Data-Driven Analysis and Testing of Software

Hakjoo Oh

Korea University

21 June 2019 @HFR

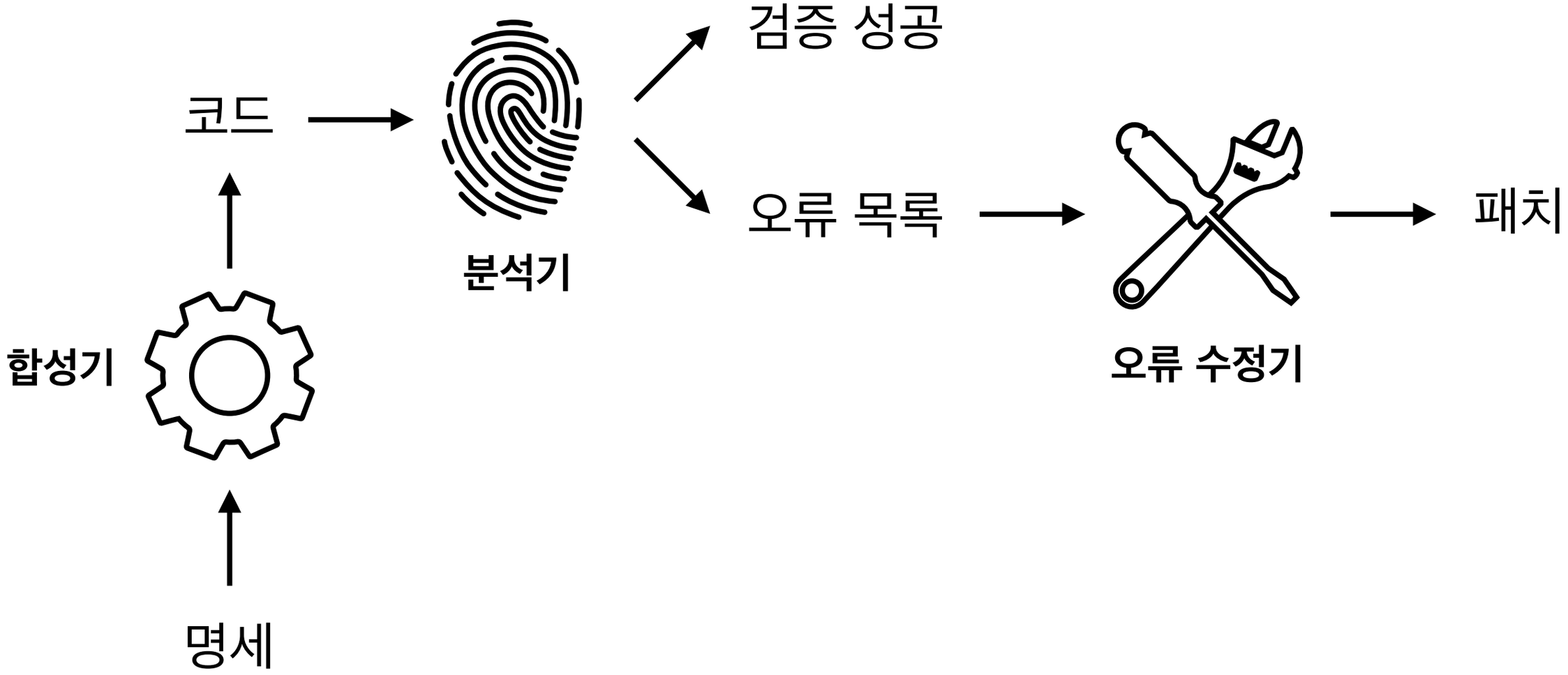
Software Analysis Lab@KU

- Research areas: programming languages, software engineering, software security
  - program analysis and testing
  - program synthesis and repair
- Publication: top-venues in PL, SE, Security, and AI:
  - PLDI('12,'14), OOPSLA('15,'17,'17,'18,'18), TOPLAS('14,'16,'17,'18,'19), ICSE('17,'18,'19), FSE('18,'19), ASE'18, S&P'17, IJCAI('17,'18), etc

http://prl.korea.ac.kr

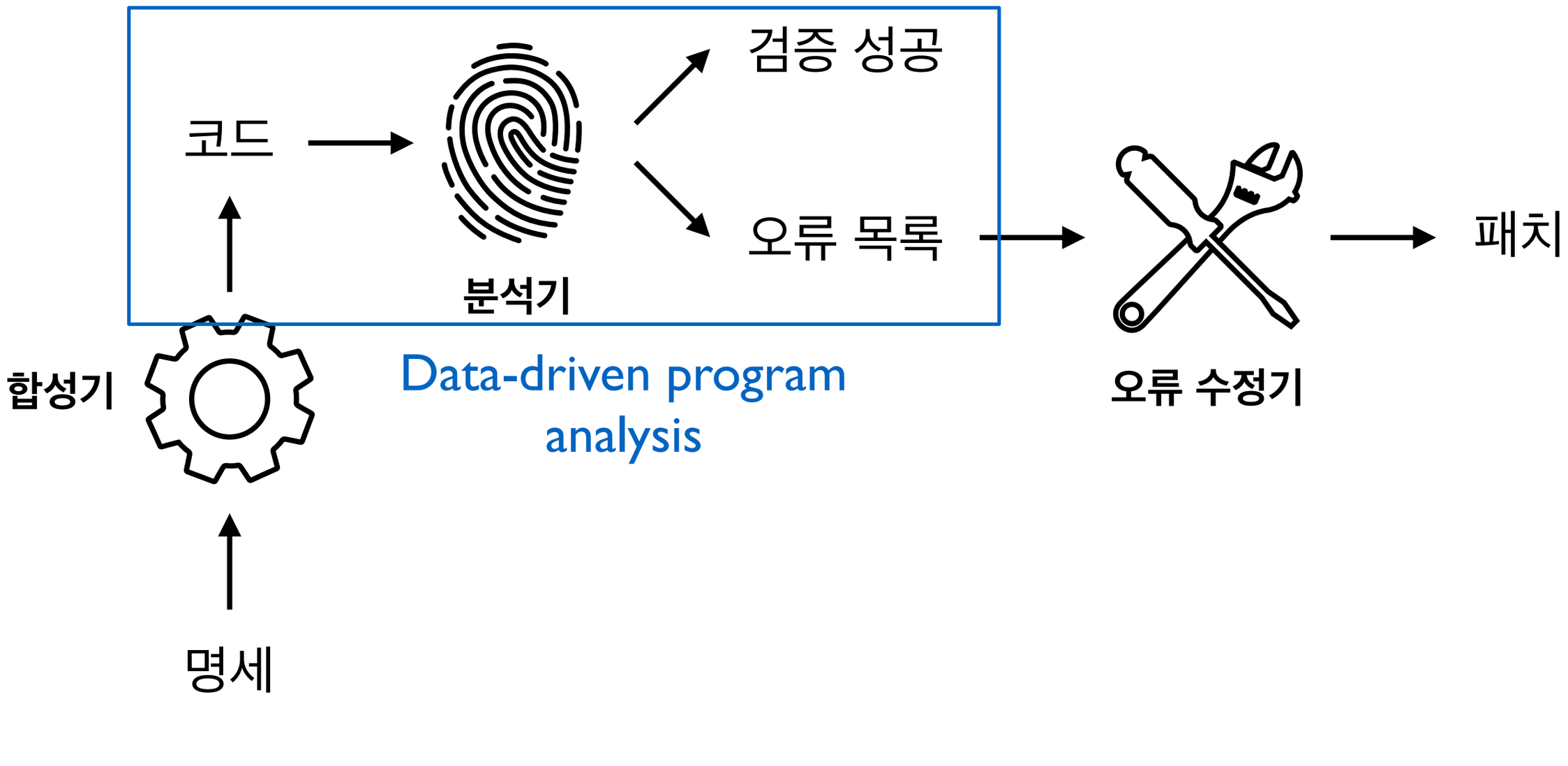
연구 방향

- Q) 어떻게 안전한 소프트웨어를 손쉽게 만들것인가?
- A) 소프트웨어 자동 **분석**, **패치**, **합성** 기술



연구 방향

- Q) 어떻게 안전한 소프트웨어를 손쉽게 만들것인가?
- A) 소프트웨어 자동 **분석**, **패치**, **합성** 기술



Heuristics in Program Analysis


- Practical program analysis tools use many heuristics
  - E.g., context/flow-sensitivity, variable clustering, unsoundness, trace partitioning, path selection/pruning, state merging, etc
- Developing a good heuristic is an art
  - Empirically done by analysis designers: nontrivial & suboptimal

Vision

Automatically Generating Analysis Heuristics from Data

- Use data to make empirical decisions in program analysis

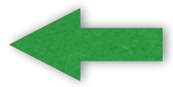
machine learning

context-sensitivity heuristics
flow-sensitivity heuristics
unsoundness heuristics
path-selection heuristics
…

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

Example: Context-Sensitivity

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

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

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

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

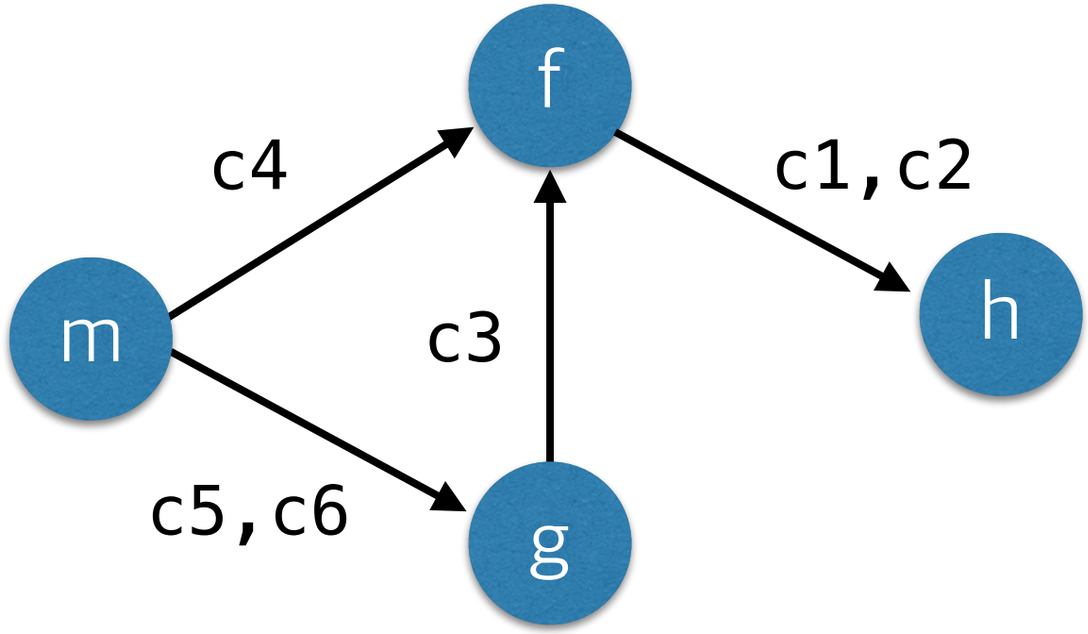


holds always

Context-Insensitive Analysis

- Merge calling contexts into single abstract context

```
    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();
    }
```

