Introduction to Software Research @Korea University

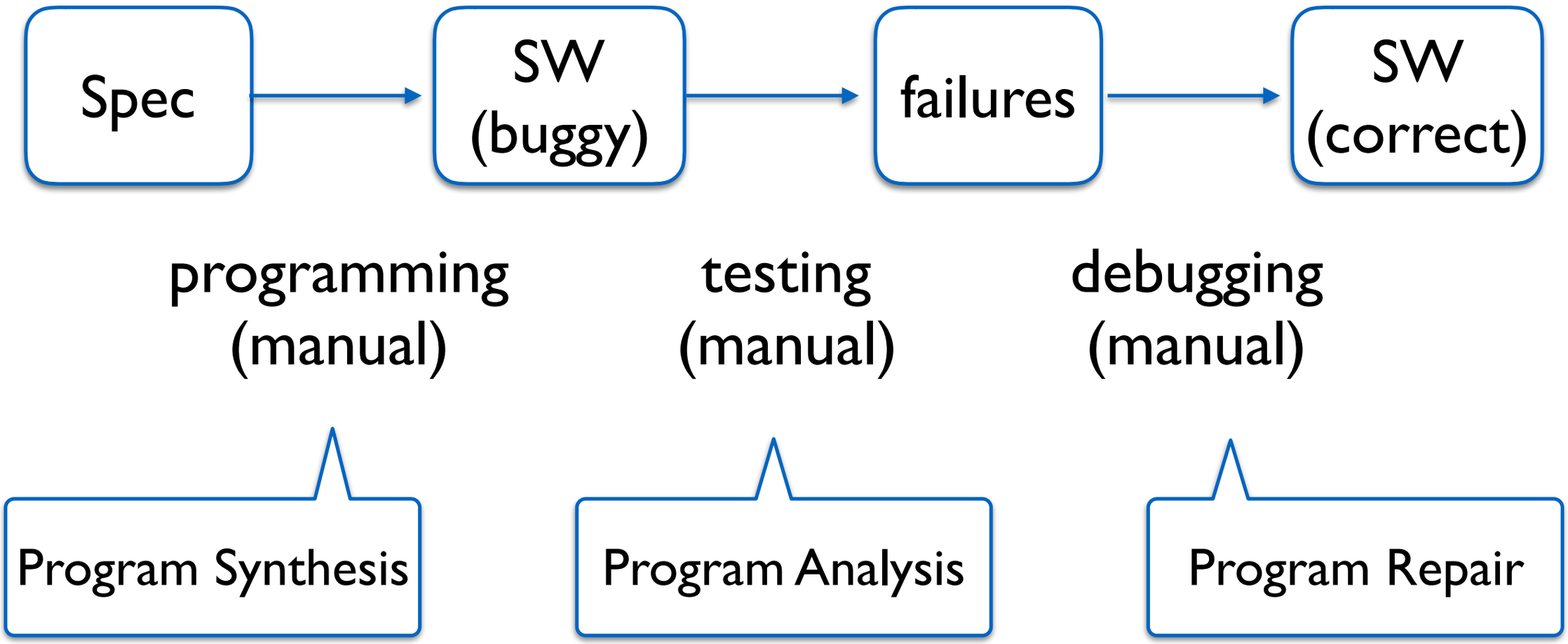
Hakjoo Oh

Korea University

11 June 2018 @S-Core

Research Goals

- Computer-aided software engineering: automated programming / testing / debugging



Data-Driven Program Analysis

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
  - Manually done by analysis designers: nontrivial & suboptimal

Goal

Automatically Generating Analysis Heuristics from Data

- Use data to make heuristic decisions in program analysis

machine learning

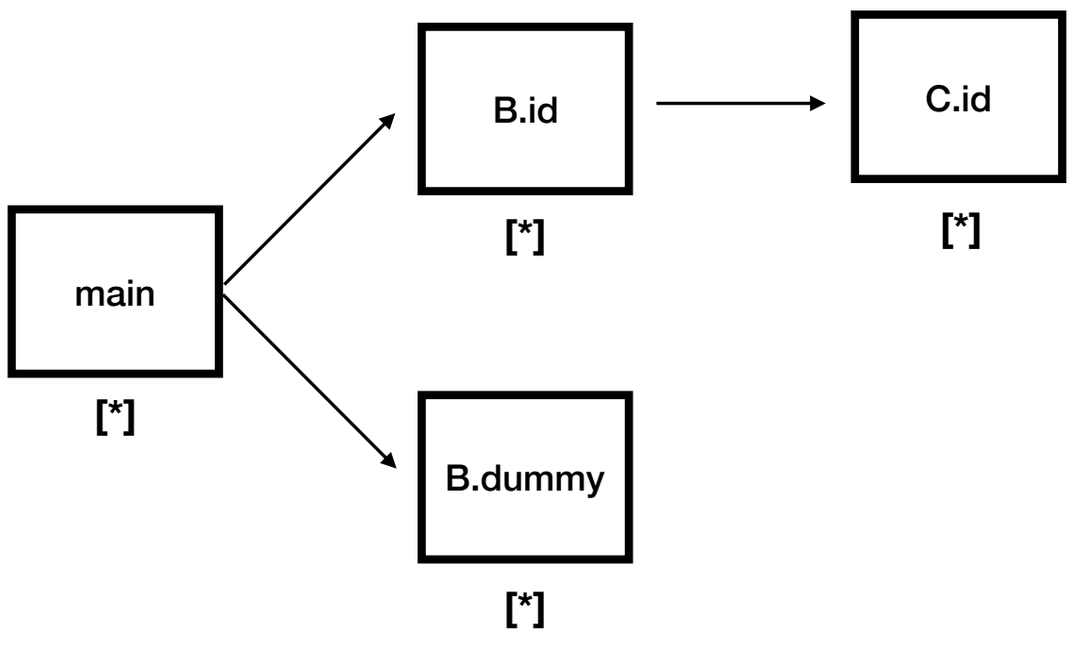
context-sensitivity heuristics
flow-sensitivity heuristics
unsoundness heuristics
path-selection heuristics
…

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

Context-Sensitivity

```
1:  class D{} class E{}
2:
3:  class C{
4:  Object id(Object v){return v;}}
5:
6:  class B{
7:  void dummy(){}
8:  Object id(Object v){
9:  C c = new C();//C1
10: return c.id(v);}}
11:
12: class A{
13: public static void main(String[] args){
14: B b1 = new B();//B1
15: B b2 = new B();//B2
16: D d = (D) b1.id1(new D());//query1
17: E e = (E) b2.id1(new E());//query2
18: b1.dummy();
19: b2.dummy();}}
```

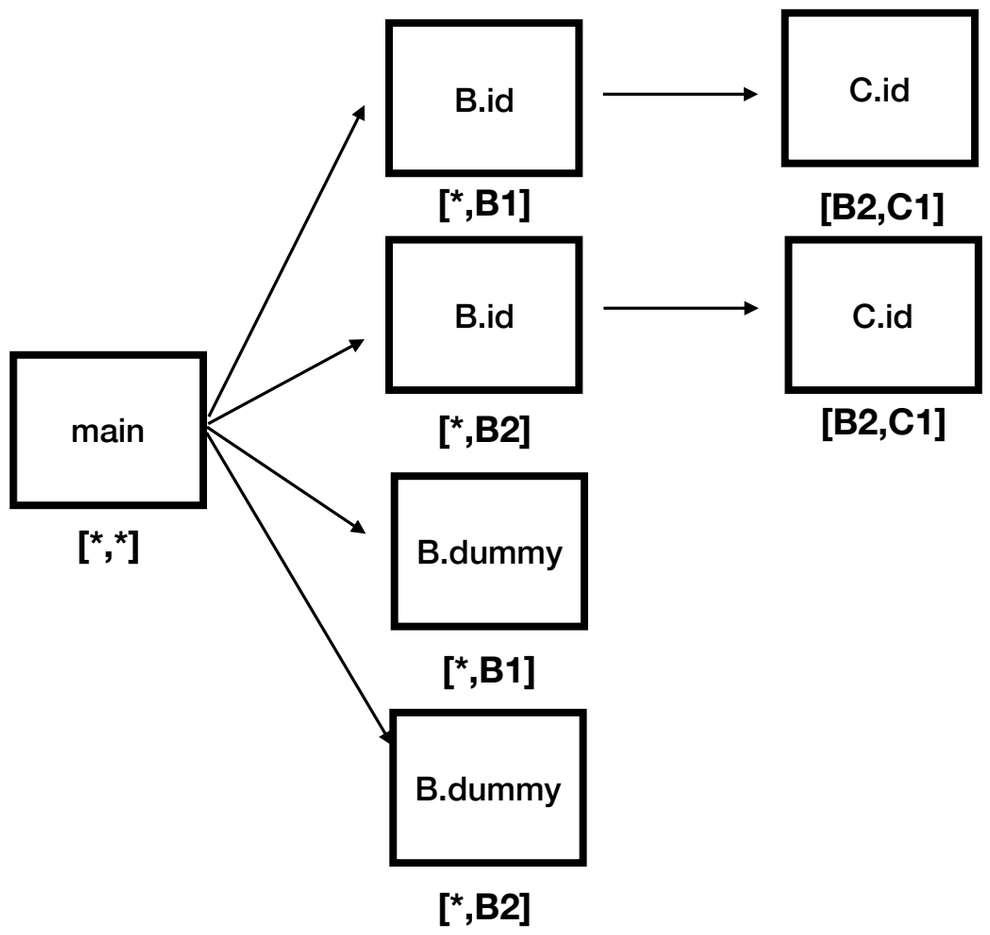
Without context-sensitivity, analysis fails to prove queries



