Data-Driven Static Analysis

Hakjoo Oh

Korea University

12 September 2017 @Shonan Meeting

(co-work with Sooyoung Cha, Kwonsoo Chae, Kihong Heo, Minseok Jeon, Sehun Jeong, Hongseok Yang, Kwangkeun Yi)

PL Research in Korea Univ.

- We research on technology for safe and reliable software.
- Research areas: programming languages, software engineering, software security
  - program analysis and testing
  - program synthesis and repair
- Publication: top-venues in PL, SE, and Security
  - PLDI('12,'14), ICSE'17, OOPSLA('15,'17,'17), Oakland'17, etc

http://prl.korea.ac.kr

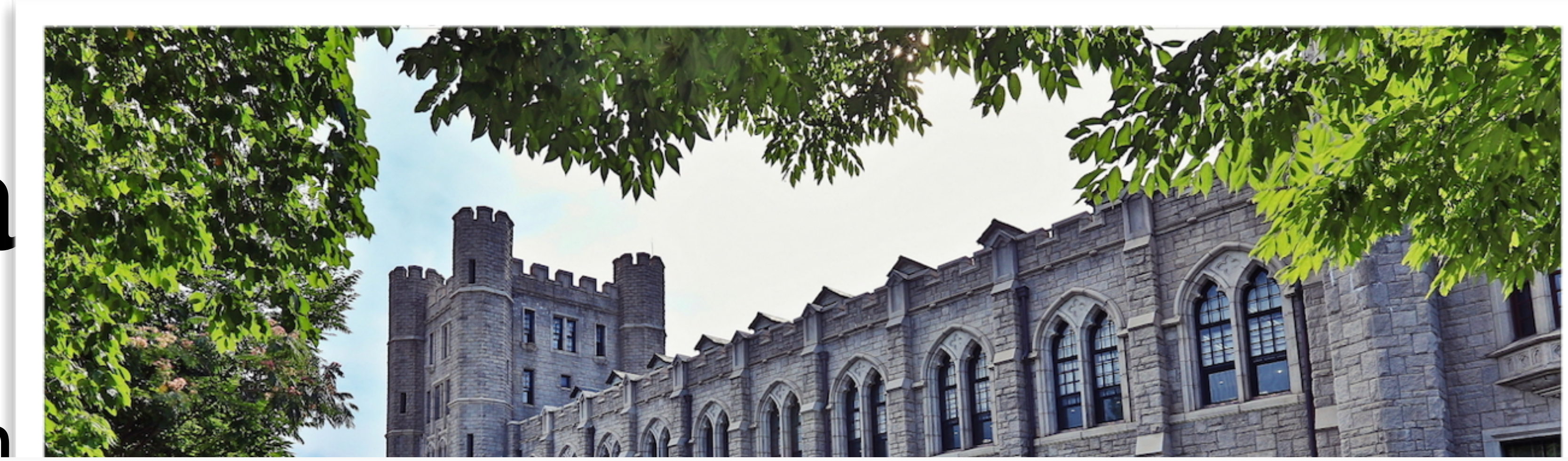
PL Resea

- We research on tech
- Research areas: progr

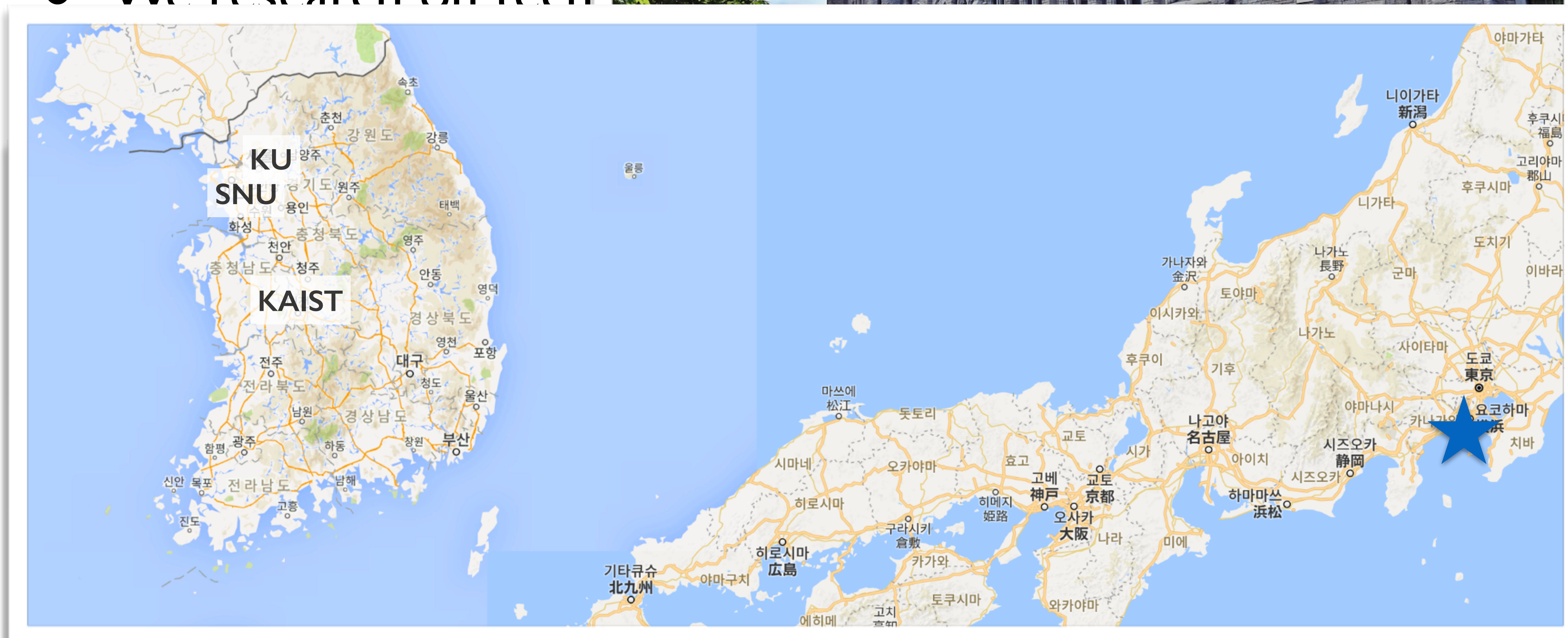
7, etc

http://prl.korea.ac.kr

PL Resea



- We research on tech



7, etc

http://prl.korea.ac.kr

Heuristics in Static Analysis

Sparrow The Early Bird

WALA T. J. WATSON LIBRARIES FOR ANALYSIS

Astrée DOOP TAJS SAFE

- Practical static analyzers involve many heuristics
  - Which procedures should be analyzed context-sensitively?
  - Which relationships between variables should be tracked?
  - When to split and merge in trace partitioning?
  - Which program parts to analyze unsoundly or soundly?, etc
- Designing a good heuristic is an art
  - Usually done by trials and error: nontrivial and suboptimal

Goal

Automatically Generating Heuristics from Data

- Automate the process: use data to make heuristic decisions in static analysis

GitHub

machine learning

context-sensitivity heuristics
flow-sensitivity heuristics
unsoundness heuristics
…

- Automatic: little reliance on analysis designers
- Powerful: machine-tuning outperforms hand-tuning
- Stable: can be generated for target programs

Context-Sensitivity

```
class D {} class E {}
class C {
  void dummy(){}
