AI-based Software Analysis and Testing

Hakjoo Oh

Korea University

9 July 2019 @Suresoft

Software Analysis Research@KU

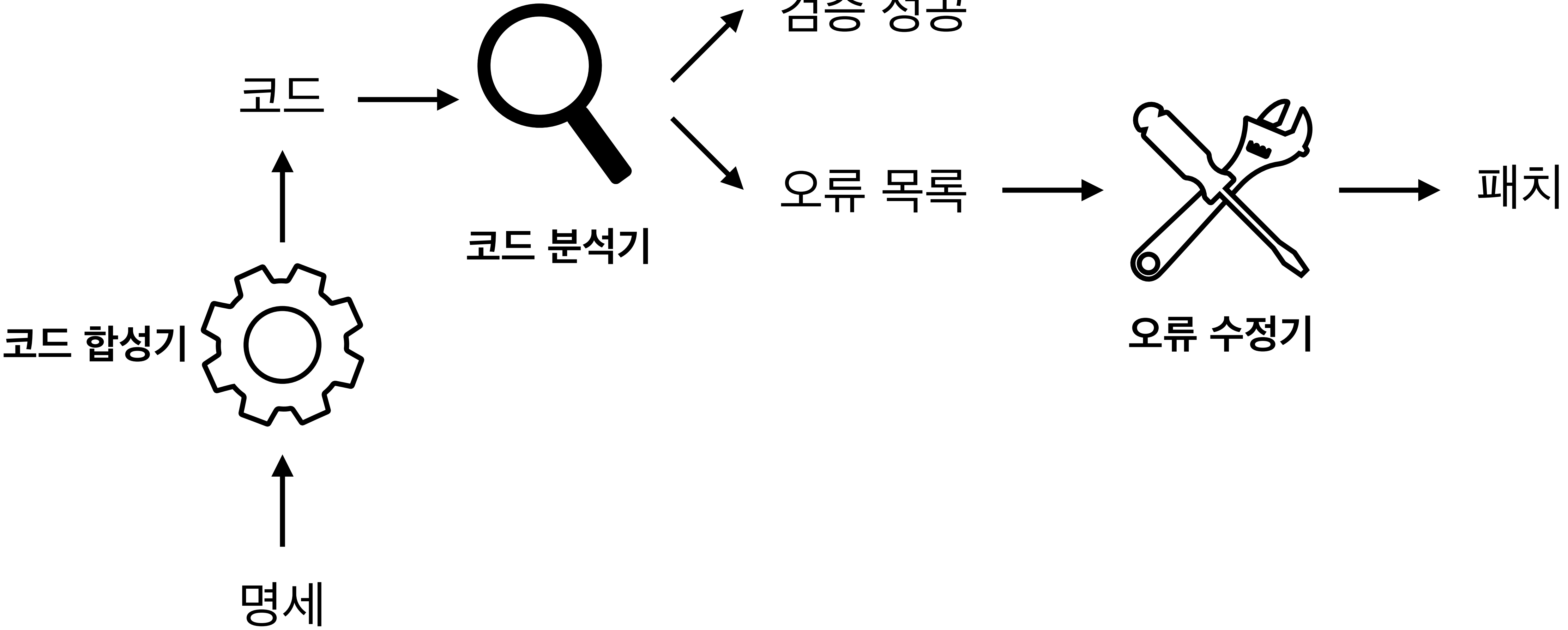
- Research areas: programming languages, software engineering, software security
  - program analysis and testing
  - program synthesis and repair

- Publication: top-venues in PL, SE, Security, and AI:
  - PLDI('12,'14), OOPSLA('15,'17a,'17b,'18a,'18b,'19), 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

Research Direction

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



명세

코드 합성기

코드

코드 분석기

검증 성공

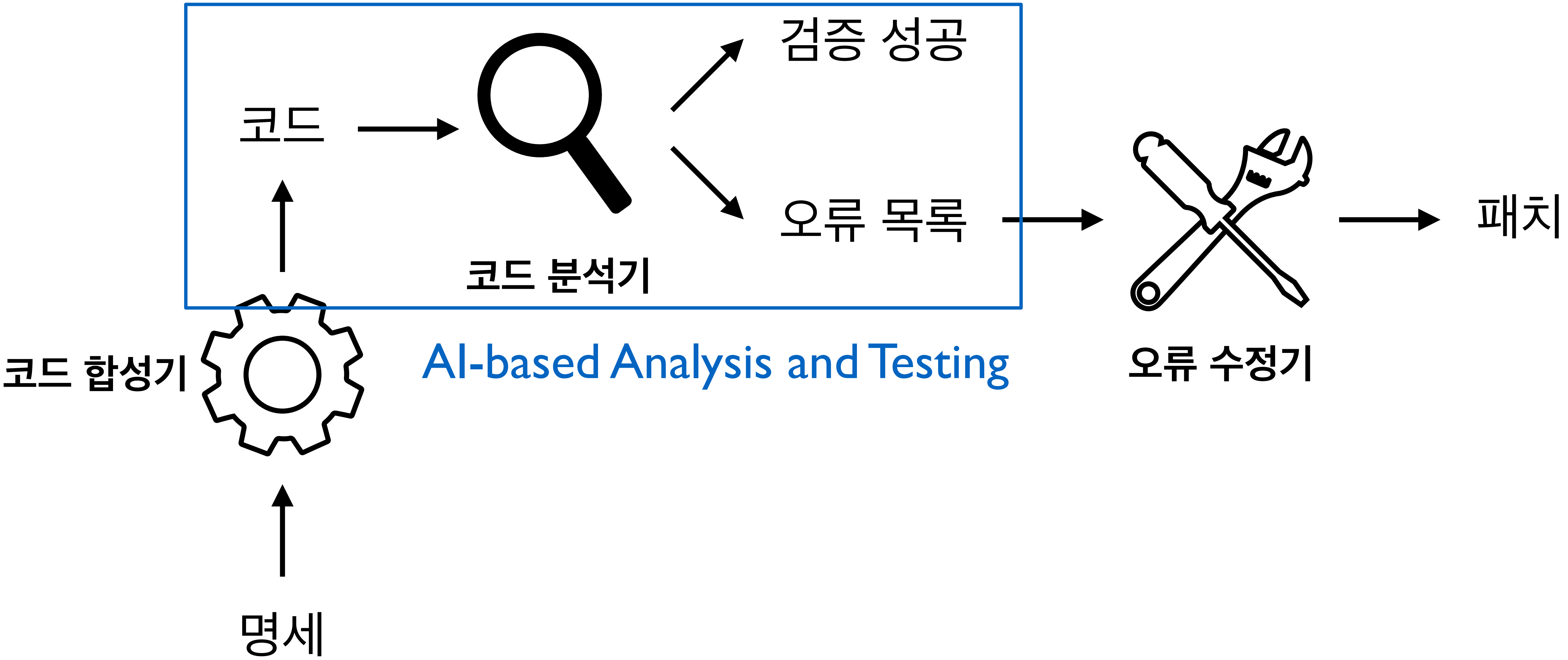
오류 목록

오류 수정기

패치

Research Direction

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



코드

검증 성공

오류 목록

코드 분석기

코드 합성기

AI-based Analysis and Testing

오류 수정기

패치

명세

Challenge in Program Analysis

Sparrow The Early Bird WALA T. J. WATSON LIBRARIES FOR ANALYSIS Infer Astrée

DOOP TAJS SAFE KLEE

- Practical program analysis tools rely on a variety of heuristics to optimize their performance
  - E.g., context/flow-sensitivity, variable clustering, unsoundness, path selection/pruning, state merging, etc
- Manuallly designing a heuristic does not pay-off
  - Nontrivial and laborious, but suboptimal and unstable

Vision

Automatically Generating Analysis Heuristics from Data

- Use data to make heuristic decisions in program analysis

AI, machine learning,
data mining, etc

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

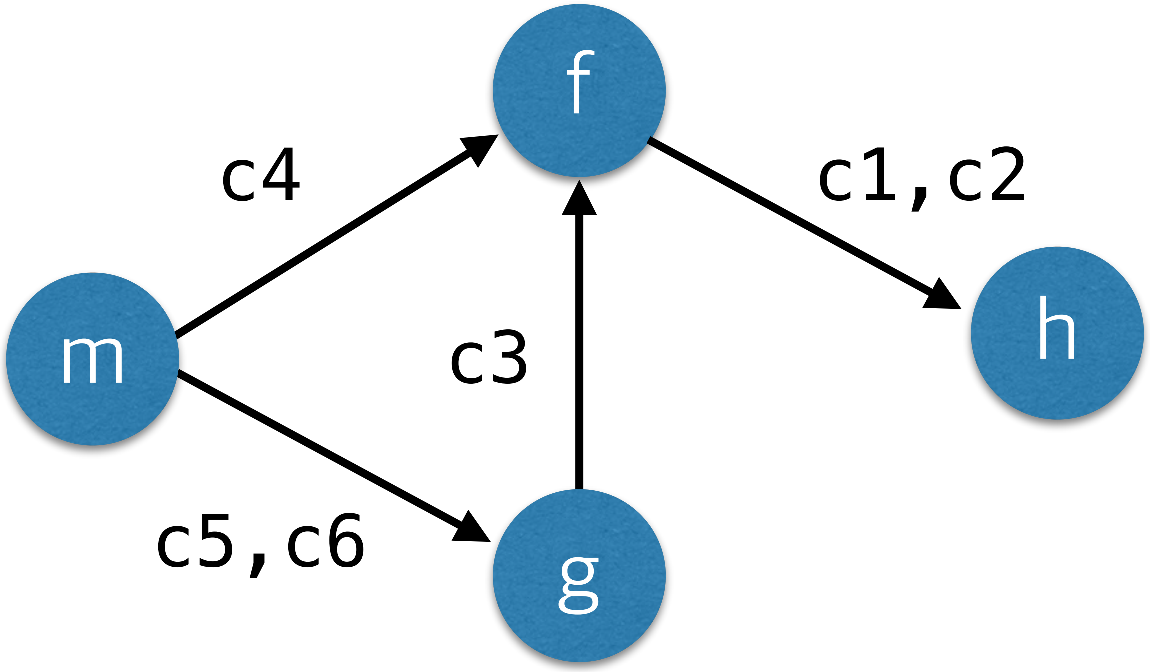
```
    int h(n) {ret n;}

    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();
    }
```

