Data-Driven Static Analysis: Combining Machine Learning and Program Analysis

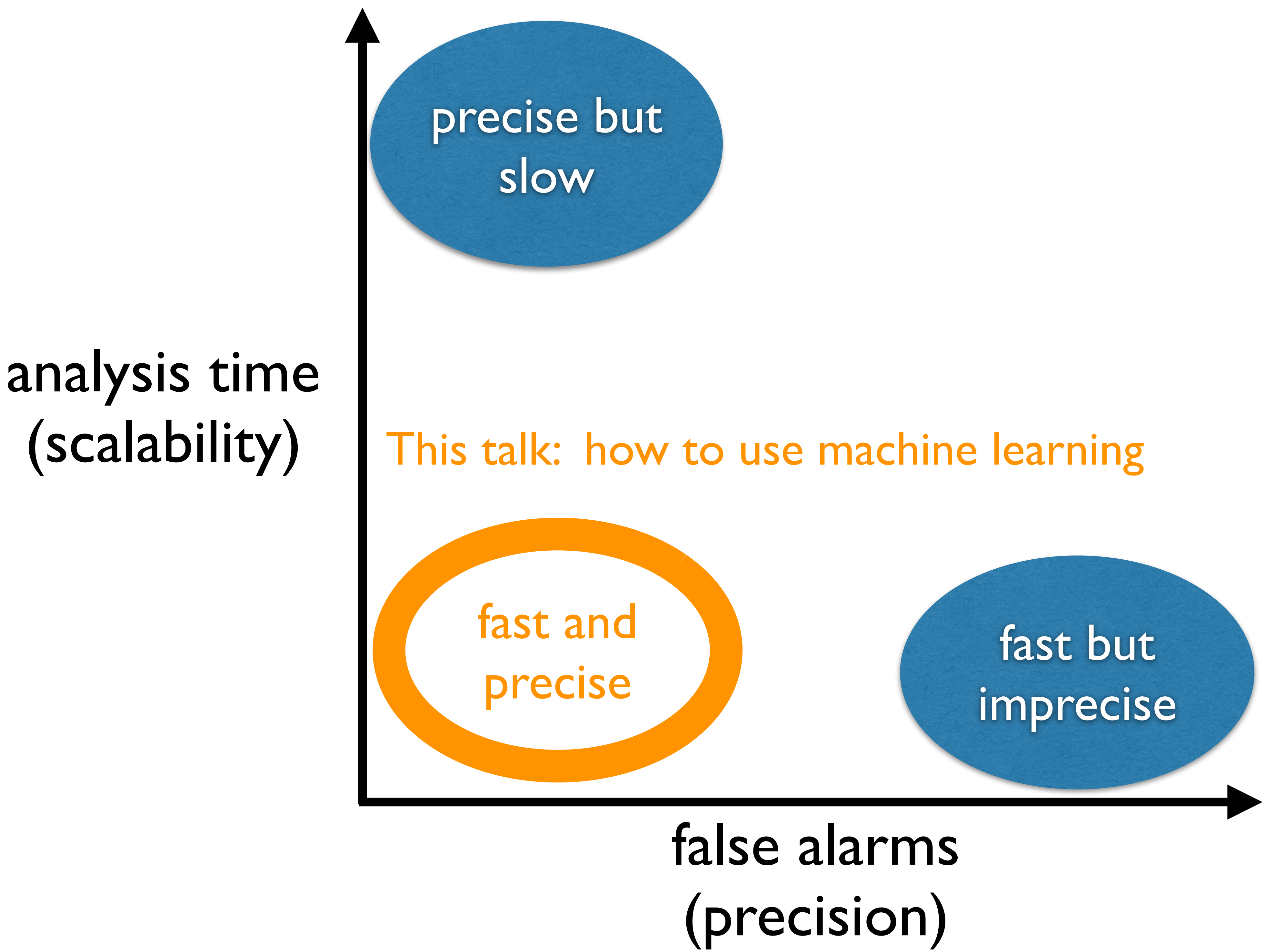
Hakjoo Oh

Korea University

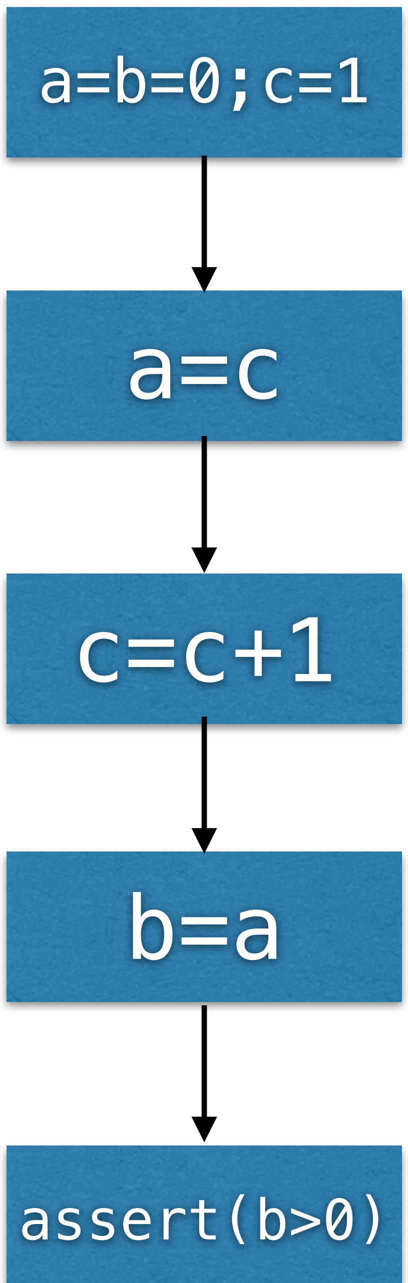
28 January 2022 @POPL'22 Virtual Workshop

(co-work with Minseok Jeon, Sehun Jeong, Sooyoung Cha, Seongjoon Hong, Junhee Lee, Kwangkeun Yi, Hongseok Yang)

Precision vs. Scalability Tradeoff in Static Analysis



Example 1: Flow Sensitivity



a	[0,0]
b	[0,0]
c	[1,1]

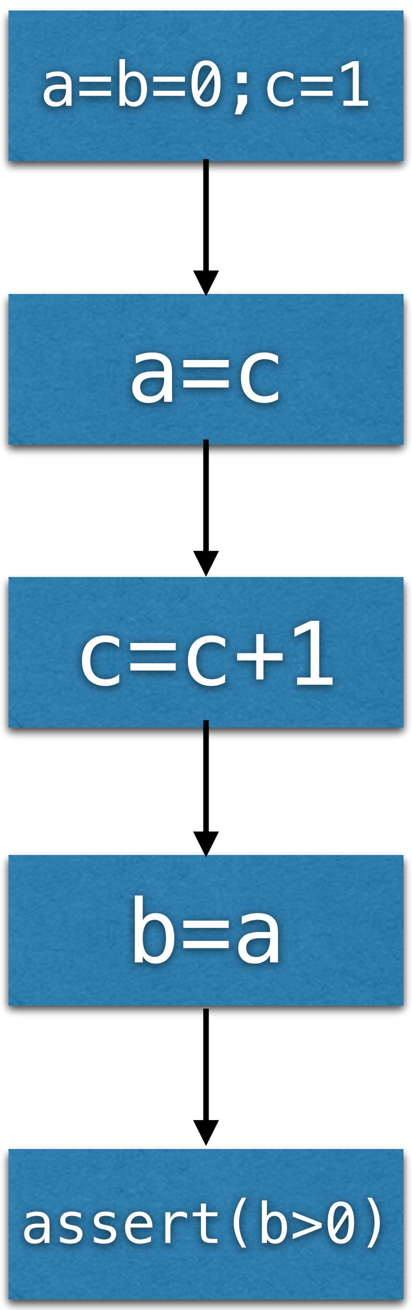
a	[1,1]
b	[0,0]
c	[1,1]

a	[1,1]
b	[0,0]
c	[2,2]

a	[1,1]
b	[1,1]
c	[2,2]

precise but costly

Flow Insensitivity



a	[0,0]
b	[0,0]
c	[1,1]

a	[1,1]
b	[0,0]
c	[1,1]

a	[1,1]
b	[0,0]
c	[2,2]

