

머신러닝을 활용한 소프트웨어 품질 검증 방안 연구

오학주

고려대학교

Apr 18, 2019 @국가보안기술연구소

연구 목표

- 머신러닝을 활용한 소프트웨어 테스팅 방안 연구
 - 대상 기술: 기호 실행 및 콘콜릭 테스팅
 - 코드 커버리지 및 오류 탐지율을 높일 수 있는 머신러닝 기법

기호 실행 (Symbolic Execution)

- A program analysis technique that executes a program with symbolic – rather than concrete – input values.
- Popular for finding software bugs and vulnerabilities: e.g.,
 - ▶ In Microsoft, 30% of bugs are discovered by symbolic execution.
 - ▶ Symbolic execution is the key technique used in DARPA Cyber Grand Challenge.
- Symbolic execution tools:
 - ▶ Stanford: KLEE
 - ▶ NASA: PathFinder
 - ▶ Microsoft: SAGE
 - ▶ UC Berkeley: CUTE
 - ▶ EPFL: S²E

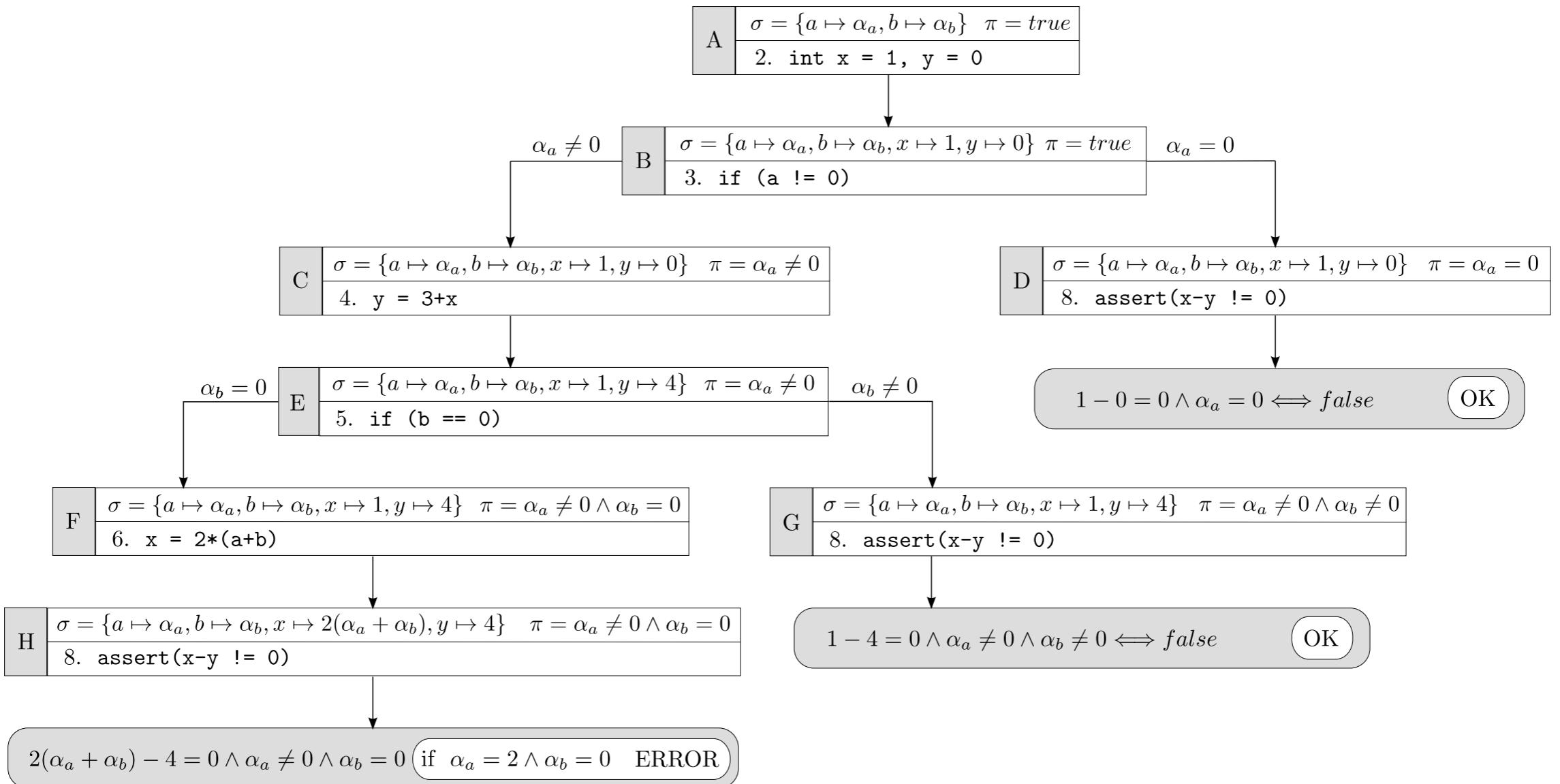
기호 실행 (Symbolic Execution)

```
1. void foobar(int a, int b) {  
2.     int x = 1, y = 0;  
3.     if (a != 0) {  
4.         y = 3+x;  
5.         if (b == 0)  
6.             x = 2*(a+b);  
7.     }  
8.     assert(x-y != 0);  
9. }
```