  Object id1(Object v){ return id2(v); }
  Object id2(Object v){ return v; }
}
class B {
  void m (){
    C c = new C();
    D d = (D)c.id1(new D());   //Query 1
    E e = (E)c.id1(new E());   //Query 2
    c.dummy();
  }
}
public class A {
  public static void main(String[] args){
    B b = new B();
    b.m();
    b.m();
  }
}
```

Contet-insensitivity fails to prove the queries

2-object-sensitivity succeeds but not scale

Selective Context-Sensitivity

```java
class D {} class E {}
class C {
  void dummy(){}
  Object id1(Object v){ return id2(v); }
  Object id2(Object v){ return v; }
}
class B {
  void m (){
    C c = new C();
    D d = (D)c.id1(new D());   //Query 1
    E e = (E)c.id1(new E());   //Query 2
    c.dummy();
  }
}
public class A {
  public static void main(String[] args){
    B b = new B();
    b.m();
    b.m();
  }
}
```

Apply 2-obj-sens: {C.id2}
Apply 1-obj-sens: {C.id1}
Apply insens: {B.m, C.dummy}

Selective Context-Sensitivity

```
class D {} class E {}
class C {
  void dummy(){}
  Object id1(Object v){ return id2(v); }
  Object id2(Object v){ return v; }
}
class B {
  void m (){
    C c = new C();
    D d = (D)c.id1(new D());   //Query 1
    E e = (E)c.id1(new E());   //Query 2
    c.dummy();
  }
}
public class A {
  public static void main(String[] args){
    B b = new B();
    b.m();
    b.m();
  }
}
```

Apply 2-obj-sens: {C.id2}
Apply 1-obj-sens: {C.id1}
Apply insens: {B.m, C.dummy}

Challenge: How to decide?
=> Data-driven approach

OOPSLA'17

Data-Driven Ctx-Sensitivity

Parametric static analyzer

Training data (programs)

Atomic features (a1,a2,…,a25)

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

Our DD Framework

OOPSLA'17

Data-Driven Ctx-Sensitivity

Parametric static analyzer

Training data (programs)

Atomic features (a1,a2,…,a25)