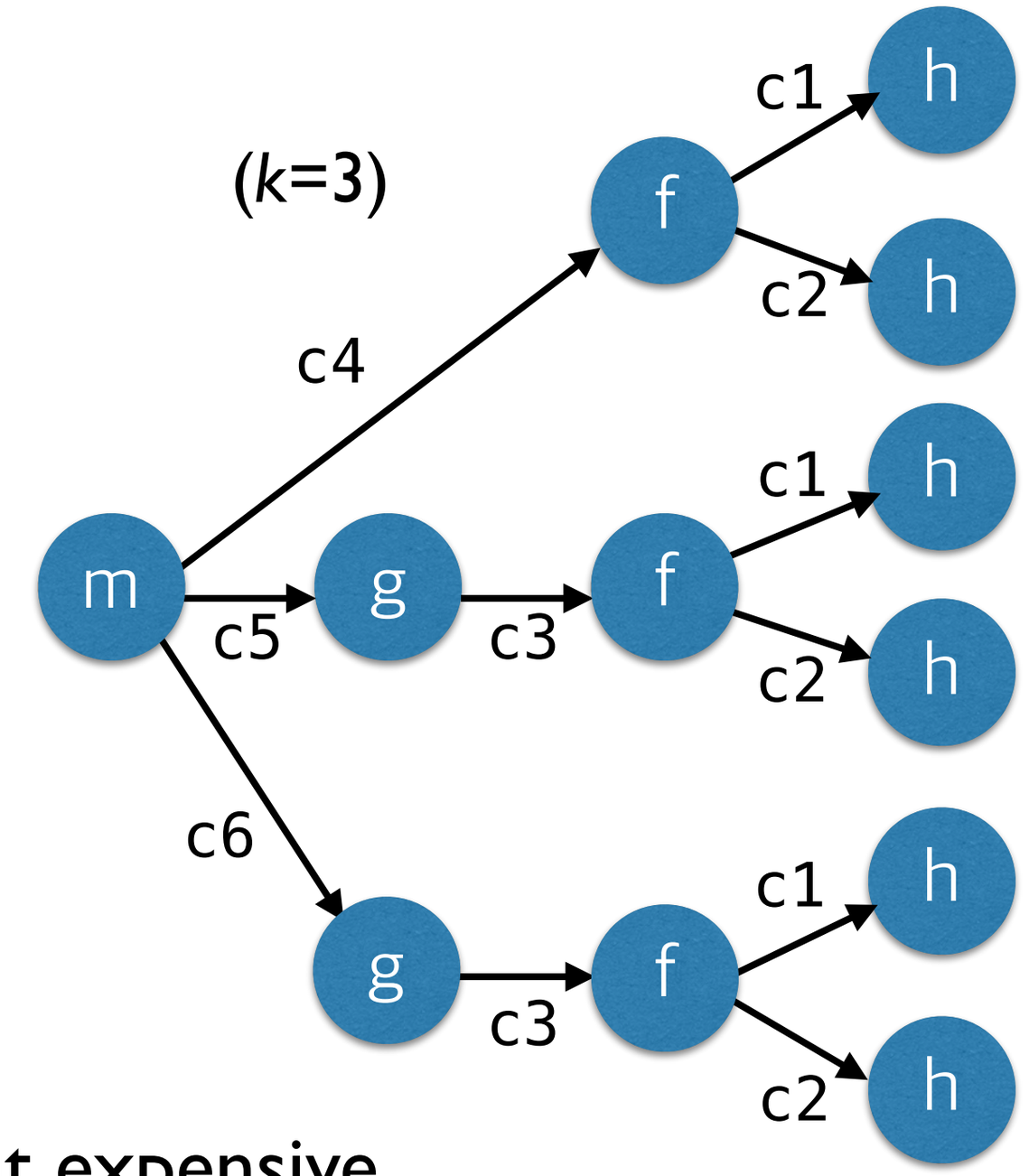


cheap but imprecise

k-Context-Sensitive Analysis

- Analyze functions separately for each calling context

```
    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();
    }
```



