wedge a_{15} \wedge \neg a_{16} \wedge \neg a_{17} \wedge \ldots) \vee$
  $(a_1 \wedge \neg a_3 \wedge \neg a_4 \wedge a_7 \wedge \neg a_9 \wedge a_{12} \wedge a_{14} \wedge a_{15} \wedge \neg a_{16} \wedge \neg a_{19} \wedge \neg a_{21}) \vee$
  $(\neg a_3 \wedge a_6 \wedge \neg a_9 \wedge a_{14} \wedge a_{15} \wedge \neg a_{18} \wedge \neg a_{19} \wedge \neg a_{23})$

Tunneling Heuristic

- A set of relations T between two methods

$$\mathcal{T} \subseteq \mathbb{M} \times \mathbb{M}$$

- T means when contexts should not be updated
- Method m is called under its parent method p
- Callee context is constructed as follows:

$$calleeCtx = \begin{cases} \lceil parCtx +\!\!+ elem \rceil_{maxK} & \text{if } (p, m) \notin \mathcal{T} \\ \lceil parCtx \rceil_{maxK} & \text{if } (p, m) \in \mathcal{T} \end{cases}$$

Learning Model for Tunneling

- Two boolean formulas <f1, f2> is our model's parameter
- Given parameter, the model generates the tunneling relation for a target program as follows:

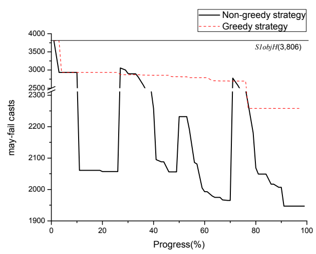
$$\{(m_1, m_2) \in \mathbb{M}_P \times \mathbb{M}_P \mid m_1 \in [\![f_1]\!]_P \vee m_2 \in [\![f_2]\!]_P\}.$$

Learning Problem

Find parameter <f1, f2> that maximizes analysis precision while it is scalable than <false, false> (i.e., without tunneling) over training programs.

IV. Learning in Non-monotonic Space

- Context tunneling heuristics are not equipped with precision order
- Our learning algorithm repeats exploration and exploitation steps to avoid local minima



V. Evaluation

- Ours (S1objH+T) is more precise and faster than conventions.

		S1objH+T	S1objH	S2objH
xalan	alarms	**572**	1,129	623
	costs	**64**	187	465
chart	alarms	**876**	2,290	915
	costs	**73**	1,299	488
bloat	alarms	**1,251**	1,931	1,326
	costs	**464**	707	2,211
jython	alarms	**837**	1,308	timeout
	costs	**425**	730	-