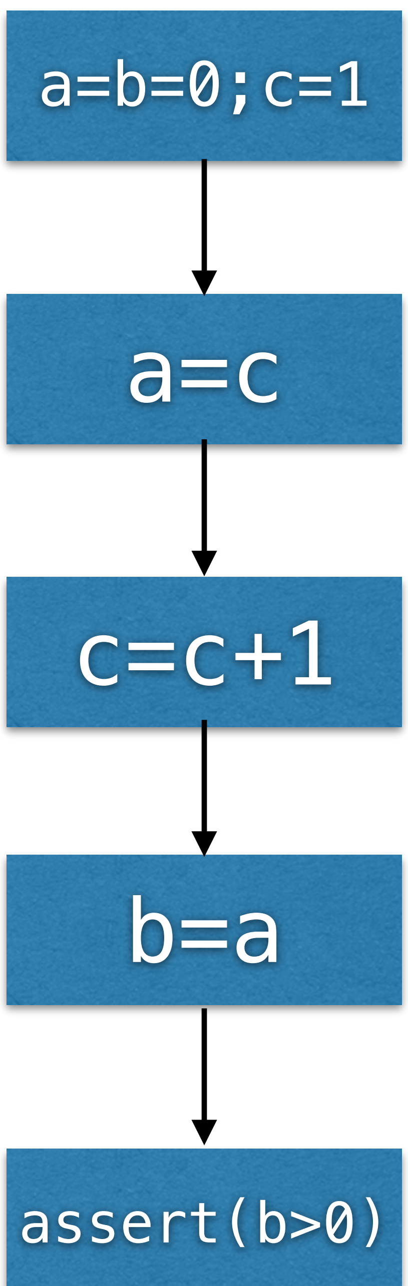
a	[1,1]
b	[1,1]
c	[2,2]

a	[0,+∞]
b	[0,+∞]
c	[1,+∞]

precise but costly

cheap but imprecise

Selective Flow Sensitivity



FS : {a,b}

a	[0,0]
b	[0,0]

a	[1,+∞]
b	[0,0]

a	[1,+∞]
b	[0,0]

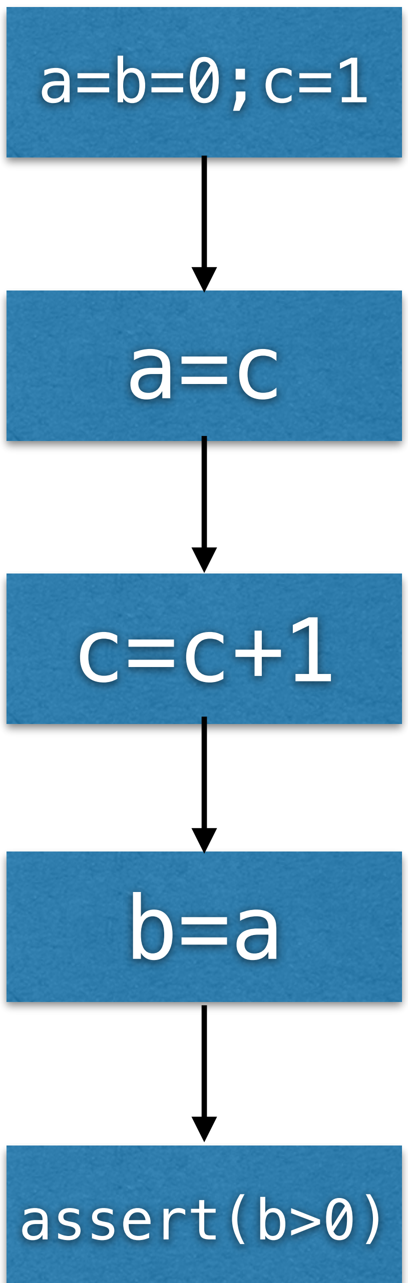
a	[1,+∞]
b	[1,+∞]

FI : {c}

c	[1,+∞]

cheap and precise

Selective Flow Sensitivity



FS : {b,c}

b	[0,0]
c	[1,1]

b	[0,0]
c	[1,1]

b	[0,0]
c	[2,2]

b	[0,+∞]
c	[2,2]

fail to prove

FI : {a}

a	[0,+∞]

Challenging Search Problem

"How to find a good program abstraction?"

- Intractably large search space, if not infinite
  - e.g., $2^{|Var|}$ difference abstractions for flow sensitivity
- Most of them are too imprecise or costly
  - P({a,b,c}) = {~~∅,{a},{b},{c}~~,{a,b},~~{b,c}~~,~~{a,c}~~,~~{a,b,c}~~}

The only nontrivial abstraction that proves assertion

A fundamental problem in static analysis
=> Use machine learning to solve this problem

Example 2: Context Sensitivity

```
    int h(n) {ret n;}

    void f(a) {
c1:   x = h(a);
      assert(x > 0);  // Query      holds always
c2:   y = h(input());
    }

c3: void g() {f(8);}

    void m() {
c4:   f(4);
c5:   g();
c6:   g();
    }
```

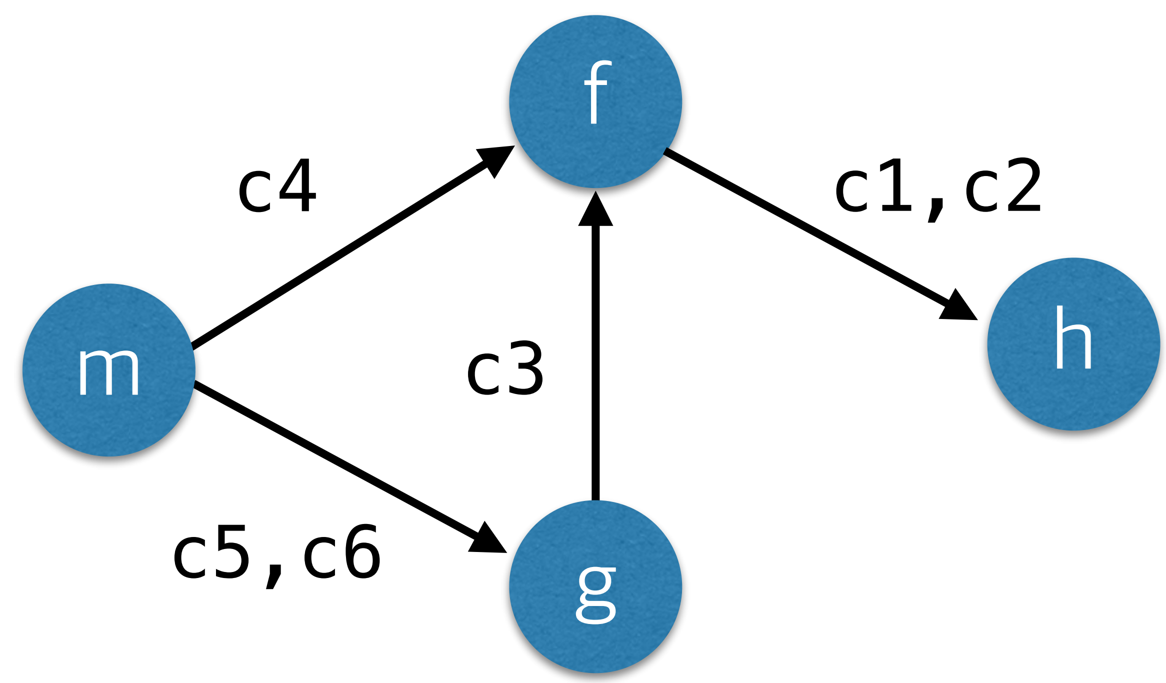
Context Insensitivity

```
    int h(n) {ret n;}

    void f(a) {
c1:   x = h(a);
      assert(x > 0);
c2:   y = h(input());
    }

c3: void g() {f(8);}

    void m() {
c4:   f(4);
c5:   g();
c6:   g();
    }
```