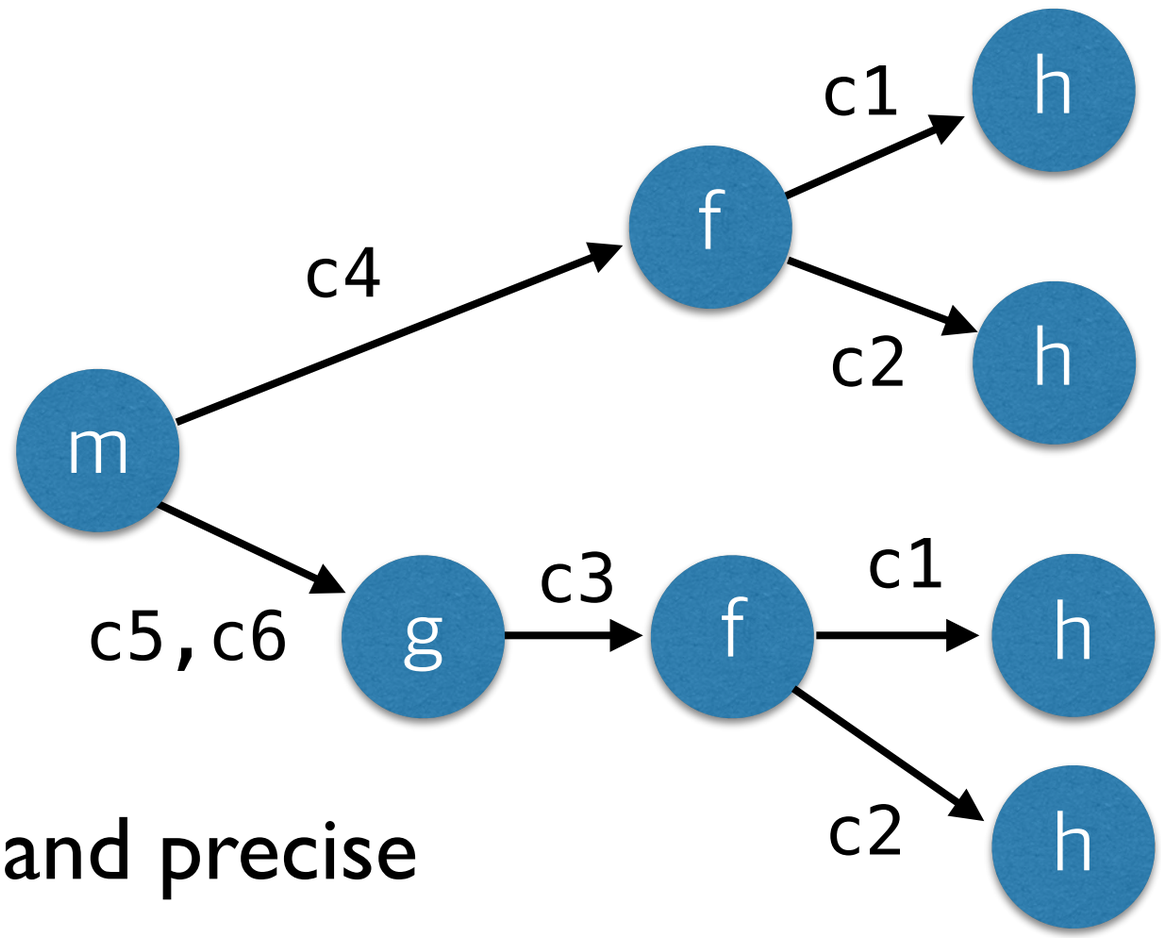
precise but expensive

Selective Context-Sensitivity

- Selectively differentiate contexts only when necessary

```
    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

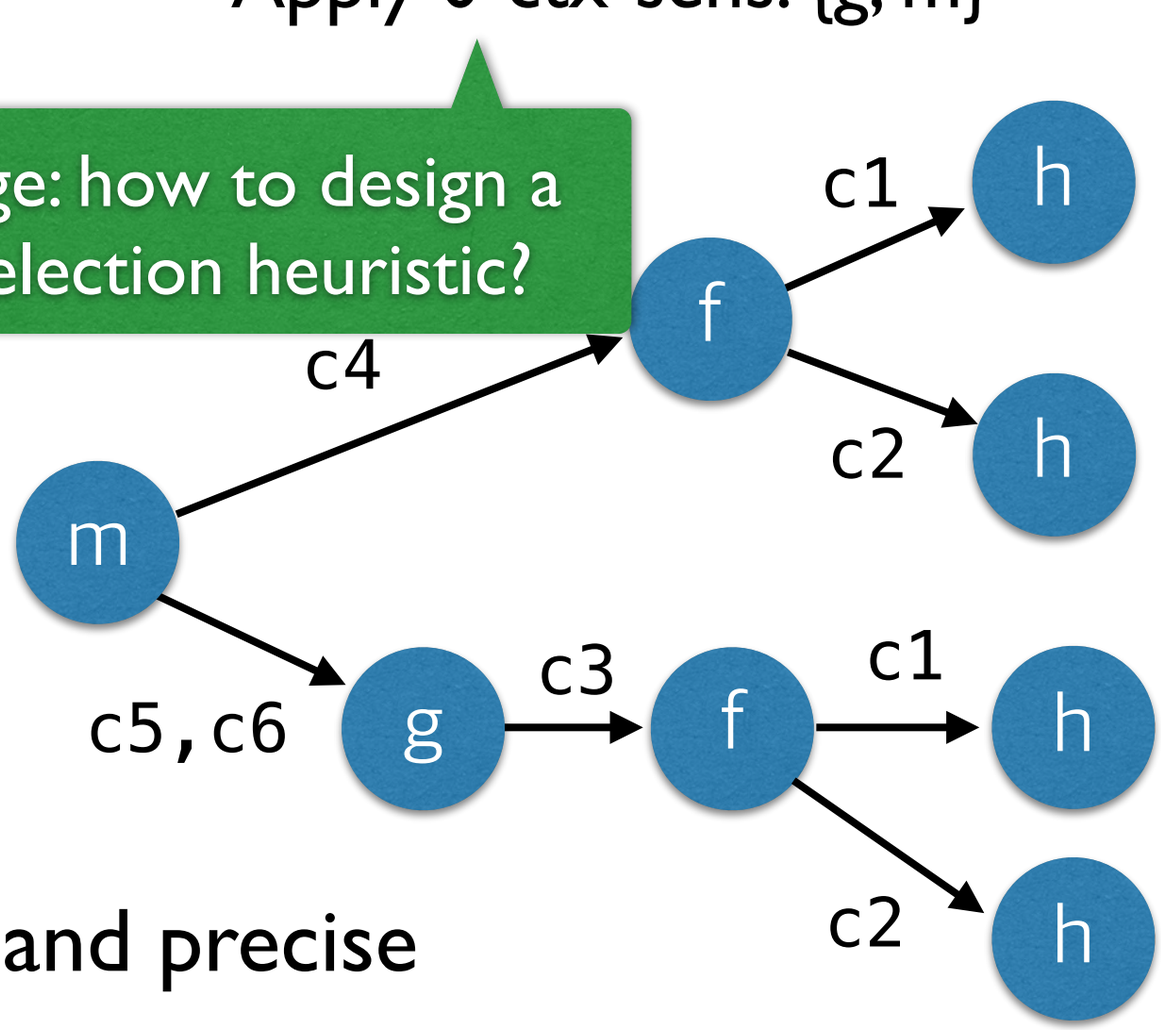
Selective Context-Sensitivity

- Selectively differentiate contexts only when necessary

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

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

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}

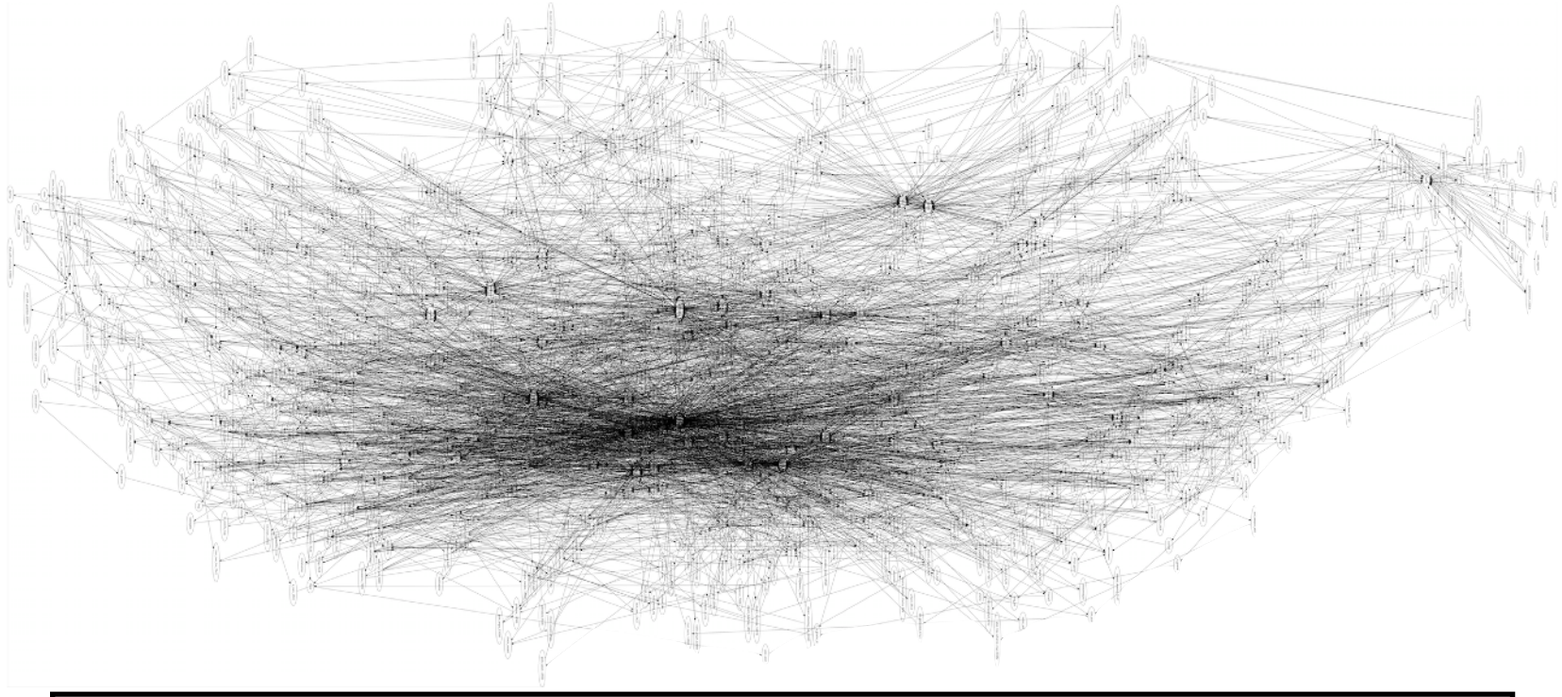
Challenge: how to design a good selection heuristic?



cheap and precise

Hard Search Problem

- Intractably large and sparse search space, if not infinite
  - e.g., S^k choices where $S = 2^{|Proc|}$ for *k*-context-sensitivity
- Real programs are complex to reason about
  - e.g., typical call-graph of real program:



A fundamental problem in program analysis
=> New data-driven approach

OOPSLA'17a
TOPLAS'19

Learning Algorithm Overview

Parametric program analyzer

Training data (programs w/o labels)

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

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

cf) Atomic Features

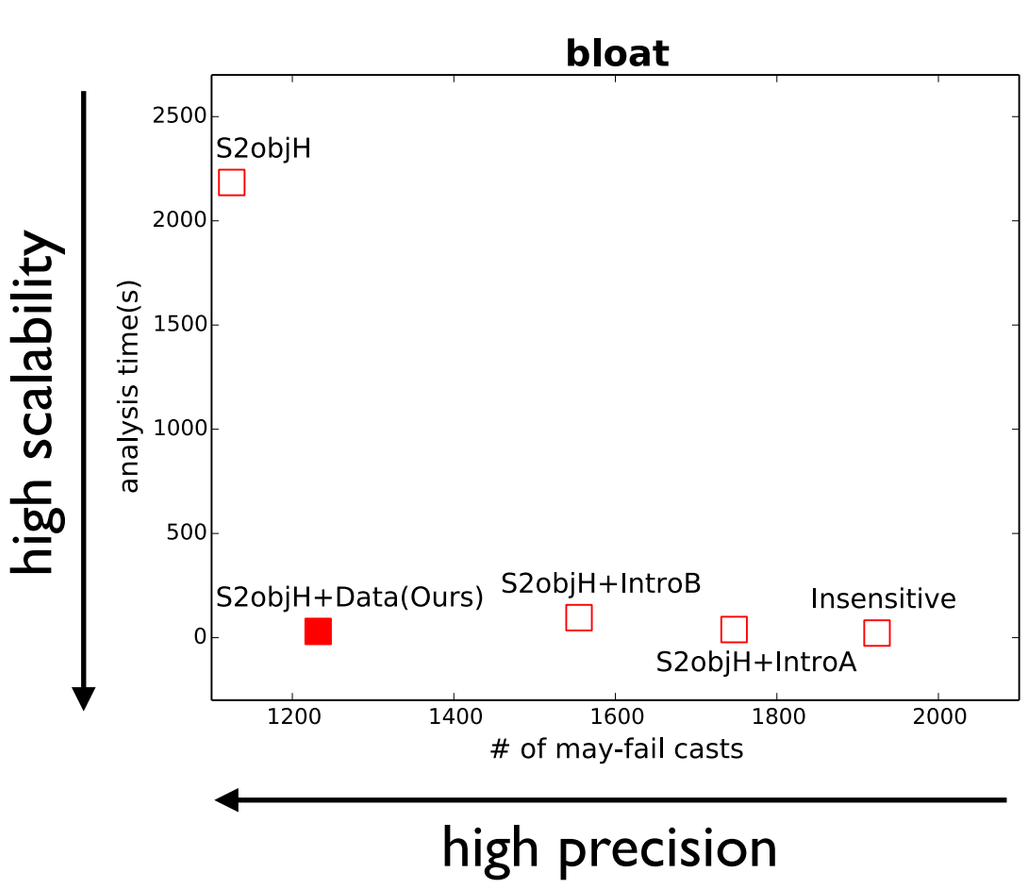
Signature features									
#1	“java”	#3	“sun”	#5	“void”	#7	“int”	#9	“String”
#2	“lang”	#4	“()”	#6	“security”	#8	“util”	#10	“init”

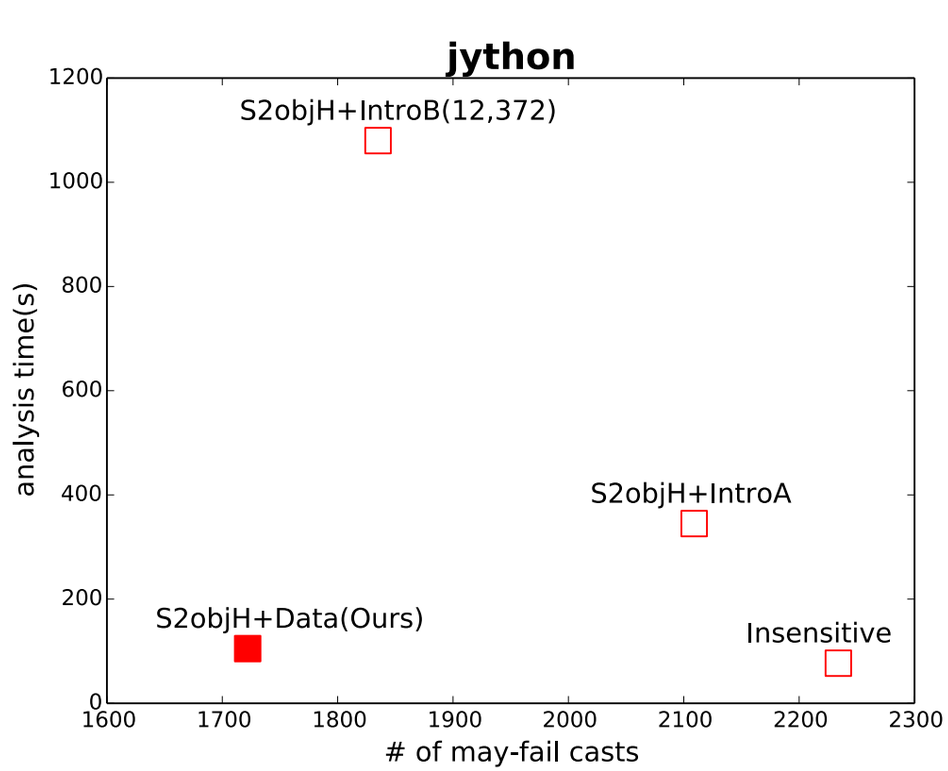
Statement features					
#11	AssignStmt	#16	BreakpointStmt	#21	LookupStmt
#12	IdentityStmt	#17	EnterMonitorStmt	#22	NopStmt
#13	InvokeStmt	#18	ExitMonitorStmt	#23	RetStmt
#14	ReturnStmt	#19	GotoStmt	#24	ReturnVoidStmt
#15	ThrowStmt	#20	IfStmt	#25	TableSwitchStmt

OOPSLA'17a

Application to Pointer Analysis

- Context-sensitive pointer analysis for Java
- Trained with 5 small programs from the DaCapo benchmark and tested with 5 remaining large programs

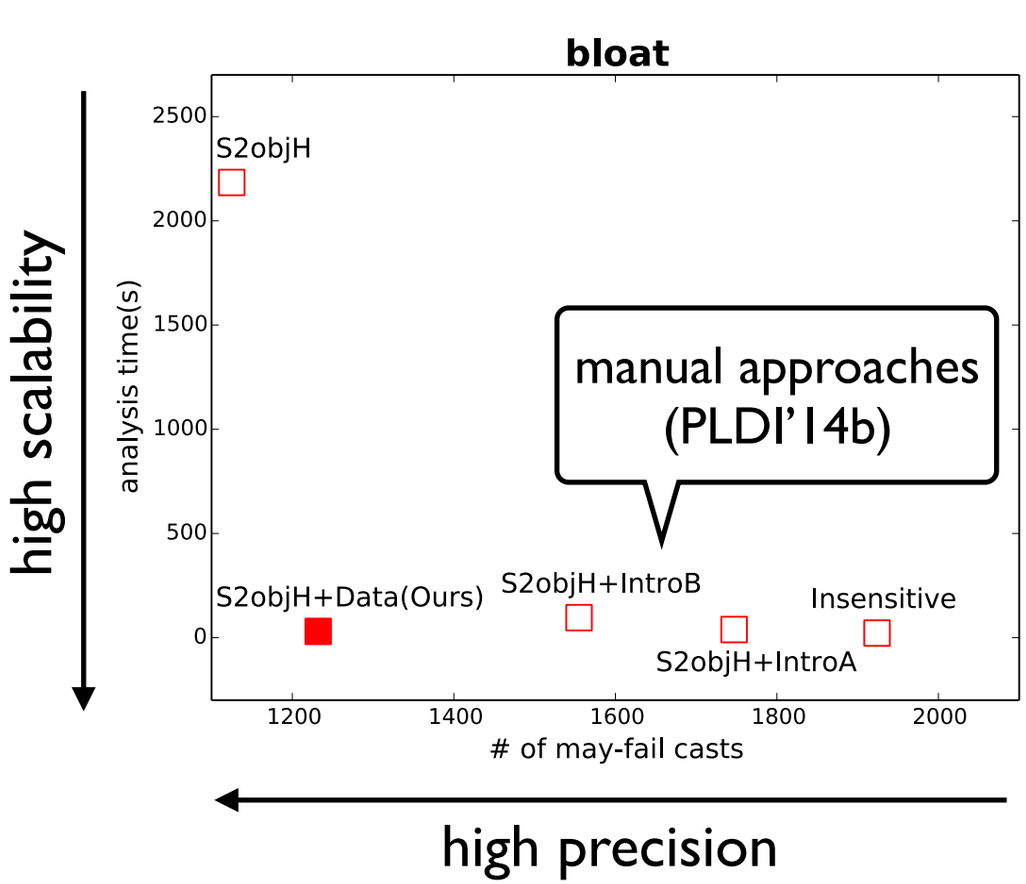


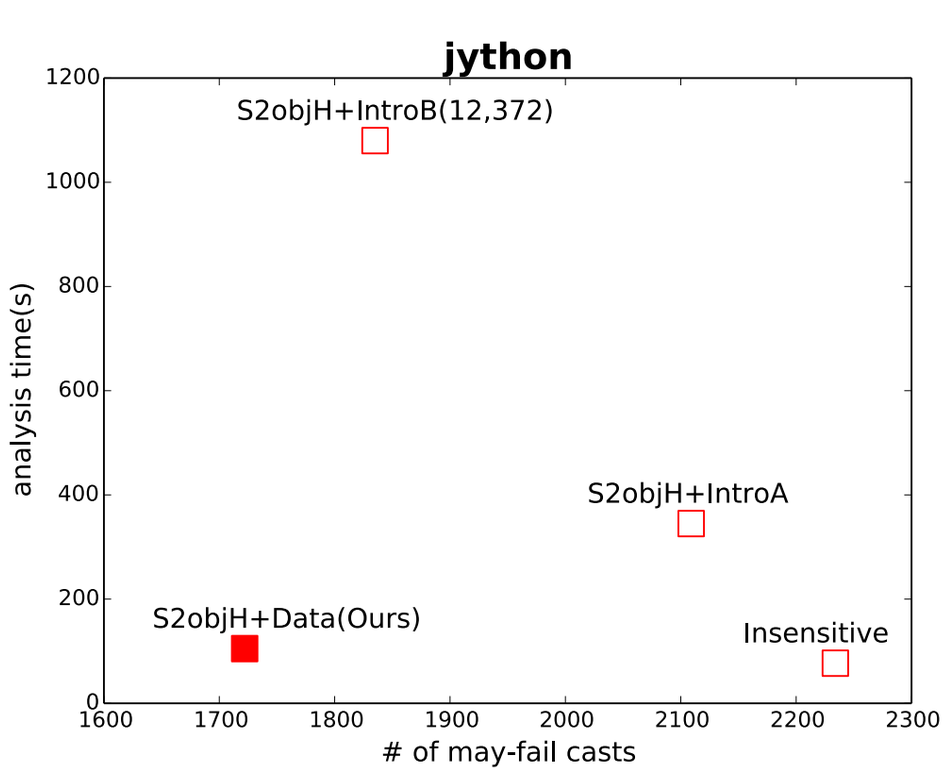


OOPSLA'17a

Application to Pointer Analysis

- Context-sensitive pointer analysis for Java
- Trained with 5 small programs from the DaCapo benchmark and tested with 5 remaining large programs





Concolic Testing (Dynamic Symbolic Execution)

- Concolic testing is an effective software testing method based on symbolic execution

- Key challenge: path explosion
- Our solution: mitigate the problem with good search heuristics

Limitation of Random Testing