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

Our DD Framework

Heuristic for applying (hybrid) object-sensitivity:

f2: Methods that require 2-object-sensitivity

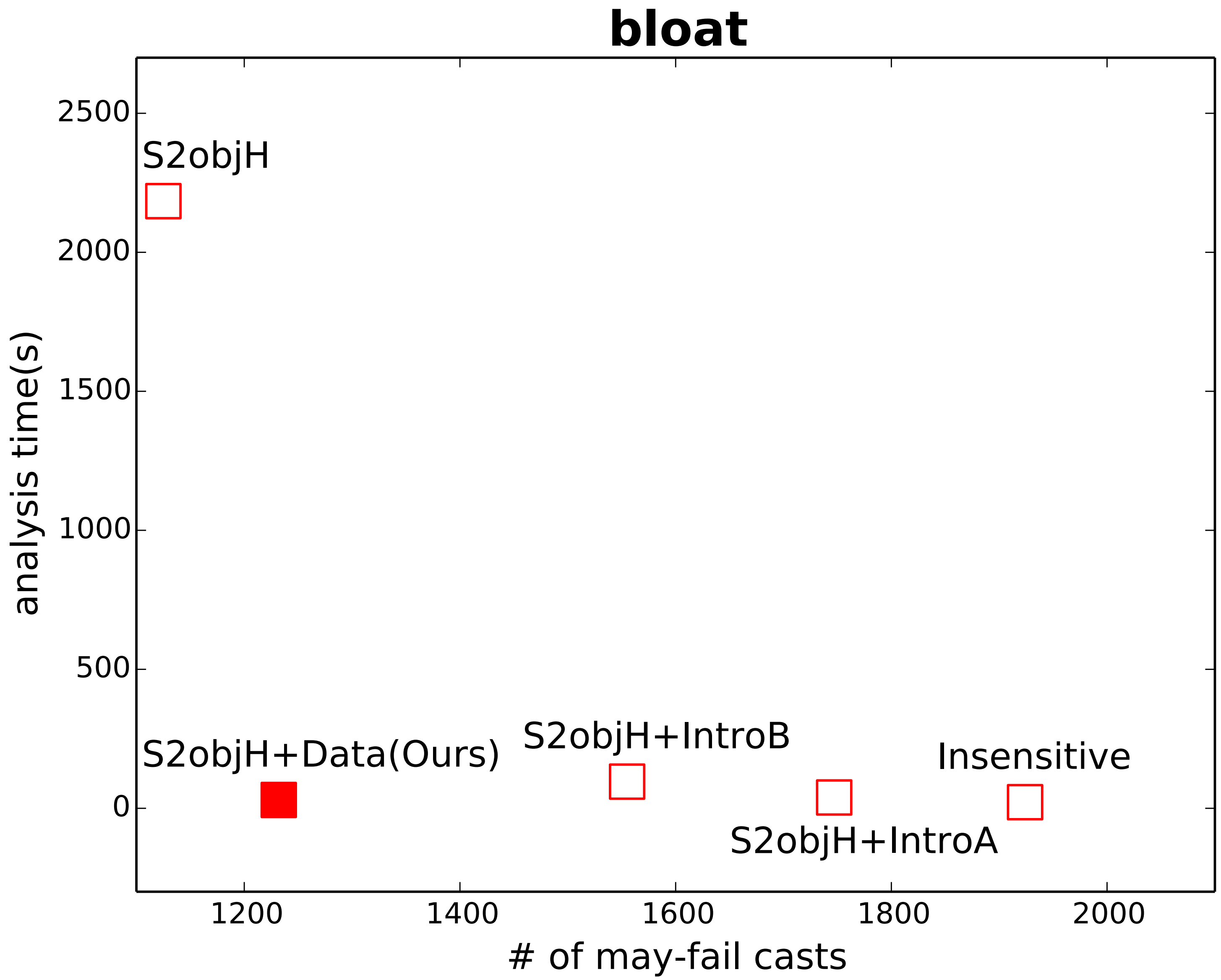
$$1 \wedge \neg 3 \wedge \neg 6 \wedge 8 \wedge \neg 9 \wedge \neg 16 \wedge \neg 17 \wedge \neg 18 \wedge \neg 19 \wedge \neg 20 \wedge \neg 21 \wedge \neg 22 \wedge \neg 23 \wedge \neg 24 \wedge \neg 25$$

f1: Methods that require 1-object-sensitivity

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

Performance

- Training with 4 small programs from DaCapo, and applied to 6 large programs
- Machine-tuning outperforms hand-tuning



Other Context-Sensitivities

- Plain (not hybrid) Object-sensitivity:
  - Depth-2 formula (f_2):

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

  - Depth-1 formula (f_1):

$$(1 \land 2 \land \neg 3 \land \neg 6 \land \neg 7 \land \neg 8 \land \neg 9 \land \neg 16 \land \neg 17 \land \neg 18 \land \neg 19 \land \neg 20 \land \neg 21 \land \neg 22 \land \neg 23 \land \neg 24 \land \neg 25) \lor$$
$$(\neg 1 \land \neg 2 \land 5 \land 8 \land \neg 9 \land 11 \land 12 \land \neg 14 \land \neg 15 \land \neg 16 \land \neg 17 \land \neg 18 \land \neg 19 \land \neg 20 \land \neg 21 \land \neg 22 \land \neg 23 \land \neg 24 \land \neg 25) \lor$$
$$(\neg 3 \land \neg 4 \land \neg 7 \land \neg 8 \land \neg 9 \land 10 \land 11 \land 12 \land \neg 16 \land \neg 17 \land \neg 18 \land \neg 19 \land \neg 20 \land \neg 21 \land \neg 22 \land \neg 23 \land \neg 24 \land \neg 25)$$