cheap but imprecise

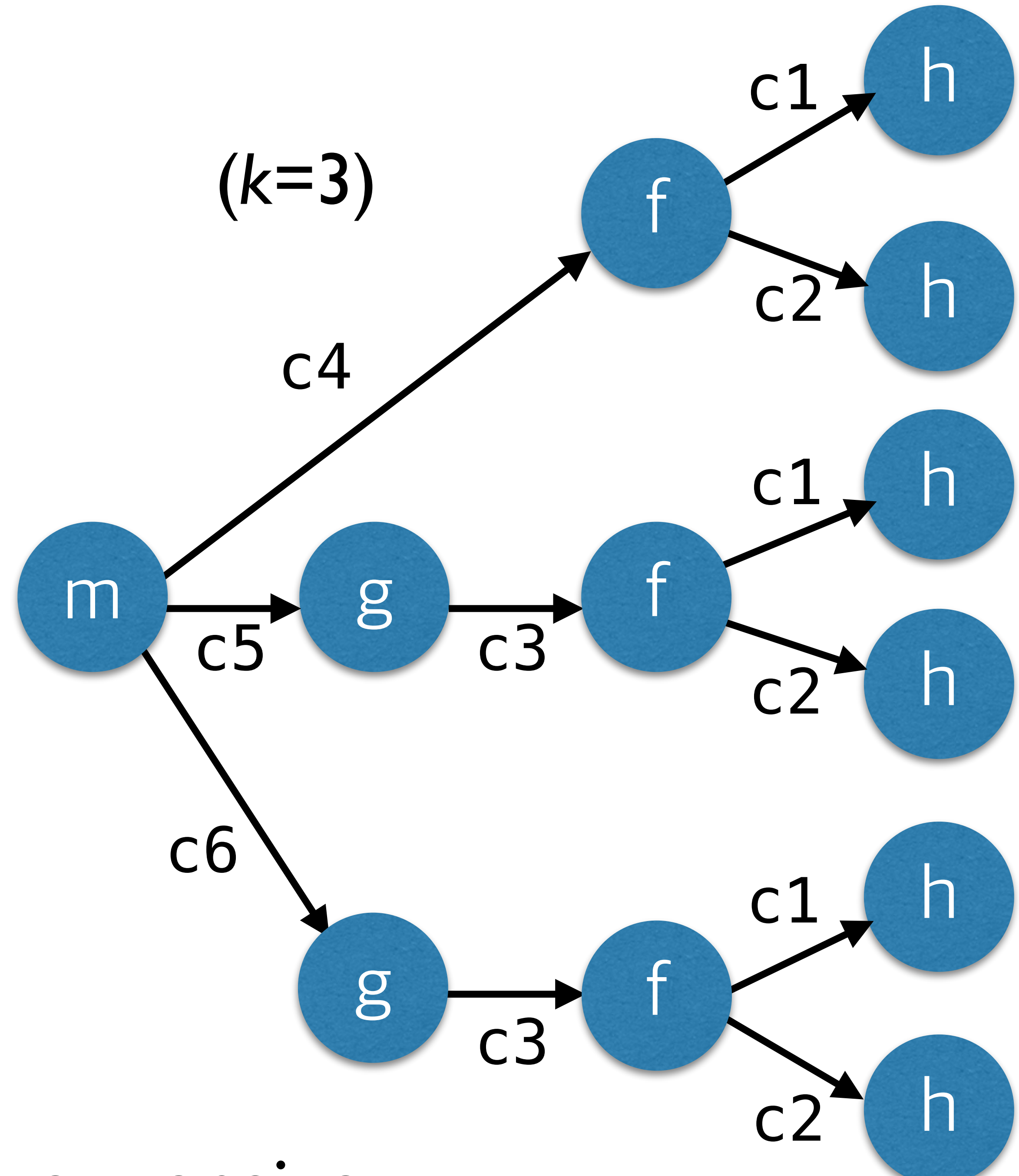
k-Bounded Context Sensitivity (*k*-CFA)

```
    int h(n) {ret n;}

    void f(a) {
c1:   x = h(a);
      assert(x > 0);
c2:   y = h(input());
    }

c3: void g() {f(8);}

    void m() {
c4:   f(4);
c5:   g();
c6:   g();
    }
```



precise but expensive

Selective *k*-CFA

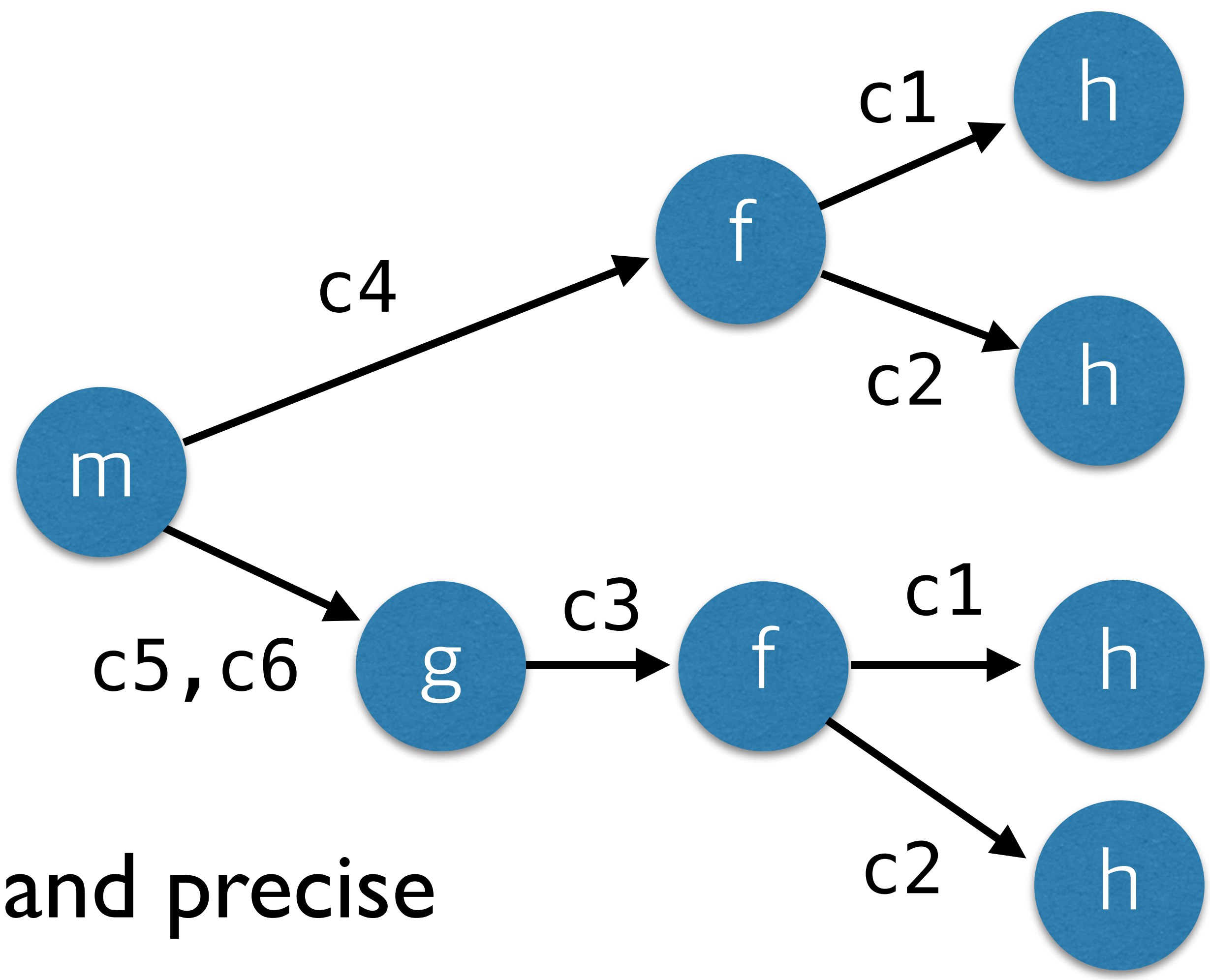
```
    int h(n) {ret n;}

    void f(a) {
c1:   x = h(a);
      assert(x > 0);
c2:   y = h(input());
    }

c3: void g() {f(8);}

    void m() {
c4:   f(4);
c5:   g();
c6:   g();
    }
```