```
int double (int v) {
  return 2*v;
}

void testme(int x, int y) {
  z := double (y);

  if (z==x) {

    if (x>y+10) {
      Error;
    }
  }
}
```

Probability of the error? ($0 \leq x,y \leq 100$)

Limitation of Random Testing

```
int double (int v) {
  return 2*v;
}

void testme(int x, int y) {
  z := double (y);

  if (z==x) {

    if (x>y+10) {
      Error;
    }
  }
}
```

Probability of the error? ($0 \leq x,y \leq 100$)

< 0.4%

Limitation of Random Testing

```
int double (int v) {
  return 2*v;
}

void testme(int x, int y) {
  z := double (y);

  if (z==x) {

    if (x>y+10) {
      Error;
    }
  }
}
```

Probability of the error? ($0 \leq x,y \leq 100$)

< 0.4%

- random testing requires 250 runs
- concolic testing finds it in 3 runs

Concolic Testing

```
int double (int v) {
  return 2*v;
}

void testme(int x, int y) {
  z := double (y);

  if (z==x) {

    if (x>y+10) {
      Error;
    }
  }
}
```

Concrete State	Symbolic State
x=22, y=7	x=α, y=β
	true

1st iteration

Concolic Testing

```
int double (int v) {
  return 2*v;
}

void testme(int x, int y) {

  z := double (y);
  if (z==x) {

    if (x>y+10) {
      Error;
    }
  }
}
```

	Concrete State	Symbolic State
	x=22, y=7, z=14	x=α, y=β, z=2*β
		true

1st iteration

Concolic Testing

```
int double (int v) {
  return 2*v;
}

void testme(int x, int y) {

  z := double (y);

  if (z==x) {

    if (x>y+10) {
      Error;
    }
  }
}
```

	Concrete State	Symbolic State
1st iteration	x=22, y=7, z=14	x=α, y=β, z=2*β 2*β ≠ α

Concolic Testing

```
int double (int v) {
  return 2*v;
}

void testme(int x, int y) {

  z := double (y);

  if (z==x) {

    if (x>y+10) {
      Error;
    }
  }
}
```

Concrete State	Symbolic State
	Solve: $2*\beta = \alpha$ Solution: $\alpha=2, \beta=1$
x=22, y=7, z=14	x=α, y=β, z=2*β $2*\beta \neq \alpha$

1st iteration

Concolic Testing

```
int double (int v) {
  return 2*v;
}

void testme(int x, int y) {
  z := double (y);

  if (z==x) {

    if (x>y+10) {
      Error;
    }
  }
}
```

Concrete State	Symbolic State
x=2, y=1	x=α, y=β
	true

2nd iteration

Concolic Testing

```
int double (int v) {
  return 2*v;
}

void testme(int x, int y) {

  z := double (y);
  if (z==x) {

    if (x>y+10) {
      Error;
    }
  }
}
```

	Concrete State	Symbolic State
	x=2, y=1, z=2	x=α, y=β,z=2*β
		true

2nd iteration

Concolic Testing

```
int double (int v) {
  return 2*v;
}

void testme(int x, int y) {

  z := double (y);

  if (z==x) {
    if (x>y+10) {
      Error;
    }
  }
}
```

Concrete State	Symbolic State
x=2, y=1, z=2	x=α, y=β, z=2*β
	2*β = α

2nd iteration

Concolic Testing

```
int double (int v) {
  return 2*v;
}

void testme(int x, int y) {

  z := double (y);

  if (z==x) {

    if (x>y+10) {
      Error;
    }
  }
}
```

	Concrete State	Symbolic State
	x=2, y=1, z=2	x=α, y=β, z=2*β
2nd iteration		2*β = α ∧ α ≤ β+10

Concolic Testing

```
int double (int v) {
  return 2*v;
}

void testme(int x, int y) {

  z := double (y);

  if (z==x) {

    if (x>y+10) {
      Error;
    }
  }
}
```

Concrete State	Symbolic State
	Solve: $2^*\beta = \alpha \wedge \alpha > \beta+10$ Solution: $\alpha=30,\ \beta=15$
$x=2, y=1,$ $z=2$	$x=\alpha, y=\beta, z=2^*\beta$ $2^*\beta = \alpha \wedge$ $\alpha \leq \beta+10$

2nd iteration

Concolic Testing

```
int double (int v) {
  return 2*v;
}

void testme(int x, int y) {
  z := double (y);

  if (z==x) {

    if (x>y+10) {
      Error;
    }
  }
}
```

Concrete State	Symbolic State
x=30, y=15	x=α, y=β
	true

3rd iteration

Concolic Testing

```
int double (int v) {
  return 2*v;
}

void testme(int x, int y) {

  z := double (y);
  if (z==x) {

    if (x>y+10) {
      Error;
    }
  }
}
```

Concrete State	Symbolic State
x=30, y=15, z=30	x=α, y=β, z=2*β
	true

3rd iteration

Concolic Testing

```
int double (int v) {
  return 2*v;
}

void testme(int x, int y) {

  z := double (y);

  if (z==x) {
    if (x>y+10) {
      Error;
    }
  }
}
```

	Concrete State	Symbolic State
←	x=30, y=15, z=30	x=α, y=β,z=2*β 2*β = α

3rd iteration

Concolic Testing

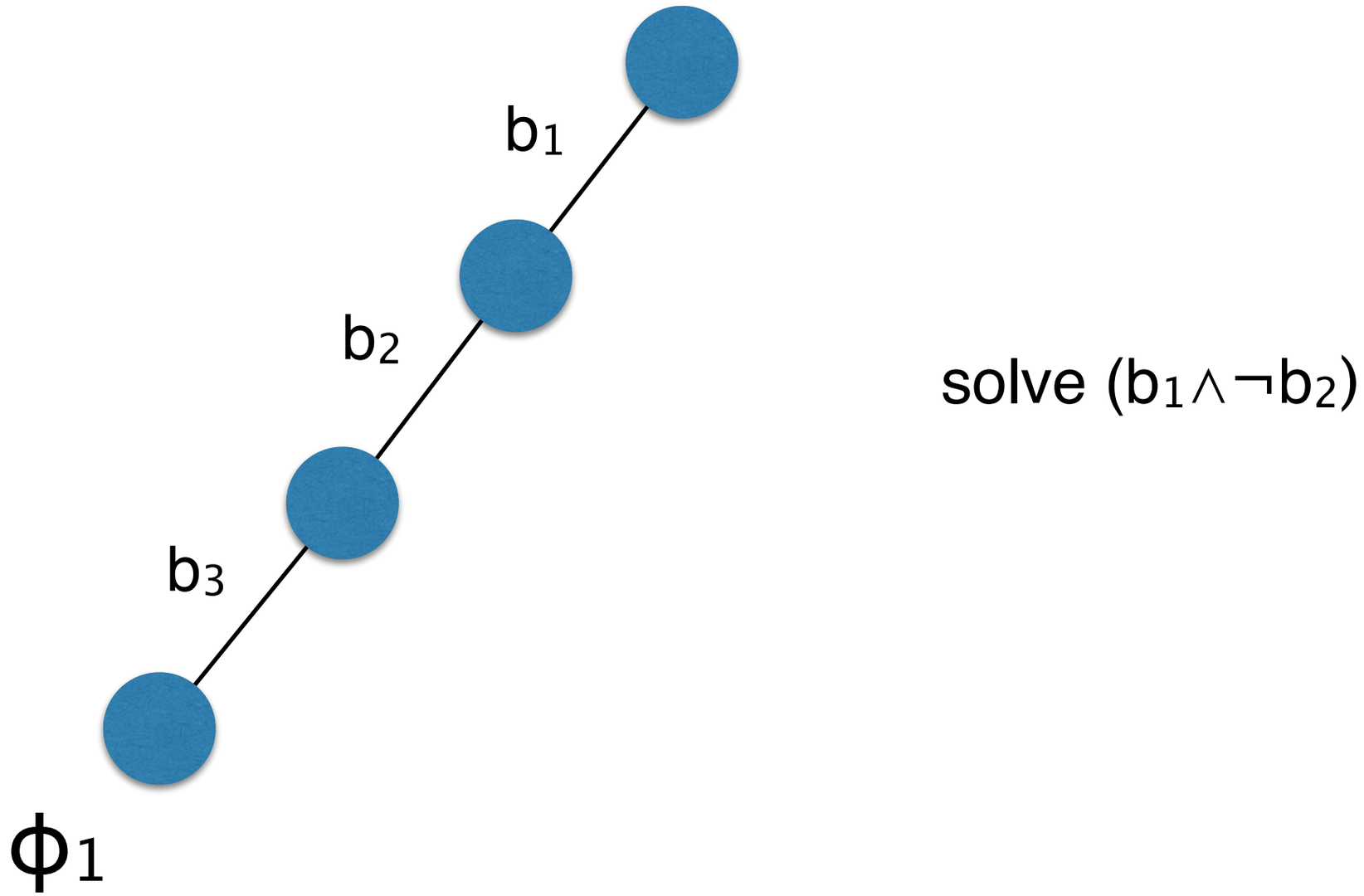
```
int double (int v) {
  return 2*v;
}