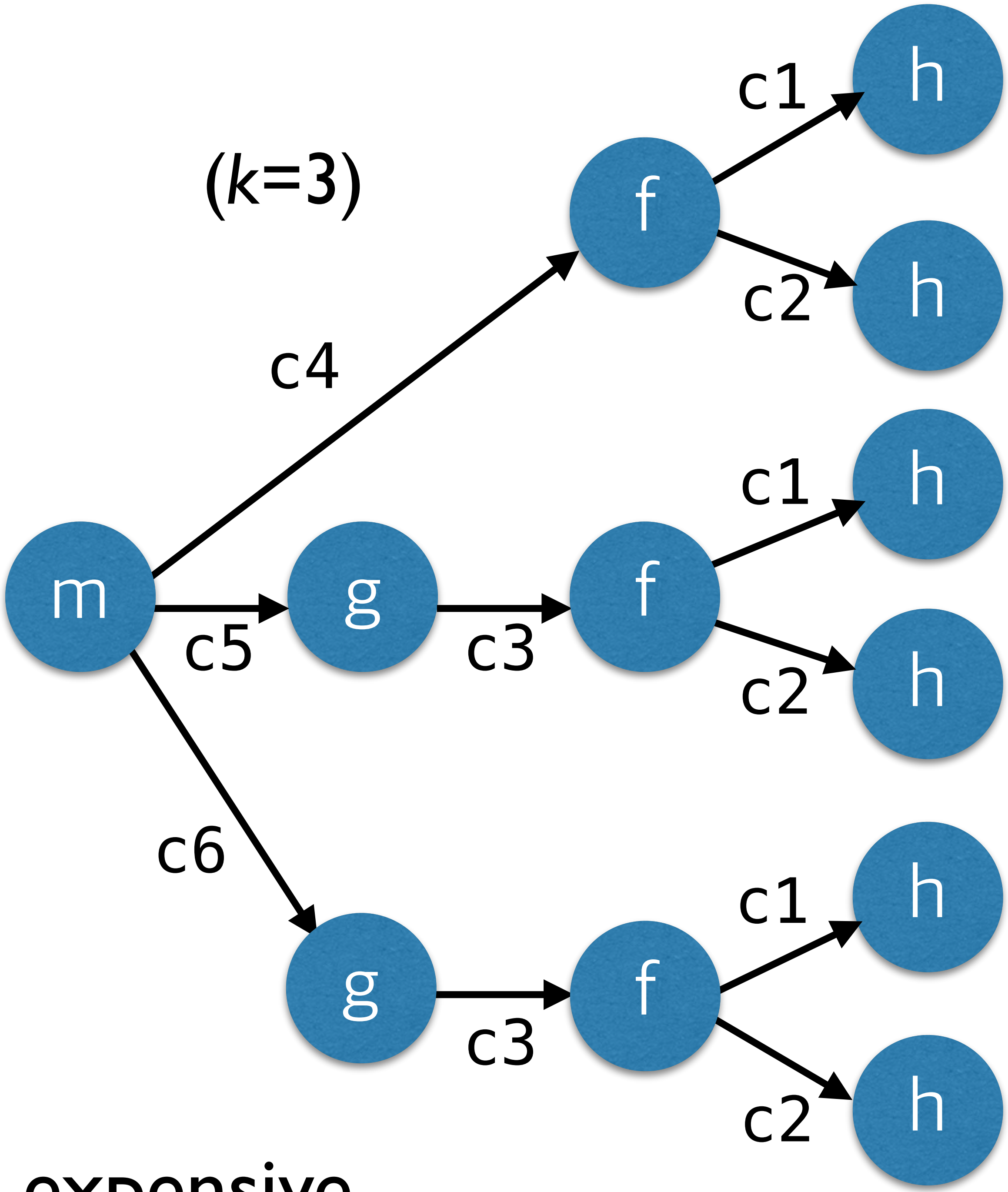


cheap but imprecise

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();
    }
```



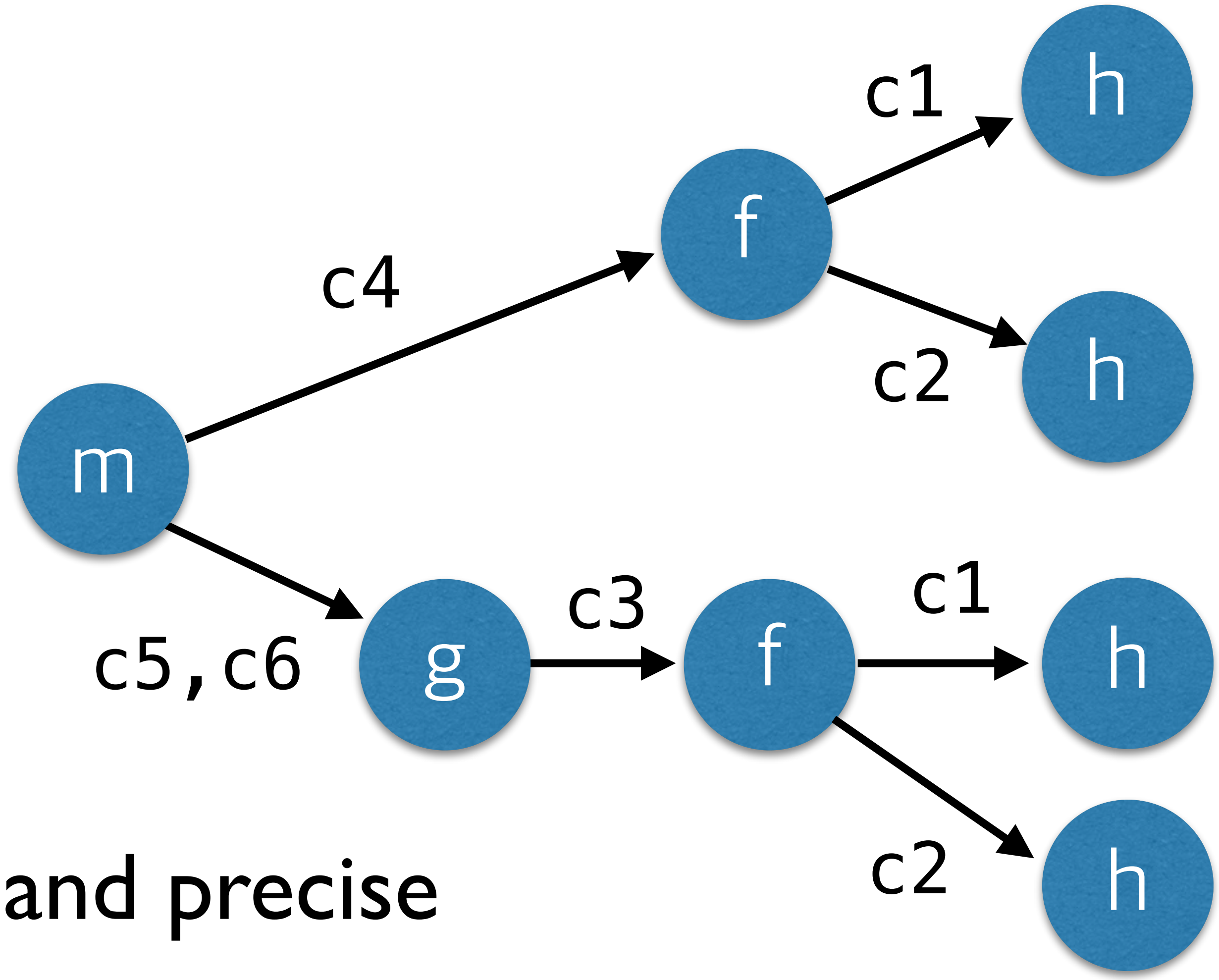
precise but expensive

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

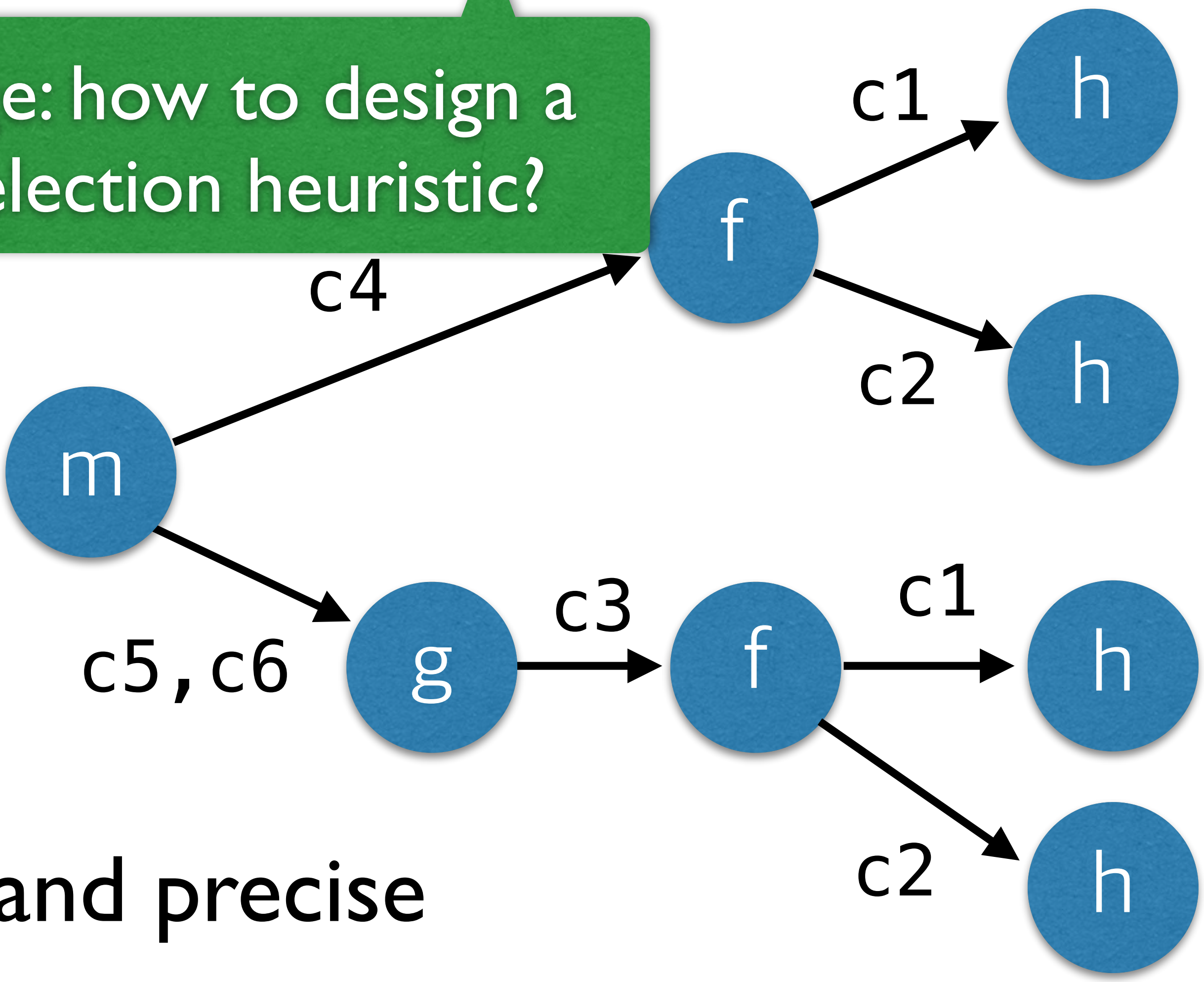
Selective Context-Sensitivity

- 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}

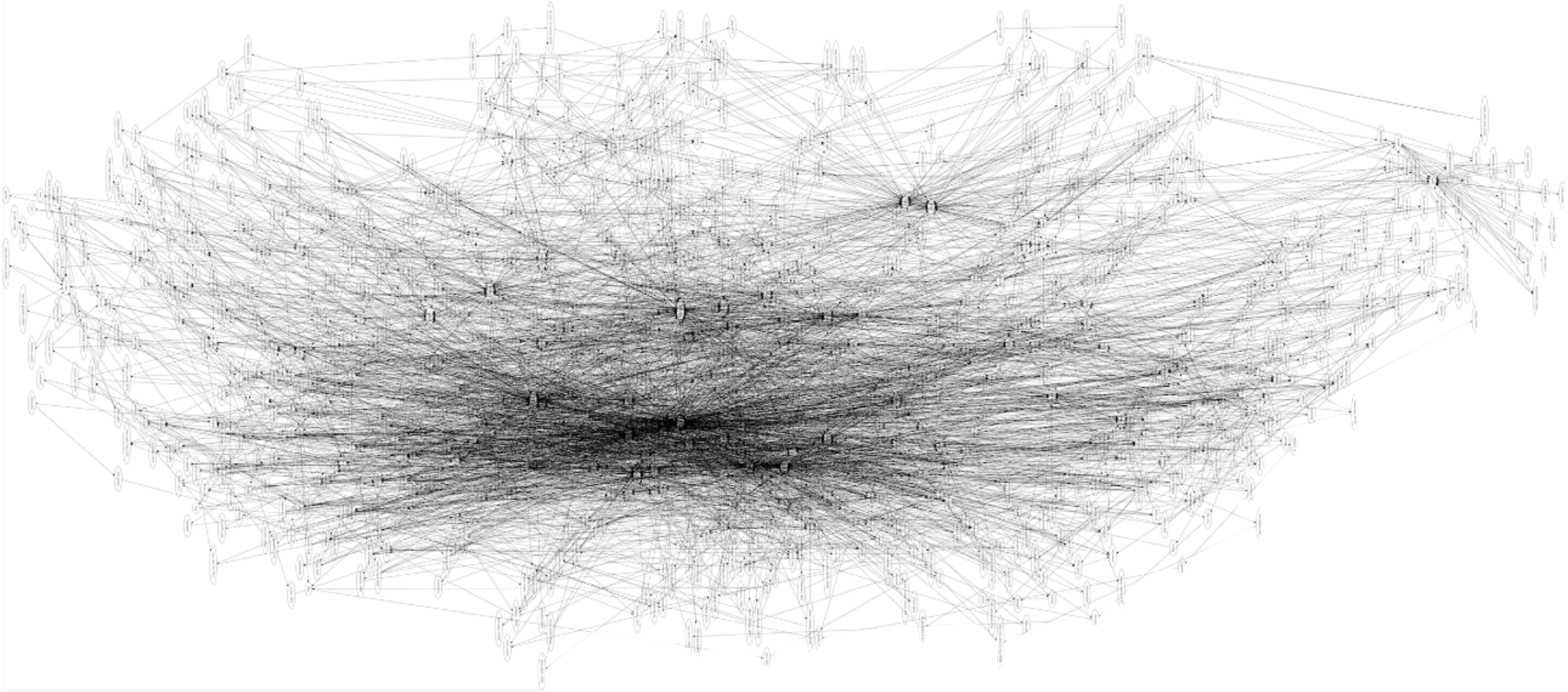
Challenge: how to design a good selection heuristic?



cheap and precise

Hard Search Problem

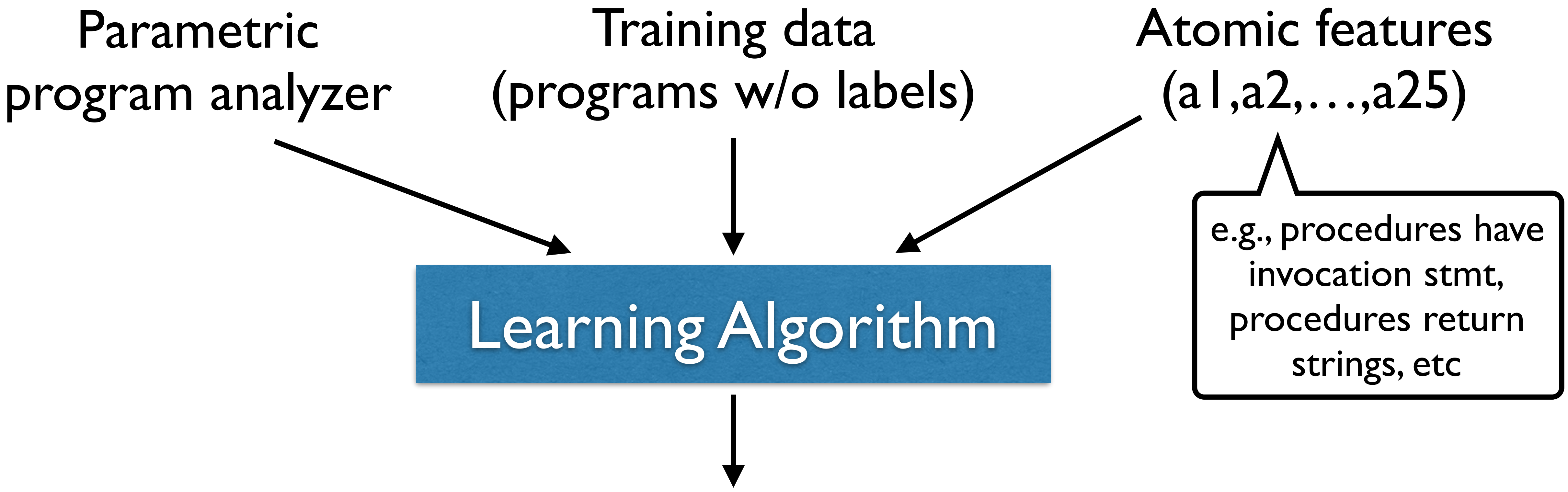
- Intractably large and sparse search space, if not infinite
  - e.g., S^k choices where $S = 2^{|Proc|}$ for *k*-context-sensitivity
- Real programs are complex to reason about
  - e.g., typical call-graph of real program:



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

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

f1: procedures to apply 1-context-sensitivity

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

cf) Atomic Features

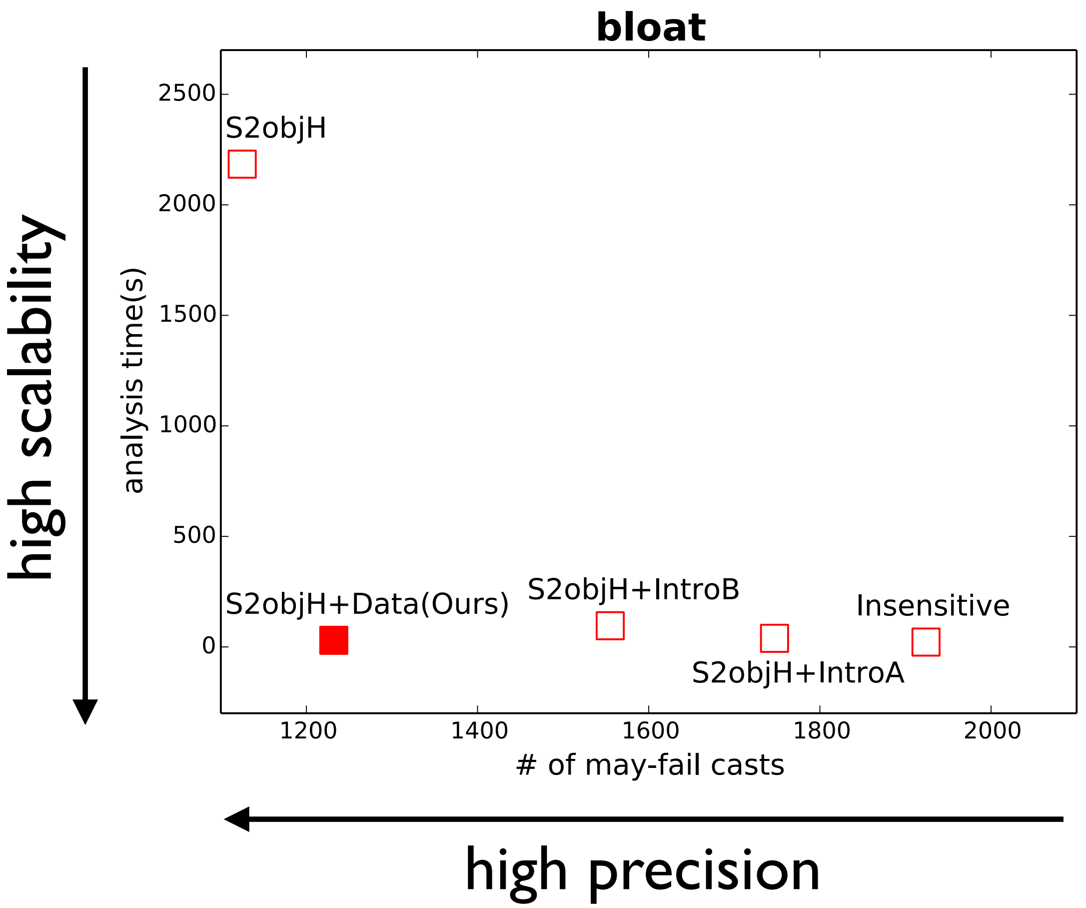
Signature features									
#1	“java”	#3	“sun”	#5	“void”	#7	“int”	#9	“String”
#2	“lang”	#4	“()”	#6	“security”	#8	“util”	#10	“init”

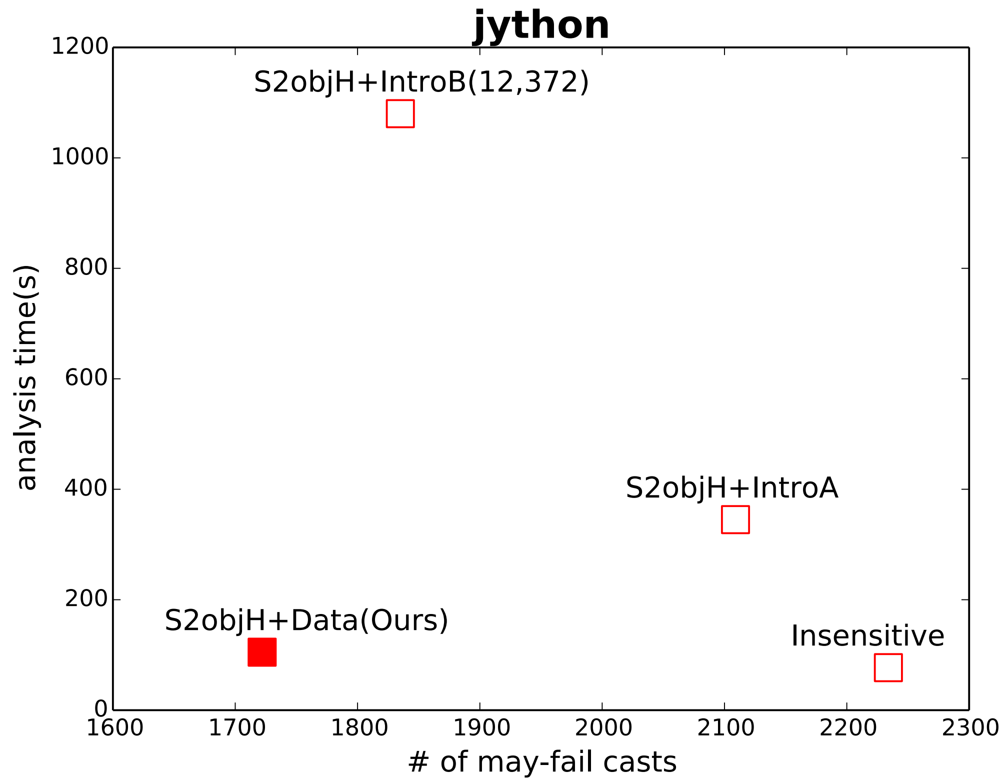
Statement features					
#11	AssignStmt	#16	BreakpointStmt	#21	LookupStmt
#12	IdentityStmt	#17	EnterMonitorStmt	#22	NopStmt