Apply 2-ctx-sens: {h}
Apply 1-ctx-sens: {f}
Apply 0-ctx-sens: {g, m}



cheap and precise

Challenging Search Problem

"How to find a good program abstraction?"

- Abstraction space:

$$Func \rightarrow \{0,1,\ldots,k\}$$

- $(k+1)^{|Func|}$ different abstractions

Our approach: use machine learning to tackle this problem

Our Data-Driven Approach

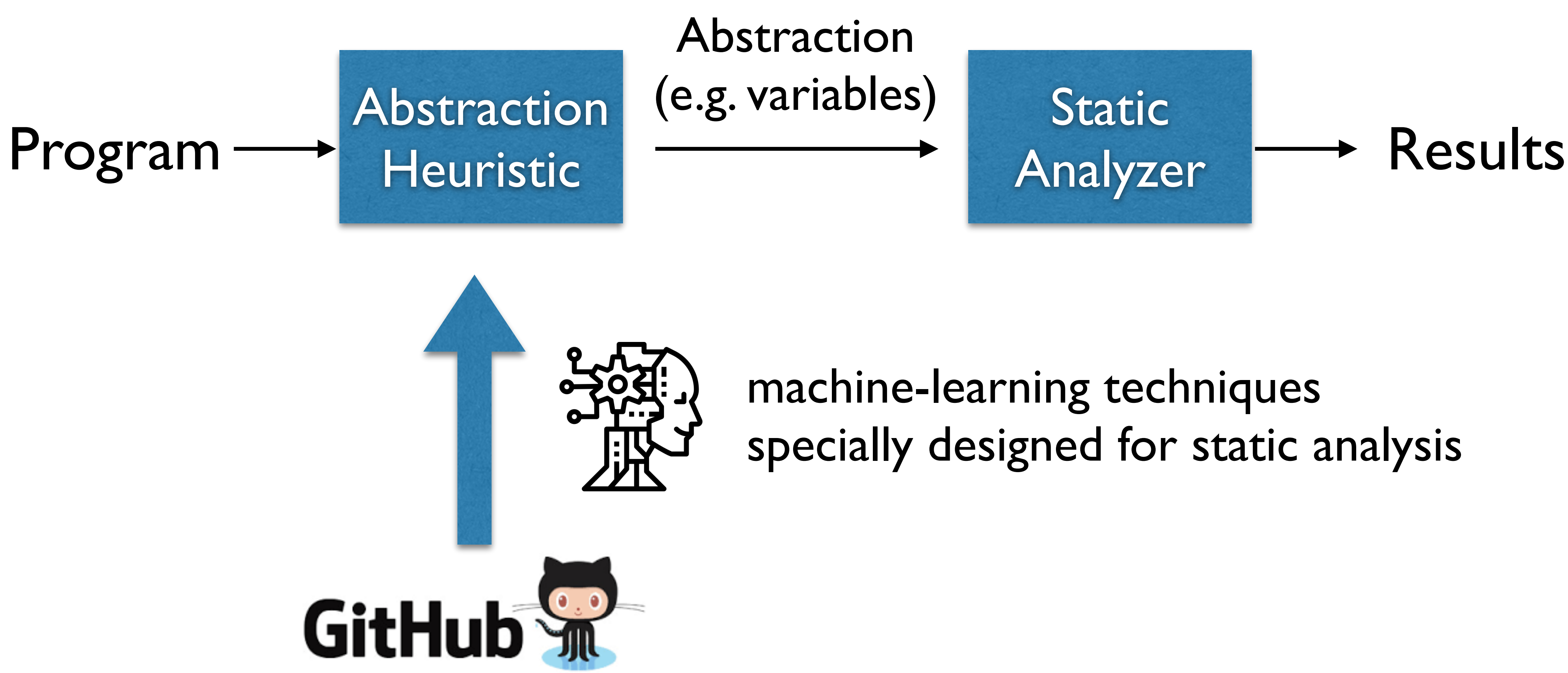
Program → **Abstraction Heuristic** → Abstraction (e.g. variables) → **Static Analyzer** → Results

Traditionally, abstraction heuristics developed manually by human experts:

… PLDI'14 POPL'17 PLDI'17 OOPSLA'18 FSE'18 OOPSLA'19 OOPSLA'21 …

=> nontrivial and time-consuming

Our Data-Driven Approach



- Automatic: little reliance on analysis designers
- Powerful: machine-tuning outperforms hand-tuning

Our Data-Driven Approach

ML tools developed for static analysis:

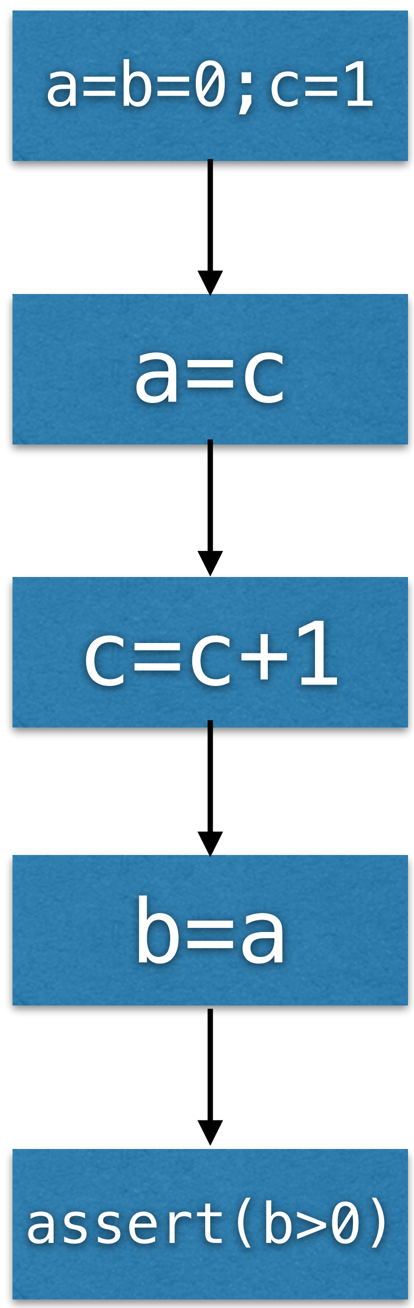
most successful, ~~focus of this talk~~

useful to illustrate

- Learning algorithm with linear model [OOPSLA'15]
- Learning algorithm with disjunctive model [OOPSLA'17a]
- Learning algorithm with automated feature generation [OOPSLA'17b]
- Learning algorithm for symbolic execution [ICSE'18]
- Learning algorithm for non-monotone analyses [OOPSLA'18]
- Learning algorithm with feature language [OOPSLA'20]
- Learning algorithm for boosting k-CFA [POPL'22]
- …