- Call-site-sensitivity:
  - Depth-2 formula (f_2):

    $1 \land \neg 6 \land \neg 7 \land 11 \land 12 \land 13 \land \neg 16 \land \neg 17 \land \neg 18 \land \neg 19 \land \neg 20 \land \neg 21 \land \neg 22 \land \neg 23 \land \neg 24 \land \neg 25$

  - Depth-1 formula (f_1):

    $(1 \land 2 \land \neg 7 \land \neg 16 \land \neg 17 \land \neg 18 \land \neg 19 \land \neg 20 \land \neg 21 \land \neg 22 \land \neg 23 \land \neg 24 \land \neg 25)$

- Type-sensitivity:
  - Depth-2 formula (f_2):

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

  - Depth-1 formula (f_1):

$$1 \land 2 \land \neg 3 \land \neg 6 \land \neg 7 \land \neg 8 \land \neg 9 \land \neg 15 \land \neg 16 \land \neg 17 \land \neg 18 \land \neg 19 \land \neg 20 \land \neg 21 \land \neg 22 \land \neg 23 \land \neg 24 \land \neg 25$$

Obj-Sens vs. Type-Sens

- In theory, obj-sens is more precise than type-sens
- The set of methods that benefit from obj-sens is a superset of the methods that benefit from type-sens
- Interestingly, our algorithm automatically discovered this rule from data:

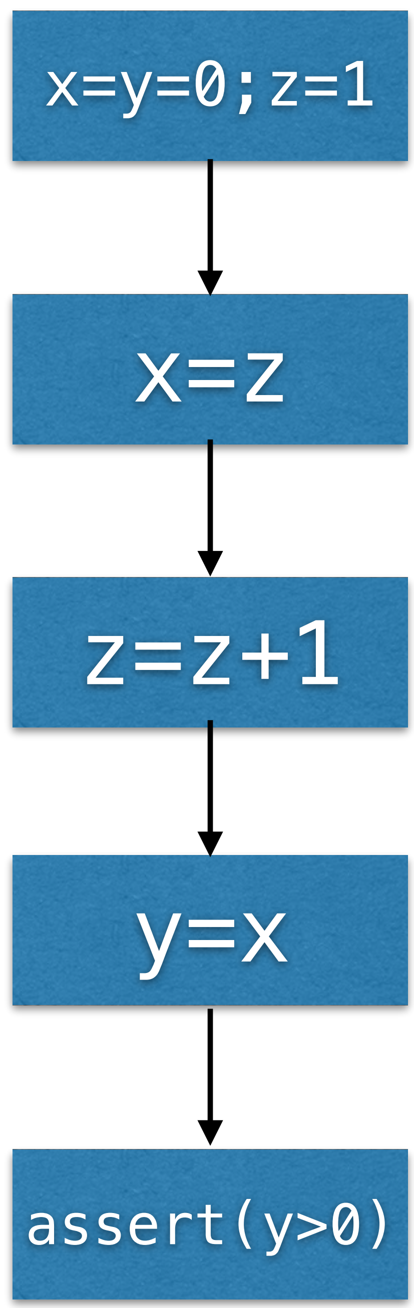
f_1 for *2objH+Data*	:	$(\mathbf{1 \wedge 2 \wedge \neg 3 \wedge \neg 6 \wedge \neg 7 \wedge \neg 8 \wedge \neg 9 \wedge \neg 16 \wedge \cdots \wedge \neg 22 \wedge \neg 23 \wedge \neg 24 \wedge \neg 25})\vee$ $(\neg 1 \wedge \neg 2 \wedge 8 \wedge 5 \wedge \neg 9 \wedge 11 \wedge 12 \wedge \cdots \wedge \neg 21 \wedge \neg 22 \wedge \neg 23 \wedge \neg 24 \wedge \neg 25)\vee$ $(\neg 3 \wedge \neg 4 \wedge \neg 7 \wedge \neg 8 \wedge \neg 9 \wedge 10 \wedge 11 \wedge \cdots \wedge \neg 21 \wedge \neg 22 \wedge \neg 23 \wedge \neg 24 \wedge \neg 25)$
f_1 for *2typeH+Data*	:	$\mathbf{1 \wedge 2 \wedge \neg 3 \wedge \neg 6 \wedge \neg 7 \wedge \neg 8 \wedge \neg 9 \wedge \neg 15 \wedge \neg 16 \wedge \cdots \wedge \neg 22 \wedge \neg 23 \wedge \neg 24 \wedge \neg 25}$