Context-Sensitivity

```
1:  class D{} class E{}
2:
3:  class C{
4:  Object id(Object v){return v;}}
5:
6:  class B{
7:  void dummy(){}
8:  Object id(Object v){
9:  C c = new C();//C1
10: return c.id(v);}}
11:
12: class A{
13: public static void main(String[] args){
14: B b1 = new B();//B1
15: B b2 = new B();//B2
16: D d = (D) b1.id1(new D());//query1
17: E e = (E) b2.id1(new E());//query2
18: b1.dummy();
19: b2.dummy();}}
```

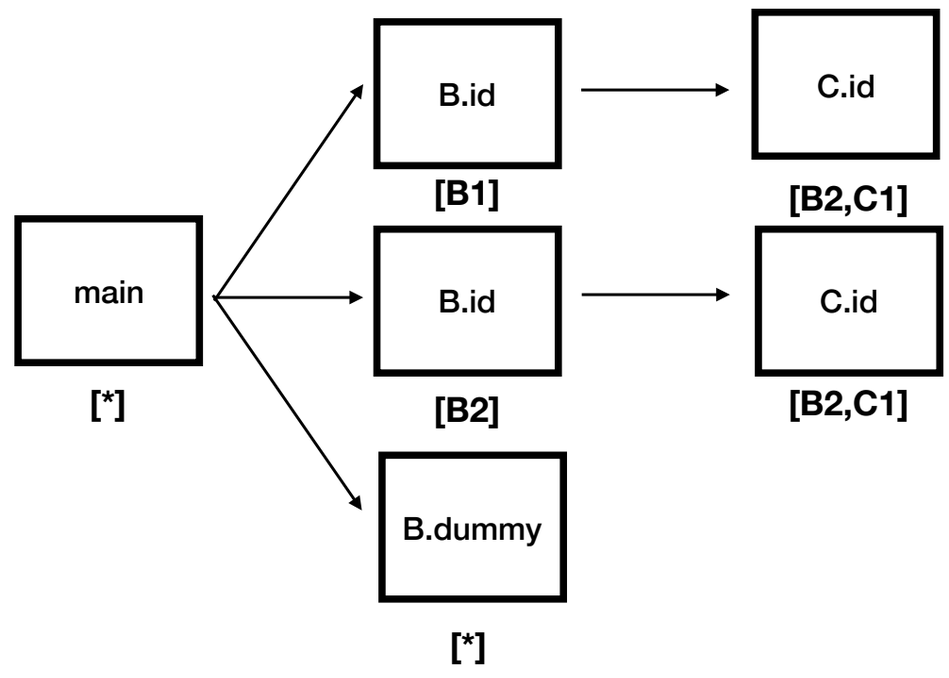
2-object-sensitivity succeeds but does not scale



Selective Context-Sensitivity

```
1:  class D{} class E{}
2:
3:  class C{
4:  Object id(Object v){return v;}}
5:
6:  class B{
7:  void dummy(){}
8:  Object id(Object v){
9:  C c = new C();//C1
10: return c.id(v);}}
11:
12: class A{
13: public static void main(String[] args){
14: B b1 = new B();//B1
15: B b2 = new B();//B2
16: D d = (D) b1.id1(new D());//query1
17: E e = (E) b2.id1(new E());//query2
18: b1.dummy();
19: b2.dummy();}}
```

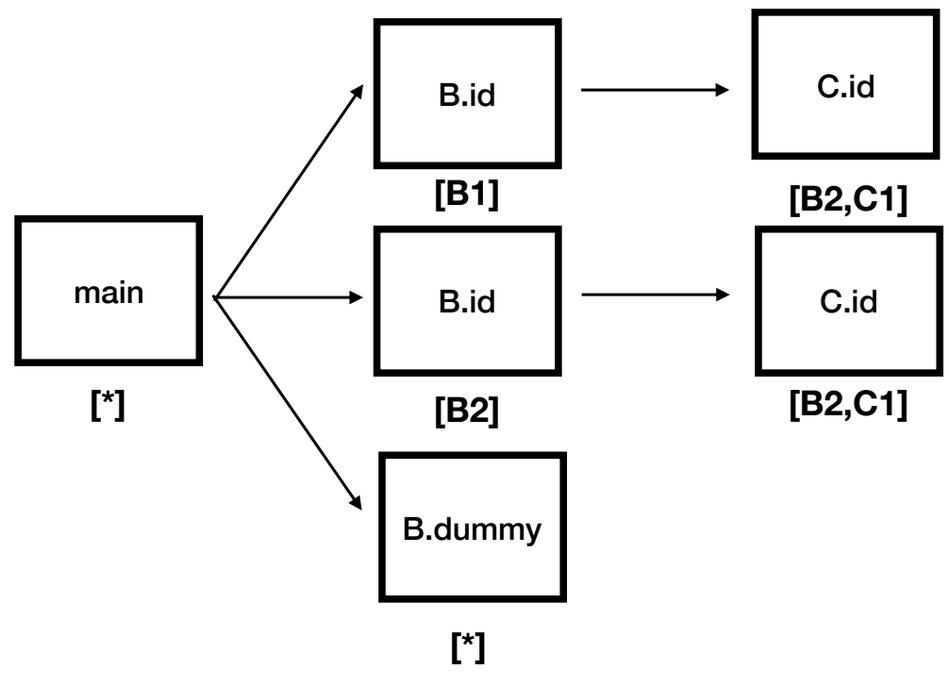
Apply 2-obj-sens: {C.id}
Apply 1-obj-sens: {B.id}
Apply insens: {B.m}



Selective Context-Sensitivity

```
1:  class D{} class E{}
2:
3:  class C{
4:  Object id(Object v){return v;}}
5:
6:  class B{
7:  void dummy(){}
8:  Object id(Object v){
9:  C c = new C();//C1
10: return c.id(v);}}
11:
12: class A{
13: public static void main(String[] args){
14: B b1 = new B();//B1
15: B b2 = new B();//B2
16: D d = (D) b1.id1(new D());//query1
17: E e = (E) b2.id1(new E());//query2
18: b1.dummy();
19: b2.dummy();}}
```

Apply 2-obj-sens: {C.id}
Apply 1-obj-sens: {B.id}
Apply insens: {B.m}



Challenge: How to decide? Data-driven approach

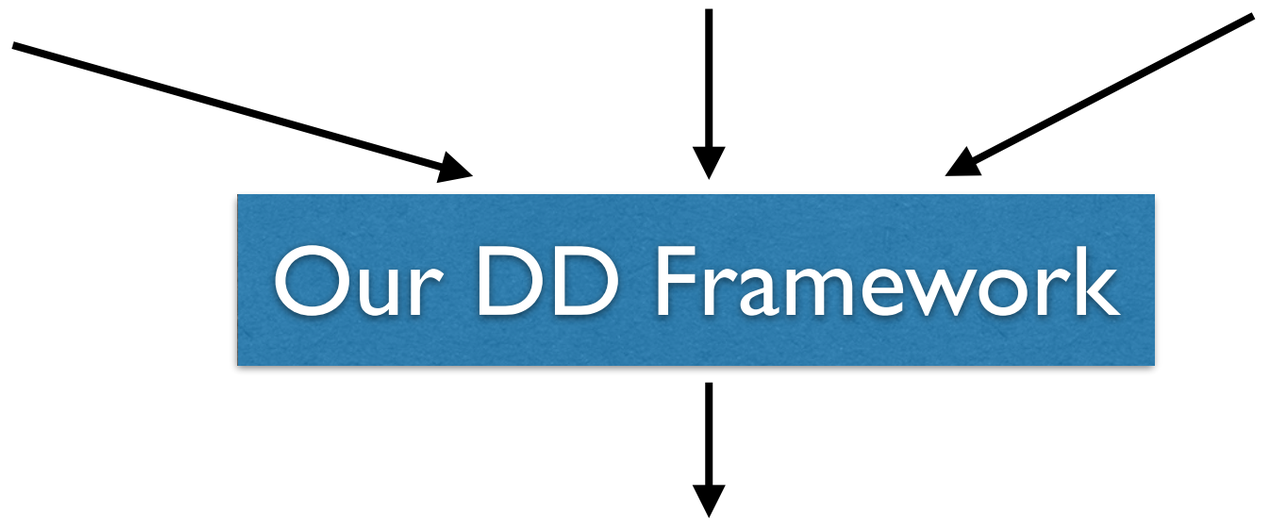
OOPSLA'17

Our Data-Driven Approach

Parametric static analyzer

Training data (programs)

Atomic features (a1,a2,…,a25)



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

OOPSLA'17

Our Data-Driven Approach

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

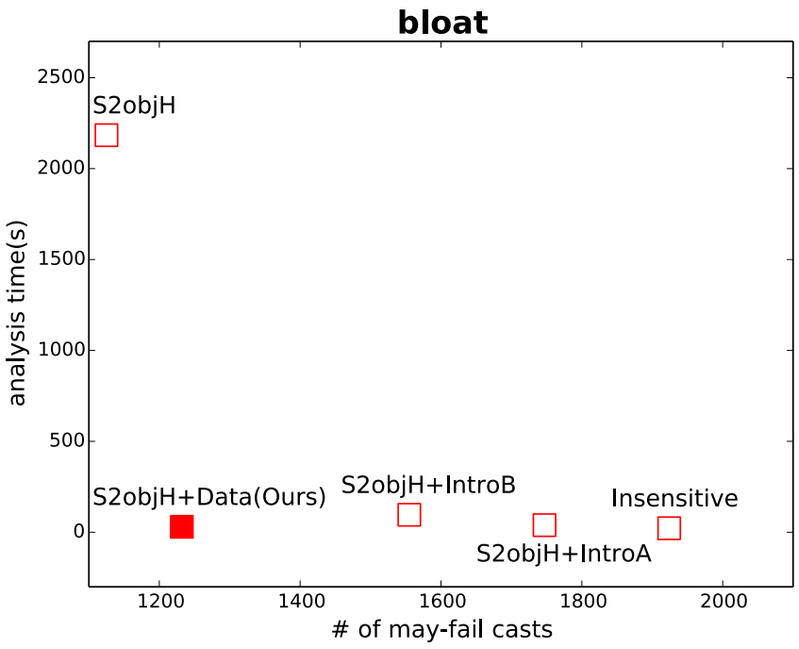
$$1 \wedge \neg 3 \wedge \neg 6 \wedge 8 \wedge \neg 9 \wedge \neg 16 \wedge \neg 17 \wedge \neg 18 \wedge \neg 19 \wedge \neg 20 \wedge \neg 21 \wedge \neg 22 \wedge \neg 23 \wedge \neg 24 \wedge \neg 25$$