Data-Driven Static Analysis with Linear Models (OOPSLA'15)

Example: Flow Sensitivity



FS : {a,b}

a	[0,0]
b	[0,0]

a	[1,+∞]
b	[0,0]

a	[1,+∞]
b	[0,0]

a	[1,+∞]
b	[1,+∞]

FI : {c}

c	[1,+∞]

Settings

- $P \in \mathbb{P}$: an input program to analyze
- $\mathscr{A}_P$: the set of abstractions for P
  - $\mathbf{a} \in \mathscr{A}_P = Var_P \to \{0,1\} = 2^{Var_P}$
- $\mathbb{Q}_P$: the set of queries (assertions) in P
  - goal of static analysis is to prove as much as possible

Settings

Static analyzer is modeled by blackbox function F_P :

$$F_P : \mathscr{A}_P \rightarrow 2^{\mathbb{Q}_P} \times \mathbb{N}$$

- $Q \in 2^{\mathbb{Q}_P}$: assertions proved by the analysis
- $\mathbb{N}$: integer denoting cost (e.g., time, memory)
- cost($F_P(\mathbf{a})$) : cost of analysis with abstraction $\mathbf{a}$
- proved($F_P(\mathbf{a})$) : precision of analysis with abstraction $\mathbf{a}$

Machine Learning: Three Steps

1. Define a parameterized heuristic $\mathscr{H}_\Pi$:

$$\mathscr{H}_\Pi(P) : 2^{Var_P}$$

2. Define a learning objective as optimization problem:

“Find Π that maximizes analysis performance”

3. Solve the optimization problem via learning algorithm

1. Parameterized Heuristic

$$\mathscr{H}_{\Pi}(P) : 2^{Var_P}$$

(1) Represent program variables as feature vectors.

(2) Compute the score of each variable.

(3) Choose top-k variables with highest scores

(1) Features

$$\mathbb{A} = \{a_1, a_2, \ldots, a_n\}$$

- A feature is a predicate over variables:

$$a_i : Var \rightarrow \{0,1\}$$

- E.g., syntactic features for programs variables
  - Is it a local variable? or global variable?
  - Is it passed to a function as argument? (e.g., f(x))
  - Is it incremented by a constant value? (e.g., x=x+1)

(1) Features

- Represent each variable as a feature vector:

$$\mathbb{A}(x) = \langle a_1(x), a_2(x), a_3(x), a_4(x), a_5(x) \rangle$$

$$\mathbb{A}(x) = \langle 1,0,1,0,0 \rangle$$

$$\mathbb{A}(y) = \langle 1,0,1,0,1 \rangle$$

$$\mathbb{A}(z) = \langle 0,0,1,1,0 \rangle$$

(2) Scoring

- The parameter Π is a real-valued vector: e.g.,

$$\Pi = \langle 0.9, 0.5, -0.6, 0.7, 0.3 \rangle$$

- Compute scores of variables by linear combination:

$$\text{score}(x) = \langle 1,0,1,0,0 \rangle \cdot \langle 0.9, 0.5, -0.6, 0.7, 0.3 \rangle = 0.3$$
$$\text{score}(y) = \langle 1,0,1,0,1 \rangle \cdot \langle 0.9, 0.5, -0.6, 0.7, 0.3 \rangle = 0.6$$
$$\text{score}(z) = \langle 0,0,1,1,0 \rangle \cdot \langle 0.9, 0.5, -0.6, 0.7, 0.3 \rangle = 0.1$$

(3) Choose Top-k Variables

- Choose the top-k(%) variables based on their scores: e.g., when k=66%,

$$\begin{array}{l} \text{score}(x) = 0.3 \\ \text{score}(y) = 0.6 \\ \text{score}(z) = 0.1 \end{array} \quad \Longrightarrow \quad \{x,y\}$$

- In practice, choosing 10% of variables strikes the precision and cost balance well

2. Optimization Problem

- Goal of learning is to find good parameter Π from data:

$$\text{Codebase } \mathbf{P} = \{P_1, P_2, \ldots, P_m\} \quad \Longrightarrow \quad \Pi \in \mathbb{R}^n$$

- Formulated as the optimization problem:

$$\text{Find } \Pi \text{ that maximizes } \sum_{P \in \mathbf{P}} \text{proved}(F_P(\mathscr{H}_\Pi(P)))$$

analysis precision over the training data

3. Learning Algorithm

- Simple algorithm based on random sampling:

repeat N times

  pick $\Pi \in \mathbb{R}^n$ randomly

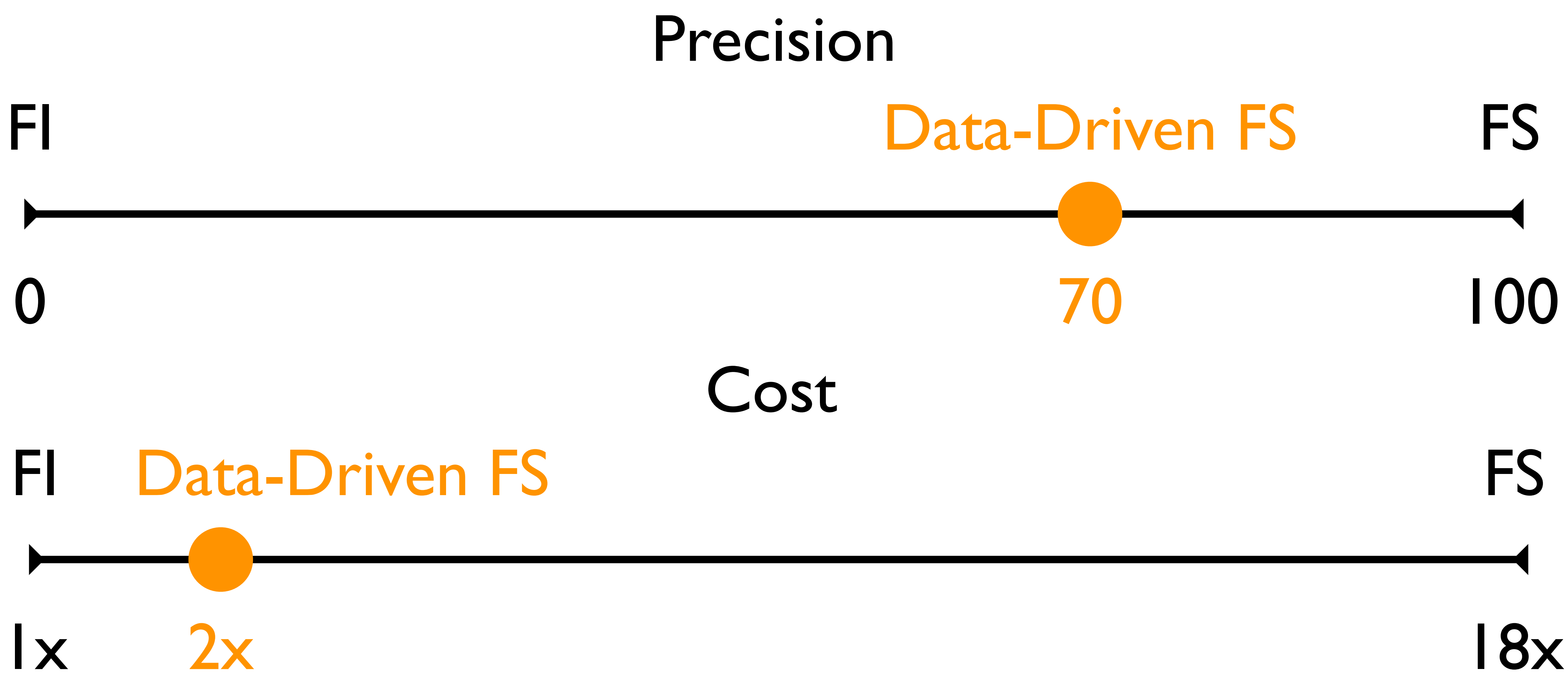
  evaluate $\sum_{P \in \mathbf{P}} \text{proved}(F_P(\mathscr{H}_\Pi(P)))$

return best Π found

- The algorithm can be improved with Bayesian optimization (details in paper)