#13	InvokeStmt	#18	ExitMonitorStmt	#23	RetStmt
#14	ReturnStmt	#19	GotoStmt	#24	ReturnVoidStmt
#15	ThrowStmt	#20	IfStmt	#25	TableSwitchStmt

OOPSLA'17a

Effectiveness

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

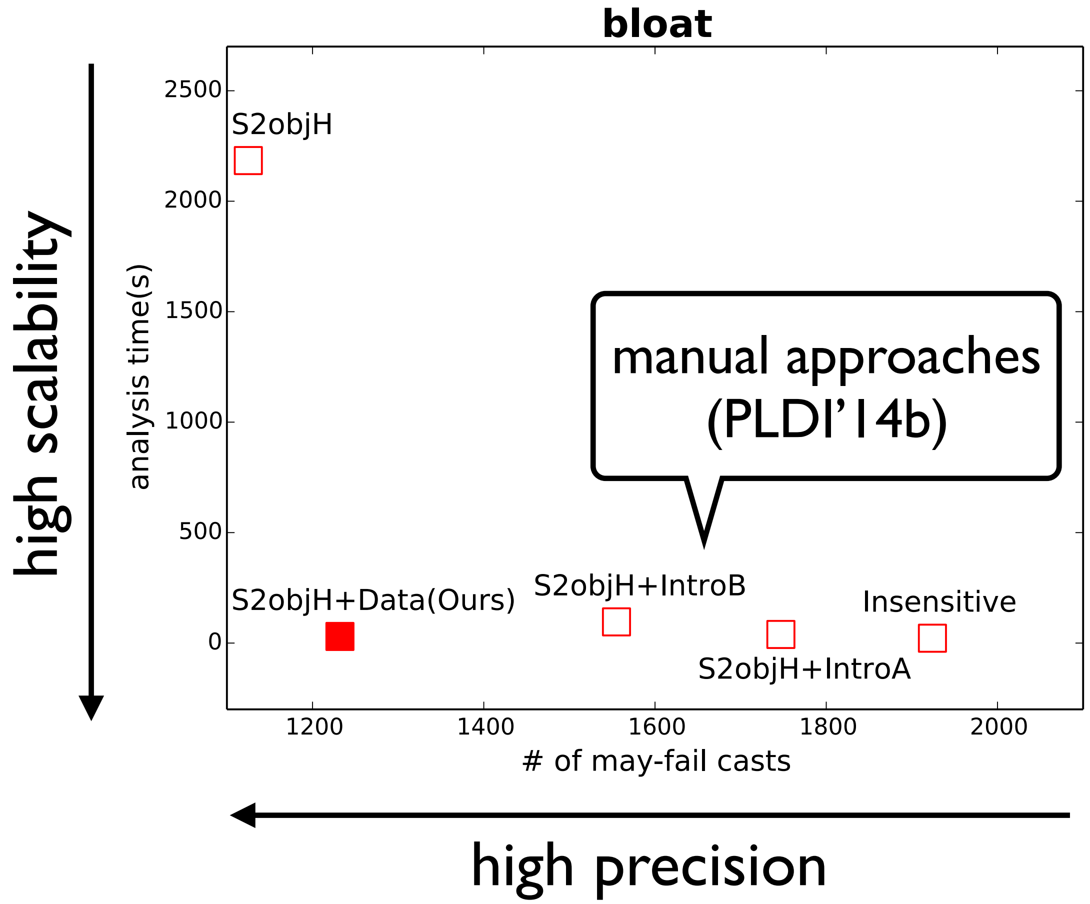




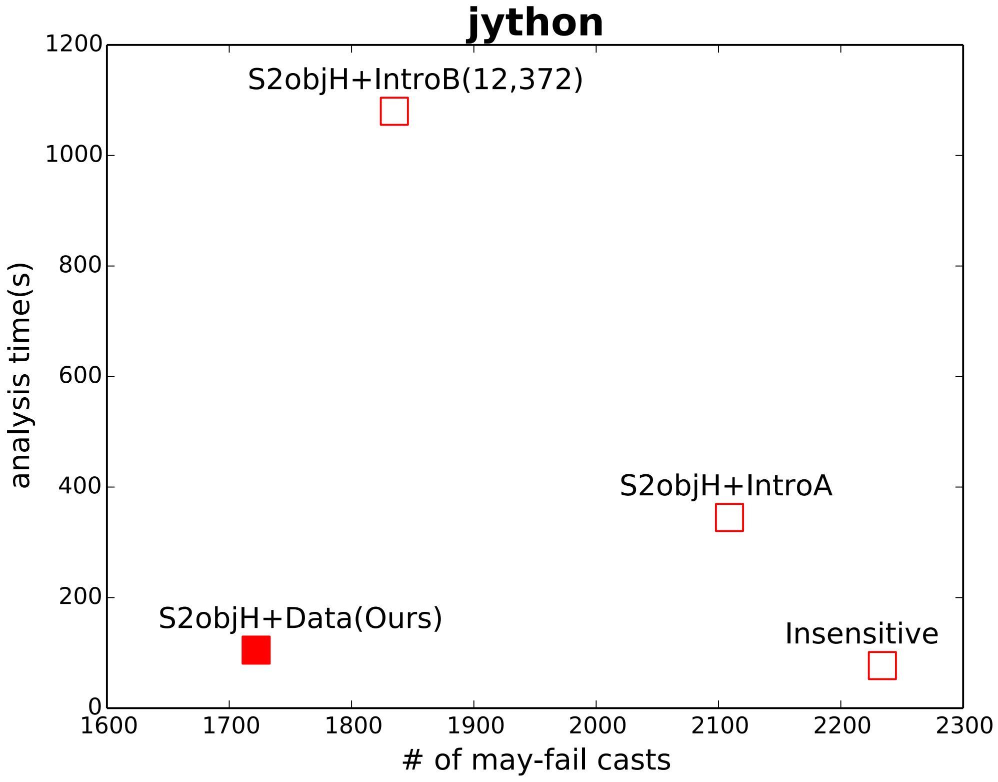
OOPSLA'17a

Effectiveness

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



manual approaches (PLDI'14b)



Concolic Testing (Dynamic Symbolic Execution)

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

KLEE S^{2}E Pex TRITON

- 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 ≤ x,y ≤ 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 ≤ x,y ≤ 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 ≤ x,y ≤ 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	$x=\alpha, y=\beta, z=2^*\beta$ $2^*\beta \neq \alpha$

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=22, y=7, z=14	x=α, y=β,z=2*β 2*β ≠ α

Solve: 2*β = α

Solution: α=2,β=1

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
$x=2, y=1, z=2$	$x=\alpha, y=\beta, z=2*\beta$ $2*\beta = \alpha$

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

x=2, y=1,
z=2

Symbolic State

$x=\alpha, y=\beta, z=2*\beta$

$2*\beta = \alpha \wedge$
$\alpha \leq \beta+10$

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=2, y=1, z=2	x=α, y=β,z=2*β $2*\beta = \alpha \land \alpha \leq \beta+10$