void testme(int x, int y) {
  z := double (y);
  if (z==x) {
    if (x>y+10) {
      Error;
    }
  }
}
```

Concrete State	Symbolic State
x=30, y=15, z=30	x=α, y=β, z=2*β
	2*β = α ∧ α > β+15

error-triggering input

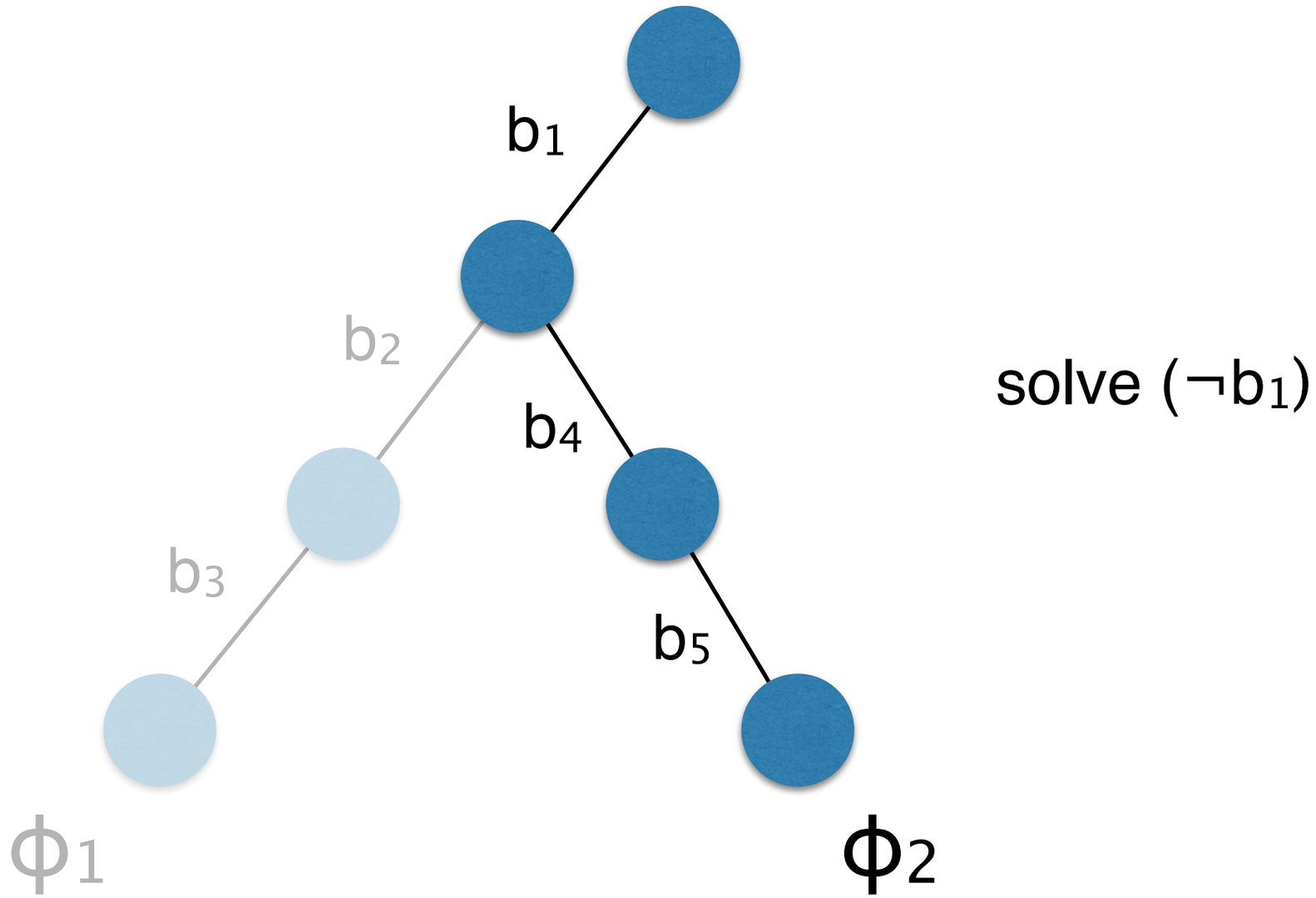
3rd iteration

Concolic Testing



execution tree

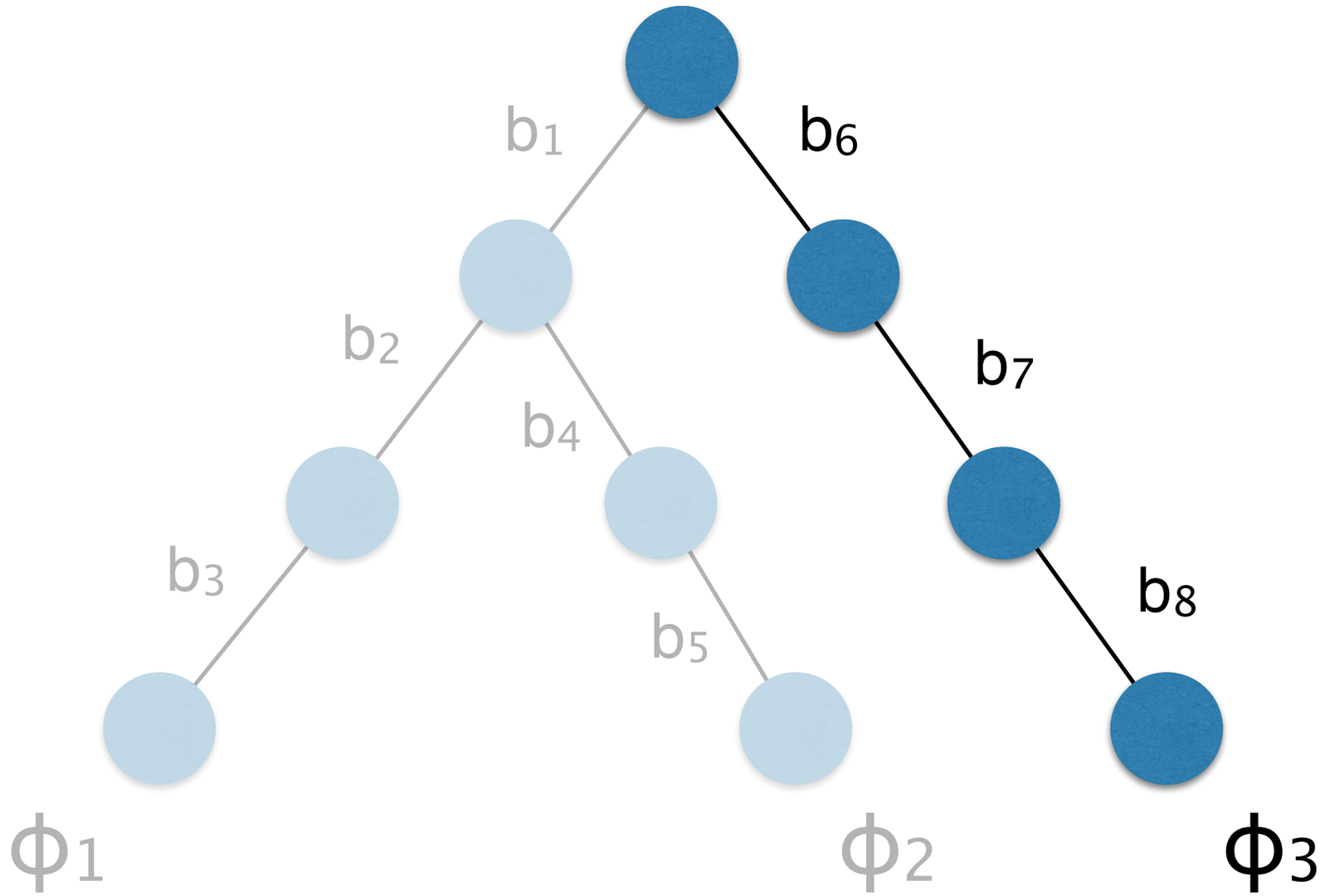
Concolic Testing



solve ($\neg b_1$)

execution tree

Concolic Testing



execution tree

Concolic Testing Algorithm

Input : Program P, initial input vector v_0, budget N
Output : The number of branches covered

1: $T \leftarrow \langle\rangle$
2: $v \leftarrow v_0$
3: **for** $m = 1$ to N **do**
4: $\quad \Phi_m \leftarrow \mathsf{RunProgram}(P, v)$
5: $\quad T \leftarrow T \cdot \Phi_m$
6: $\quad$ **repeat**
7: $\quad\quad (\Phi, \phi_i) \leftarrow \mathsf{Choose}(T) \qquad (\Phi = \phi_1 \wedge \cdots \wedge \phi_n)$
8: $\quad$ **until** $\mathsf{SAT}(\bigwedge_{j<i} \phi_j \wedge \neg\phi_i)$
9: $\quad v \leftarrow \mathsf{model}(\bigwedge_{j<i} \phi_j \wedge \neg\phi_i)$
10: **end for**
11: **return** $|\mathsf{Branches}(T)|$

Concolic Testing Algorithm

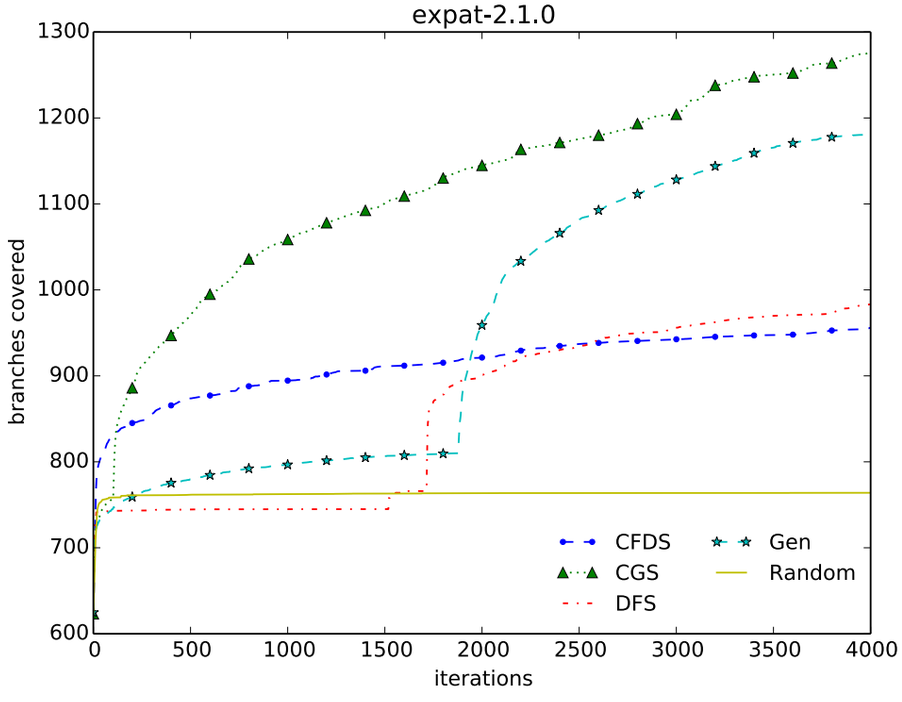
Input : Program P, initial input vector v_0, budget N
Output : The number of branches covered

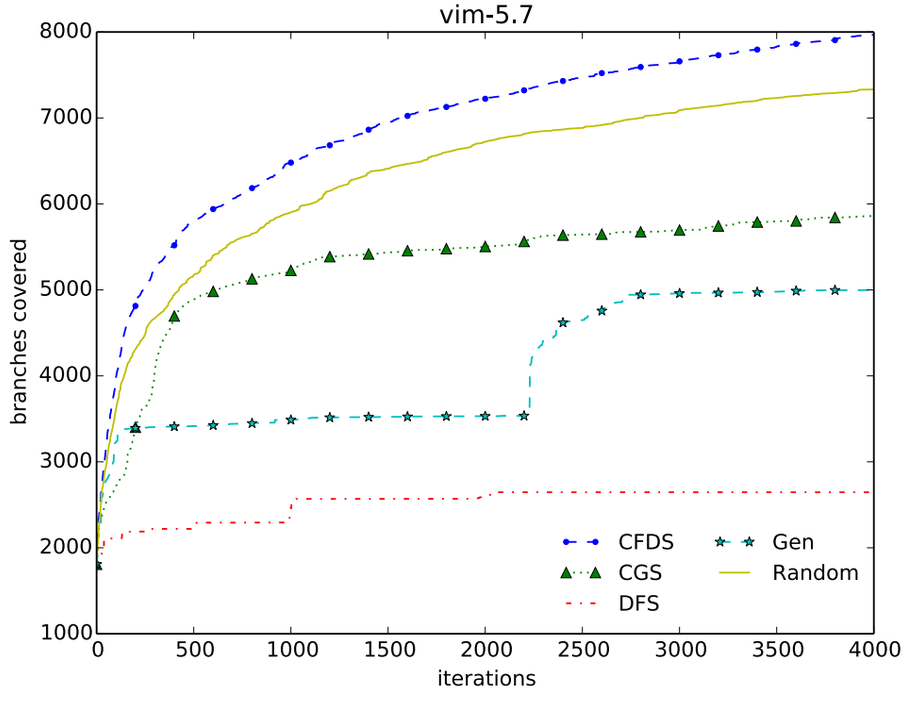
1: $T \leftarrow \langle\rangle$
2: $v \leftarrow v_0$
3: **for** $m = 1$ to N **do**
4: $\quad \Phi_m \leftarrow$ RunProgram(P
5: $\quad T \leftarrow T \cdot \Phi_m$
6: $\quad$ **repeat**
7: $\quad\quad (\Phi, \phi_i) \leftarrow$ Choose(T) $\qquad (\Phi = \phi_1 \wedge \cdots \wedge \phi_n)$
8: $\quad$ **until** SAT($\bigwedge_{j<i} \phi_j \wedge \neg\phi_i$)
9: $\quad v \leftarrow$ model($\bigwedge_{j<i} \phi_j \wedge \neg\phi_i$)