Data-Driven Static Analysis

- Techniques
  - Learning via black-box optimization [OOPSLA'15]
  - Learning with disjunctive model [OOPSLA'17]
  - Learning with automatically generated features [OOPSLA'17]
  - Learning with supervison [ICSE'17,SAS'16,APLAS'16]
- Applications
  - context-sensitivity, flow-sensitivity, variable clustering, widening thresholds, unsoundness, search strategy in symbolic execution, etc

Learing via
Black-Box Optimization
(OOPSLA'15)

Selective Flow-Sensitivity



FS : {x,y}

x	[0,0]
y	[0,0]

x	[1,+∞]
y	[0,0]

x	[1,+∞]
y	[0,0]

x	[1,+∞]
y	[1,+∞]

FI : {z}

z	[1,+∞]

Static Analyzer

number of
proved assertions

$$F(p, a) \Rightarrow n$$

abstraction
(e.g., a set of variables)

Overall Approach

- Parameterized heuristic

$$H_w : \text{pgm} \rightarrow 2^{\text{Var}}$$

Overall Approach

- Parameterized heuristic

$$H_W : \text{pgm} \rightarrow 2^{\text{Var}}$$

- Learn a good parameter W from existing codebase

$$\boxed{P_1, P_2, \ldots, P_m} \quad \Rightarrow \quad W$$

Codebase

Overall Approach

- Parameterized heuristic

$$H_W : \text{pgm} \rightarrow 2^{\text{Var}}$$

- Learn a good parameter W from existing codebase

$$\boxed{P_1, P_2, \ldots, P_m} \quad \Rightarrow \quad W$$

Codebase

- For new program P, run static analysis with $H_W(\text{P})$

1. Parameterized Heuristic

$$H_w : \text{pgm} \rightarrow 2^{\text{Var}}$$

(1) Represent program variables as feature vectors.

(2) Compute the score of each variable.

(3) Choose the top-k variables based on the score.

(1) Features

- Predicates over variables:

$$f = \{f_1, f_2, \ldots, f_5\} \qquad (f_i : \mathrm{Var} \to \{0,1\})$$

- We used 45 simple syntactic features for variables
  - e.g., local / global variable, passed to / returned from malloc, incremented by constants, etc

(1) Features

- Represent each variable as a feature vector:

$$f(x) = \langle f_1(x), f_2(x), f_3(x), f_4(x), f_5(x) \rangle$$

$$f(x) = \langle 1,0,1,0,0 \rangle$$
$$f(y) = \langle 1,0,1,0,1 \rangle$$
$$f(z) = \langle 0,0,1,1,0 \rangle$$

(2) Scoring

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

$$\mathbf{w} = \langle 0.9, 0.5, -0.6, 0.7, 0.3 \rangle$$

- Compute scores of variables:

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

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

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

(3) Choose Top-k Variables

- Choose the top-k variables based on their scores: e.g., when k=2,

score(x) = 0.3
score(y) = 0.6 $\Rightarrow$ {x,y}
score(z) = 0.1

- In experiments, we choose 10% of variables with highest scores.

2. Learn a Good Parameter

$$\boxed{P_1, P_2, \ldots, P_m} \quad \Rightarrow \quad W$$

Codebase

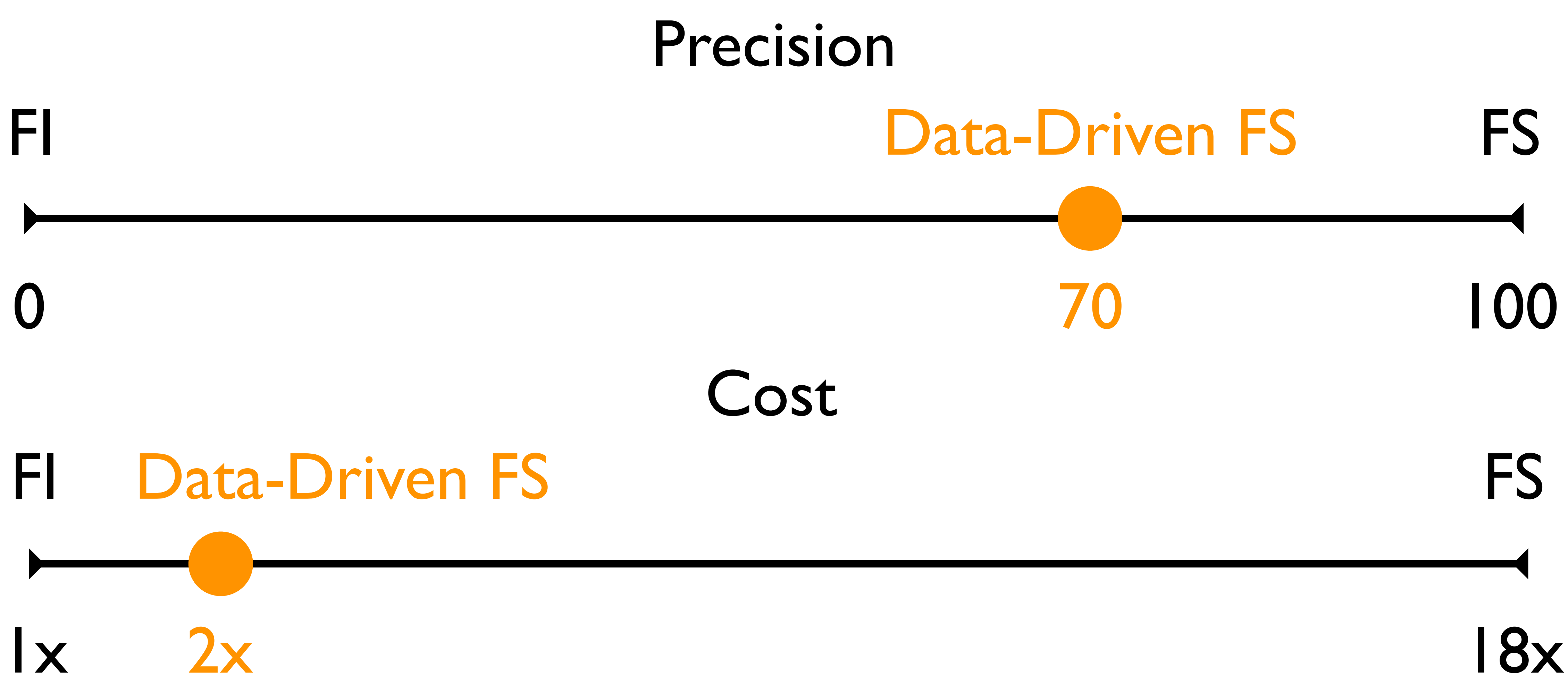
- Formulated as the optimization problem:

Find **w** that maximizes $\displaystyle\sum_{P_i} F(P_i, S_{\mathbf{w}}(P_i))$

- We solve it via Bayesian optimization (details in paper)

Effectiveness on Sparrow

- Implemented in Sparrow, an interval analyzer for C
- Evaluated on 30 open-source programs
  - Training with 20 programs (12 hours)
  - Evaluation with the remaining 10 programs



Limitations & Follow-ups

- Limited expressiveness due to linear heuristic
  - Disjunctive heuristic [OOPSLA'17]
- Semi-automatic due to manual feature engineering
  - Automated feature engineering [OOPSLA'17]
- High learning cost due to black-box approach
  - Supervised approaches [SAS'16,APLAS'16,ICSE'17]

OOPSLA'17

Learning with Disjunctive Heuristics

- The linear heuristic cannot express disjunctive properties:

$$
\begin{aligned}
x &: \{a_1, a_2\} \\
y &: \{a_1\} \\
z &: \{a_2\} \\
w &: \emptyset
\end{aligned}
$$

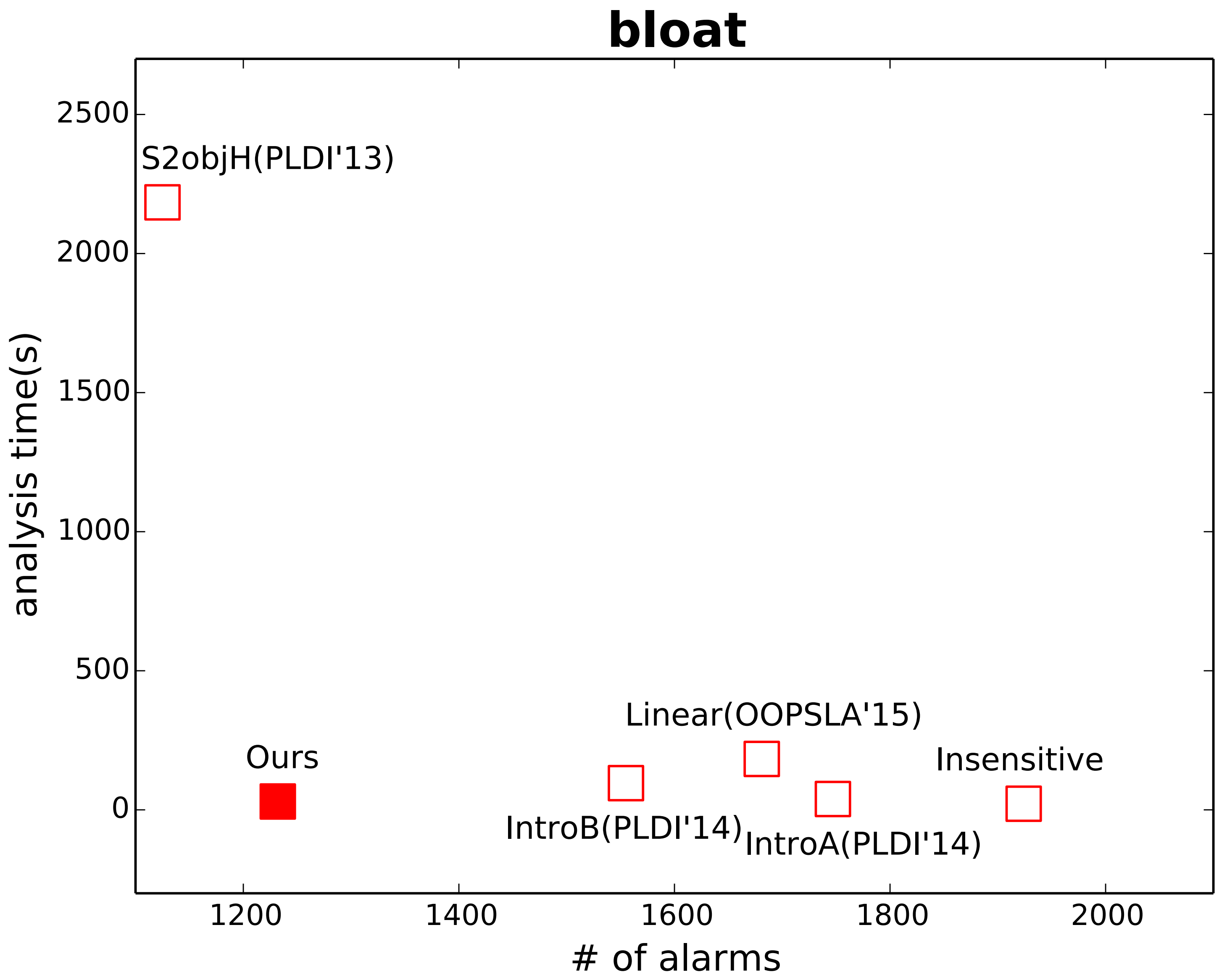
$$\text{Goal: } \{x, w\}$$

$$(a_1 \land a_2) \lor (\neg a_1 \land \neg a_2)$$

- Disjunctive heuristic + algorithm for learning boolean formulas

Performance

- Applied to context-sensitive points-to analysis for Java
- Without disjunction, the learned heuristic lags behind hand-tuning becaused of limited expressiveness



Manual Feature Engineering

- The success of ML heavily depends on the "features"
- Feature engineering is nontrivial and time-consuming
- Features do not generalize to other analyses

Type	#	Features
A	1	local variable
	2	global variable
	3	structure field
	4	location created by dynamic memory allocation
	5	defined at one program point
	6	location potentially generated in library code
	7	assigned a constant expression (e.g., x = c1 + c2)