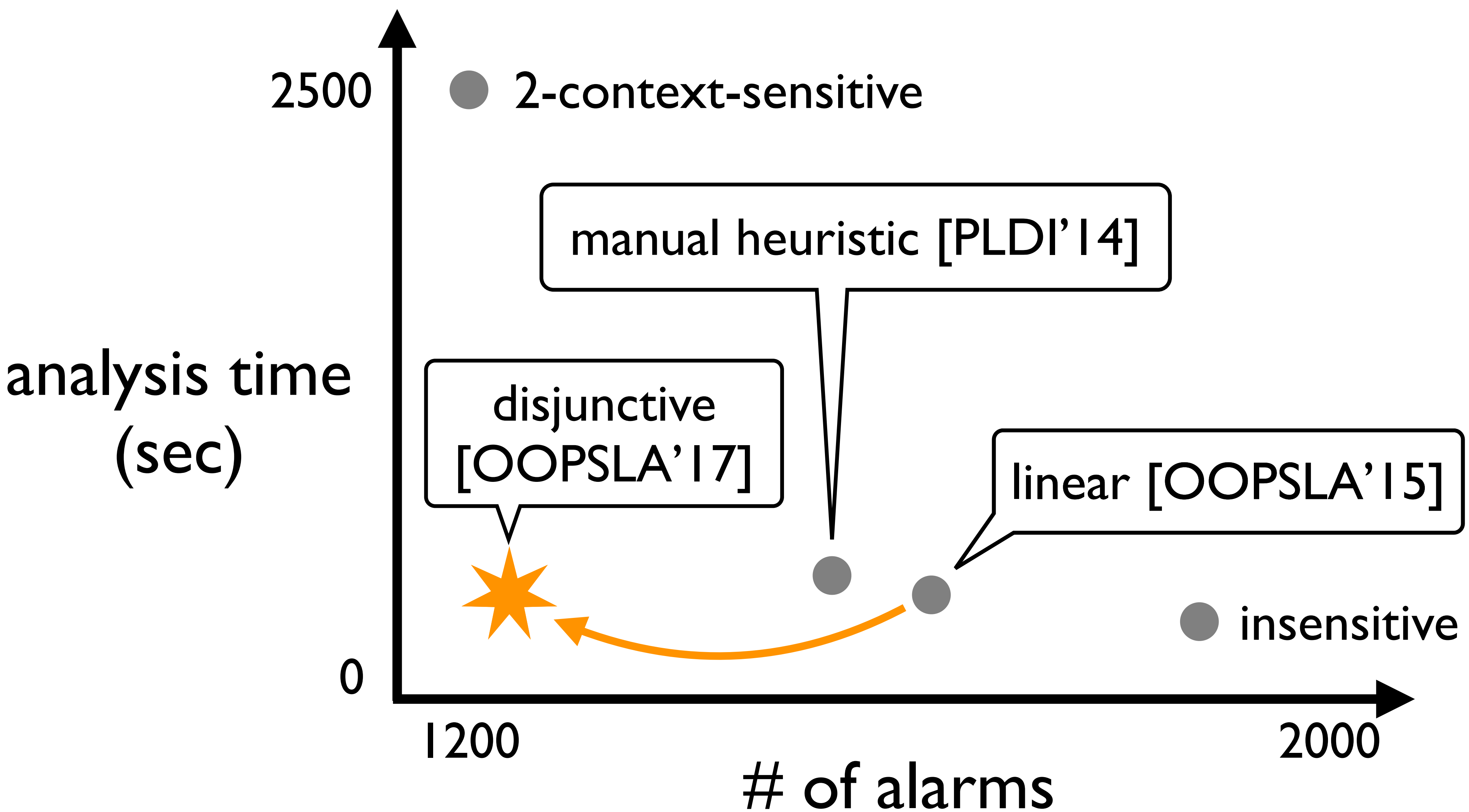
Effectiveness

- Implemented in Sparrow, an interval analyzer for C
  - 45 syntactic features for program variables
- Evaluated on 30 open-source programs
  - Training with 20 programs (12 hours) and evalation on 10



Limitation of Linear Method

- Not effective enough to beat manual heuristics
- E.g., context-sensitive pointer analysis (Java bloat)



Data-Driven Static Analysis with Disjunctive Models (OOPSLA'17)

Key Limitation

- Linear heuristic is not expressive to capture complex program properties

$$x : \{a_1, a_2\}$$
$$y : \{a_1\}$$
$$z : \{a_2\}$$
$$w : \emptyset$$

Can we select $\{x, w\}$?

- We need a method that allows disjunctions

$$(a_1 \wedge a_2) \vee (\neg a_1 \wedge \neg a_2)$$

Example: Context Sensitivity

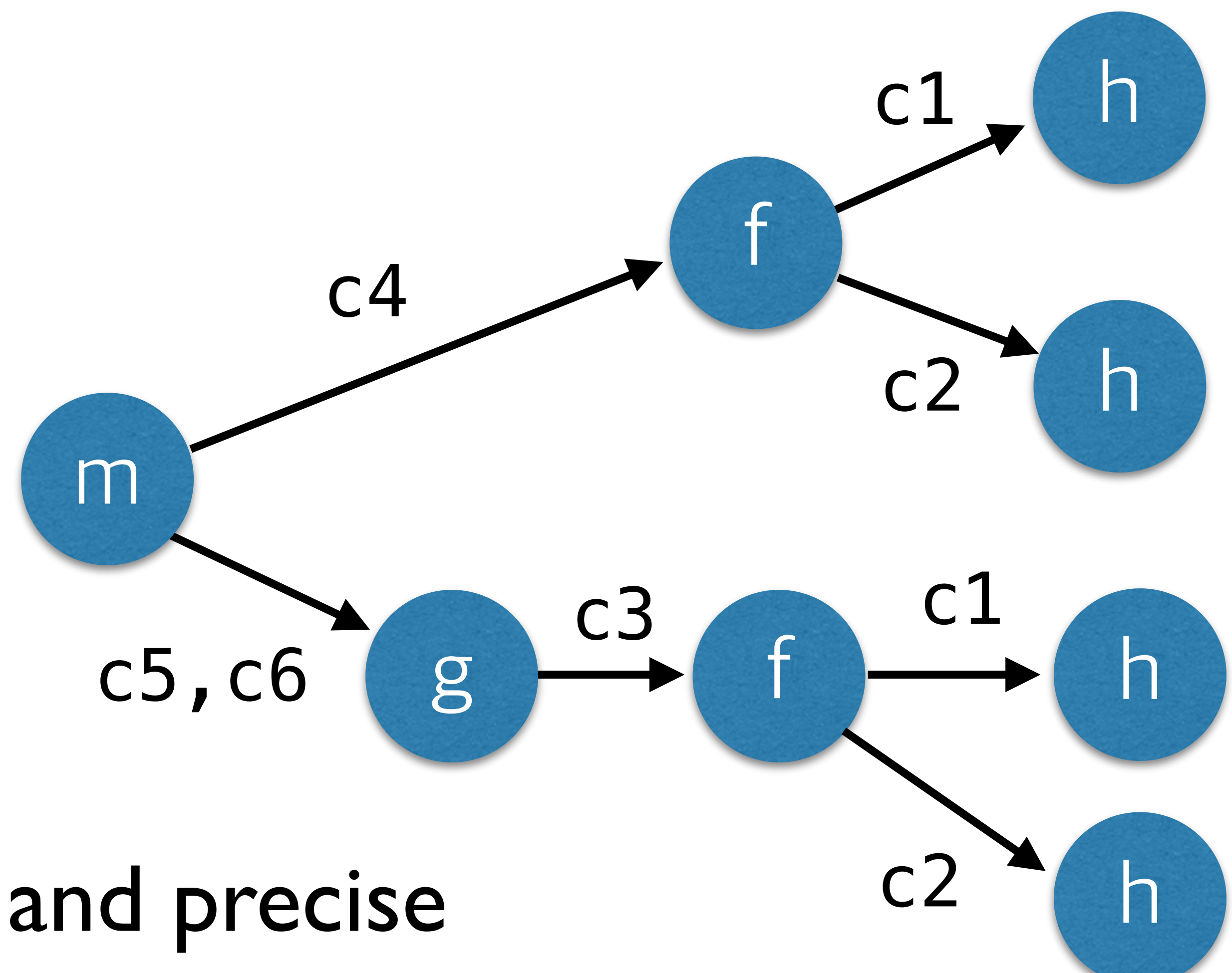
```
    int h(n) {ret n;}

    void f(a) {
c1:   x = h(a);
      assert(x > 0);
c2:   y = h(input());
    }

c3: void g() {f(8);}

    void m() {
c4:   f(4);
c5:   g();
c6:   g();
    }
```