The goal is to find the inputs that make the assertion fail.

- Random testing with concrete values unlikely generate the inputs.
- Symbolic execution overcomes the limitation of random testing by reasoning on *classes of inputs*, rather than single input values.

Symbolic Execution Tree



콘콜릭 테스팅

An approach that combines concrete and symbolic execution to address the limitations of symbolic execution.

- external calls
- constraint solving
- pointers

Approaches to concolic execution:

- Dynamic symbolic execution (e.g. DART, SAGE, KLEE)
- Selective symbolic execution (e.g. S²E)

콘콜릭 테스팅

```
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=22, y=7

Symbolic
State

x=a, y=β

true

1st iteration

콘콜릭 테스팅

```
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=22, y=7,
z=14

Symbolic
State

x=a, y=β, z=2*β
true

1st iteration

콘콜릭 테스팅

```
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=22, y=7,
z=14

1st iteration

Symbolic
State

x=a, y=β, z=2*β
2*β ≠ a

콘콜릭 테스팅

```
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

Solve: $2^*\beta = a$
Solution: $a=2, \beta=1$

$x=22, y=7,$
 $z=14$

Symbolic
State

$x=a, y=\beta, z=2^*\beta$
 $2^*\beta \neq a$

1st iteration

기계 학습 적용 대상

- 패스 선별 휴리스틱 (기호 실행)
- 입력 템플릿 선별 휴리스틱 (기호 실행, 콘콜릭 테스팅)
- 브랜치 선별 휴리스틱 (콘콜릭 테스팅)

패스 선별 휴리스틱

Since enumerating all paths of a program can be prohibitively expensive, symbolic execution prioritizes the most promising paths. Several strategies for selecting the next path to be explored have been proposed: e.g.,