Solve: $2*\beta = \alpha \land \alpha > \beta+10$

Solution: α=30, β=15

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

x=30, y=15

Symbolic State

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

x=30, y=15,
z=30

Symbolic
State

$x=\alpha, y=\beta, z=2*\beta$

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

x=30, y=15,
z=30

Symbolic State

$x=\alpha, y=\beta, z=2*\beta$

$2*\beta = \alpha$

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

error-triggering input

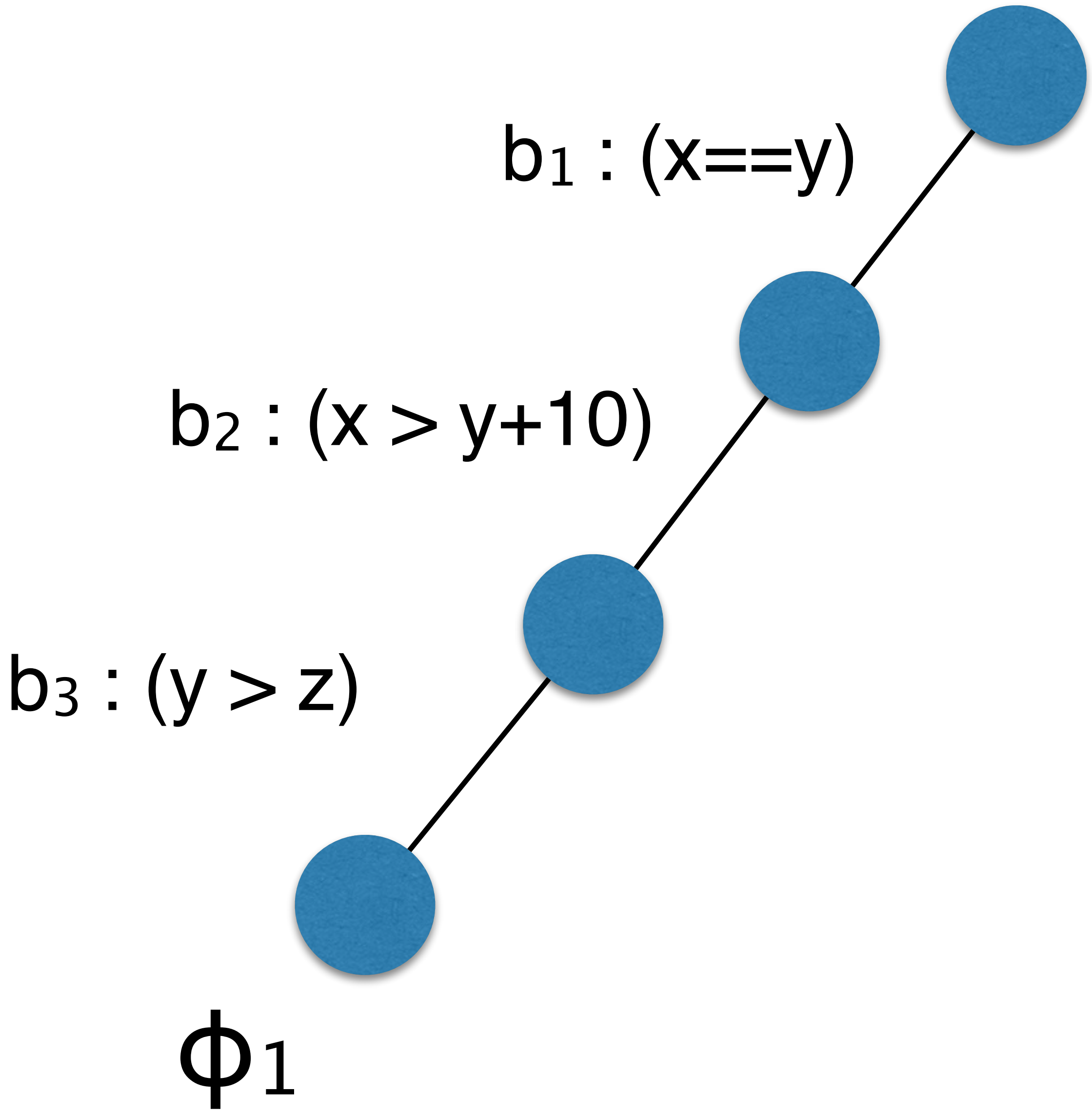
x=30, y=15, z=30

x=α, y=β,z=2*β

$2*\beta = \alpha \land$
$\alpha > \beta+15$

3rd iteration

Concolic Testing



$b_1 : (x==y)$

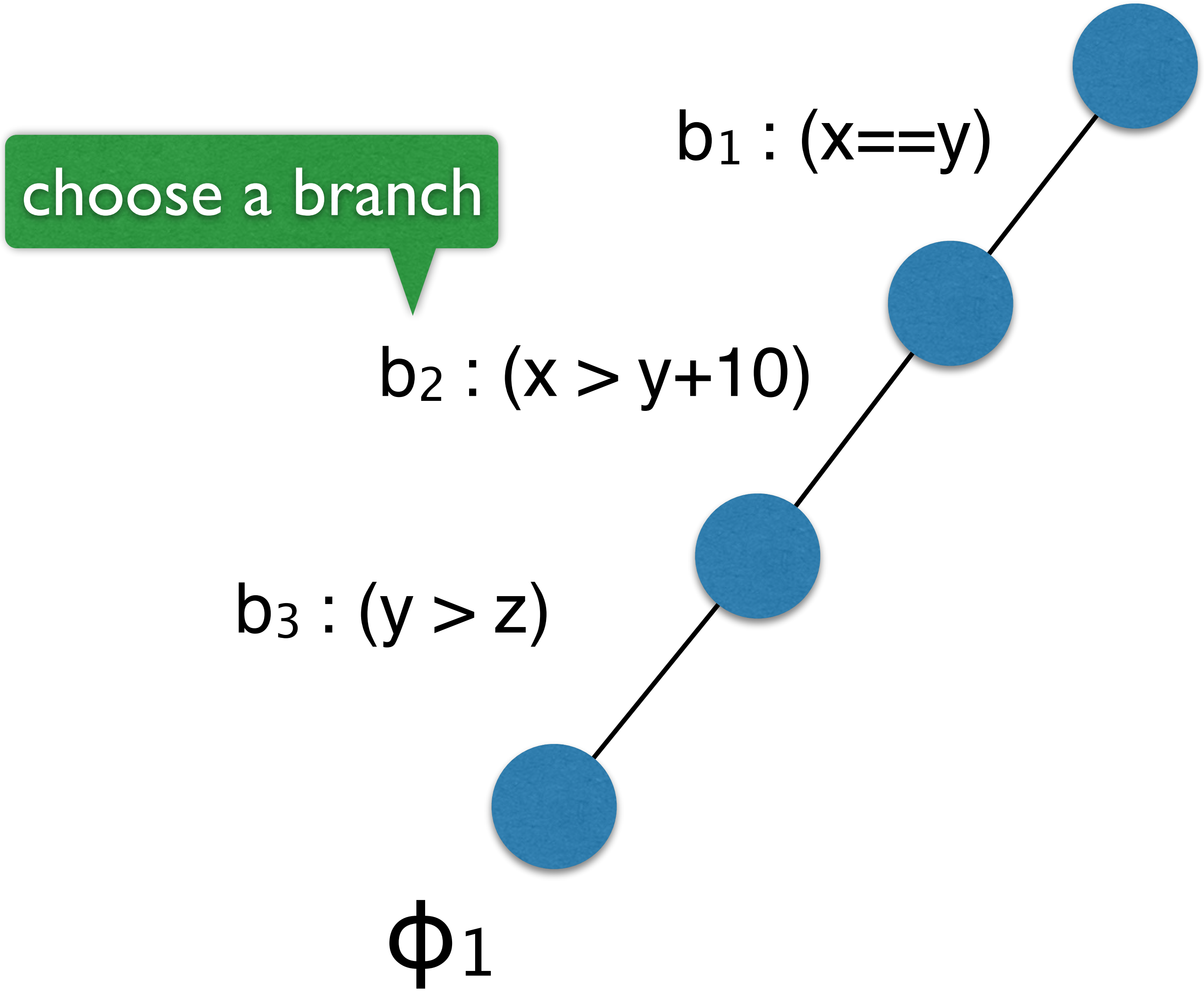
$b_2 : (x > y+10)$

$b_3 : (y > z)$

ϕ_1

execution tree

Concolic Testing



choose a branch

$b_1 : (x==y)$

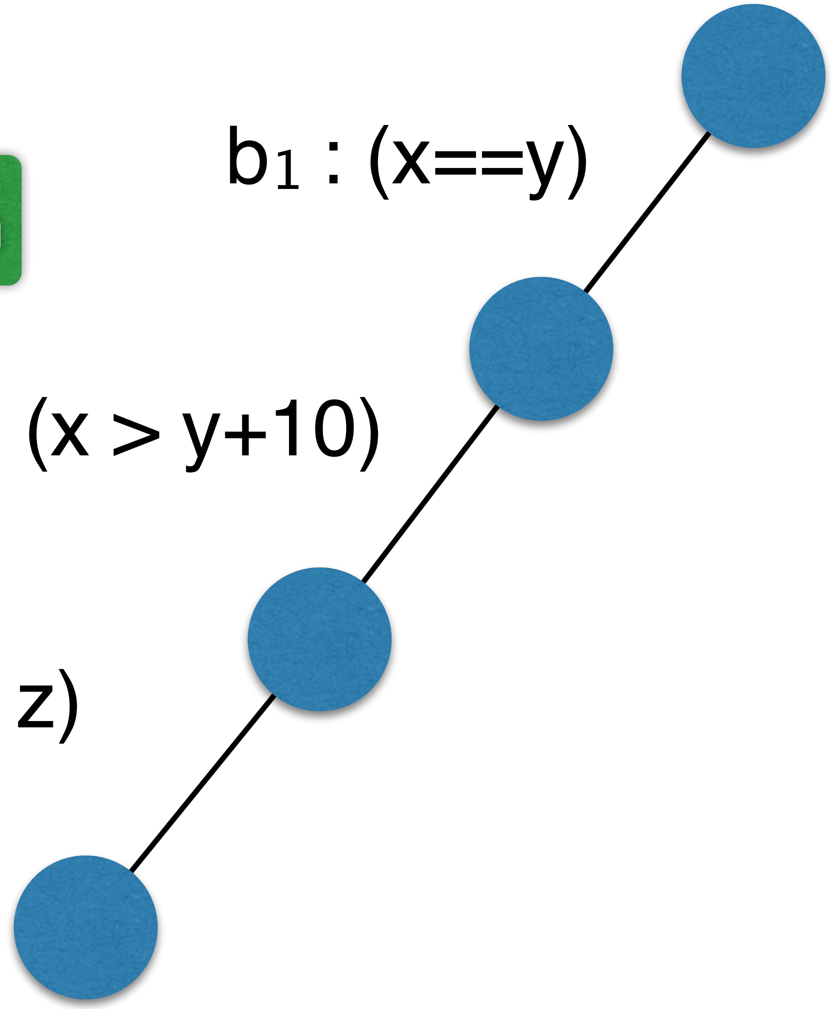
$b_2 : (x > y+10)$

$b_3 : (y > z)$

ϕ_1

execution tree

Concolic Testing



choose a branch

$b_1 : (x==y)$

$b_2 : (x > y+10)$

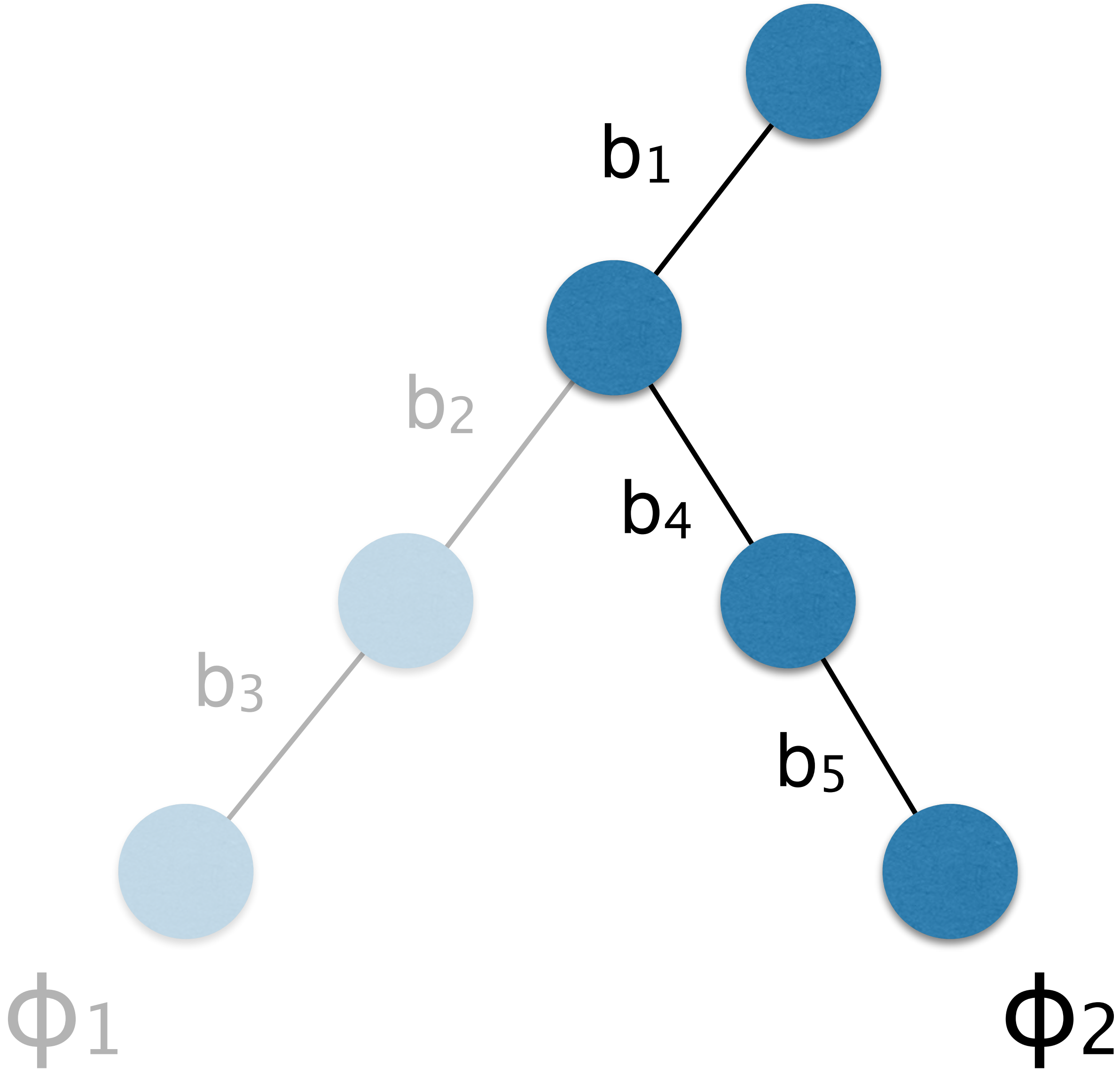
$b_3 : (y > z)$

ϕ_1

solve $(b_1 \wedge \neg b_2)$

execution tree

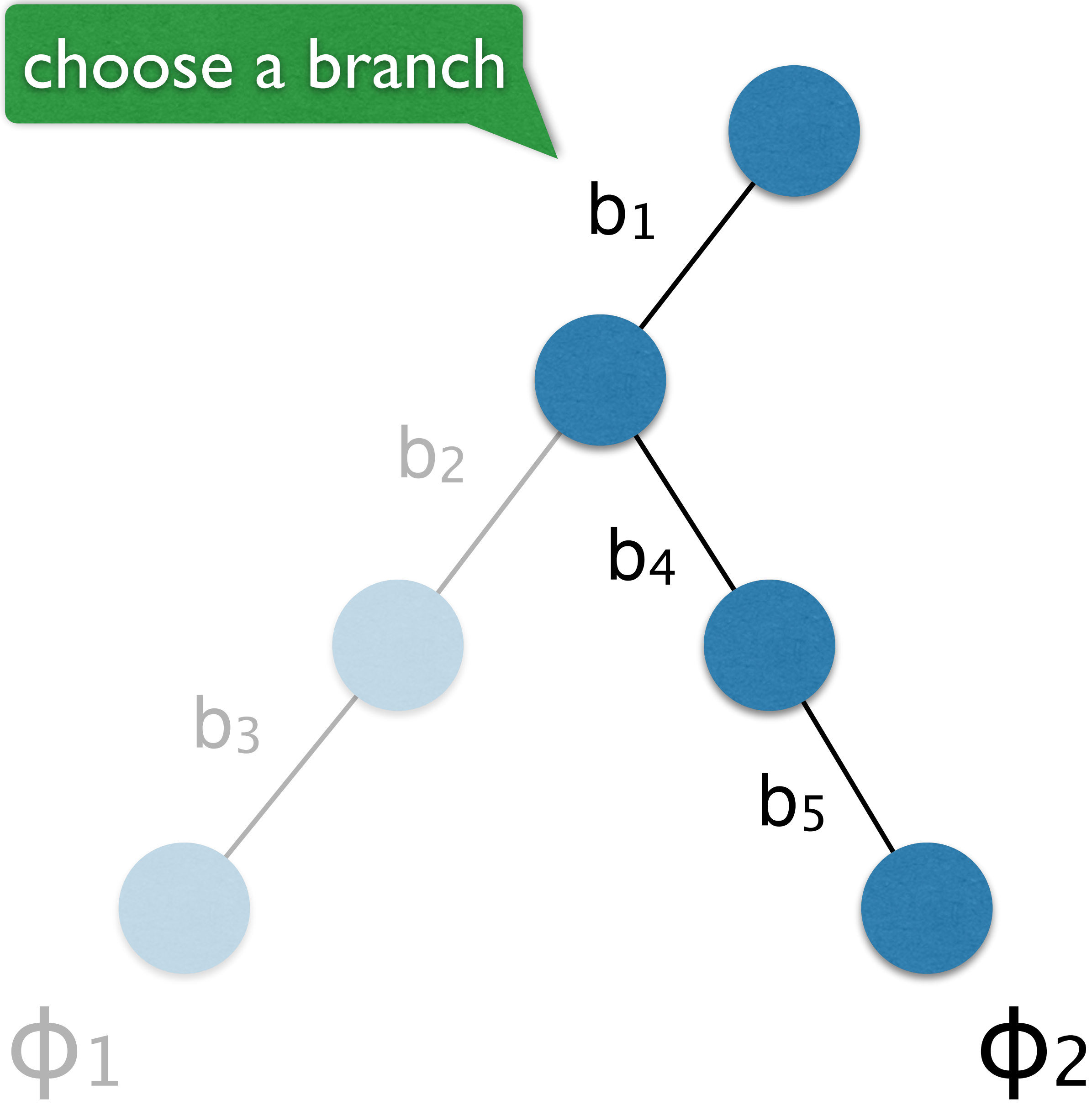
Concolic Testing



execution tree

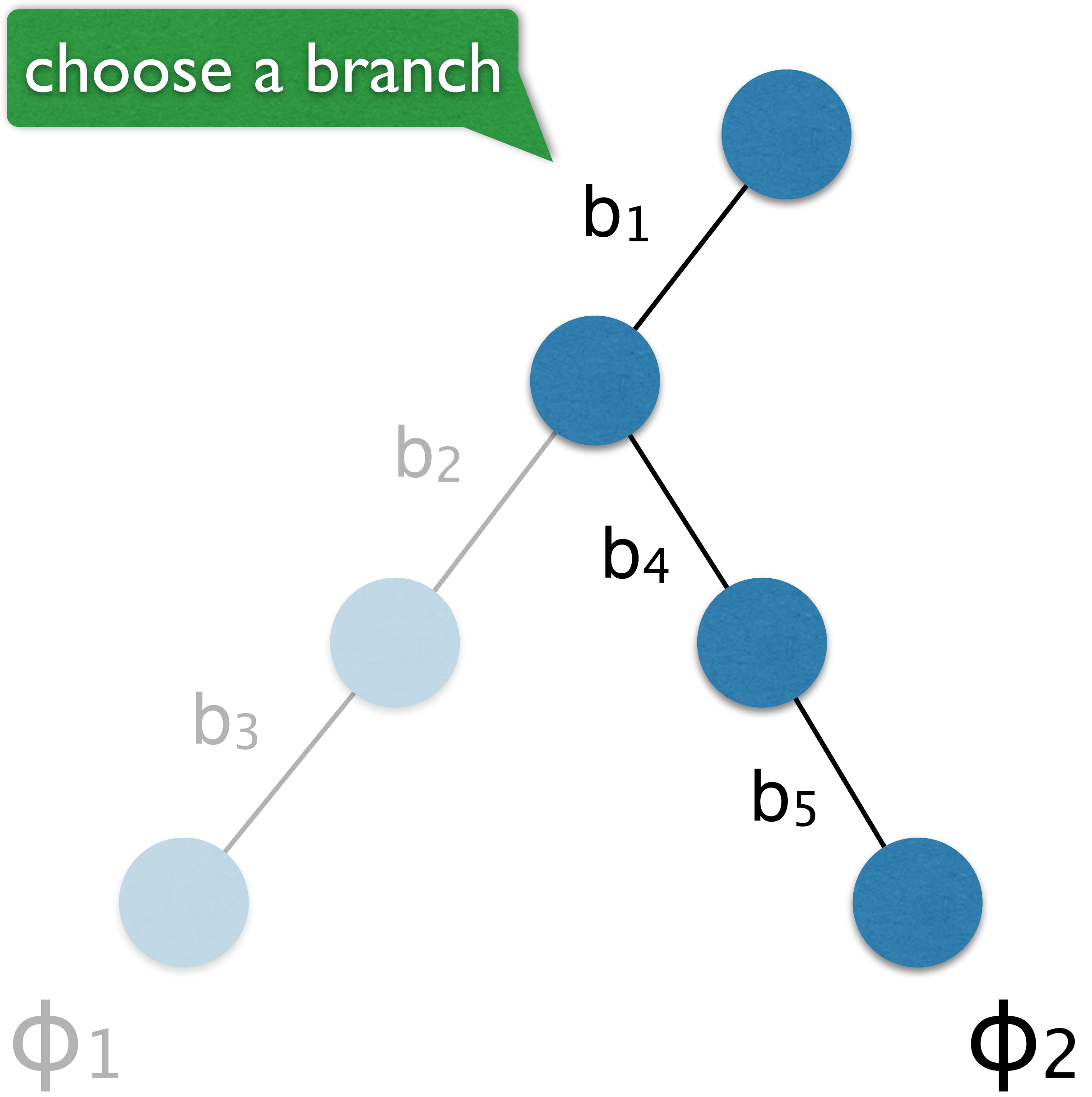
Concolic Testing

choose a branch



execution tree

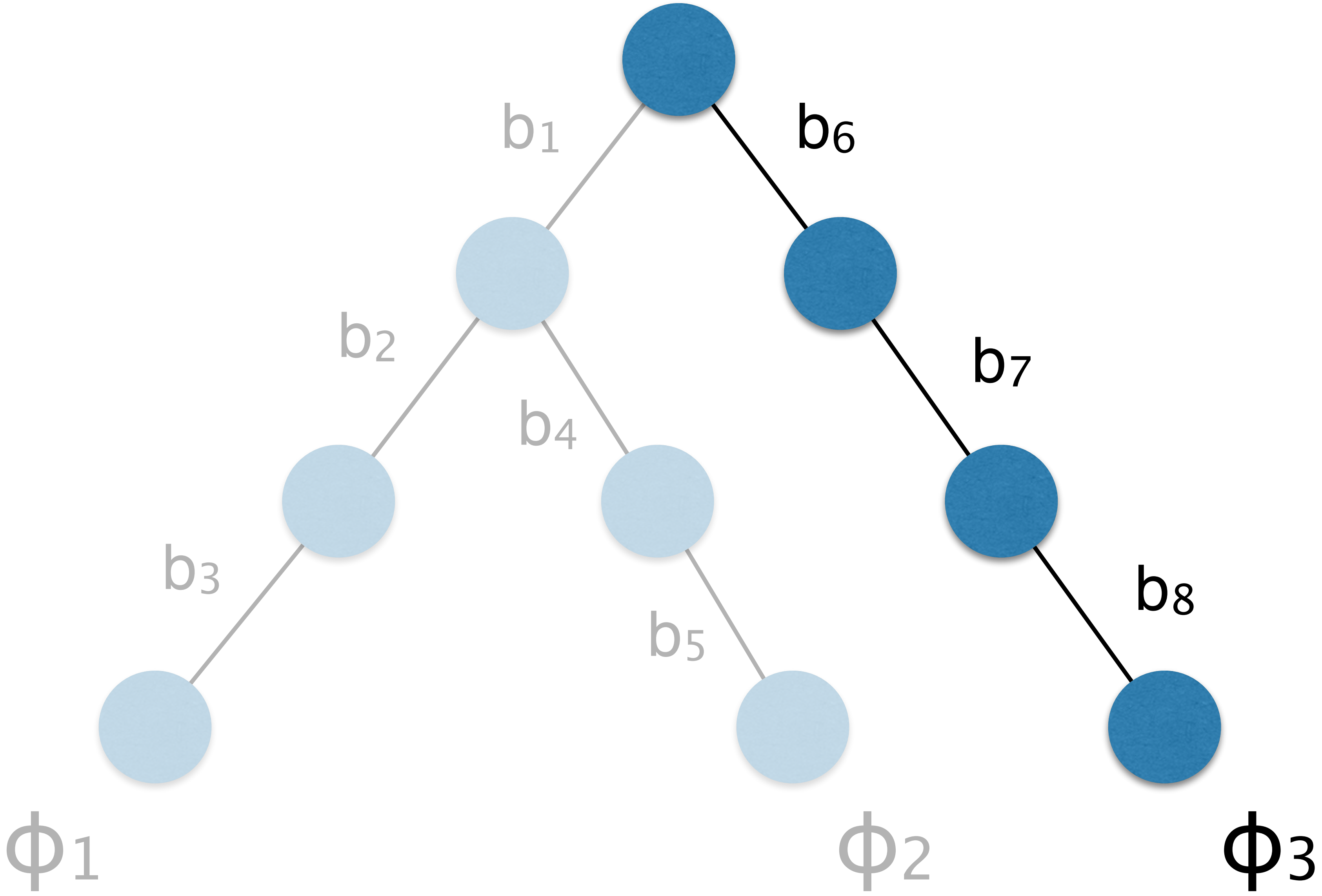
Concolic Testing



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