Apply 2-ctx-sens: {h}
Apply 1-ctx-sens: {f}
Apply 0-ctx-sens: {g, m}



cheap and precise

Learning Algorithm Overview

Static analyzer

Training data
(a set of programs)

Atomic features
(a1,a2,…,a25)

e.g., procedures have invocation stmt, procedures return strings, etc

Learning Algorithm

Learned heuristic for applying context-sensitivity:

f2: procedures to apply 2-context-sensitivity

$$1 \wedge \neg 3 \wedge \neg 6 \wedge 8 \wedge \neg 9 \wedge \neg 16 \wedge \neg 17 \wedge \neg 18 \wedge \neg 19 \wedge \neg 20 \wedge \neg 21 \wedge \neg 22 \wedge \neg 23 \wedge \neg 24 \wedge \neg 25$$

f1: procedures to apply 1-context-sensitivity

$$\begin{aligned}
&(1 \wedge \neg 3 \wedge \neg 4 \wedge \neg 7 \wedge \neg 8 \wedge 6 \wedge \neg 9 \wedge \neg 15 \wedge \neg 16 \wedge \neg 17 \wedge \neg 18 \wedge \neg 19 \wedge \neg 20 \wedge \neg 21 \wedge \neg 22 \wedge \neg 23 \wedge \neg 24 \wedge \neg 25) \vee \\
&(\neg 3 \wedge \neg 4 \wedge \neg 7 \wedge \neg 8 \wedge \neg 9 \wedge 10 \wedge 11 \wedge 12 \wedge 13 \wedge \neg 16 \wedge \neg 17 \wedge \neg 18 \wedge \neg 19 \wedge \neg 20 \wedge \neg 21 \wedge \neg 22 \wedge \neg 23 \wedge \neg 24 \wedge \neg 25) \vee \\
&(\neg 3 \wedge \neg 9 \wedge 13 \wedge 14 \wedge 15 \wedge \neg 16 \wedge \neg 17 \wedge \neg 18 \wedge \neg 19 \wedge \neg 20 \wedge \neg 21 \wedge \neg 22 \wedge \neg 23 \wedge \neg 24 \wedge \neg 25) \vee \\
&(1 \wedge 2 \wedge \neg 3 \wedge 4 \wedge \neg 5 \wedge \neg 6 \wedge \neg 7 \wedge \neg 8 \wedge \neg 9 \wedge \neg 10 \wedge \neg 13 \wedge \neg 15 \wedge \neg 16 \wedge \neg 17 \wedge \neg 18 \wedge \neg 19 \wedge \neg 20 \wedge \neg 21 \wedge \neg 22 \\
&\wedge \neg 23 \wedge \neg 24 \wedge \neg 25)
\end{aligned}$$

Settings

- $P \in \mathbb{P}$: a program to analyze
- k : the degree of abstraction
  - e.g., $k = 2$ for 2-CFA
  - e.g., $k = 1$ for flow sensitivity
- $\mathscr{A}_P$: the set of abstractions for P
  - $a \in \mathscr{A}_P = Func_P \rightarrow \{0, 1, \ldots, k\}$
- $\mathbb{Q}_P$: the set of queries (assertions) in P

Input 1: Static Analyzer

Static analyzer is modeled by blackbox function F_P :

$$F_P : \mathscr{A}_P \to 2^{\mathbb{Q}_P} \times \mathbb{N}$$

- $Q \in 2^{\mathbb{Q}_P}$: assertions proved by the analysis
- $\mathbb{N}$: integer denoting cost (e.g., time, memory)
- cost$(F_P(\mathbf{a}))$: cost of analysis with abstraction $\mathbf{a}$
- proved$(F_P(\mathbf{a}))$: precision of analysis with abstraction $\mathbf{a}$

Input 2, 3: Programs and Features

- Training data $\mathbf{P} = \{P_1, P_2, \ldots, P_m\}$
- Atomic features $\mathbb{A} = \{a_1, a_2, \ldots, a_n\}$
  - $a_i : Func \rightarrow \{true, false\}$
  - A feature denotes a set of functions:

$$[\![a_i]\!]_P = \{m \in Func \mid a_i(m) = true\}$$

Output: Abstraction Heuristic

- An abstraction heuristic $\mathscr{H}$:

$$\mathscr{H}(P) : \mathscr{A}_P = Func_P \rightarrow \{0,1,\ldots,k\}$$

- The heuristic is used to analyze new program P :

$$F_P(\mathscr{H}(P))$$

Machine Learning: Three Steps

1. Define a parameterized heuristic $\mathscr{H}_\Pi$:

$$\mathscr{H}_\Pi(P) : 2^{Func_P}$$

2. Define a learning objective as optimization problem:

"Find Π that maximizes analysis performance"

3. Solve the optimization problem via learning algorithm

1. Parameterized Heuristics

- The heuristic $\mathscr{H}_\Pi$ has k boolean formulas as learnable parameters:

$$\Pi = \langle f_1, f_2, \ldots, f_k \rangle$$

- Each formula f_i is defined over atomic features ($\mathbb{A}$):

$$f \rightarrow true \mid false \mid a_i \in \mathbb{A} \mid \neg f \mid f_1 \wedge f_2 \mid f_1 \vee f_2$$

- A formula denotes a set of functions:

$$[\![true]\!] = Func \qquad [\![a_i]\!]_P = \{m \in Func \mid a_i(m) = true\}$$

$$[\![false]\!] = \varnothing \qquad [\![\neg f\,]\!] = Func \ \setminus \ [\![f\,]\!]$$

$$[\![f_1 \wedge f_2]\!] = [\![f_1]\!] \cap [\![f_2]\!] \qquad [\![f_1 \vee f_2]\!] = [\![f_1]\!] \cup [\![f_2]\!]$$