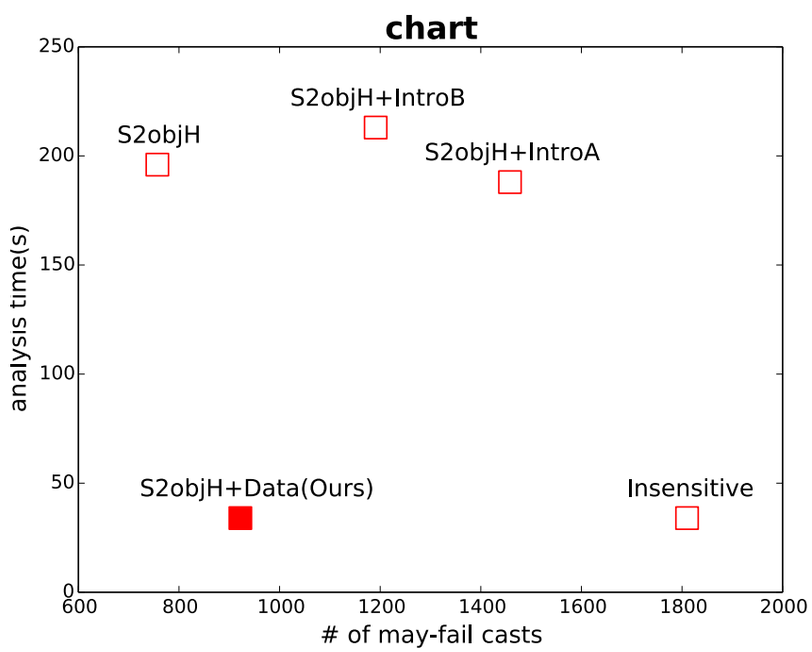
f1: Methods that require 1-object-sensitivity

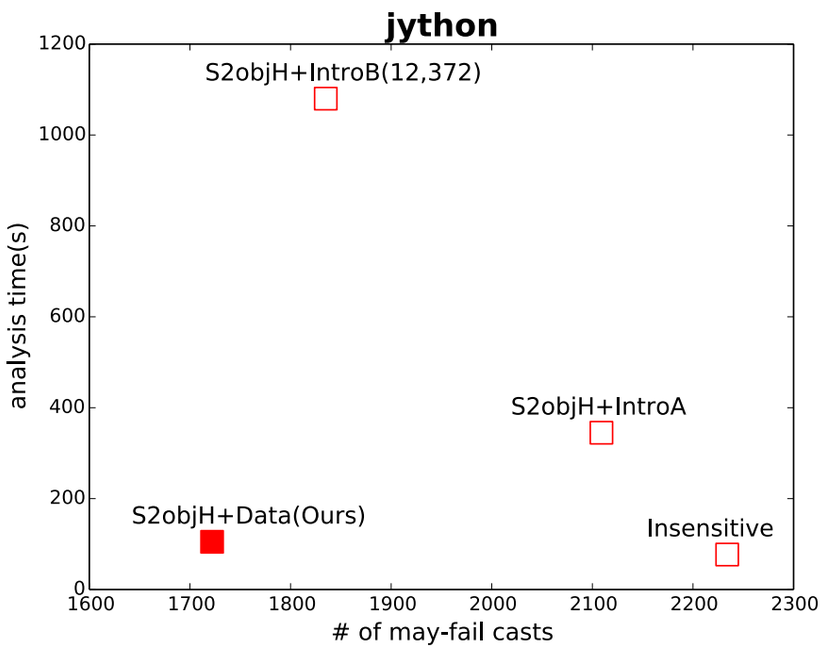
$$(1 \wedge \neg 3 \wedge \neg 4 \wedge \neg 7 \wedge \neg 8 \wedge 6 \wedge \neg 9 \wedge \neg 15 \wedge \neg 16 \wedge \neg 17 \wedge \neg 18 \wedge \neg 19 \wedge \neg 20 \wedge \neg 21 \wedge \neg 22 \wedge \neg 23 \wedge \neg 24 \wedge \neg 25) \vee$$
$$(\neg 3 \wedge \neg 4 \wedge \neg 7 \wedge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

Performance

- Training with 4 small programs from DaCapo, and applied to 6 large programs (1 for validation)
- Machine-tuning outperforms hand-tuning







Other Context-Sensitivities

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

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

  - Depth-1 formula (f_1):

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

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

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

  - Depth-1 formula (f_1):

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

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

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

  - Depth-1 formula (f_1):

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

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:

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

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
	x=22, y=7, z=14	x=α, y=β, z=2*β 2*β ≠ α

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
	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