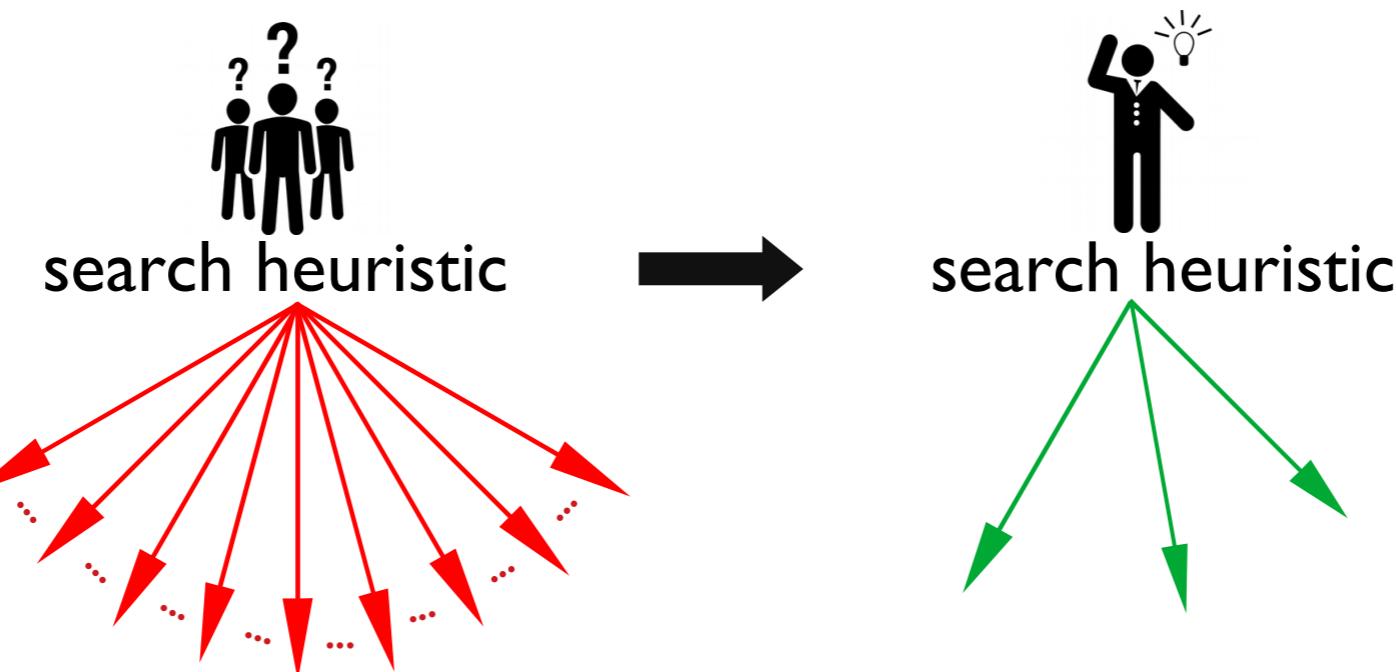
- Depth-first search
- Breadth-first search
- Random path selection
- Coverage optimize search
- Subpath-guided search
- Buggy-path first search
- ...

목표: 프로그램마다 최적의 패스 선별 휴리스틱을 자동 학습

입력 템플릿 선별 허리스틱

- 기호로 취급할 입력을 선별하여 탐색 공간을 줄임

α_1	α_2	α_3	α_4	α_5	α_6	α_7	α_8
→							
α_1	α_2	'-	'S'	α_5	'A'	α_7	'L'



목표: 기호를 대체할 최적의 입력값을 자동 학습

브랜치 선별 휴리스틱

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



연구 내용

- 기계학습을 이용한 패스 선별 휴리스틱 자동 생성
- 기계학습을 이용한 입력 템플릿 자동 생성
- 기계학습을 이용한 브랜치 선별 휴리스틱 자동 생성

연구 보고서 목차(안)

1. 연구개요
2. SW 테스팅 기법
 - (1) 기호 실행
 - (2) 콘콜릭 테스팅
3. SW 테스팅을 위한 기계 학습
 - (1) 기계학습을 이용한 패스 선별 휴리스틱 자동 생성
 - (2) 기계학습을 이용한 입력 템플릿 자동 생성
 - (3) 기계학습을 이용한 브랜치 선별 휴리스틱 자동 생성
4. 결론

연구 일정 및 결과물

- 기간: 2019.04.01 ~ 2019.10.31 (7개월)
- 연구결과물

결과물 유형	결과물 명칭	규격	수량	제출예정일
중간 보고서	머신러닝을 활용한 소프트웨어 품질 검증 방안 연구 중간 보고서	20 ~ 30 pages	5부	2019.07 .31
최종 보고서	머신러닝을 활용한 소프트웨어 품질 검증 방안 연구 최종 보고서	30 ~ 40 pages	5부	2019.10 .31