	8	compared with a constant expression (e.g., x < c)
	9	compared with an other variable (e.g., x < y)
	10	negated in a conditional expression (e.g., if (!x))
	11	directly used in malloc (e.g., malloc(x))
	12	indirectly used in malloc (e.g., y = x; malloc(y))
	13	directly used in realloc (e.g., realloc(x))
	14	indirectly used in realloc (e.g., y = x; realloc(y))
	15	directly returned from malloc (e.g., x = malloc(e))
	16	indirectly returned from malloc
	17	directly returned from realloc (e.g., x = realloc(e))
	18	indirectly returned from realloc
	19	incremented by one (e.g., x = x + 1)
	20	incremented by a constant expr. (e.g., x = x + (1+2))
	21	incremented by a variable (e.g., x = x + y)
	22	decremented by one (e.g., x = x - 1)
	23	decremented by a constant expr (e.g., x = x - (1+2))
	24	decremented by a variable (e.g., x = x - y)
	25	multiplied by a constant (e.g., x = x * 2)
	26	multiplied by a variable (e.g., x = x * y)
	27	incremented pointer (e.g., p++)
	28	used as an array index (e.g., a[x])
	29	used in an array expr. (e.g., x[e])
	30	returned from an unknown library function
	31	modified inside a recursive function
	32	modified inside a local loop
	33	read inside a local loop
B	34	$1 \land 8 \land (11 \lor 12)$
	35	$2 \land 8 \land (11 \lor 12)$
	36	$1 \land (11 \lor 12) \land (19 \lor 20)$
	37	$2 \land (11 \lor 12) \land (19 \lor 20)$
	38	$1 \land (11 \lor 12) \land (15 \lor 16)$
	39	$2 \land (11 \lor 12) \land (15 \lor 16)$
	40	$(11 \lor 12) \land 29$
	41	$(15 \lor 16) \land 29$
	42	$1 \land (19 \lor 20) \land 33$
	43	$2 \land (19 \lor 20) \land 33$
	44	$1 \land (19 \lor 20) \land \neg 33$
	45	$2 \land (19 \lor 20) \land \neg 33$