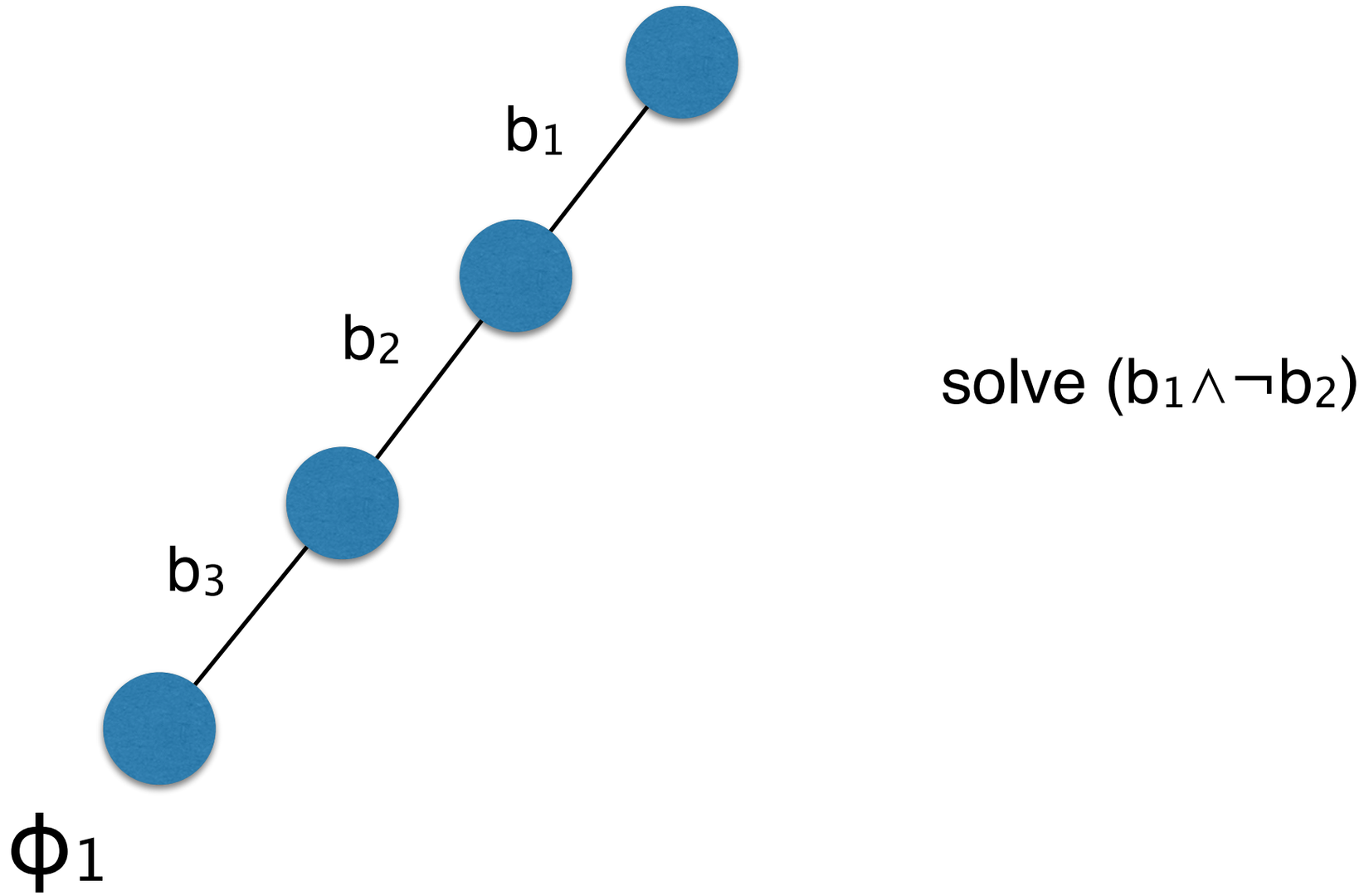
```
int double (int v) {
  return 2*v;
}

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

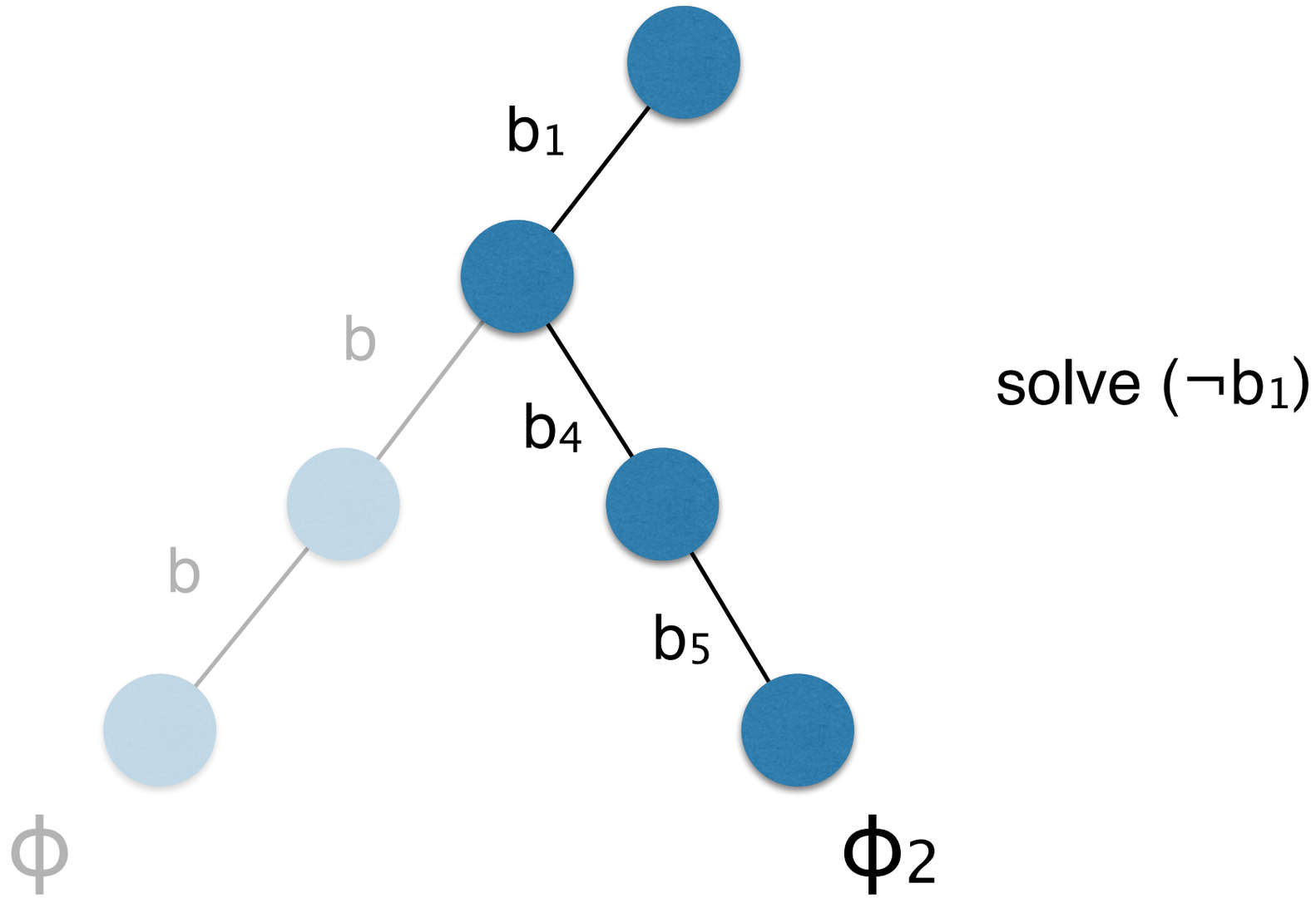
3rd iteration

Concolic Testing



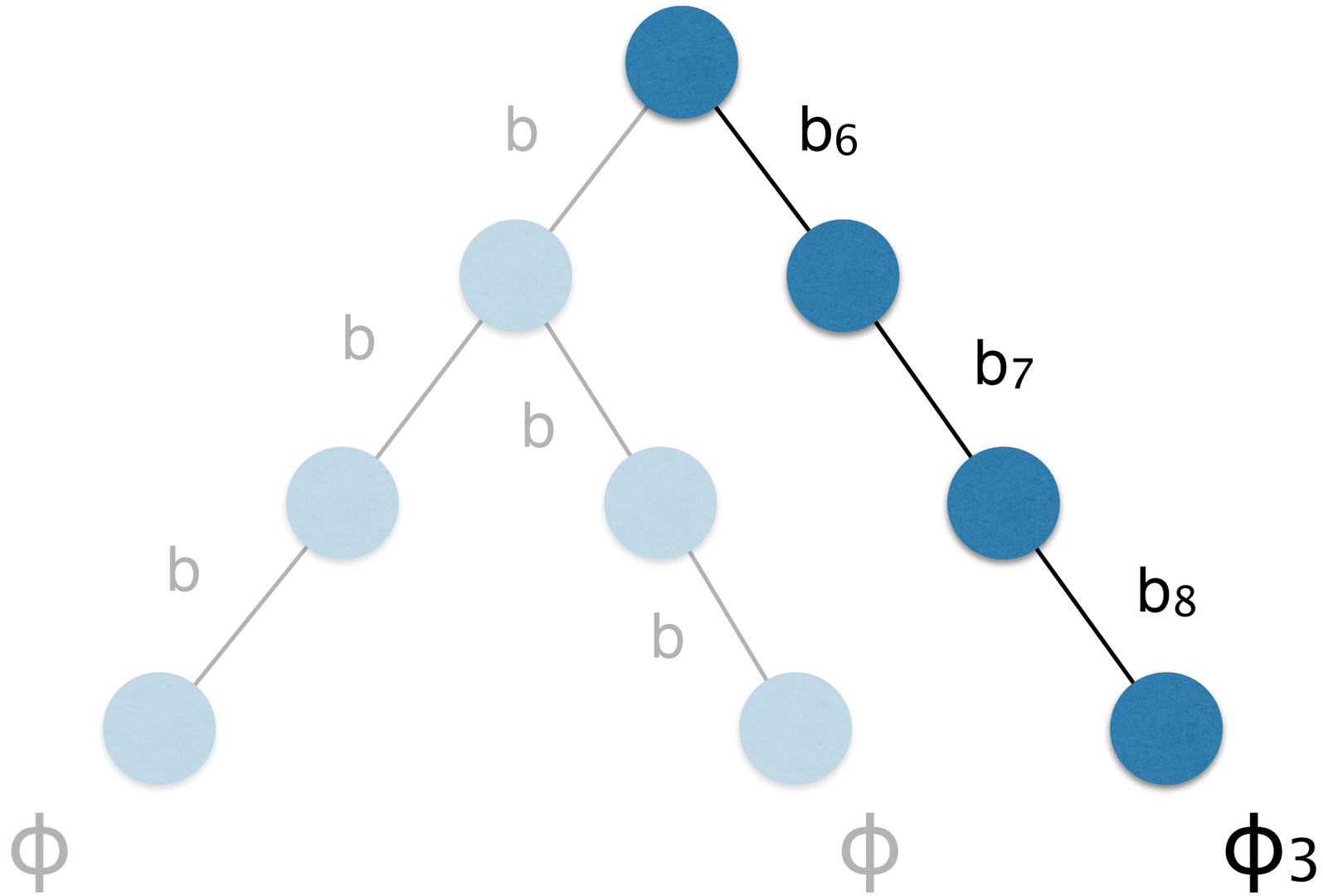
execution tree

Concolic Testing



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

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: $\quad \Phi_m \leftarrow \text{RunProgram}(P$
5: $\quad T \leftarrow T \cdot \Phi_m$
6: $\quad$ **repeat**
7: $\quad\quad (\Phi, \phi_i) \leftarrow \text{Choose}(T)$ $\quad (\Phi = \phi_1 \wedge \cdots \wedge \phi_n)$
8: $\quad$ **until** $\text{SAT}(\bigwedge_{j<i} \phi_j \wedge \neg\phi_i)$
9: $\quad v \leftarrow \text{model}(\bigwedge_{j<i} \phi_j \wedge \neg\phi_i)$
10: **end for**
11: **return** $|\text{Branches}(T)|$

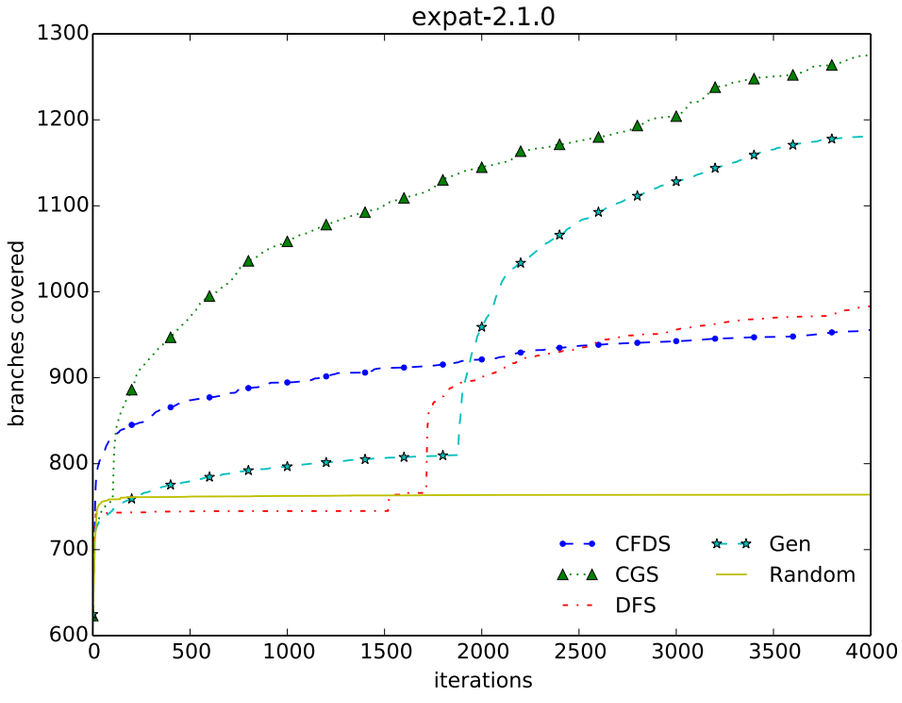
Search Heuristic

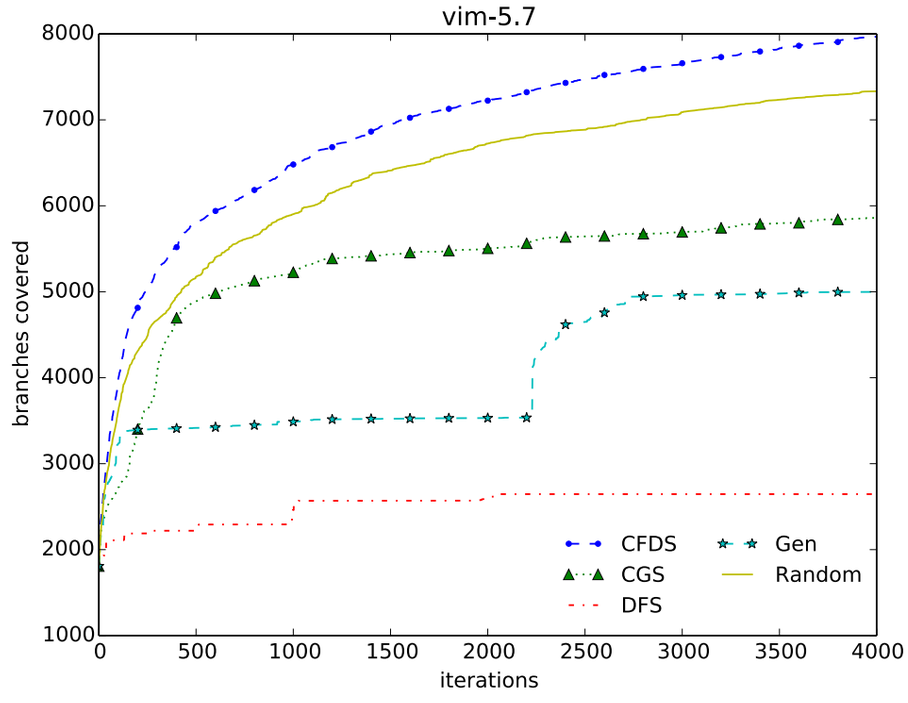
Search Heuristics

- Concolic testing relies on search heuristics to maximize code coverage in a limited time budget.
- Key but the most manual and ad-hoc component of concolic testing
- Numerous heuristics have been proposed:
  - DFS [PLDI'05], BFS, Random, CFDS [ASE'08], Generational [NDSS'08], CarFast[FSE'12], CGS [FSE'14], …

Limitations of Existing Search Heuristics

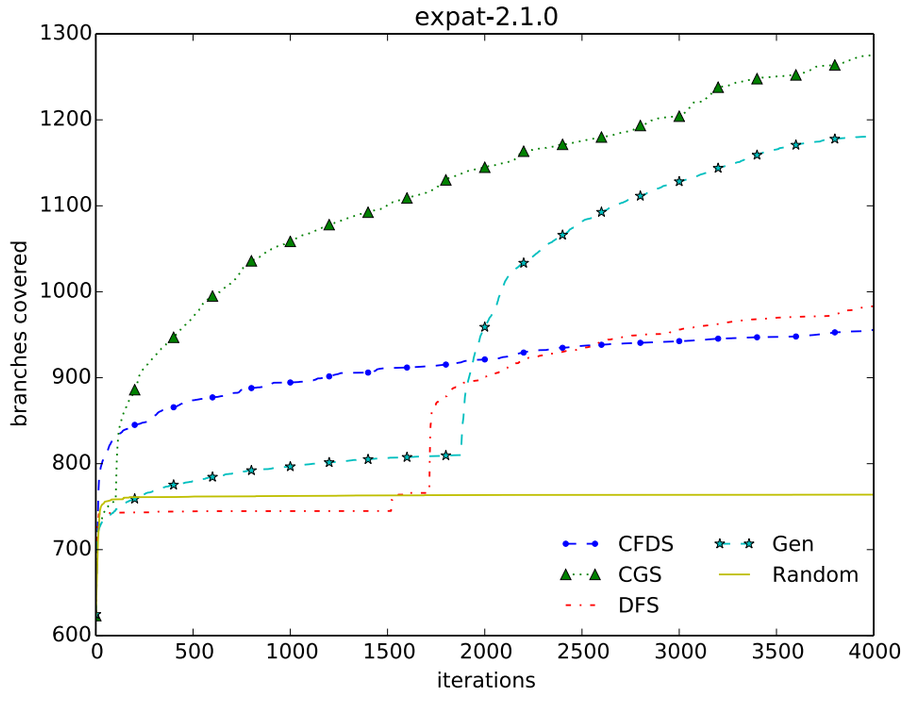