10: **end for**
11: **return** |Branches(T)|

Search Heuristic

Application to Symbolic Execution

- Exissting search heuristics have been hand-tuned:
  - e.g., CGS [FSE'14], CarFast [FSE'12], CFDS [ASE'08], Generational [NDSS'08], DFS [PLDI'05], …



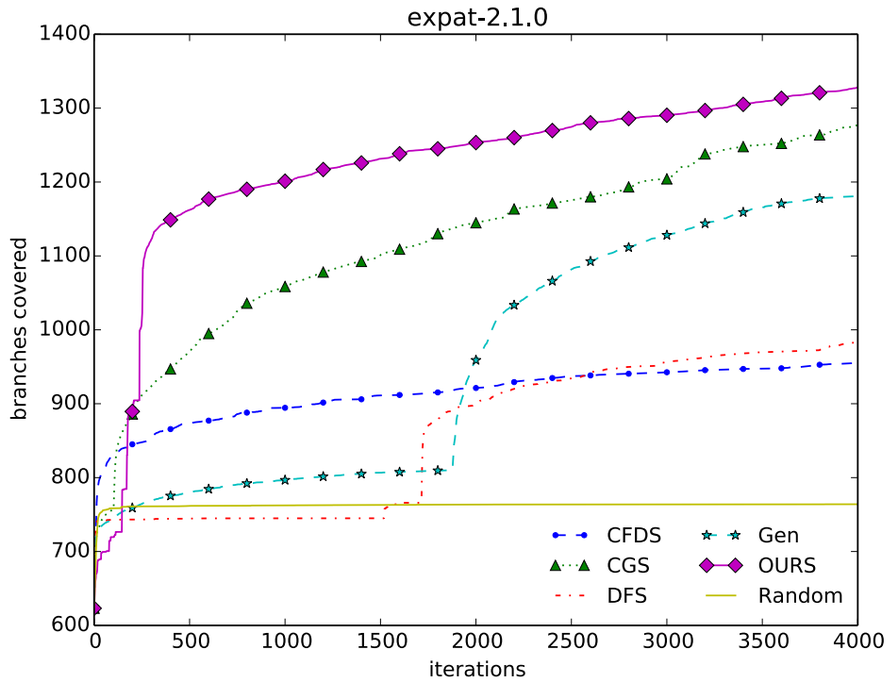


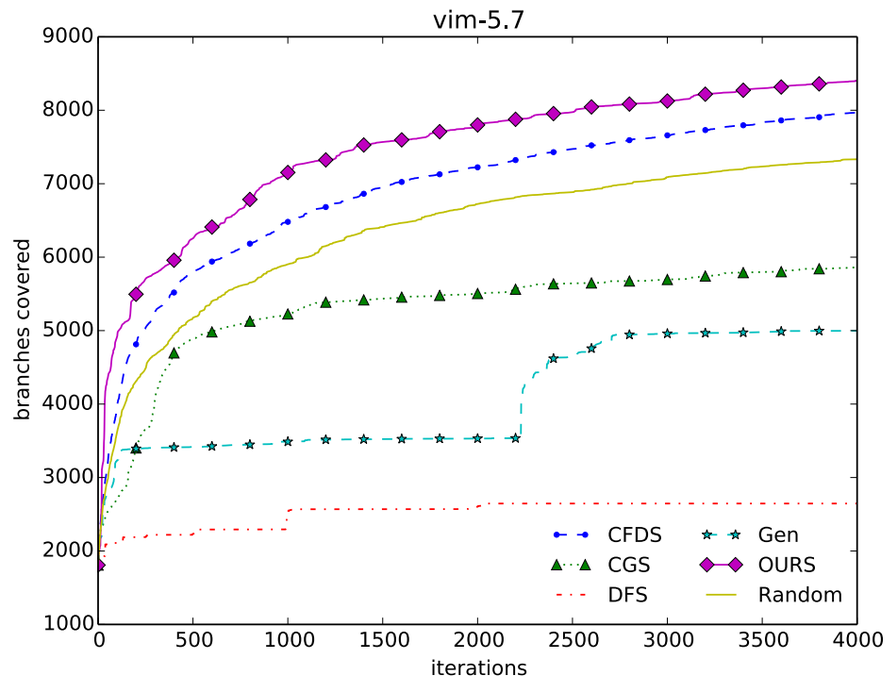
Our goal: automatically generating path-selection heuristics

ICSE'18

Application to Symbolic Execution

- Developed "data-driven symbolic execution"
  - considerable increase in code coverage





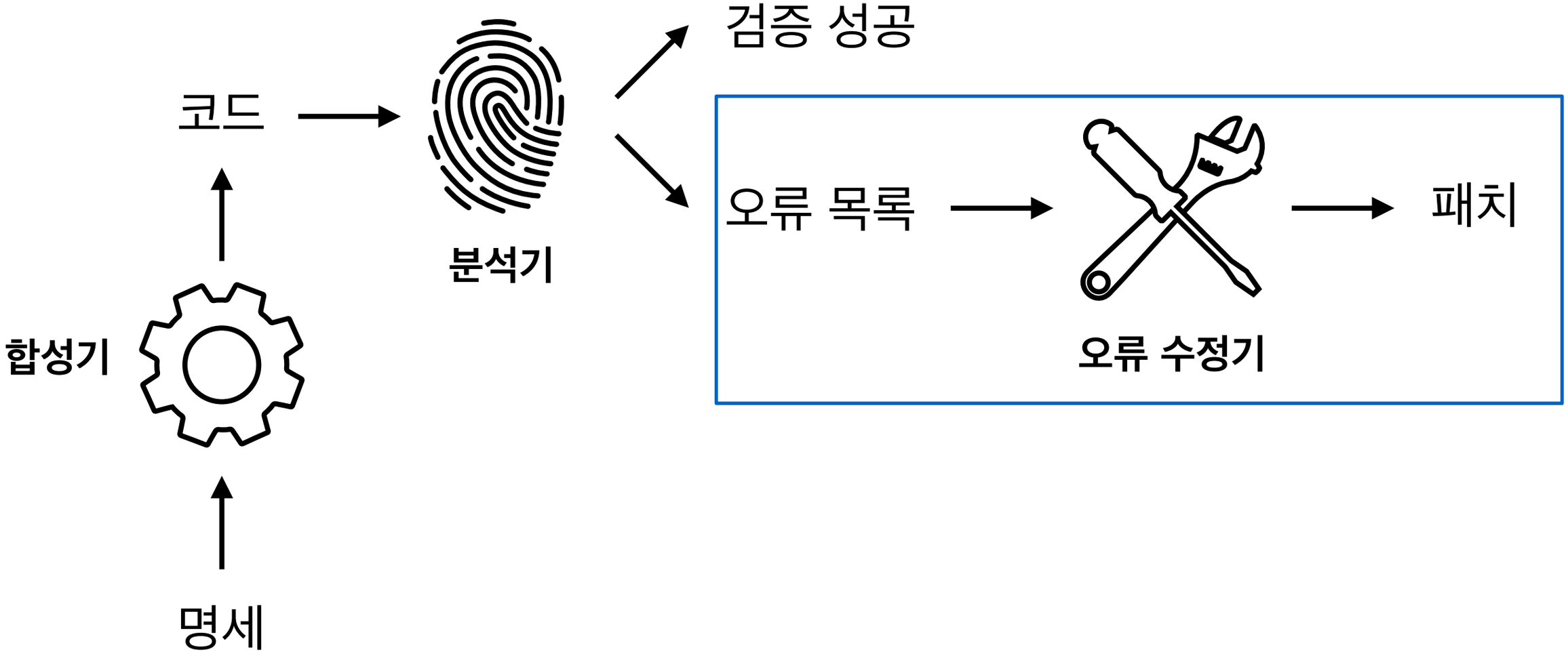
- dramatic increase in bug-finding capability

	OURS	CFDS	CGS	Random	Gen	DFS
gawk-3.0.3	**100/100**	0/100	0/100	0/100	0/100	0/100
grep-2.2	**47/100**	0/100	5/100	0/100	0/100	0/100

	Phenomenons	**Bug-Triggering Inputs**	**Version**
sed	Memory Exhaustion	'**H** **g** **;D**'	4.4(latest)
sed	Infinite File Write	'**H** **w {-** **x; D**'	4.4(latest)
grep	Segmentation Fault	'**\(\)\1\+****'	3.1(latest)
grep	Non-Terminating	'**?(^(|^+*)*\+\{8957\}**'	3.1(latest)
gawk	Memory Exhaustion	'**$6672467e2=E7**'	4.21(latest)

연구 방향

- Q) 어떻게 안전한 소프트웨어를 손쉽게 만들것인가?
- A) 소프트웨어 자동 **분석**, **패치**, **합성** 기술



자동 디버깅 기술의 필요성

- 소프트웨어 개발에서 디버깅은 전체 시간의 절반을 차지
  - 상용 소프트웨어 오류 수정에 평균 200일 소요[1)]
  - 오류/취약점은 해마다 증가 개수: e.g., CVE 등록수 4,000('10년), 6,000('15년)
- 다른 개발 단계에 비해 자동화된 도구 지원이 가장 적음
  - cf) 소프트웨어 오류 탐지 분야는 지난 30여년간 눈부신 발전
  - 개발자에 전적으로 의존할수 밖에 없지만 가장 어렵고 부담스러운 단계

1) Kim and Whitehead. How long did it take to fix bugs? MSR 2006

메모리 해제 오류

- 메모리 관리를 수동으로 해야하는 언어(e.g., C/C++) 발생
  - Memory-leak (CWE-401): 메모리를 너무 늦게 해제
  - Use-after-free (CWE-416): 메모리를 너무 빨리 해제
  - Double-free (CWE-415): 메모리를 여러번 해제

Memory-Leak