```
$T \leftarrow \langle\rangle$
$v \leftarrow v_0$
for $m = 1$ to $N$ do
    $\Phi_m \leftarrow \mathrm{RunProgram}(P, v)$
    $T \leftarrow T \cdot \Phi_m$
    repeat
        $(\Phi, \phi_i) \leftarrow \mathrm{Choose}(T)$        $(\Phi = \phi_1 \wedge \cdots \wedge \phi_n)$
    until $\mathrm{SAT}(\bigwedge_{j<i} \phi_j \wedge \neg\phi_i)$
    $v \leftarrow \mathrm{model}(\bigwedge_{j<i} \phi_j \wedge \neg\phi_i)$
end for
return $|\mathrm{Branches}(T)|$
```

Concolic Testing Algorithm

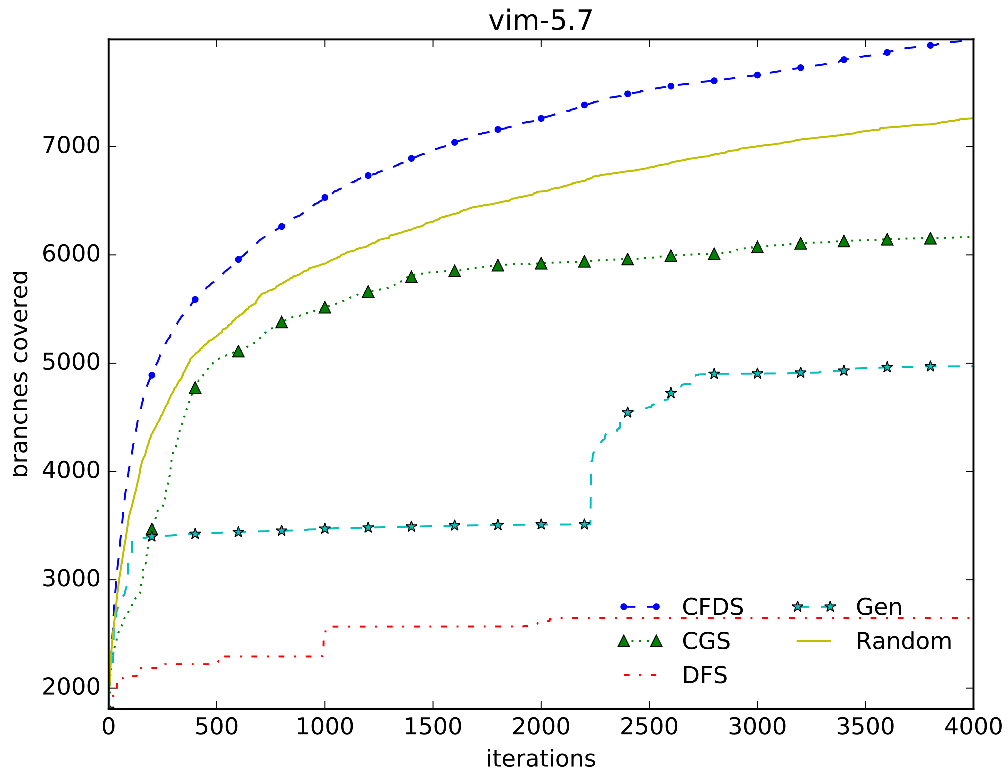
Input : Program P, initial input vector v_0, budget N
Output : The number of branches covered

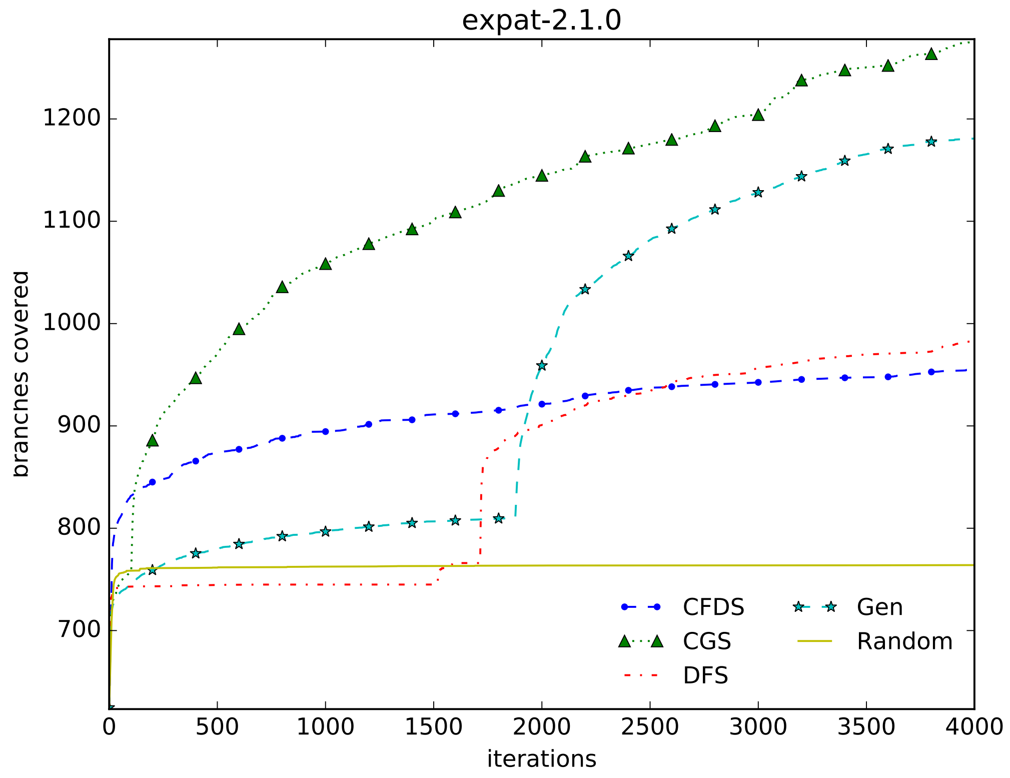
```
$T \leftarrow \langle\rangle$
$v \leftarrow v_0$
for $m = 1$ to $N$ do
    $\Phi_m \leftarrow \text{RunProgram}(P$
    $T \leftarrow T \cdot \Phi_m$
    repeat
        $(\Phi, \phi_i) \leftarrow \text{Choose}(T)$        $(\Phi = \phi_1 \wedge \cdots \wedge \phi_n)$
    until $\text{SAT}(\bigwedge_{j<i} \phi_j \wedge \neg\phi_i)$
    $v \leftarrow \text{model}(\bigwedge_{j<i} \phi_j \wedge \neg\phi_i)$
end for
return $|\text{Branches}(T)|$
```

Search Heuristic

Existing Search Heuristics

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





Data-Driven Symbolic Execution

- Goal: Automatically generating heuristics for symbolic execution heuristics
- Application: search heuristic, path pruning heuristic, state merging heuristic, symbolization heuristic, etc

Automatically Generating Search Heuristics for Concolic Testing

Sooyoung Cha
Korea University
sooyoungcha@korea.ac.kr

Seongjoon Hong
Korea University
seongjoon@korea.ac.kr

Junhee Lee
Korea University
junhee_lee@korea.ac.kr

Hakjoo Oh*
Korea University
hakjoo_oh@korea.ac.kr

ABSTRACT

We present a technique to automatically generate search heuristics for concolic testing. A key challenge in concolic testing is how to effectively explore the program's execution paths to achieve high code coverage in a limited time budget. Concolic testing employs a search heuristic to address this challenge, which favors exploring particular types of paths that are most likely to maximize the final coverage. However, manually designing a good search heuristic is nontrivial and typically ends up with suboptimal and unstable outcomes. The goal of this paper is to overcome this shortcoming of concolic testing by automatically generating search heuristics. We define a class of search heuristics, namely a parameterized heuristic, and present an algorithm that efficiently finds an optimal heuristic for each subject program. Experimental results with open-source C programs show that our technique successfully generates search heuristics that significantly outperform existing manually-crafted heuristics in terms of branch coverage and bug-finding.

CCS CONCEPTS

- **Software and its engineering → Software testing and debugging**;

ACM Reference Format:
Sooyoung Cha, Seongjoon Hong, Junhee Lee, and Hakjoo Oh. 2018. Automatically Generating Search Heuristics for Concolic Testing. In *ICSE '18: ICSE '18: 40th International Conference on Software Engineering , May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3180155.3180166

1 INTRODUCTION

Concolic testing [15, 28] has emerged as an effective software-testing method with diverse applications [1, 7, 21, 30, 33]. The idea of concolic testing is to symbolically execute a program alongside the concrete execution, where the main job of the symbolic execution is to collect path conditions. Initially, the program is executed with a random input. After the program finishes, a branch of the current path is selected and negated to find an input that drives the next program execution to follow a previously unexplored path. This way concolic testing systematically explores the execution paths of the program, greatly improving random testing.

A key component of concolic testing is the so-called search heuristic. Because of the path-explosion problem, exploring all execution paths of a nontrivial program is simply impossible. Instead, concolic testing relies on a search heuristic to maximize code coverage in a limited time budget. A search heuristic has a criterion and steers concolic testing by choosing the best branch to negate according to the criterion. For example, the CFDS (Control-Flow Directed Search) heuristic [3] picks the branch that is closest to the uncovered regions of the program and the CGS (Context-Guided Search) heuristic [29] selects a branch only if it is in a new context. It is well-known that the effectiveness of concolic testing depends heavily on the choice of the search heuristic [3, 21, 27, 29].

However, manually designing such a heuristic is challenging. It is not only nontrivial but also likely to deliver sub-optimal and unstable results. As we demonstrate in this paper, no manually-designed existing heuristics consistently achieve good code coverage in practice. For example, the CGS heuristic is arguably a state-of-the-art and outperforms existing approaches for a number of programs [29]. However, we found that CGS is sometimes brittle and inferior even to a random heuristic. Furthermore, existing search heuristics came from a huge amount of engineering effort and domain expertise. The difficulty of manually coming up with a good search heuristic is a major remaining challenge in concolic testing.

To address this challenge, this paper presents a new approach that automatically generates search heuristics for concolic testing. To this end, we use two key ideas. First, we define a *parameterized search heuristic*, which creates a large class of search heuristics. The parameterized heuristic reduces the problem of designing a good search heuristic into a problem of finding a good parameter value. Second, we present a search algorithm specialized to concolic testing. The search space that the parameterized heuristic poses is intractably large. Our algorithm effectively guides the search by iteratively refining the search space based on the feedback from previous runs of concolic testing.

Experimental results show that automatically-generated heuristics by our approach outperform existing manually-crafted heuristics for a range of C programs. We have implemented our technique in CREST [3] and evaluated it on 10 C programs (0.5–150KLoC). For every benchmark program, our technique has successfully generated a search heuristic that achieves considerably higher branch coverage than the existing state-of-the-art techniques. We also demonstrate that the increased coverage by our technique leads to more effective finding of real bugs.

This paper makes the following contributions:

- We present a new approach for automatically generating search heuristics for concolic testing. Our work represents a significant departure from prior work; while existing work (e.g. [3, 21, 27, 29]) focuses on manually developing a particular search heuristic, our goal is to automate the very process of generating such a heuristic.

---

*Corresponding author

ICSE '18, May 27-June 3, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5638-1/18/05…$15.00
https://doi.org/10.1145/3180155.3180166

ICSE'18

Template-Guided Concolic Testing via Online Learning

Sooyoung Cha
Korea University
Republic of Korea
sooyoungcha@korea.ac.kr

Seonho Lee
Korea University
Republic of Korea
seonho_lee@korea.ac.kr

Hakjoo Oh*
Korea University
Republic of Korea
hakjoo_oh@korea.ac.kr

ABSTRACT

We present template-guided concolic testing, a new technique for effectively reducing the search space in concolic testing. Addressing the path-explosion problem has been a significant challenge in concolic testing. Diverse search heuristics have been proposed to mitigate this problem but using search heuristics alone is not sufficient to substantially improve code coverage for real-world programs. The goal of this paper is to complement existing techniques and achieve higher coverage by exploiting templates in concolic testing. In our approach, a template is a partially symbolized input vector whose job is to reduce the search space. However, choosing a right set of templates is nontrivial and significantly affects the final performance of our approach. We present an algorithm that automatically learns useful templates online, based on data collected from previous runs of concolic testing. The experimental results with open-source programs show that our technique achieves greater branch coverage and finds bugs more effectively than conventional concolic testing.

CCS CONCEPTS

- **Software and its engineering → Software testing and debugging**;

KEYWORDS

Concolic Testing, Online Learning

ACM Reference Format:
Sooyoung Cha, Seonho Lee, and Hakjoo Oh. 2018. Template-Guided Concolic Testing via Online Learning. In *Proceedings of the 2018 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE '18), September 3–7, 2018, Montpellier, France.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3238147.3238227

1 INTRODUCTION

Concolic testing [11, 22] is a popular software testing method that effectively and systematically achieves high code coverage and finds bugs. The key idea of concolic testing is to simultaneously execute a program concretely and symbolically, where new test cases are systematically generated by symbolic execution enhanced with concrete execution. Recently, concolic testing has been used in diverse application domains such as operating systems [18], firmware [8, 16, 31], and binary code [1, 25] among many others.

A major open challenge in concolic testing is how to effectively explore the search space. As the number of execution paths in a realistic program grows exponential, concolic testing must be able to favor and explore the paths that are most likely to benefit the final testing results. However, guiding concolic testing effectively is nontrivial and many different approaches exist with the goal of mitigating the path-explosion problem: e.g., path pruning [2, 3, 17, 28], search heuristics [4, 5, 19, 23, 29], and so on.

In this paper, we present template-guided concolic testing, a new technique for adaptively reducing the search space of concolic testing. The key idea is to guide concolic testing with templates, which restrict the input space by selectively generating symbolic variables. Unlike conventional concolic testing that tracks all input values symbolically, our technique treats a set of selected input values as symbolic and fixes unselected inputs with particular concrete inputs, thereby reducing the original search space. A challenge, however, is choosing input values to track symbolically and replacing the remaining inputs with appropriate values. To address this challenge, we develop an algorithm that performs concolic testing while automatically generating, using, and refining templates. The algorithm is based on two key ideas. First, by using the sequential pattern mining [9], we generate the candidate templates from a set of effective test-cases, where the test-cases contribute to improving code coverage and are collected while conventional concolic testing is performed. Second, we use an algorithm that learns effective templates from the candidates during concolic testing. Our algorithm iteratively ranks the candidates based on the effectiveness of templates that were evaluated in the previous runs. Our technique is orthogonal to the existing techniques and can be fruitfully combined with them, in particular with the state-of-the-art search heuristics.

Experimental results show that our approach outperforms conventional concolic testing in term of branch coverage and bug-finding. We have implemented our approach in CREST [7] and compared our technique with conventional concolic testing for open-source C programs of medium size (up to 165K LOC). For all benchmarks, our technique achieves significantly higher branch coverage compared to conventional concolic testing. For example, for vim-5.7, we have performed both techniques for 70 hours, where our technique exclusively covered 883 branches that conventional concolic testing failed to reach. Our technique also succeeded in finding real bugs that can be triggered in the latest versions of three open-source C programs: sed-4.4, grep-3.1 and gawk-4.21.

---

*Corresponding author

ASE '18, September 3–7, 2018, Montpellier, France

ACM ISBN 978-1-4503-5937-5/18/09…$15.00
https://doi.org/10.1145/3238147.3238227

ASE'18

Concolic Testing with Adaptively Changing Search Heuristics

Sooyoung Cha
Korea University
Republic of Korea
sooyoungcha@korea.ac.kr

Hakjoo Oh*
Korea University
Republic of Korea
hakjoo_oh@korea.ac.kr

ABSTRACT

We present Chameleon, a new approach for adaptively changing search heuristics during concolic testing. Search heuristics play a central role in concolic testing as they mitigate the path-explosion problem by focusing on particular program paths that are likely to increase code coverage as quickly as possible. A variety of techniques for search heuristics have been proposed over the past decade. However, existing approaches are limited in that they use the same search heuristics throughout the entire testing process, which is inherently insufficient to exercise various execution paths. Chameleon overcomes this limitation by adapting search heuristics on the fly via an algorithm that learns new search heuristics based on the knowledge accumulated during concolic testing. Experimental results show that the transition from the traditional non-adaptive approaches to ours greatly improves the practicality of concolic testing in terms of both code coverage and bug-finding.

CCS CONCEPTS

- **Software and its engineering → Software testing and debugging**.

KEYWORDS

Concolic Testing, Dynamic Symbolic Execution, Online Learning

ACM Reference Format:
Sooyoung Cha and Hakjoo Oh. 2019. Concolic Testing with Adaptively Changing Search Heuristics. In *Proceedings of the 27th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '19), August 26–30, 2019, Tallinn, Estonia.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3338906.3338964

1 INTRODUCTION

Concolic testing [11, 27] is a promising software testing technique popular in both academia and industry [1, 5, 6, 19, 20, 30, 32, 33]. The technique aims to increase code coverage as quickly as possible, ultimately enabling effective bug-finding in a limited time budget. To do so, unlike random testing or fuzzing, concolic testing systematically generates test-cases by repeating the following process: (1) it *concolically* executes the subject program to collect the path condition, i.e., the sequence of symbolic branch conditions exercised by the current program execution, (2) it produces a new path condition by selecting and negating a branch of the current path condition, and (3) it solves the resulting path condition to generate a new test-case that guides the next program execution towards the opposite of the selected branch. Because of this systematic nature, concolic testing is increasingly used in diverse domains, including operating systems [19], embedded systems [10, 14], and even neural networks [30], among others.

Concolic testing includes search heuristics as a critical ingredient. To be practical for real-world applications, concolic testing must be able to adequately address the path-explosion problem; because real-world programs exhibit infinitely many different paths, it is impossible to exercise all of them by testing. To address this challenge, concolic testing uses a search heuristic, a branch selection strategy that takes a path condition and selects a branch based on its own criterion (it is used in the second step of the concolic testing process described in the preceding paragraph). Search heuristics allow concolic testing to preferentially explore particular classes of execution paths that they think are most effective to maximize code coverage within a given time limit. It has been well-known that how to choose and use search heuristics is critically important, and diverse approaches have been proposed to improve concolic testing in practice over the past decade [3–5, 19, 22, 26, 28].

In this paper, we propose a new approach, called Chameleon, for effectively employing search heuristics during concolic testing. The key novelty of Chameleon is *adaptively* changing search heuristics on the fly, so that the branch-selection criterion changes as necessary throughout concolic testing in a way that maximizes the final performance. By contrast, all of the existing approaches for employing search heuristics [3–5, 19, 22, 26, 28] are not adaptive as they use the same search heuristics over the whole process of concolic testing. In this paper, we demonstrate that this is a key limiting factor of the existing approaches, and we can make concolic testing much more practical for real-world applications by being adaptive. We illustrate the limitation of existing search heuristics in more detail in Section 2.

To enable adaptation, we present an algorithm that automatically learns and switches search heuristics during concolic testing. The algorithm maintains a set of search heuristics and continuously changes them during the testing process. To do so, we first define the space of possible search heuristics using the idea of parametric search heuristic recently proposed in prior work [5]. A technical challenge is how to adaptively switch search heuristics in the predefined space. We address this challenge with a new concolic testing algorithm that (1) accumulates the knowledge about the previously evaluated search heuristics, (2) learns the probabilistic distributions of the effective and ineffective search heuristics from the accumulated knowledge, and (3) samples a new set of search heuristics

---

*Corresponding author

ESEC/FSE '19, August 26–30, 2019, Tallinn, Estonia

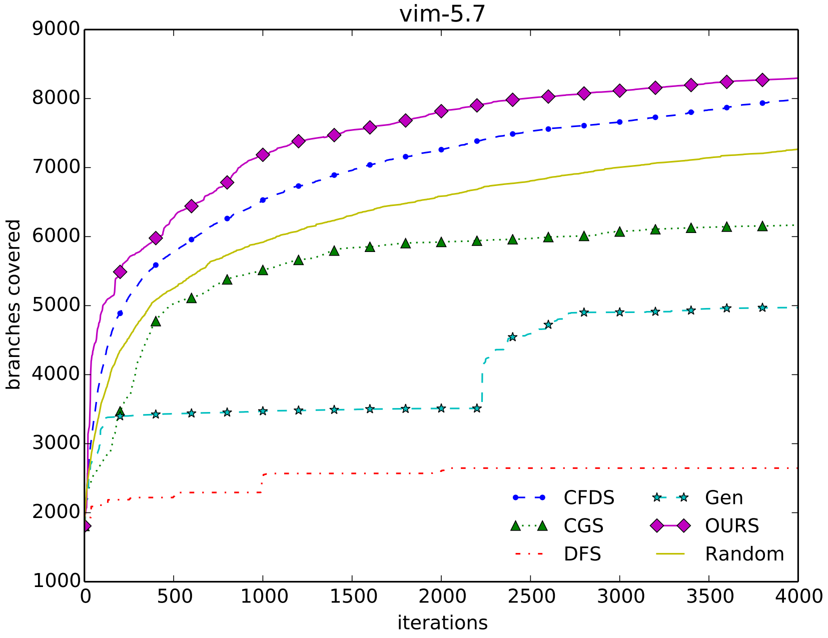
ACM ISBN 978-1-4503-5572-8/19/08…$15.00
https://doi.org/10.1145/3338906.3338964

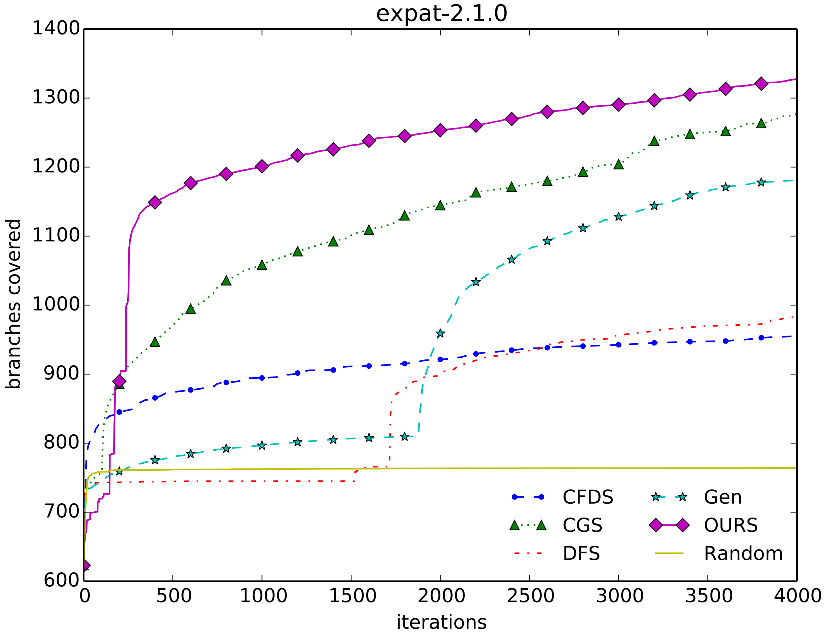
FSE'19

Effectiveness

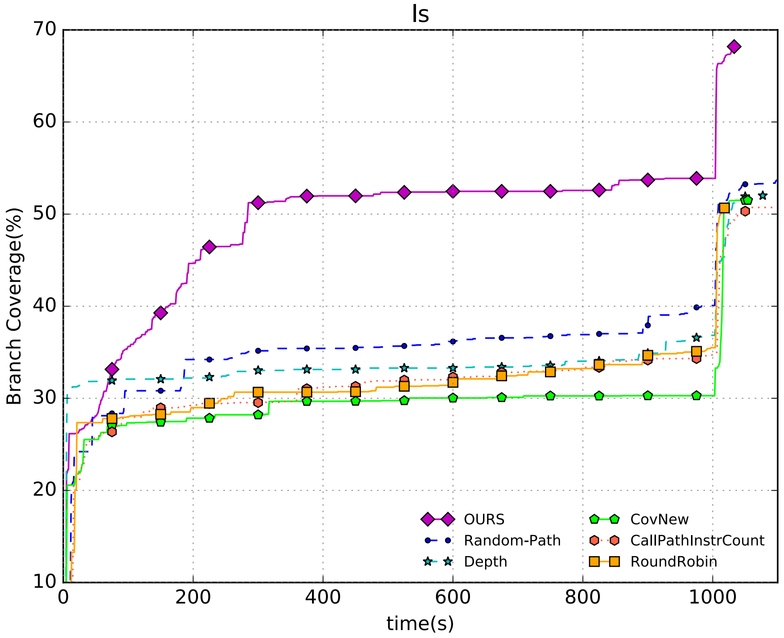
- Improved code coverage

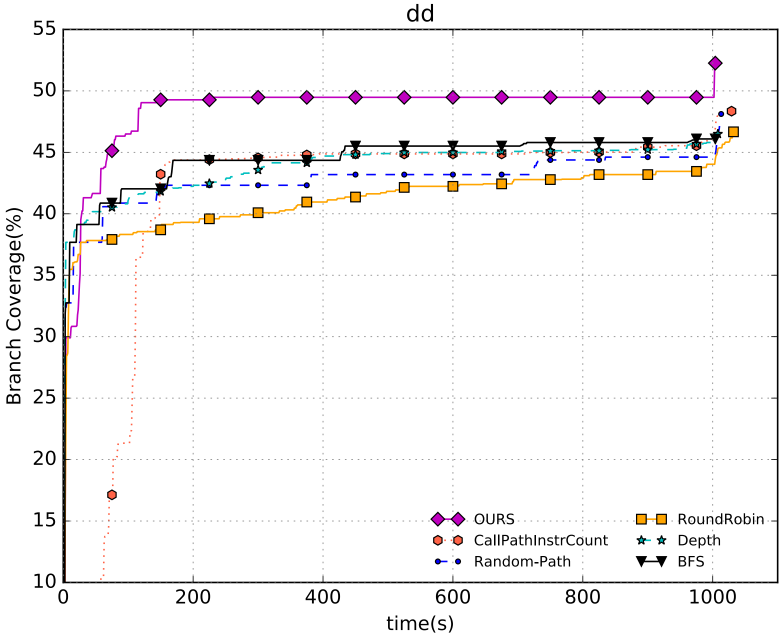
CREST





KLEE





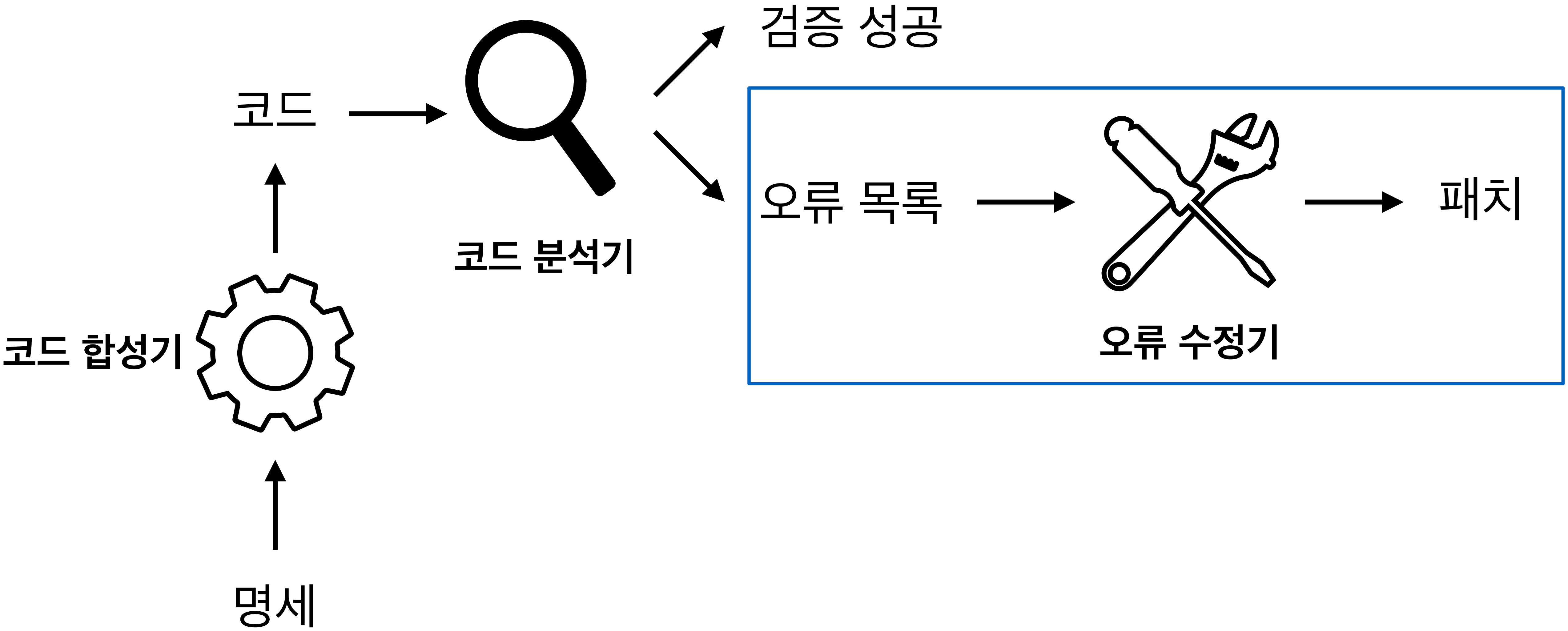
Effectiveness

- Increased bug-finding capability

<table>
<tr><th>Benchmarks</th><th>Versions</th><th>Error Types</th><th>Bug-Triggering Inputs</th><th>OURS</th><th>Param</th><th>RR</th><th>CGS</th><th>CFDS</th><th>Gen</th><th>Random</th></tr>
<tr><td>vim</td><td>8.1*</td><td>Non-termination</td><td>K1!10001001001111110(</td><td>✓</td><td>✗</td><td>✗</td><td>✗</td><td>✗</td><td>✗</td><td>✗</td></tr>
<tr><td></td><td>5.7</td><td>Abnormal-termination</td><td>H:w>>`"`\ [press ‘Enter’]</td><td>✓</td><td>✓</td><td>✗</td><td>✗</td><td>✗</td><td>✓</td><td>✓</td></tr>
<tr><td></td><td></td><td>Segmentation fault</td><td>=ipI\-9~q0qw</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✗</td><td>✗</td><td>✓</td></tr>
<tr><td></td><td></td><td>Non-termination</td><td>v(ipaprq&T$T</td><td>✓</td><td>✓</td><td>✓</td><td>✗</td><td>✗</td><td>✗</td><td>✓</td></tr>
<tr><td>gawk</td><td>4.2.1*</td><td>Memory-exhaustion</td><td>'+E_Q$h+w$8==++$6E8#'</td><td>✓</td><td>✗</td><td>✗</td><td>✗</td><td>✗</td><td>✗</td><td>✗</td></tr>
<tr><td></td><td>3.0.3</td><td>Abnormal-termination</td><td>'f[][][][][y]^/#['</td><td>✓</td><td>✗</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td></tr>
<tr><td></td><td></td><td>Non-termination</td><td>'$g?E2^=-E-2"?^+$=":/?/#["'</td><td>✓</td><td>✓</td><td>✗</td><td>✗</td><td>✓</td><td>✗</td><td>✗</td></tr>
<tr><td>grep</td><td>3.1*</td><td>Abnormal-termination</td><td>'\(\)\1*?*?\|\W*\1W*'</td><td>✓</td><td>✗</td><td>✗</td><td>✗</td><td>✗</td><td>✗</td><td>✗</td></tr>
<tr><td></td><td></td><td>Segmentation fault</td><td>'\(\)\1^*@*\?\1*\+*\?'</td><td>✓</td><td>✗</td><td>✗</td><td>✓</td><td>✗</td><td>✗</td><td>✗</td></tr>
<tr><td></td><td>2.2</td><td>Segmentation fault</td><td>"_^^*9\|^\(\)\'\1*$"</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td></tr>
<tr><td></td><td></td><td>Non-termination</td><td>'\({**+**\)*\++*\1*\+'</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✗</td></tr>
<tr><td>sed</td><td>1.17</td><td>Segmentation fault</td><td>'{:};:C;b'</td><td>✓</td><td>✗</td><td>✓</td><td>✗</td><td>✓</td><td>✓</td><td>✓</td></tr>
</table>

Research Direction

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



명세

코드 합성기

코드

코드 분석기

검증 성공

오류 목록

오류 수정기

패치

자동 디버깅 기술의 필요성

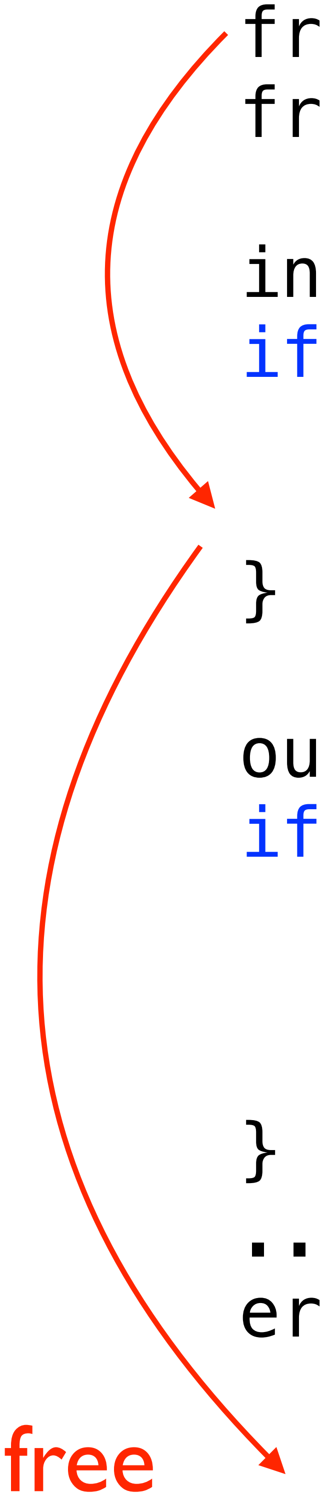
- 소프트웨어 개발에서 디버깅은 가장 어렵고 부담스러운 단계
  - 상용 소프트웨어 오류 수정에 평균 200일 소요[1]
- 다른 개발 단계에 비해 자동화된 도구 지원이 가장 적음
  - 소프트웨어 오류 탐지 분야는 지난 30여년간 눈부신 발전을 이룸
  - 디버깅은 현재 개발자에 전적으로 의존하는 상황

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

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;
```