- No existing heuristics perform well in practice
- Developing a heuristic requires a huge amount of engineering effort and expertise.

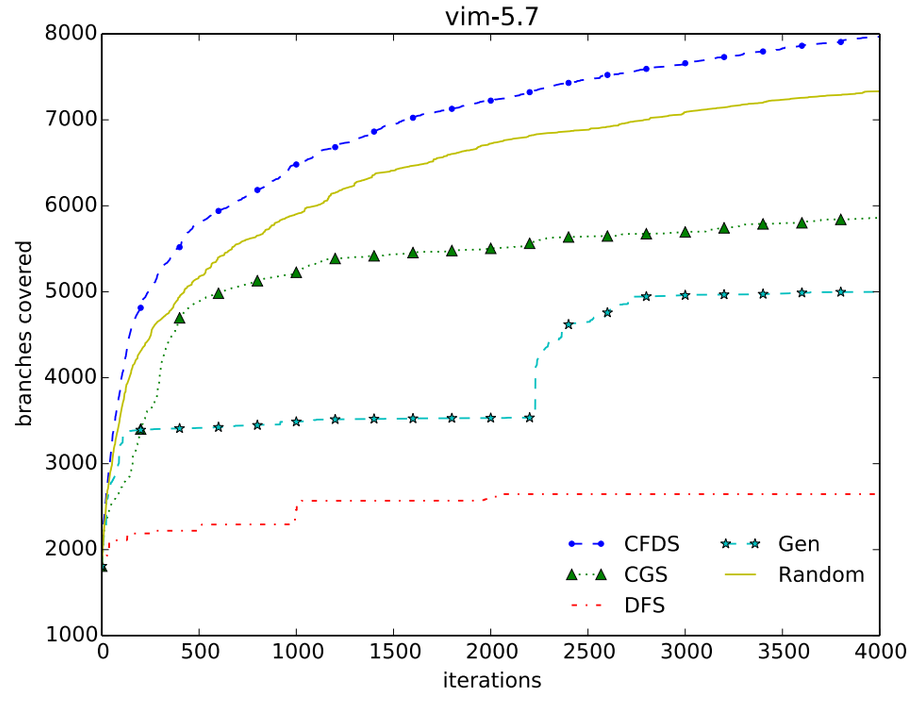




Limitations of Existing Search Heuristics

- No existing heuristics perform well in practice
- Developing a heuristic requires a huge amount of engineering effort and expertise.



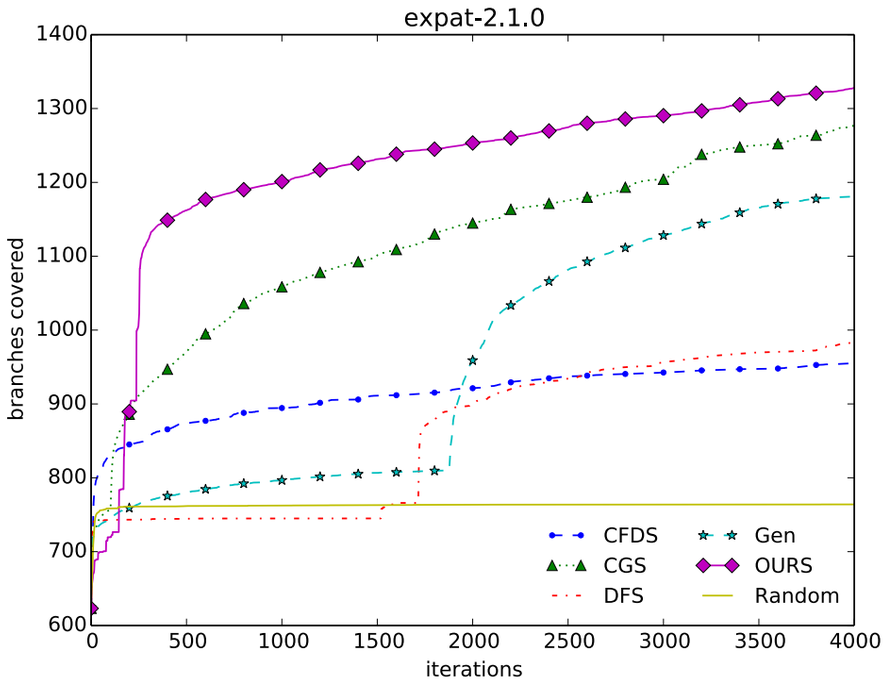


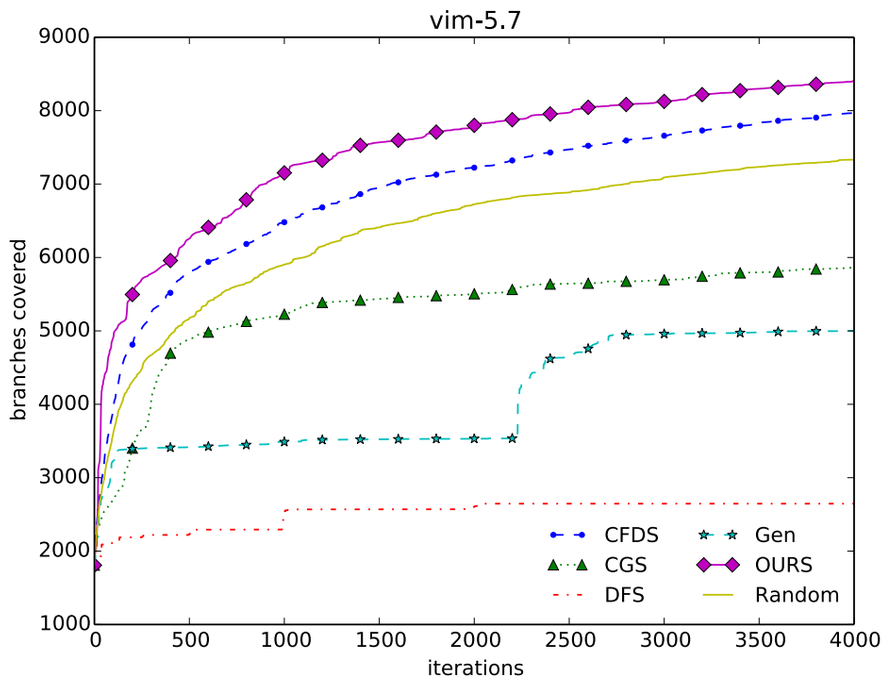
Our goal: automatically generating search heuristics

ICSE'18

Effectiveness of Our Method

- Considerable increase in branch coverage

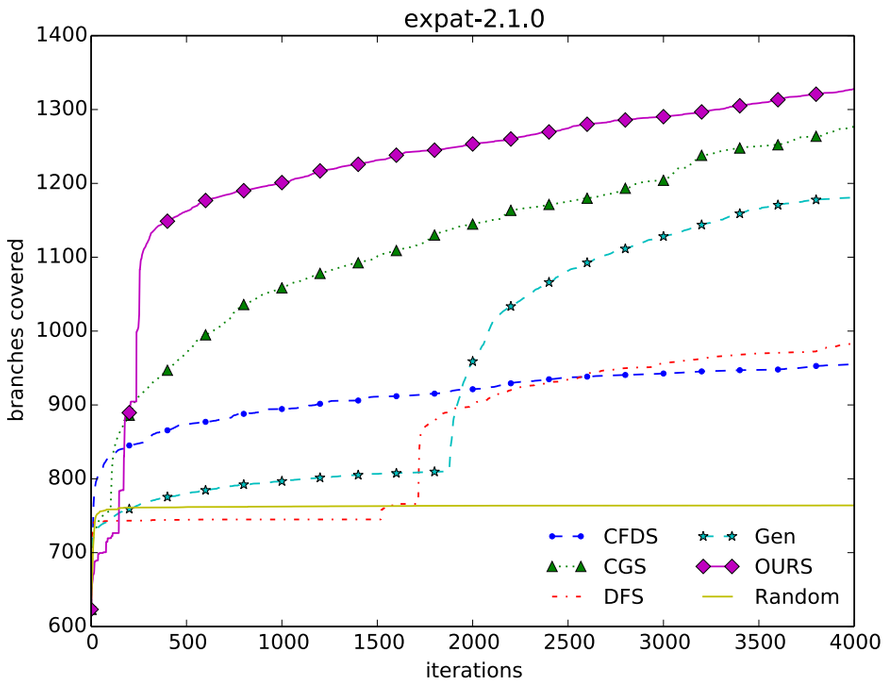


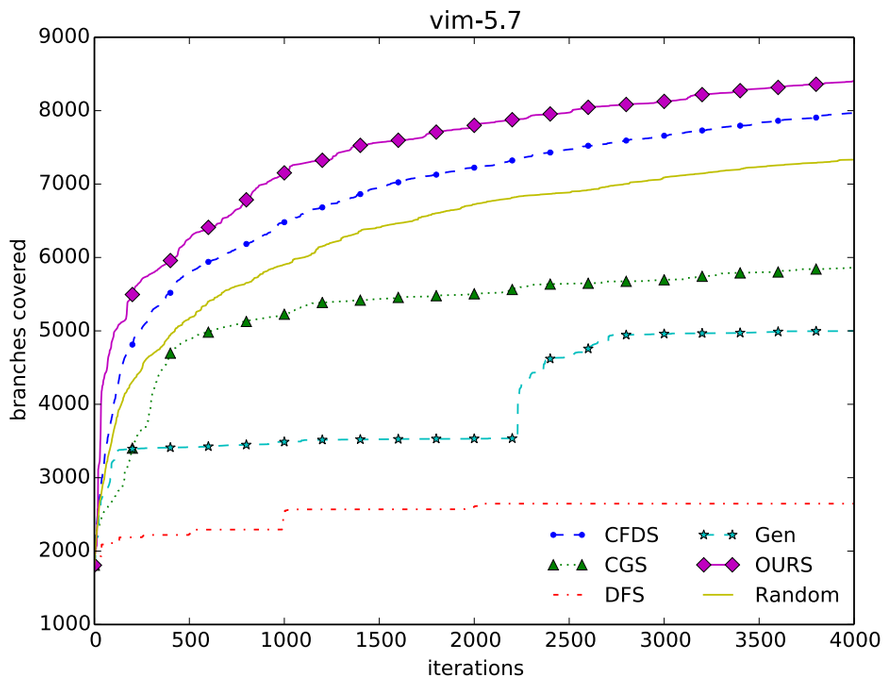


ICSE'18

Effectiveness of Our Method

- Considerable increase in branch coverage





- Dramatic increase in bug-finding

	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

Automatic Debugging (Automatic Program Repair)

MemFix: 메모리 관리 오류 자동 수정기

- 메모리 관리 오류: C/C++에서 빈번하게 발생

Memory Leak

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

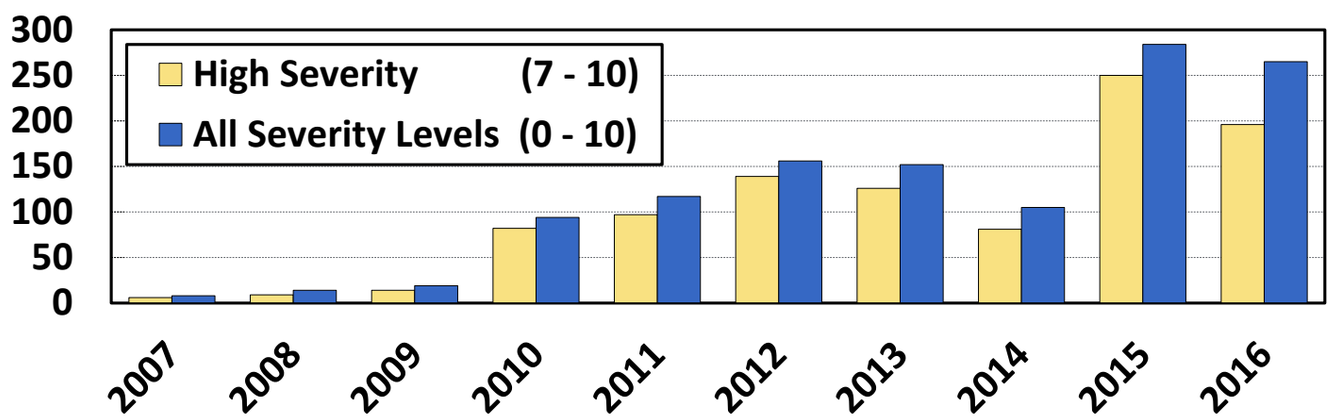
Use-After-Free

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

Double-Free

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

- 심각한 소프트웨어 보안취약점의 주요 원인



(Yan et al. ICSE 2018)

- 사전 탐지 및 정확한 수정이 매우 어려움

Example (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;
```