```
p = malloc(1);
...
return;
```

Use-After-Free

```
p = malloc(1);
...
free(p);
...
use(p);
```

Double-Free

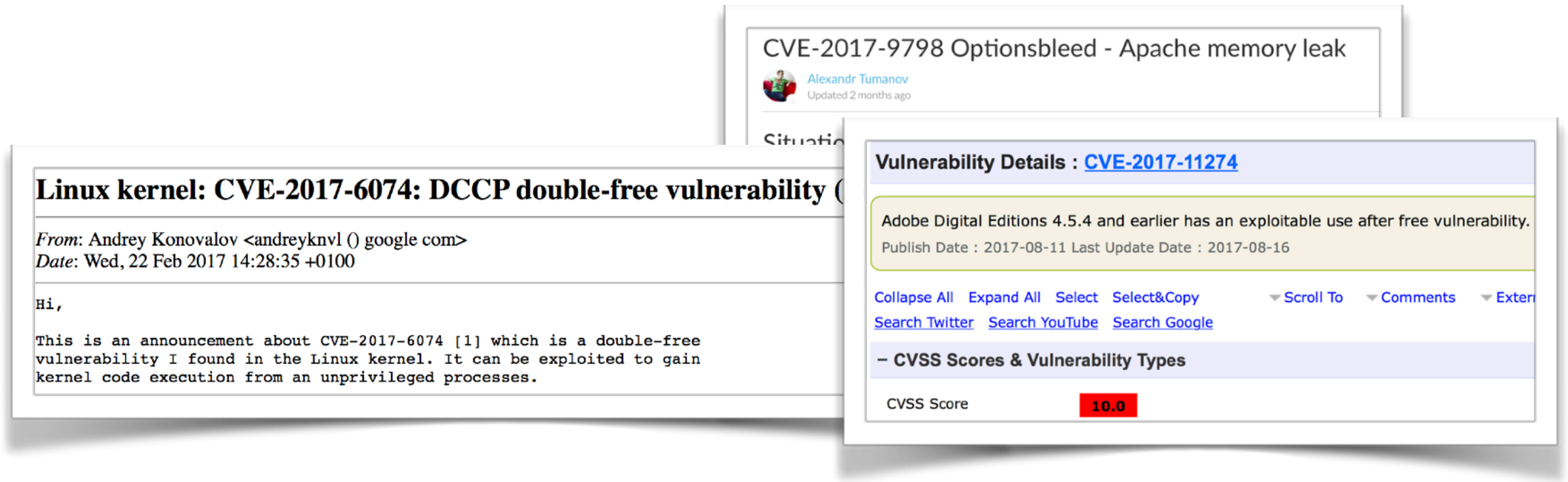
```
p = malloc(1);
...
free(p);
...
free(p);
```

메모리 해제 오류

- C/C++ 프로그램에서 가장 골칫거리중 하나

Repository	#commits	ML	DF	UAF	Total	*-overflow
linux	721,119	3,740	821	1,986	**6,363**	5,092
openssl	21,009	220	36	12	**264**	61
numpy	17,008	58	2	2	**59**	53
php	105,613	1,129	148	197	**1,449**	649
git	49,475	350	19	95	**442**	258

- 소프트웨어 결함의 주요 원인이지만 정확한 수정이 까다로움



실제 사례
(Linux Kernel)

```
in = malloc(1);
out = malloc(1);
... // use in, out
free(out);
free(in);

in = malloc(2);
if (in == NULL) {

  goto err;
}
```

```
out = malloc(2);
if (out == NULL) {
  free(in);

  goto err;
}
... // use in, out
err:
  free(in);
  free(out);
  return;
```

실제 사례
(Linux Kernel)

```
in = malloc(1);
out = malloc(1);
... // use in, out
free(out);
free(in);

in = malloc(2);
if (in == NULL) {

  goto err;
}

out = malloc(2);
if (out == NULL) {
  free(in);

  goto err;
}
... // use in, out
err:
  free(in);
  free(out);
  return;
```

double-free

실제 사례
(Linux Kernel)

```
in = malloc(1);
out = malloc(1);
... // use in, out
free(out);
free(in);

in = malloc(2);
if (in == NULL) {

  goto err;
}

out = malloc(2);
if (out == NULL) {
  free(in);

  goto err;
}
... // use in, out
err:
  free(in);
  free(out);
  return;
```

double-free

실제 사례
(Linux Kernel)


```
in = malloc(1);
out = malloc(1);
... // use in, out
free(out);
free(in);

in = malloc(2);
if (in == NULL) {
  out = NULL;
  goto err;
}
```

```
out = malloc(2);
if (out == NULL) {
  free(in);
  in = NULL;
  goto err;
}
... // use in, out
err:
  free(in);
  free(out);
  return;
```

실제 사례
(Linux Kernel)

수동 디버깅의 문제 1:
오류가 사라졌는지 확신하기 어려움

```
in = malloc(1);
out = malloc(1);
... // use in, out
free(out);
free(in);

in = malloc(2);
if (in == NULL) {
  out = NULL;
  goto err;
}

out = malloc(2);
if (out == NULL) {
  free(in);
  in = NULL;
  goto err;
}
... // use in, out
err:
  free(in);
  free(out);
  return;
```

실제 사례
(Linux Kernel)


```
in = malloc(1);
out = malloc(1);
... // use in, out
// removed
free(in);

in = malloc(2);
if (in == NULL) {
  out = NULL;
  goto err;
}
free(out);
out = malloc(2);
if (out == NULL) {
  free(in);
  in = NULL;
  goto err;
}
... // use in, out
err:
  free(in);
  free(out);
  return;
```

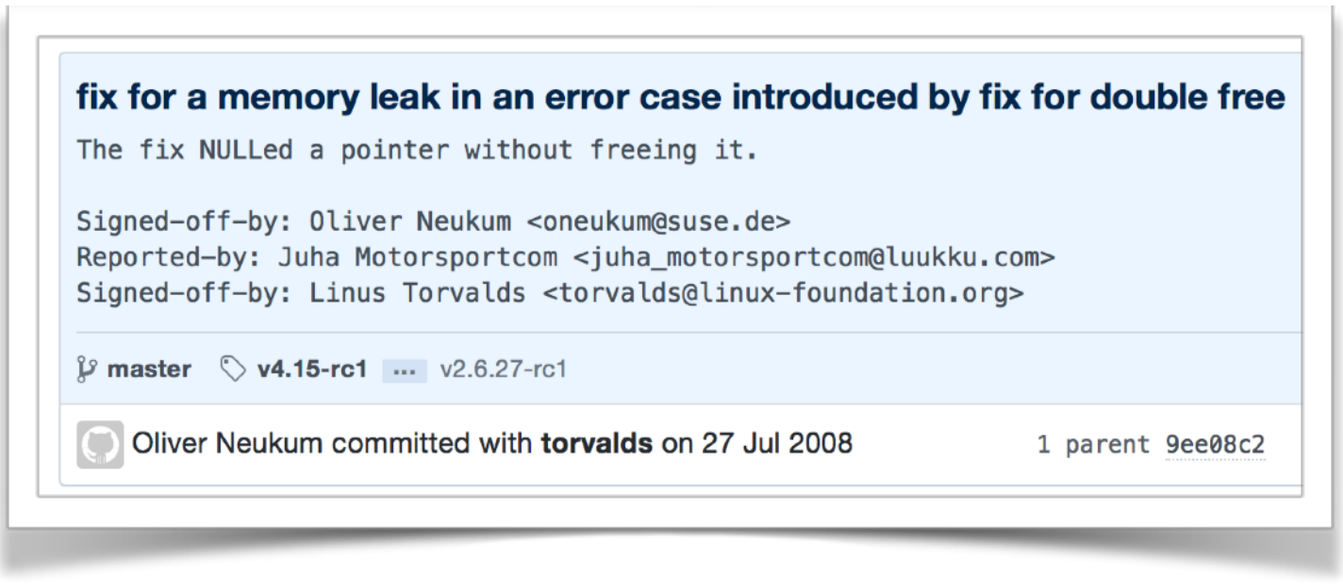
실제 사례
(Linux Kernel)

수동 디버깅의 문제 2:
고치는 과정에서 새로운 오류가 발생

memory leak

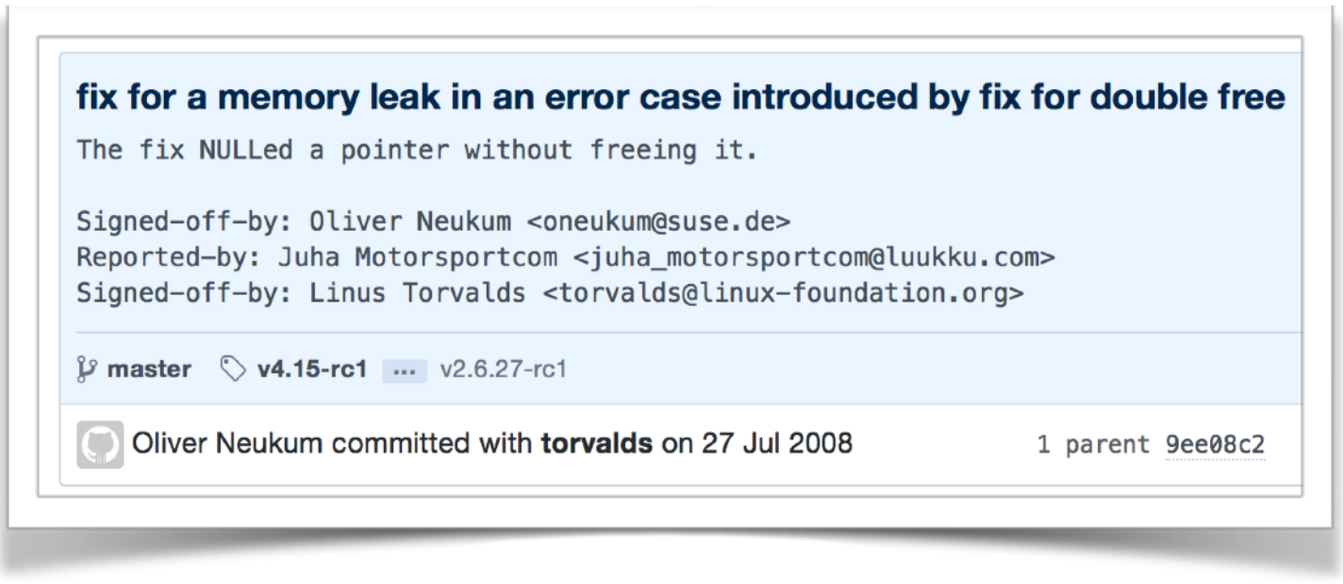