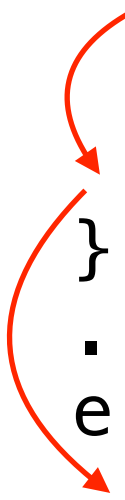
실제 사례
(Linux Kernel)



```
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;
```

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)

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>
```

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

```
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)

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>
```

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

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)

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>

Oliver Neukum committed with **gregkh** on 30 Jun 2008

```
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;
```

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>
```

Oliver Neukum committed with **gregkh** on 30 Jun 2008

실제 사례 (Linux Kernel)

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

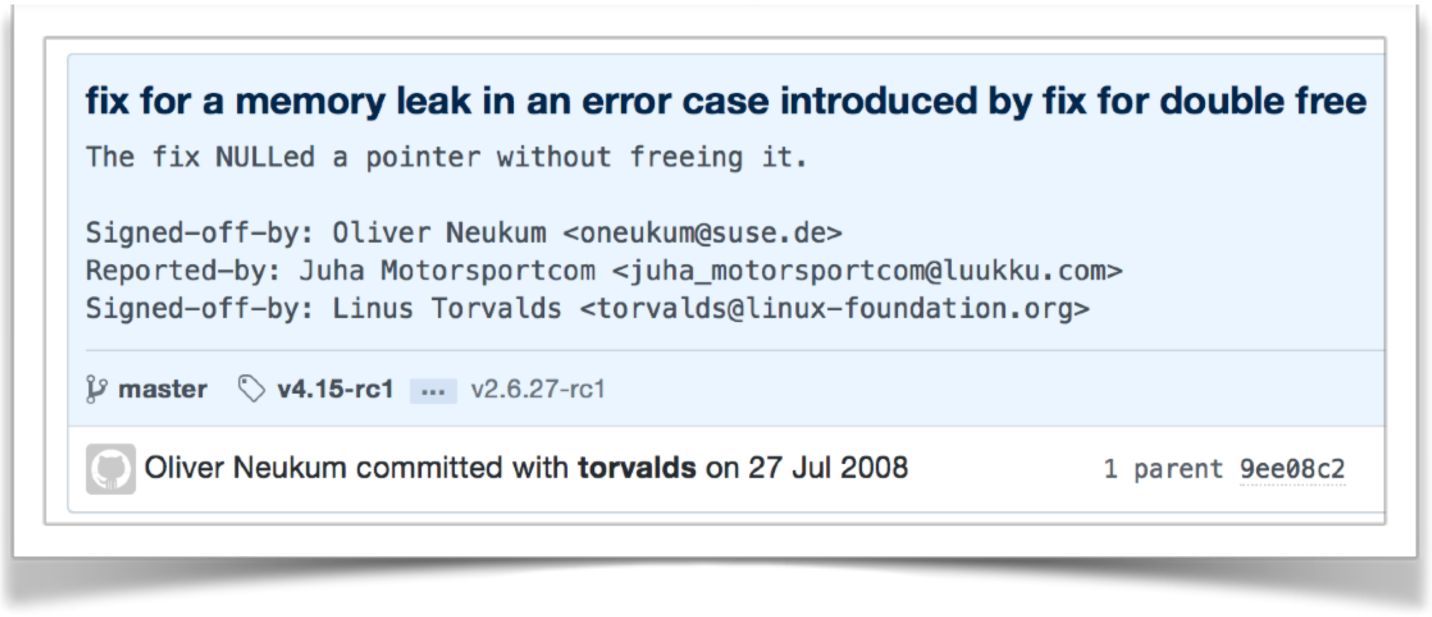
```
The fix NULLed a pointer without freeing it.

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>
```

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;
```

실제 사례 (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>

Oliver Neukum committed with **torvalds** on 27 Jul 2008

1 parent 9ee08c2

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;
```

소프트웨어 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;
```

소프트웨어 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

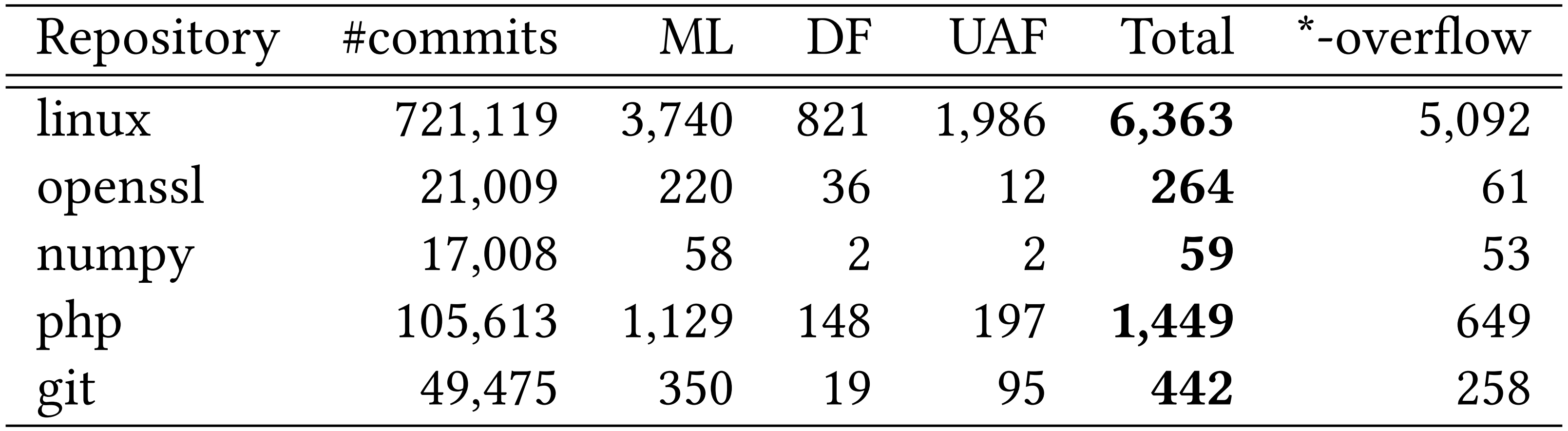
수동 디버깅의 문제 해결:

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

```
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;
```

대상: 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
- 시스템 소프트웨어 결함의 주요 원인



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

FSE'18

MemFix

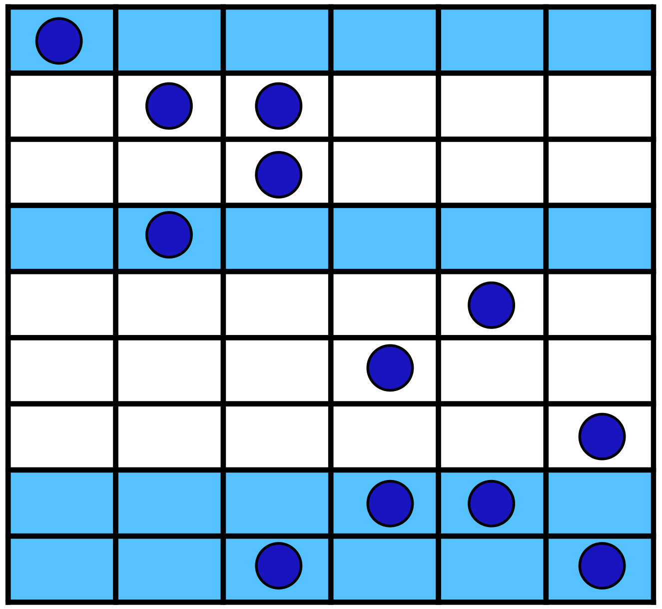
- Automatically repairs deallocation errors
  - **memory-leak**, **double-free** and **use-after-free**
- Key features
  - **sound**: generated patch is guaranteed to be correct
  - **safe**: no new errors are introduced
- Approach: **Static Analysis** + **Exact Cover Problem**

Key Insight

```
out = malloc(1);
in = 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);
```

Find a set of free-statements

⦀



Solve an Exact Cover Problem

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

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

    goto err;
}
free(out); // +
out = malloc(2);
if(out == NULL) {
    // -

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

Performance

state-of-the-art (ICSE'18)

Ours

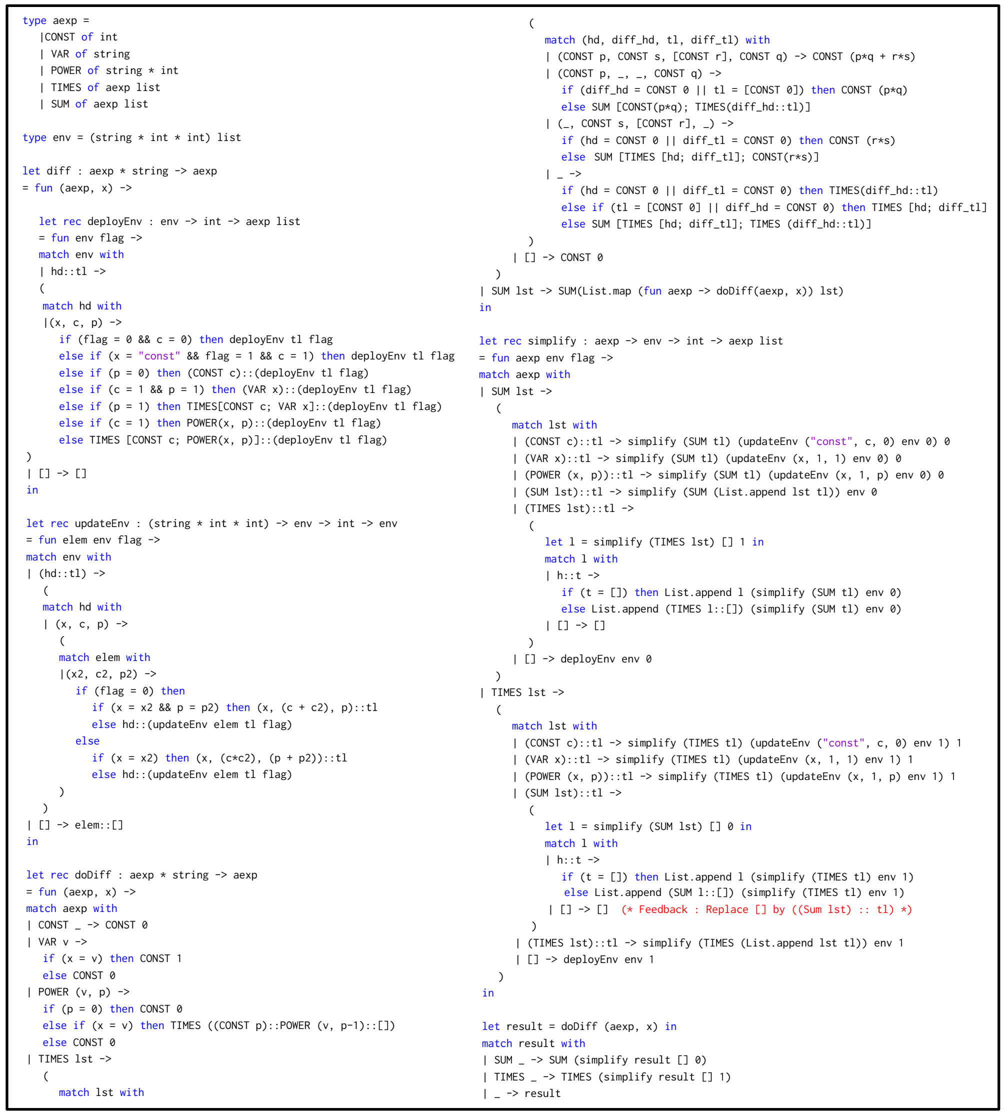
		FootPatch			SAVER		
Projects	TP FP	Generated	Correct	Unsafe	Generated	Correct	Unsafe
rappel (2.1 KLoC)	1	1	1	0	1	1	0
	0	0	-	0	0	-	0
Swoole (44.5 KLoC)	15	9	7	2	12	12	0
	5	2	-	2	0	-	0
lxc (63.0 KLoC)	3	0	0	0	3	3	0
	5	1	-	1	0	-	0
Total	19	10	8	2	16	16	0
	10	3	-	3	0	-	0

Application to Intelligent Tutoring System

- 오류 수정 기술을 함수형 프로그래밍 교육에 적용
- 현재 코딩 교육 자동 도구들의 한계: 개인화된 피드백 제공 못함