Example (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

Example (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;
```

Example (Linux Kernel)

USB: fix double frees in error code paths of ipaq driver

the error code paths can be enter with buffers to freed buffers.
Serial core would do a kfree() on memory already freed.

Signed-off-by: Oliver Neukum <oneukum@suse.de>
Signed-off-by: Greg Kroah-Hartman <gregkh@suse.de>

master v4.15-rc1 ... v2.6.24-rc1

Oliver Neukum committed with **gregkh** on 18 Sep 2007 1 par

```
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;
```

Example (Linux Kernel)

USB: fix double kfree in ipaq in error case

in the error case the ipaq driver leaves a dangling pointer to already freed memory that will be freed again.

Signed-off-by: Oliver Neukum <oneukum@suse.de>
Signed-off-by: Greg Kroah-Hartman <gregkh@suse.de>

master v4.15-rc1 ··· v2.6.27-rc1

Oliver Neukum committed with **gregkh** on 30 Jun 2008 1 parent 35

```
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;
```

Example (Linux Kernel)

USB: fix double kfree in ipaq in error case

in the error case the ipaq driver leaves a dangling pointer to already freed memory that will be freed again.

Signed-off-by: Oliver Neukum <oneukum@suse.de>
Signed-off-by: Greg Kroah-Hartman <gregkh@suse.de>

master v4.15-rc1 ··· v2.6.27-rc1

Oliver Neukum committed with **gregkh** on 30 Jun 2008 1 parent 35

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;
```

Example (Linux Kernel)

fix for a memory leak in an error case introduced by fix for double free

The fix NULLed a pointer without freeing it.