1. Parameterized Heuristics

$$\mathscr{H}_{\Pi}(P) : \mathscr{A}_P = Func_P \to \{0,1,\ldots,k\}$$

- With $\Pi = \langle f_1, f_2, \ldots, f_k \rangle$:

$$\mathscr{H}_{\Pi}(P) = \lambda m \,.\, i \textit{ such that } m \in [\![f_i]\!]$$

(when $m \in [\![f_i]\!]$ and $m \in [\![f_j]\!], max(i, j)$)

Example

```
int h(n) {ret n;}

void f(a) {
  x = h(a);
  assert(x > 0);
  y = h(input());
}

void g() {f(8);}

void m() {
  f(4);
  g();
  g();
}
```

$$\mathbb{A} = \{a_1, a_2, a_3, a_4, a_5\}$$

$$\texttt{h} : \{a_1, a_3, a_5\} \qquad \texttt{f} : \{a_3, a_5\}$$

$$\texttt{g} : \{a_1, a_2, a_3\} \qquad \texttt{m} : \{a_2, a_3, a_4\}$$

Heuristic $\mathscr{H}_{\langle f_1, f_2 \rangle}$ with

$$f_1 = \neg a_4 \wedge a_5, \quad f_2 = (a_1 \wedge a_5) \vee (a_2 \wedge \neg a_3)$$

$$([\![f_1]\!] = \{\texttt{f}, \texttt{h}\}, \quad [\![f_2]\!] = \{\texttt{h}\})$$

produces the abstraction:

$$\{\texttt{h} \mapsto 2,\ f \mapsto 1,\ g \mapsto 0,\ m \mapsto 0\}$$

2. Optimization Problem

Find Π that minimizes $\sum_{P \in \mathbf{P}} \text{cost}(F_P(\mathscr{H}_\Pi(P)))$

while ensuring a user-provided precision constraint.

E.g., "maintain 90% precision of 2-CFA"

# of assertions proved by the current abstraction

$$\frac{\sum_{P \in \mathbf{P}} |\text{proved}(F_P(\mathscr{H}_\Pi(P)))|}{\sum_{P \in \mathbf{P}} |\text{proved}(F_P(\lambda m.2))|} \geq 0.9$$

# of assertions proved by the most precise abstraction (2-CFA)

3. Learning Algorithm

We learn each formula via iterative refinement

1. Initialize f to the most general formula in DNF:
   $f = a_1 \vee \neg a_1 \vee a_2 \vee \neg a_2 \vee \ldots \vee a_n \vee \neg a_n \quad (\equiv true)$
2. Repeat the following (until no refinement is possible)

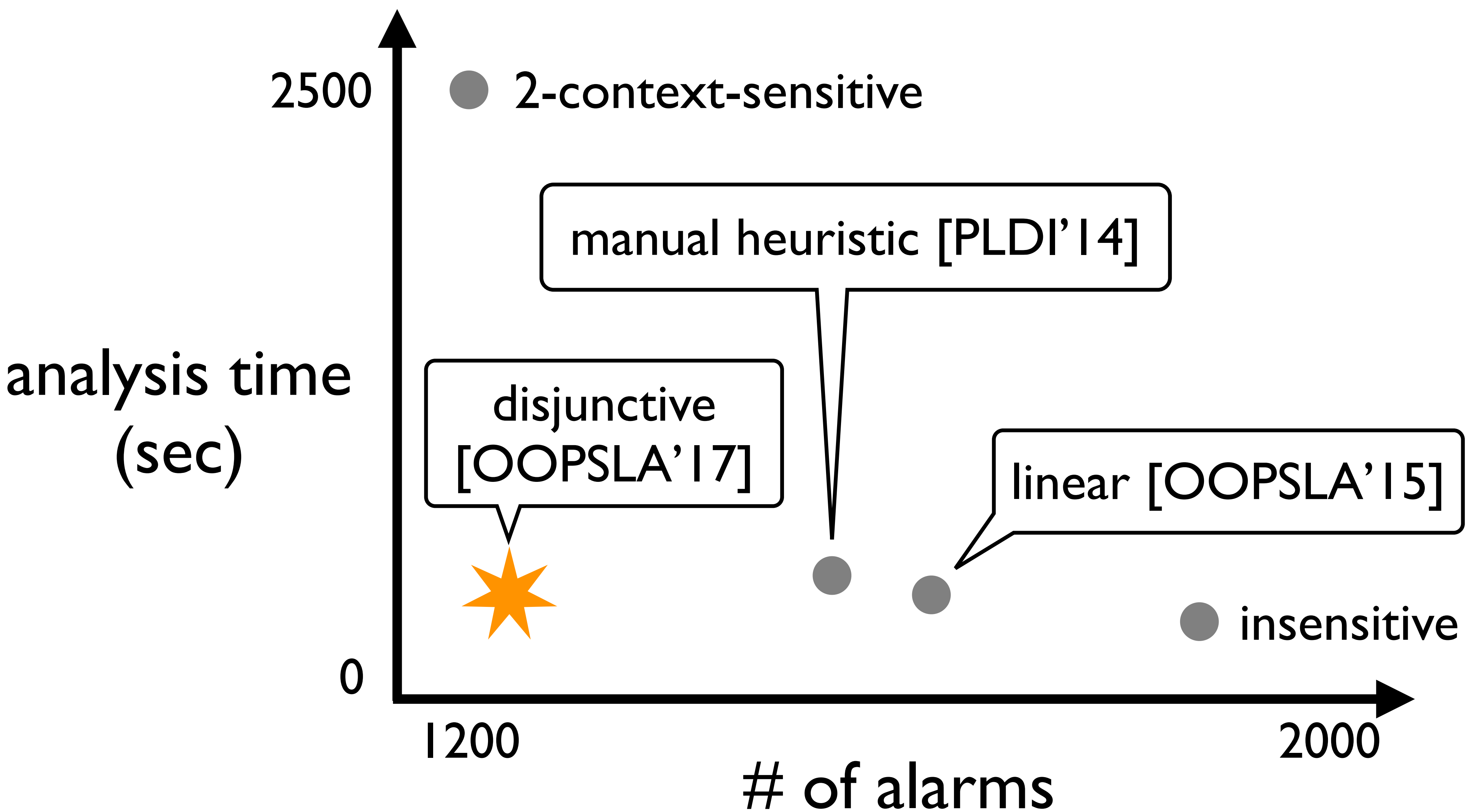
$$f = c_1 \vee c_2 \vee \ldots \vee c_m$$

   1. Choose the most expensive conjunct, say c_i
   2. Refine the conjunct with some feature a_j:
   $$f = c_1 \vee c_2 \vee \ldots \vee (c_i \wedge a_j) \vee \ldots \vee c_m$$
   3. Check the precision constraint: If not, revert the last change.

(details in paper)

Effectiveness

- Now data-driven approach beats hand-tuning
- E.g., context-sensitive pointer analysis for Java (bloat)

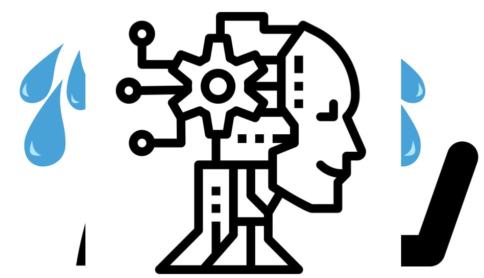


Summary

Data-Driven Static Analysis

- A general framework for generating analysis heuristics:

Static analyzer →
Data (programs) →



→ context-sensitivity heuristics
flow-sensitivity heuristics
…

- More recent results available at http://prl.korea.ac.kr
  - Without handcrafted features [OOPSLA'20]
  - Non-traditional applications [OOPSLA'18, POPL'22]
  - Beyond static analysis [ICSE'18, ICSE'22]

Thank you!