flow-sensitivity

Type	#	Features
A	1	leaf function
	2	function containing malloc
	3	function containing realloc
	4	function containing a loop
	5	function containing an if statement
	6	function containing a switch statement
	7	function using a string-related library function
	8	write to a global variable
	9	read a global variable
	10	write to a structure field
	11	read from a structure field
	12	directly return a constant expression
	13	indirectly return a constant expression
	14	directly return an allocated memory
	15	indirectly return an allocated memory
	16	directly return a reallocated memory
	17	indirectly return a reallocated memory
	18	return expression involves field access
	19	return value depends on a structure field
	20	return void
	21	directly invoked with a constant
	22	constant is passed to an argument
	23	invoked with an unknown value
	24	functions having no arguments
	25	functions having one argument
	26	functions having more than one argument
	27	functions having an integer argument
	28	functions having a pointer argument
	29	functions having a structure as an argument
B	30	$2 \land (21 \lor 22) \land (14 \lor 15)$
	31	$2 \land (21 \lor 22) \land \neg(14 \lor 15)$
	32	$2 \land 23 \land (14 \lor 15)$
	33	$2 \land 23 \land \neg(14 \lor 15)$
	34	$2 \land (21 \lor 22) \land (16 \lor 17)$
	35	$2 \land (21 \lor 22) \land \neg(16 \lor 17)$
	36	$2 \land 23 \land (16 \lor 17)$
	37	$2 \land 23 \land \neg(16 \lor 17)$
	38	$(21 \lor 22) \land \neg 23$

context-sensitivity

Type	#	Features
A	1	used in array declarations (e.g., a[c])
	2	used in memory allocation (e.g., malloc(c))
	3	used in the righthand-side of an assignment (e.g., x = c)
	4	used with the less-than operator (e.g, x < c)
	5	used with the greater-than operator (e.g., x > c)
	6	used with $\leq$ (e.g., x $\leq$ c)
	7	used with $\geq$ (e.g., x $\geq$ c)
	8	used with the equality operator (e.g., x == c)
	9	used with the not-equality operator (e.g., x ! = c)
	10	used within other conditional expressions (e.g., x < c+y)
	11	used inside loops
	12	used in return statements (e.g., return c)
	13	constant zero
B	14	$(1 \lor 2) \land 3$
	15	$(1 \lor 2) \land (4 \lor 5 \lor 6 \lor 7)$
	16	$(1 \lor 2) \land (8 \lor 9)$
	17	$(1 \lor 2) \land 11$
	18	$(1 \lor 2) \land 12$
	19	$13 \land 3$
	20	$13 \land (4 \lor 5 \lor 6 \lor 7)$
	21	$13 \land (8 \lor 9)$
	22	$13 \land 11$
	23	$13 \land 12$

widening thresholds

OOPSLA'17

Automating Feature Engineering

Before (OOPSLA'15)

Codebase → Hand-crafted features → Parameter values → Analysis Heuristic

New method (OOPSLA'17)

Codebase → Auto-crafted Features → Parameter values → Analysis Heuristic

Key Ideas

- Use program reducer to capture the key reason why FS succeeds but FI fails.

program (>10KLoC)

FS proves but FI fails

C-Reduce

```
for (i=1;i<50;i++)
  assert (i<100);
```

Key Ideas

- Use program reducer to capture the key reason why FS succeeds but FI fails.

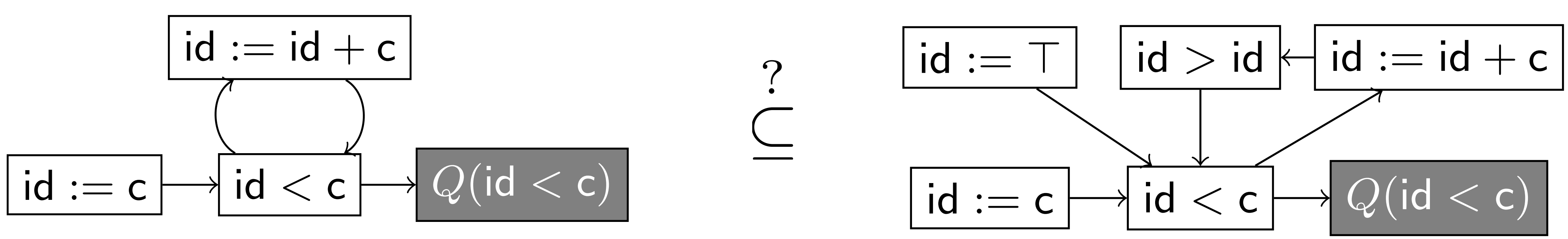
program (>10KLoC)

FS proves but FI fails

C-Reduce

```
for (i=1;i<50;i++)
  assert (i<100);
```

- Generalize the programs by abstract data flow graphs and check graph-inclusion



feature program

target program

Summary: Long-Term Vision

- Static analyzers are designed by analysis designers based on their limited insights on target programs
  - Not tuned for programs that are actually analyzed
- Our vision: "Synthesize" static analyzers from data
  - Every design decisions is parameterized and learned from actual data

GitHub → fully parameterized static analysis → static analyzer tuned for "real-world"

Summary: Long-Term Vision

- Static analyzers are designed by analysis designers based on their limited insights on target programs
  - Not tuned for programs that are actually analyzed
- Our vision: "Synthesize" static analyzers from data
  - Every design decisions is parameterized and learned from actual data

GitHub → fully parameterized static analysis → static analyzer tuned for "real-world"

Thank you