Signed-off-by: Oliver Neukum <oneukum@suse.de>
Reported-by: Juha Motorsportcom <juha_motorsportcom@luukku.com>
Signed-off-by: Linus Torvalds <torvalds@linux-foundation.org>

master v4.15-rc1 ... v2.6.27-rc1

Oliver Neukum committed with **torvalds** on 27 Jul 2008 1 parent 9ee08c2

```
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;
```

```
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;
```

MemFix

```
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;
```

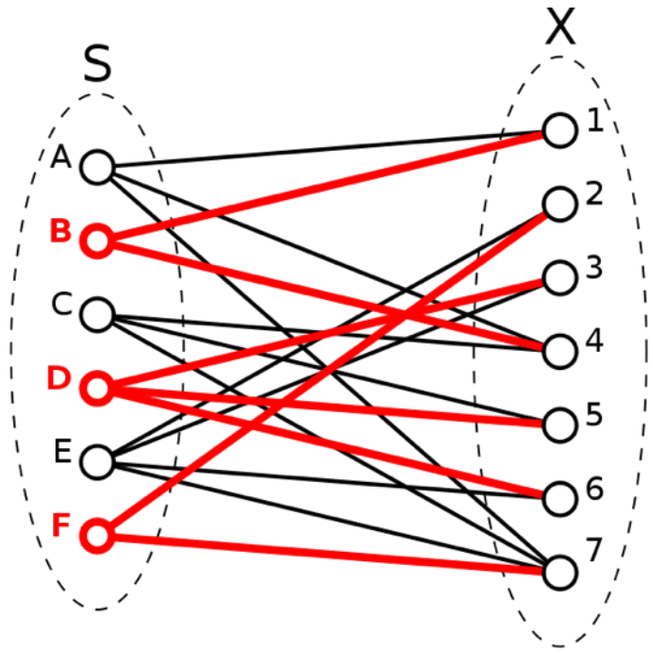
FSE'18

MemFix 알고리즘

approx. by static analysis

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

  goto err;
}
```

Fixing memory errors
(undecidable)



Exact cover problem
(NP-complete)

SAT encoding

$$\varphi_1 = \bigwedge_{j=1}^{m} \bigvee_{i=1}^{n} T_{ij} \wedge S_i$$

$$\varphi_2 = \bigwedge_{j=1}^{m} \bigwedge_{i_1=1}^{n} \bigwedge_{i_2=1}^{n} \left((i_1 \neq i_2)\right.$$

Boolean satisfiability
(NP-complete)

MemFix 알고리즘

- Soundness and safety proved formally
  - Soundness: the patch gurantees to fix the error
  - Safety: no new errors are introduced

Automatic Feedback Generation for Programming Assignments

- In my programming language course,
  - students hardly receive personalized feedback, and
  - instructor's solutions are not very helpful.

모범 답안

```
let rec map f (l,var) =
    match l with
    | [] -> []
    | hd::tl -> (f (hd,var))::(map f (tl,var))
...
| Sum lst -> Sum (map diff (lst,var))
...
```

오답 코드

```
...
| Sum plus ->
 (match plus with
   []    -> Const 0
   | [hd]   -> diff( hd, var)
   | hd::tl -> Sum [diff(hd, var); diff(Times tl, var)]
 ) ...
```

Sum

학생 제출 답안

```
type aexp =
   |CONST of int
   | VAR of string
   | POWER of string * int
   | TIMES of aexp list
   | SUM of aexp list

type env = (string * int * int) list

let diff : aexp * string -> aexp
= fun (aexp, x) ->

   let rec deployEnv : env -> int -> aexp list
   = fun env flag ->
   match env with
   | hd::tl ->
   (
    match hd with
    |(x, c, p) ->
       if (flag = 0 && c = 0) then deployEnv tl flag
       else if (x = "const" && flag = 1 && c = 1) then deployEnv tl flag
       else if (p = 0) then (CONST c)::(deployEnv tl flag)
       else if (c = 1 && p = 1) then (VAR x)::(deployEnv tl flag)
       else if (p = 1) then TIMES[CONST c; VAR x]::(deployEnv tl flag)
       else if (c = 1) then POWER(x, p)::(deployEnv tl flag)
       else TIMES [CONST c; POWER(x, p)]::(deployEnv tl flag)
)
| [] -> []
in

let rec updateEnv : (string * int * int) -> env -> int -> env
= fun elem env flag ->
match env with
| (hd::tl) ->
   (
   match hd with
   | (x, c, p) ->
      (
      match elem with
      |(x2, c2, p2) ->
         if (flag = 0) then
            if (x = x2 && p = p2) then (x, (c + c2), p)::tl
            else hd::(updateEnv elem tl flag)
         else
            if (x = x2) then (x, (c*c2), (p + p2))::tl
            else hd::(updateEnv elem tl flag)
      )
   )
| [] -> elem::[]
in

let rec doDiff : aexp * string -> aexp
= fun (aexp, x) ->
match aexp with
| CONST _ -> CONST 0
| VAR v ->
   if (x = v) then CONST 1
   else CONST 0
| POWER (v, p) ->
   if (p = 0) then CONST 0
   else if (x = v) then TIMES ((CONST p)::POWER (v, p-1)::[])
   else CONST 0
| TIMES lst ->
   (
      match lst with
         (
            match (hd, diff_hd, tl, diff_tl) with
            | (CONST p, CONST s, [CONST r], CONST q) -> CONST (p*q + r*s)
            | (CONST p, _, _, CONST q) ->
               if (diff_hd = CONST 0 || tl = [CONST 0]) then CONST (p*q)
               else SUM [CONST(p*q); TIMES(diff_hd::tl)]
            | (_, CONST s, [CONST r], _) ->
               if (hd = CONST 0 || diff_tl = CONST 0) then CONST (r*s)
               else  SUM [TIMES [hd; diff_tl]; CONST(r*s)]
            | _ ->
               if (hd = CONST 0 || diff_tl = CONST 0) then TIMES(diff_hd::tl)
               else if (tl = [CONST 0] || diff_hd = CONST 0) then TIMES [hd; diff_tl]
               else SUM [TIMES [hd; diff_tl]; TIMES (diff_hd::tl)]
         )
      | [] -> CONST 0
   )
| SUM lst -> SUM(List.map (fun aexp -> doDiff(aexp, x)) lst)
in

let rec simplify : aexp -> env -> int -> aexp list
= fun aexp env flag ->
match aexp with
| SUM lst ->
   (
      match lst with
      | (CONST c)::tl -> simplify (SUM tl) (updateEnv ("const", c, 0) env 0) 0
      | (VAR x)::tl -> simplify (SUM tl) (updateEnv (x, 1, 1) env 0) 0
      | (POWER (x, p))::tl -> simplify (SUM tl) (updateEnv (x, 1, p) env 0) 0
      | (SUM lst)::tl -> simplify (SUM (List.append lst tl)) env 0
      | (TIMES lst)::tl ->
         (
            let l = simplify (TIMES lst) [] 1 in
            match l with
            | h::t ->
               if (t = []) then List.append l (simplify (SUM tl) env 0)
               else List.append (TIMES l::[]) (simplify (SUM tl) env 0)
            | [] -> []
         )
      | [] -> deployEnv env 0
   )
| TIMES lst ->
   (
      match lst with
      | (CONST c)::tl -> simplify (TIMES tl) (updateEnv ("const", c, 0) env 1) 1
      | (VAR x)::tl -> simplify (TIMES tl) (updateEnv (x, 1, 1) env 1) 1
      | (POWER (x, p))::tl -> simplify (TIMES tl) (updateEnv (x, 1, p) env 1) 1
      | (SUM lst)::tl ->
         (
            let l = simplify (SUM lst) [] 0 in
            match l with
            | h::t ->
               if (t = []) then List.append l (simplify (TIMES tl) env 1)
                else List.append (SUM l::[]) (simplify (TIMES tl) env 1)
             | [] -> []  (* Feedback : Replace [] by ((Sum lst) :: tl) *)
         )
      | (TIMES lst)::tl -> simplify (TIMES (List.append lst tl)) env 1
      | [] -> deployEnv env 1
   )
in

let result = doDiff (aexp, x) in
match result with
| SUM _ -> SUM (simplify result [] 0)
| TIMES _ -> TIMES (simplify result [] 1)
| _ -> result
```

모범답안

```
let rec diff : aexp * string -> aexp
= fun (e, x) ->
   match e with
   | Const n -> Const 0
   | Var a -> if (a <> x) then Const 0 else Const 1
   | Power (a, n) -> if (a <> x) then Const 0 else Times [Const n; Power (a, n-1)]
   | Times l ->
      begin
      match l with
      | [] -> Const 0
      | hd::tl -> Sum [Times ((diff (hd, x))::tl); Times [hd; diff (Times tl, x)]]
      end
   | Sum l -> Sum (List.map (fun e -> diff (e,x)) l)
```

학생 제출 답안