```
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)
```

제공된 솔루션



```
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
```

학생 제출 답안

Application to Intelligent Tutoring System

- 오류 수정 기술을 함수형 프로그래밍 교육에 적용
- 현재 코딩 교육 자동 도구들의 한계: 개인화된 피드백 제공 못함

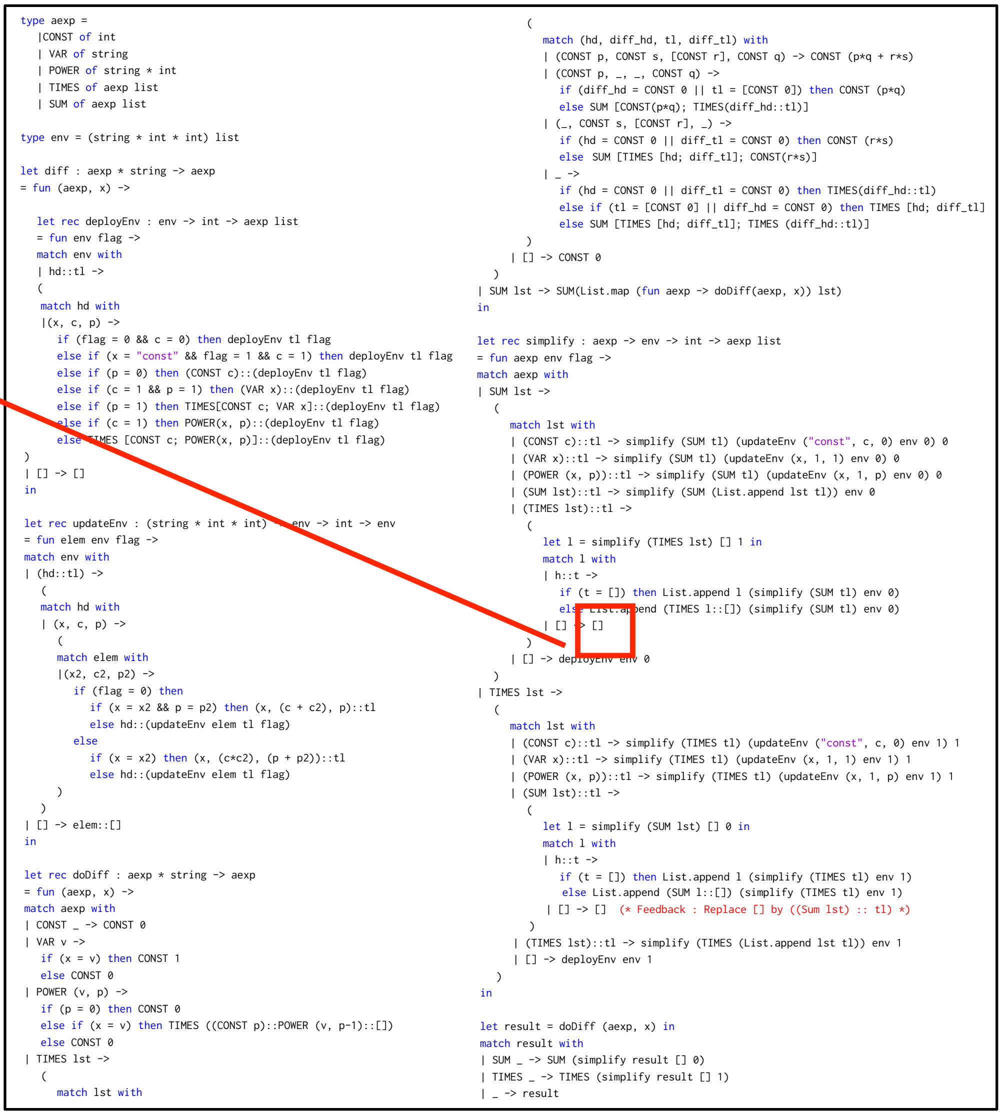
CodeOnWeb

oncoder

FixML-generated feedback: ((Sum lst)::tl)

```
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)
```

제공된 솔루션

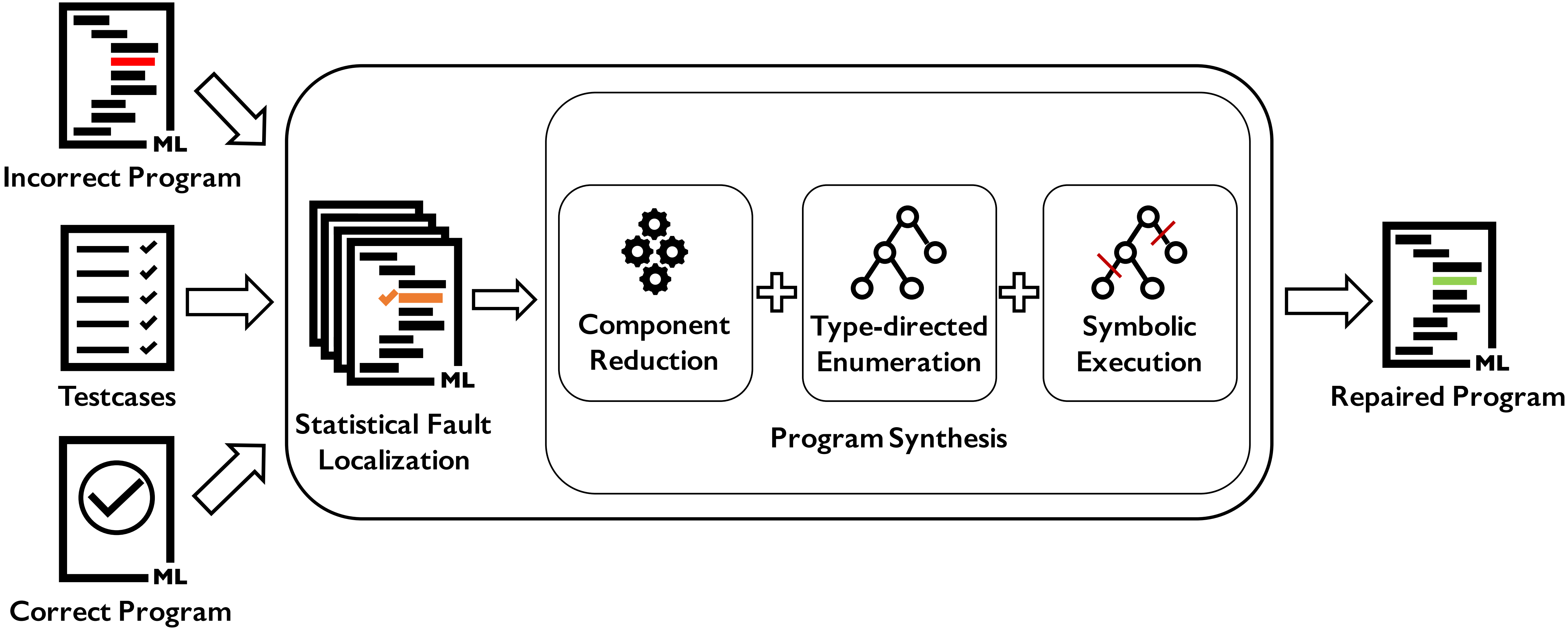


```
type aexp =
  |CONST of int
  | VAR of string
  | POWER of string * int
  | TIMES of aexp list
  | SUM of aexp list

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
```

학생 제출 답안

OOPSLA'18

FixML



Examples

```
let rec sigma f a b =
  if f a != f b then
    let induction = f b in
      induction + sigma f a (b-1)
  else f b
```

```
sigma (fun x -> x) 1 10 = 55
sigma (fun x -> x*x) 1 7 = 140
sigma (fun x -> x mod 3) 1 10 = 10
```

Examples

```
a != b
```

```
let rec sigma f a b =
  if f a != f b then
    let induction = f b in
      induction + sigma f a (b-1)
  else f b
```

```
sigma (fun x -> x) 1 10 = 55
sigma (fun x -> x*x) 1 7 = 140
sigma (fun x -> x mod 3) 1 10 = 10
```

Examples

`a != b`

```
let rec sigma f a b =
  if f a != f b then
    let induction = f b in
      induction + sigma f a (b-1)
  else f b
```

```
sigma (fun x -> x) 1 10 = 55
sigma (fun x -> x*x) 1 7 = 140
sigma (fun x -> x mod 3) 1 10 = 10
```

```
type btree =
  | Empty
  | Node of int * btree * btree

let rec mem n tree =
  match tree with
  | Empty -> false
  | Node (a, b, c) ->
      if a = n then true
      else if a < n then mem n b
           else mem n c
```

```
mem 1 (Node(2,Empty,Empty)) = false
mem 2 (Node(3,Node(2,Empty,Empty),Empty)) = true
```

Examples

```
a != b
```

```
let rec sigma f a b =
  if f a != f b then
    let induction = f b in
      induction + sigma f a (b-1)
  else f b
```

```
sigma (fun x -> x) 1 10 = 55
sigma (fun x -> x*x) 1 7 = 140
sigma (fun x -> x mod 3) 1 10 = 10
```

```
type btree =
  | Empty
  | Node of int * btree * btree

let rec mem n tree =
  match tree with
  | Empty -> false
  | Node (a, b, c) ->
      if a = n then true
      else if a < n then mem n b
           else mem n c
```

```
mem 1 (Node(2,Empty,Empty)) = false
mem 2 (Node(3,Node(2,Empty,Empty),Empty)) = true
```

```
mem n b || mem n c
```

Examples

```
type exp =
  | Num of int
  | Plus of exp * exp
  | Minus of exp * exp

type formula =
  | True
  | False
  | Not of formula
  | AndAlso of formula * formula
  | OrElse of formula * formula
  | Imply of formula * formula
  | Equal of exp * exp

let rec exp_to_int : exp -> int
= fun e ->
  match e with
  | Num n -> n
  | Plus (n1, n2) -> exp_to_int n1 + exp_to_int n2
  | Minus (n1, n2) -> exp_to_int n1 - exp_to_int n2

let rec eval : formula -> bool
= fun f ->
  match f with
  | True -> true
  | False -> false
  | Not f1 -> not (eval f1)
  | AndAlso (f1, f2) -> eval f1 && eval f2
  | OrElse (f1, f2) -> eval f1 || eval f2
  | Imply (f1, f2) ->
    (match (f1, f2) with
     | (True, False) -> false
     | _ -> true)
  | Equal (e1, e2) -> exp_to_int e1 = exp_to_int e2
```

```
eval (Imply(AndAlso(True,False),True)) = true
eval (Equal(Plus(Num 1,Num 2),Num 3)) = true
```

Examples

```
type exp =
  | Num of int
  | Plus of exp * exp
  | Minus of exp * exp

type formula =
  | True
  | False
  | Not of formula
  | AndAlso of formula * formula
  | OrElse of formula * formula
  | Imply of formula * formula
  | Equal of exp * exp

let rec exp_to_int : exp -> int
= fun e ->
  match e with
  | Num n -> n
  | Plus (n1, n2) -> exp_to_int n1 + exp_to_int n2
  | Minus (n1, n2) -> exp_to_int n1 - exp_to_int n2

let rec eval : formula -> bool
= fun f ->
  match f with
  | True -> true
  | False -> false
  | Not f1 -> not (eval f1)
  | AndAlso (f1, f2) -> eval f1 && eval f2
  | OrElse (f1, f2) -> eval f1 || eval f2
  | Imply (f1, f2) ->
   (match (f1, f2) with
    | (True, False) -> false
    | _ -> true)
  | Equal (e1, e2) -> exp_to_int e1 = exp_to_int e2
```

```
eval (Imply(AndAlso(True,False),True)) = true
eval (Equal(Plus(Num 1,Num 2),Num 3)) = true
```

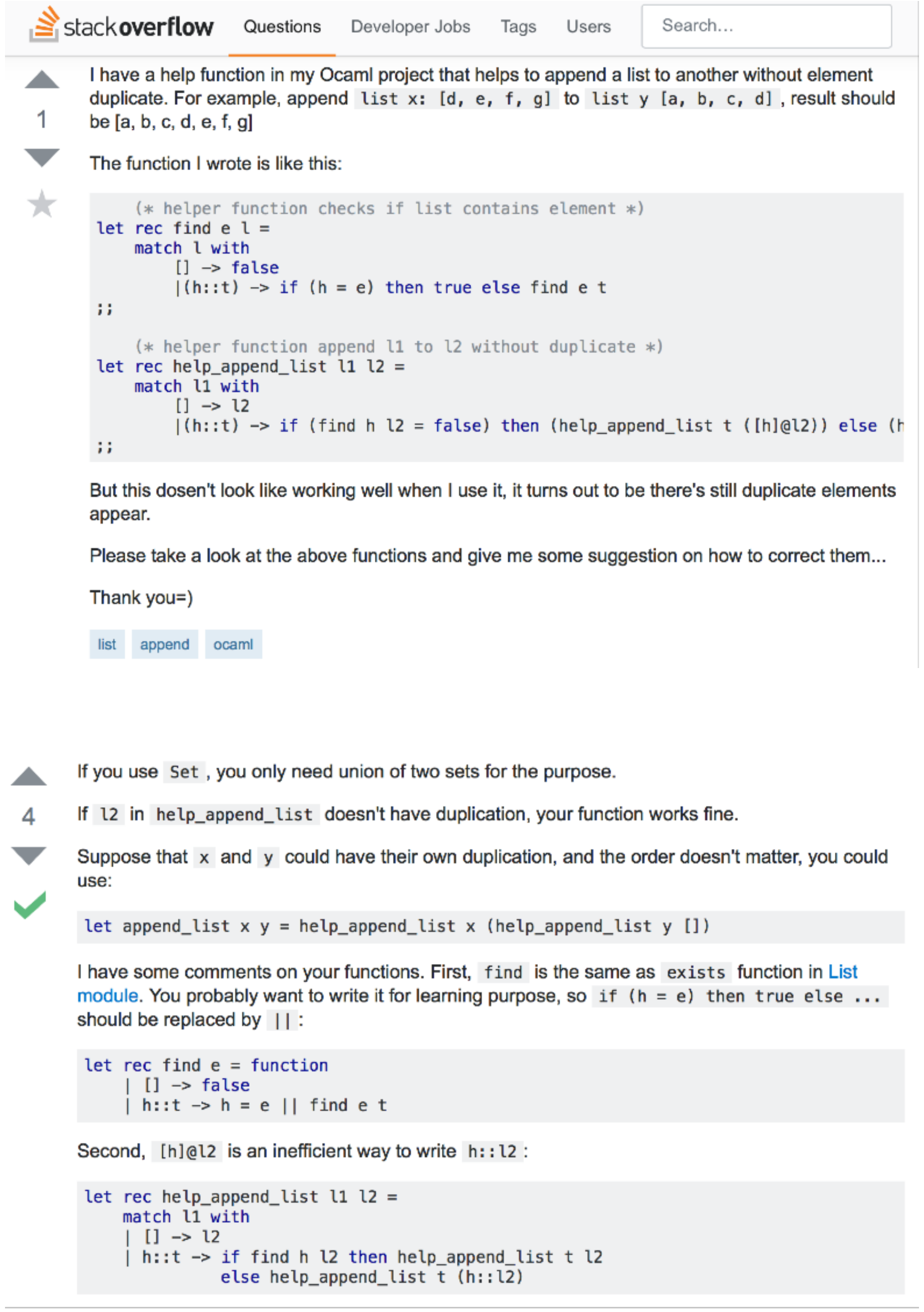
```
not (eval f1) || eval f2
```

Examples

Q) Append lists without duplicates

```
append_list ['d';'e';'f';'g'] ['a';'b';'c';'d']
= ['a'; 'b'; 'c'; 'd'; 'e'; 'f'; ‘g’]

append_list [1;3;5;4;3] [3;5;6;6;4] = [3; 5; 6; 4; 1]
```

```
let rec find e l =
  match l with
  | [] -> false
  | h::t -> if h = e then true else find e t

let rec help_append_list l1 l2 =
  match l1 with
  | [] -> l2
  | h::t ->
    if find h l2 = false then help_append_list t (l2@[h])
    else help_append_list t l2

let append_list x y = help_append_list x y
```



I have a help function in my Ocaml project that helps to append a list to another without element duplicate. For example, append `list x: [d, e, f, g]` to `list y [a, b, c, d]`, result should be [a, b, c, d, e, f, g]

The function I wrote is like this:

```
    (* helper function checks if list contains element *)
let rec find e l =
    match l with
        [] -> false
        |(h::t) -> if (h = e) then true else find e t
;;

    (* helper function append l1 to l2 without duplicate *)
let rec help_append_list l1 l2 =
    match l1 with
        [] -> l2
        |(h::t) -> if (find h l2 = false) then (help_append_list t ([h]@l2)) else (h
;;
```

But this dosen't look like working well when I use it, it turns out to be there's still duplicate elements appear.

Please take a look at the above functions and give me some suggestion on how to correct them...

Thank you=)

If you use `Set`, you only need union of two sets for the purpose.

If `l2` in `help_append_list` doesn't have duplication, your function works fine.

Suppose that `x` and `y` could have their own duplication, and the order doesn't matter, you could use:

```
let append_list x y = help_append_list x (help_append_list y [])
```

I have some comments on your functions. First, `find` is the same as `exists` function in List module. You probably want to write it for learning purpose, so `if (h = e) then true else ...` should be replaced by `||`:

```
let rec find e = function
    | [] -> false
    | h::t -> h = e || find e t
```

Second, `[h]@l2` is an inefficient way to write `h::l2`:

```
let rec help_append_list l1 l2 =
    match l1 with
    | [] -> l2
    | h::t -> if find h l2 then help_append_list t l2
              else help_append_list t (h::l2)
```

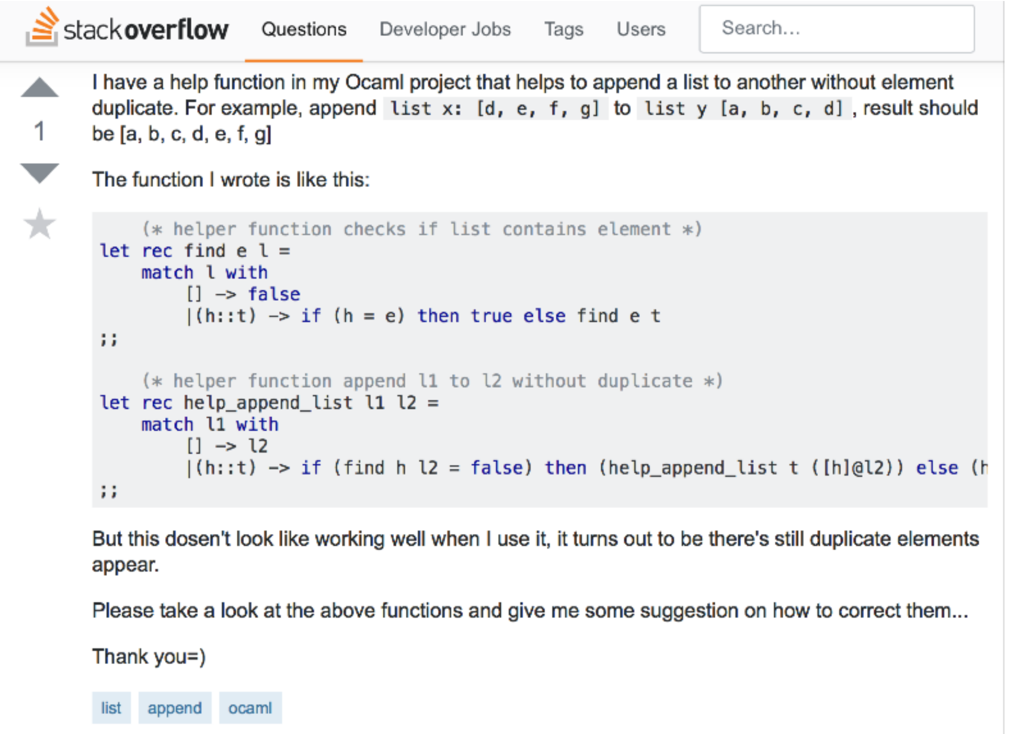
Examples

Q) Append lists without duplicates

```
append_list ['d';'e';'f';'g'] ['a';'b';'c';'d']
= ['a'; 'b'; 'c'; 'd'; 'e'; 'f'; ‘g’]

append_list [1;3;5;4;3] [3;5;6;6;4] = [3; 5; 6; 4; 1]
```

```
let rec find e l =
  match l with
  | [] -> false
  | h::t -> if h = e then true else find e t

let rec help_append_list l1 l2 =
  match l1 with
  | [] -> l2
  | h::t ->
    if find h l2 = false then help_append_list t (l2@[h])
    else help_append_list t l2

let append_list x y = help_append_list x y
```

```
(help_append_list y [])
```



I have a help function in my Ocaml project that helps to append a list to another without element duplicate. For example, append `list x: [d, e, f, g]` to `list y [a, b, c, d]`, result should be [a, b, c, d, e, f, g]

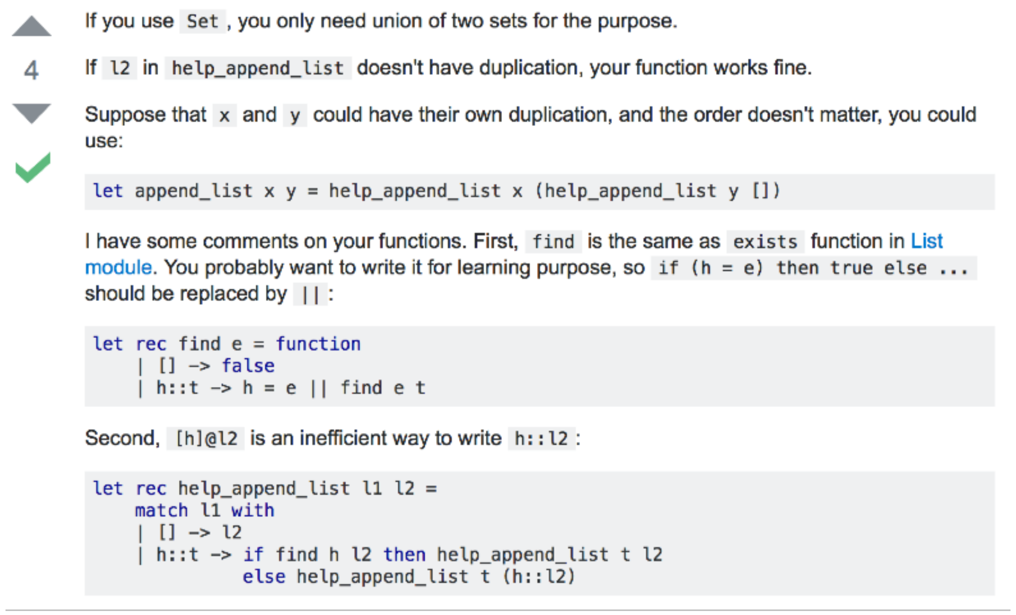
The function I wrote is like this:

```
    (* helper function checks if list contains element *)
let rec find e l =
    match l with
        [] -> false
        |(h::t) -> if (h = e) then true else find e t
;;

    (* helper function append l1 to l2 without duplicate *)
let rec help_append_list l1 l2 =
    match l1 with
        [] -> l2
        |(h::t) -> if (find h l2 = false) then (help_append_list t ([h]@l2)) else (h
;;
```

But this dosen't look like working well when I use it, it turns out to be there's still duplicate elements appear.

Please take a look at the above functions and give me some suggestion on how to correct them...

Thank you=)



If you use `Set`, you only need union of two sets for the purpose.

If `l2` in `help_append_list` doesn't have duplication, your function works fine.

Suppose that `x` and `y` could have their own duplication, and the order doesn't matter, you could use:

```
let append_list x y = help_append_list x (help_append_list y [])
```

I have some comments on your functions. First, `find` is the same as `exists` function in List module. You probably want to write it for learning purpose, so `if (h = e) then true else ...` should be replaced by `||`:

```
let rec find e = function
    | [] -> false
    | h::t -> h = e || find e t
```

Second, `[h]@l2` is an inefficient way to write `h::l2`:

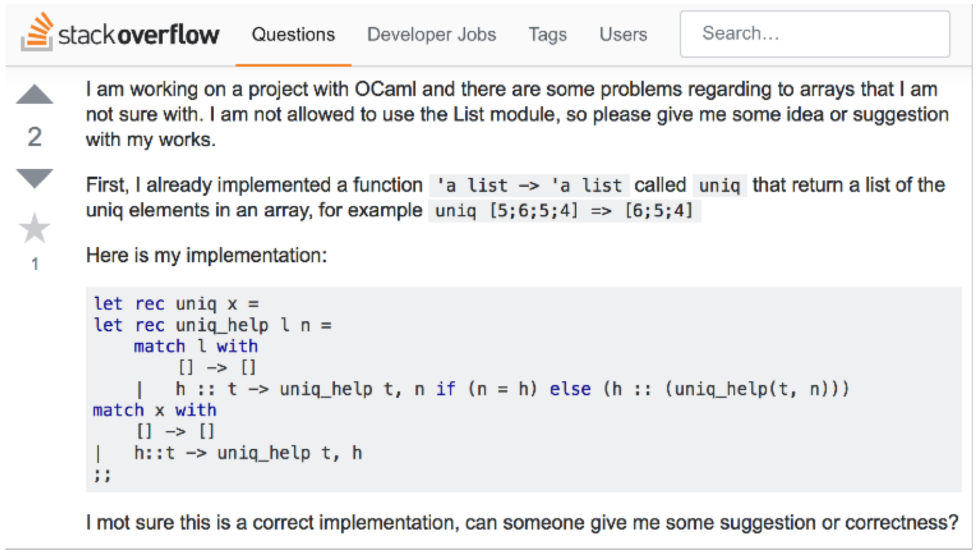
```
let rec help_append_list l1 l2 =
    match l1 with
    | [] -> l2
    | h::t -> if find h l2 then help_append_list t l2
              else help_append_list t (h::l2)
```

Examples

Q) Find unique elements

```
uniq [5;6;5;4] = [5;6;4]
uniq [3;5;7;5;7;4;8] = [3;5;7;4;8]
```

```
let rec uniq_help : int list -> int -> int list
= fun l n ->
  match l with
  | [] -> []
  | h::t -> if n = h then uniq_help t n
            else h::(uniq_help t n)

let rec uniq : int list -> int list
= fun x ->
  match x with
  | [] -> []
  | hd::tl -> uniq_help tl hd
```



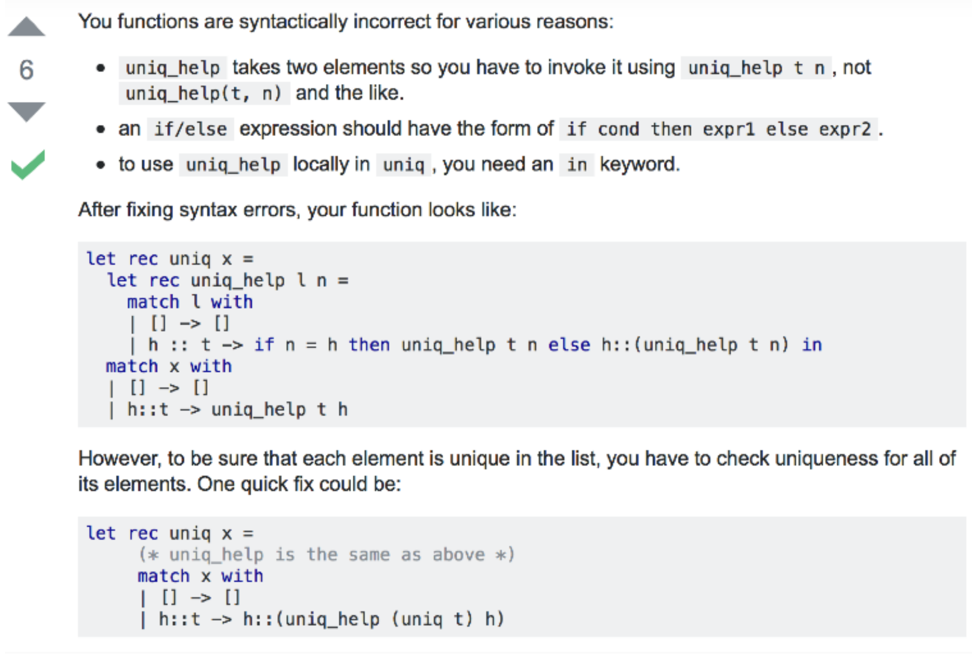
I am working on a project with OCaml and there are some problems regarding to arrays that I am not sure with. I am not allowed to use the List module, so please give me some idea or suggestion with my works.

First, I already implemented a function `'a list -> 'a list` called `uniq` that return a list of the uniq elements in an array, for example `uniq [5;6;5;4] => [6;5;4]`

Here is my implementation:

```
let rec uniq x =
let rec uniq_help l n =
    match l with
        [] -> []
    |   h :: t -> uniq_help t, n if (n = h) else (h :: (uniq_help(t, n)))
match x with
    [] -> []
|   h::t -> uniq_help t, h
;;
```

I mot sure this is a correct implementation, can someone give me some suggestion or correctness?



You functions are syntactically incorrect for various reasons:

- `uniq_help` takes two elements so you have to invoke it using `uniq_help t n`, not `uniq_help(t, n)` and the like.
- an `if/else` expression should have the form of `if cond then expr1 else expr2`.
- to use `uniq_help` locally in `uniq`, you need an `in` keyword.

After fixing syntax errors, your function looks like:

```
let rec uniq x =
  let rec uniq_help l n =
    match l with
    | [] -> []
    | h :: t -> if n = h then uniq_help t n else h::(uniq_help t n) in
  match x with
  | [] -> []
  | h::t -> uniq_help t h
```

However, to be sure that each element is unique in the list, you have to check uniqueness for all of its elements. One quick fix could be:

```
let rec uniq x =
     (* uniq_help is the same as above *)
     match x with
     | [] -> []
     | h::t -> h::(uniq_help (uniq t) h)
```

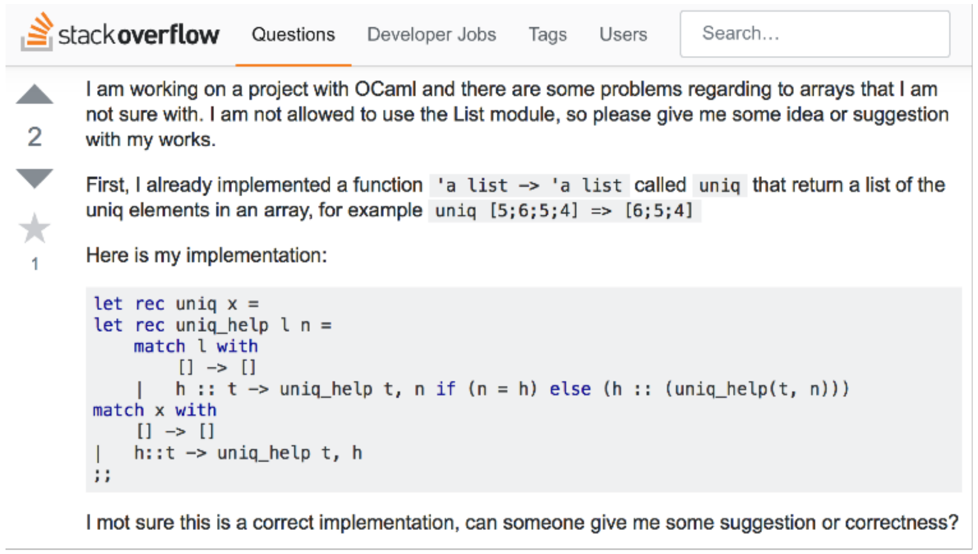
Examples

Q) Find unique elements

```
uniq [5;6;5;4] = [5;6;4]
uniq [3;5;7;5;7;4;8] = [3;5;7;4;8]
```

```
let rec uniq_help : int list -> int -> int list
= fun l n ->
  match l with
  | [] -> []
  | h::t -> if n = h then uniq_help t n
            else h::(uniq_help t n)

let rec uniq : int list -> int list
= fun x ->
  match x with
  | [] -> []
  | hd::tl -> uniq_help tl hd
```

`uniq_help tl hd` → `hd::(uniq_help (uniq tl) hd)`



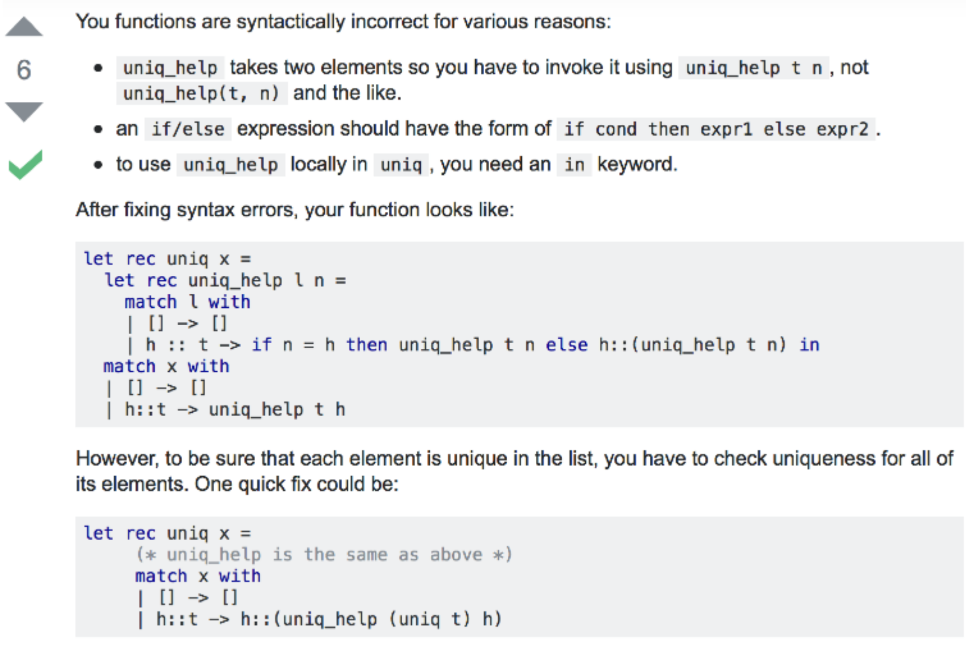
I am working on a project with OCaml and there are some problems regarding to arrays that I am not sure with. I am not allowed to use the List module, so please give me some idea or suggestion with my works.

First, I already implemented a function `'a list -> 'a list` called `uniq` that return a list of the uniq elements in an array, for example `uniq [5;6;5;4] => [6;5;4]`

Here is my implementation:

```
let rec uniq x =
let rec uniq_help l n =
    match l with
        [] -> []
    |   h :: t -> uniq_help t, n if (n = h) else (h :: (uniq_help(t, n)))
match x with
    [] -> []
|   h::t -> uniq_help t, h
;;
```

I mot sure this is a correct implementation, can someone give me some suggestion or correctness?



You functions are syntactically incorrect for various reasons:

- `uniq_help` takes two elements so you have to invoke it using `uniq_help t n`, not `uniq_help(t, n)` and the like.
- an `if/else` expression should have the form of `if cond then expr1 else expr2`.
- to use `uniq_help` locally in `uniq`, you need an `in` keyword.

After fixing syntax errors, your function looks like:

```
let rec uniq x =
  let rec uniq_help l n =
    match l with
    | [] -> []
    | h :: t -> if n = h then uniq_help t n else h::(uniq_help t n) in
  match x with
  | [] -> []
  | h::t -> uniq_help t h
```

However, to be sure that each element is unique in the list, you have to check uniqueness for all of its elements. One quick fix could be:

```
let rec uniq x =
     (* uniq_help is the same as above *)
     match x with
     | [] -> []
     | h::t -> h::(uniq_help (uniq t) h)
```

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,'17a,'17b,'18a,'18b,'19),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