```
in = malloc(1);
out = malloc(1);
... // use in, out
// removed
free(in);

in = malloc(2);
if (in == NULL) {
  out = NULL;
  goto err;
}
free(out);
out = malloc(2);
if (out == NULL) {
  free(in);
  in = NULL;
  goto err;
}
... // use in, out
err:
  free(in);
  free(out);
  return;
```

실제 사례
(Linux Kernel)




```
in = malloc(1);
out = malloc(1);
... // use in, out
free(out);
free(in);
out = NULL;
in = malloc(2);
if (in == NULL) {
  out = NULL;
  goto err;
}
// removed
out = malloc(2);
if (out == NULL) {
  free(in);
  in = NULL;
  goto err;
}
... // use in, out
err:
  free(in);
  free(out);
  return;
```

실제 사례
(Linux Kernel)



수동 디버깅의 문제 3: 수정된 코드가 복잡

```
in = malloc(1);
out = malloc(1);
... // use in, out
free(out);
free(in);
out = NULL;
in = malloc(2);
if (in == NULL) {
  out = NULL;
  goto err;
}
// removed
out = malloc(2);
if (out == NULL) {
  free(in);
  in = NULL;
  goto err;
}
... // use in, out
err:
  free(in);
  free(out);
  return;
```

메모리 오류 자동 수정기

```
in = malloc(1);
out = malloc(1);
... // use in, out
free(out);
free(in);

in = malloc(2);
if (in == NULL) {
  goto err;
}

out = malloc(2);
if (out == NULL) {
  free(in);

  goto err;
}
... // use in, out
err:
  free(in);
  free(out);
  return;
```

패치 자동 생성

```
in = malloc(1);
out = malloc(1);
... // use in, out
// removed
free(in);

in = malloc(2);
if (in == NULL) {
  goto err;
}
free(out);
out = malloc(2);
if (out == NULL) {
  // removed

  goto err;
}
... // use in, out
err:
  free(in);
  free(out);
  return;
```

메모리 오류 자동 수정기

```
in = malloc(1);
out = malloc(1);
... // use in, out
free(out);
free(in);

in = malloc(2);
if (in == NULL) {
  goto err;
}

out = malloc(2);
if (out == NULL) {
  free(in);

  goto err;
}
... // use in, out
err:
  free(in);
  free(out);
  return;
```

패치 자동 생성

수동 디버깅의 문제 해결:

1. 대상 오류가 반드시 제거됨
2. 새로운 오류가 발생하지 않음
3. 간결한 패치 (최소한의 변경)

=> **수학적 보장**.
추가적인 리뷰 불필요.

```
in = malloc(1);
out = malloc(1);
... // use in, out
// removed
free(in);

in = malloc(2);
if (in == NULL) {
  goto err;
}
free(out);
out = malloc(2);
if (out == NULL) {
  // removed

  goto err;
}
... // use in, out
err:
  free(in);
  free(out);
  return;
```

FSE'18

MemFix

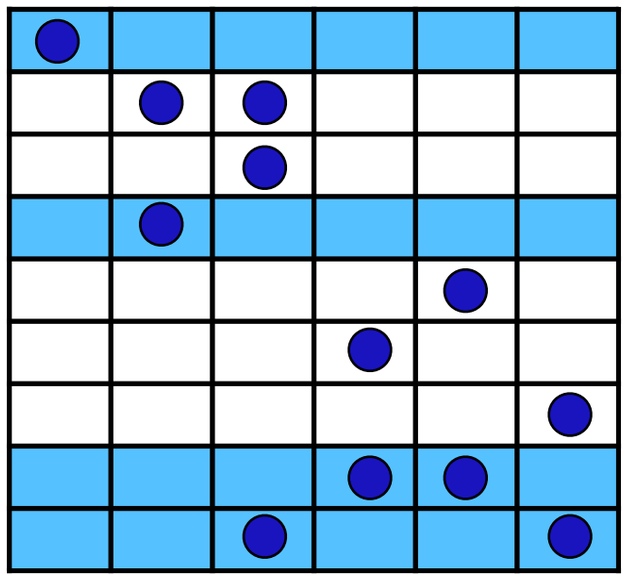
- Automatically repairs deallocation errors
  - **memory-leak**, **double-free** and **use-after-free**
- Key features
  - **sound**: generated patch is guaranteed to be correct
  - **safe**: no new errors are introduced
- Approach: **Static Analysis** + **Exact Cover Problem**

Key Insight

```
1   out = malloc(1);
2   in = malloc(1);
3   … // use in, out
4   free(out);
5   free(in);
6
7   in = malloc(2);
8   if(in == NULL) {
9
10      goto err;
11  }
12
13  out = malloc(2);
14  if(out == NULL) {
15      free(in);
16
17      goto err;
18  }
19  … // use in, out
20 err:
21  free(in);
22  free(out);
```

Find a set of free-statements

|||



Solve an Exact Cover Problem

```
1   out = malloc(1);
2   in = malloc(1);
3   … // use in, out
4   // -
5   free(in);
6
7   in = malloc(2);
8   if(in == NULL) {
9
10      goto err;
11  }
12  free(out); // +
13  out = malloc(2);
14  if(out == NULL) {
15      // -
16
17      goto err;
18  }
19  … // use in, out
20 err:
21  free(in);
22  free(out);
```

SAS'17

Synthesizing Imperative Programs

- Specification is given as test cases

$$\text{reverse}(12) = 21,\ \text{reverse}(123) = 321$$

```
reverse(n){
  r := 0;
  while (          ){
    
  };
  return r;
}
```

2.5s

```
reverse(n){
  r := 0;
  while ( n > 0 ){
    x := n % 10;
    r := r * 10;
    r := r + x;
    n := n / 10;
  };
  return r;
}
```

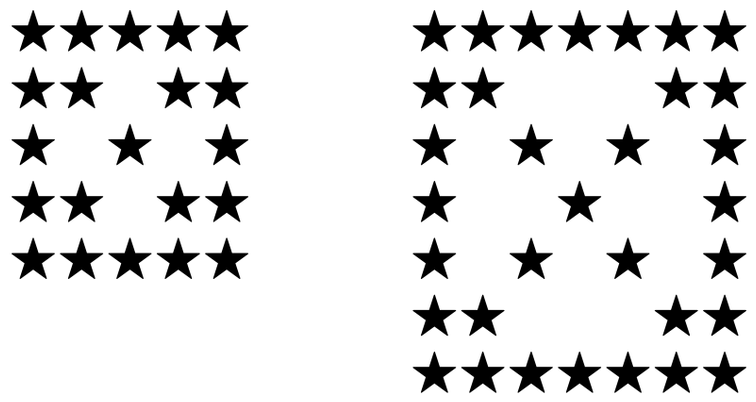
Performance

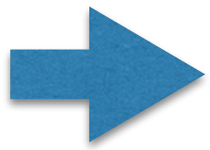
- Better than humans for introductory programming tasks

Domain	No	Description	Vars		Ints	Exs	Time (sec)		
			IVars	AVars			Base	Base+Opt	Ours
Integer	1	Given n, return $n!$.	2	0	2	4	0.0	0.0	0.0
	2	Given n, return $n!!$ (i.e., double factorial).	3	0	3	4	0.0	0.0	0.0
	3	Given n, return $\sum_{i=1}^{n} i$.	3	0	2	4	0.1	0.0	0.0
	4	Given n, return $\sum_{i=1}^{n} i^2$.	4	0	2	3	122.4	18.1	0.3
	5	Given n, return $\prod_{i=1}^{n} i^2$.	4	0	2	3	102.9	13.6	0.2
	6	Given a and n, return a^n.	4	0	2	4	0.7	0.1	0.1
	7	Given n and m, return $\sum_{i=n}^{m} i$.	3	0	2	3	0.2	0.0	0.0
	8	Given n and m, return $\prod_{i=n}^{m} i$.	3	0	2	3	0.2	0.0	0.1
	9	Count the number of digit for an integer.	3	0	3	3	0.0	0.0	0.0
	10	Sum the digits of an integer.	3	0	3	4	5.2	2.2	1.3
	11	Calculate product of digits of an intger.	3	0	3	3	0.7	2.3	0.3
	12	Count the number of binary digit of an integer.	2	0	3	3	0.0	0.0	0.0
	13	Find the nth Fibonacci number.	3	0	3	4	98.7	13.9	2.6
	14	Given n, return $\sum_{i=1}^{n}(\sum_{m=1}^{i} m))$.	3	0	2	4	$\perp$	324.9	37.6
	15	Given n, return $\prod_{i=1}^{n}(\prod_{m=1}^{i} m))$.	3	0	2	4	$\perp$	316.6	86.9
	16	Reverse a given integer.	3	0	3	3	$\perp$	367.3	2.5
Array	17	Find the sum of all elements of an array.	3	1	2	2	8.1	3.6	0.9
	18	Find the product of all elements of an array.	3	1	2	2	7.6	3.9	0.9
	19	Sum two arrays of same length into one array.	3	2	2	2	44.6	29.9	0.2
	20	Multiply two arrays of same length into one array.	3	2	2	2	47.4	26.4	0.3
	21	Cube each element of an array.	3	1	1	2	1283.3	716.1	13.0
	22	Manipulate each element into 4th power.	3	1	1	2	1265.8	715.5	13.0
	23	Find a maximum element.	3	1	2	2	0.9	0.7	0.4
	24	Find a minimum element.	3	1	2	2	0.8	0.3	0.1
	25	Add 1 to each element.	2	1	1	3	0.3	0.0	0.0
	26	Find the sum of square of each element.	3	1	2	2	2700.0	186.2	11.5
	27	Find the multiplication of square of each element.	3	1	1	2	1709.8	1040.3	12.6
	28	Sum the products of matching elements of two arrays.	3	2	1	3	20.5	38.7	1.5
	29	Sum the absolute values of each element.	2	1	1	2	45.0	50.5	12.1
	30	Count the number of each element.	3	1	3	2	238.9	1094.1	0.2
	Average						$>$ 616.8	165.5	6.6

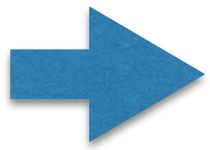
IJCAI'18

Synthesizing Pattern Programs





for i in N do:
  for j in N do:
    if ($i = 1 \,||\, i = N \,||\, j = 1 \,||\, j = i \,||$
      $j = N - i + 1 \,||\, j = N$): print ★
    else: print ␣
  print ⮐



for i in N do:
  for j in $4 * N - i - 2$ do:
    if ($j = 2 * N - i \,||\, j = 2 * N + i - 2 \,||$
      $j = 4 * N - i - 2 \,||\, j = i$): print ★
    else: print ␣
  print ⮐

Thank You!

- Research areas: programming languages, software engineering, software security
  - program analysis and testing
  - program synthesis and repair
- Publication: top-venues in PL, SE, Security, and AI:
  - PLDI('12,'14), OOPSLA('15,'17,'17,'18,'18), TOPLAS('14,'16,'17,'18,'19), ICSE('17,'18,'19), FSE('18,'19), ASE'18, S&P'17, IJCAI('17,'18), etc

http://prl.korea.ac.kr