```
type aexp =
   |CONST of int
   | VAR of string
   | POWER of string * int
   | TIMES of aexp list
   | SUM of aexp list

type env = (string * int * int) list

let diff : aexp * string -> aexp
= fun (aexp, x) ->

   let rec deployEnv : env -> int -> aexp list
   = fun env flag ->
   match env with
   | hd::tl ->
   (
    match hd with
    |(x, c, p) ->
       if (flag = 0 && c = 0) then deployEnv tl flag
       else if (x = "const" && flag = 1 && c = 1) then deployEnv tl flag
       else if (p = 0) then (CONST c)::(deployEnv tl flag)
       else if (c = 1 && p = 1) then (VAR x)::(deployEnv tl flag)
       else if (p = 1) then TIMES[CONST c; VAR x]::(deployEnv tl flag)
       else if (c = 1) then POWER(x, p)::(deployEnv tl flag)
       else TIMES [CONST c; POWER(x, p)]::(deployEnv tl flag)
)
| [] -> []
in

let rec updateEnv : (string * int * int) -> env -> int -> env
= fun elem env flag ->
match env with
| (hd::tl) ->
   (
   match hd with
   | (x, c, p) ->
      (
      match elem with
      |(x2, c2, p2) ->
         if (flag = 0) then
            if (x = x2 && p = p2) then (x, (c + c2), p)::tl
            else hd::(updateEnv elem tl flag)
         else
            if (x = x2) then (x, (c*c2), (p + p2))::tl
            else hd::(updateEnv elem tl flag)
      )
   )
| [] -> elem::[]
in

let rec doDiff : aexp * string -> aexp
= fun (aexp, x) ->
match aexp with
| CONST _ -> CONST 0
| VAR v ->
   if (x = v) then CONST 1
   else CONST 0
| POWER (v, p) ->
   if (p = 0) then CONST 0
   else if (x = v) then TIMES ((CONST p)::POWER (v, p-1)::[])
   else CONST 0
| TIMES lst ->
   (
      match lst with
         (
            match (hd, diff_hd, tl, diff_tl) with
            | (CONST p, CONST s, [CONST r], CONST q) -> CONST (p*q + r*s)
            | (CONST p, _, _, CONST q) ->
               if (diff_hd = CONST 0 || tl = [CONST 0]) then CONST (p*q)
               else SUM [CONST(p*q); TIMES(diff_hd::tl)]
            | (_, CONST s, [CONST r], _) ->
               if (hd = CONST 0 || diff_tl = CONST 0) then CONST (r*s)
               else  SUM [TIMES [hd; diff_tl]; CONST(r*s)]
            | _ ->
               if (hd = CONST 0 || diff_tl = CONST 0) then TIMES(diff_hd::tl)
               else if (tl = [CONST 0] || diff_hd = CONST 0) then TIMES [hd; diff_tl]
               else SUM [TIMES [hd; diff_tl]; TIMES (diff_hd::tl)]
         )
      | [] -> CONST 0
   )
| SUM lst -> SUM(List.map (fun aexp -> doDiff(aexp, x)) lst)
in

let rec simplify : aexp -> env -> int -> aexp list
= fun aexp env flag ->
match aexp with
| SUM lst ->
   (
      match lst with
      | (CONST c)::tl -> simplify (SUM tl) (updateEnv ("const", c, 0) env 0) 0
      | (VAR x)::tl -> simplify (SUM tl) (updateEnv (x, 1, 1) env 0) 0
      | (POWER (x, p))::tl -> simplify (SUM tl) (updateEnv (x, 1, p) env 0) 0
      | (SUM lst)::tl -> simplify (SUM (List.append lst tl)) env 0
      | (TIMES lst)::tl ->
         (
            let l = simplify (TIMES lst) [] 1 in
            match l with
            | h::t ->
               if (t = []) then List.append l (simplify (SUM tl) env 0)
               else List.append (TIMES l::[]) (simplify (SUM tl) env 0)
            | [] -> []
         )
      | [] -> deployEnv env 0
   )
| TIMES lst ->
   (
      match lst with
      | (CONST c)::tl -> simplify (TIMES tl) (updateEnv ("const", c, 0) env 1) 1
      | (VAR x)::tl -> simplify (TIMES tl) (updateEnv (x, 1, 1) env 1) 1
      | (POWER (x, p))::tl -> simplify (TIMES tl) (updateEnv (x, 1, p) env 1) 1
      | (SUM lst)::tl ->
         (
            let l = simplify (SUM lst) [] 0 in
            match l with
            | h::t ->
               if (t = []) then List.append l (simplify (TIMES tl) env 1)
               else List.append (SUM l::[]) (simplify (TIMES tl) env 1)
            | [] -> [] (* Feedback : Replace [] by ((Sum lst) :: tl) *)
         )
      | (TIMES lst)::tl -> simplify (TIMES (List.append lst tl)) env 1
      | [] -> deployEnv env 1
   )
in

let result = doDiff (aexp, x) in
match result with
| SUM _ -> SUM (simplify result [] 0)
| TIMES _ -> TIMES (simplify result [] 1)
| _ -> result
```

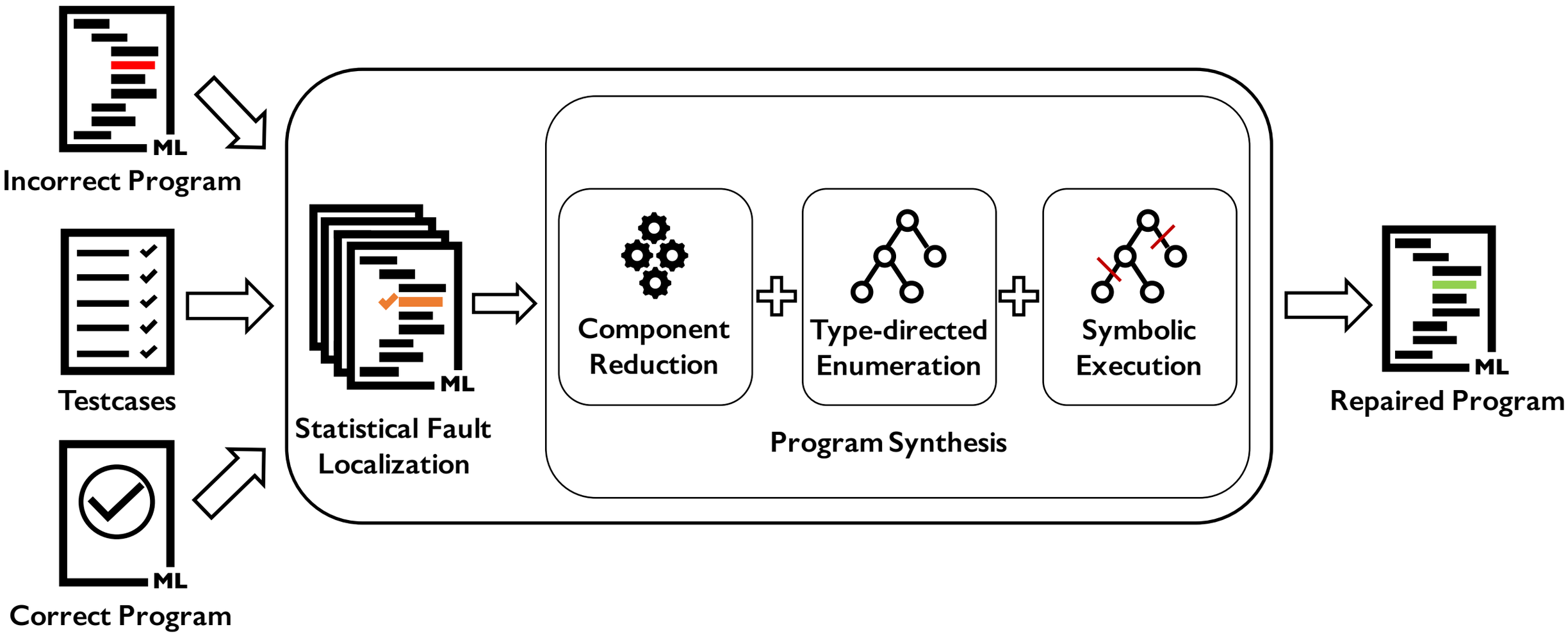
모범답안

```
let rec diff : aexp * string -> aexp
= fun (e, x) ->
   match e with
   | Const n -> Const 0
   | Var a -> if (a <> x) then Const 0 else Const 1
   | Power (a, n) -> if (a <> x) then Const 0 else Times [Const n; Power (a, n-1)]
   | Times l ->
      begin
      match l with
      | [] -> Const 0
      | hd::tl -> Sum [Times ((diff (hd, x))::tl); Times [hd; diff (Times tl, x)]]
      end
   | Sum l -> Sum (List.map (fun e -> diff (e,x)) l)
```

((Sum lst)::tl)

submitted to OOPSLA'18

The FixML System



Program Synthesis

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;
}
```

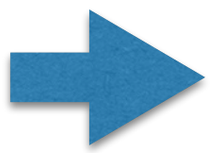
Performance

- Better than humans for introductory programming tasks

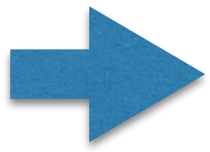
Domain	No	Description	Vars		Ints	Exs	Time (sec)		
			IVars	AVars			Base	Base+Opt	Ours
Integer	1	Given n, return $n!$.	2	0	2	4	0.0	0.0	0.0
	2	Given n, return $n!!$ (i.e., double factorial).	3	0	3	4	0.0	0.0	0.0
	3	Given n, return $\sum_{i=1}^{n} i$.	3	0	2	4	0.1	0.0	0.0
	4	Given n, return $\sum_{i=1}^{n} i^2$.	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), TOPLAS('14,'16,'17), ICSE('17,'18), FSE'18, S&P'17, IJCAI('17,'18), etc

http://prl.korea.ac.kr