AI 기반 소프트웨어 결함 검출 기술에 관한 연구

오학주

고려대학교

Apr 18, 2019 @국가보안기술연구소

연구 목표

- AI 기반 소프트웨어 결함 검출 기술에 관한 연구
 - AI를 이용한 고성능 결함 검출을 위한 정적 분석 기술
 - AI를 이용한 결함 자동 수정을 위한 기술

연구 범위



AI 기반 정적 분석

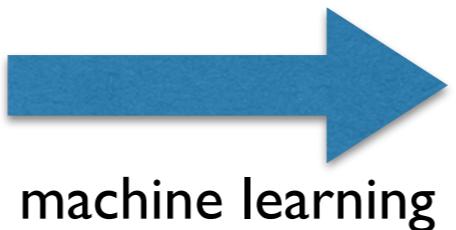
- 전통적 로직 기반 정적 분석의 한계



WALA
T. J. WATSON LIBRARIES FOR ANALYSIS

DOOP TAJS

- AI를 이용하여 정적 분석의 현재 한계를 극복



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

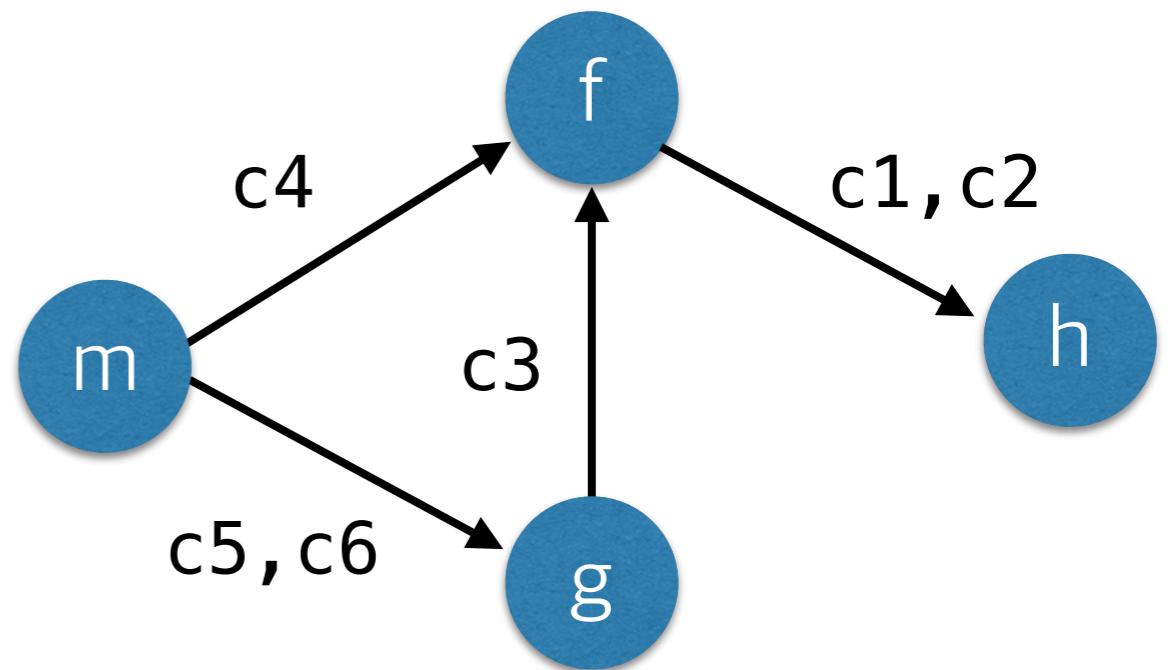
Example: Context-Sensitivity

```
int h(n) {ret n;}\n\nvoid f(a) {\nc1:  x = h(a);\n      assert(x > 0); // Query ← holds always\nc2:  y = h(input());\n}\n\n\nc3: void g() {f(8);}\n\nvoid m() {\nc4:  f(4);\nc5:  g();\nc6:  g();\n}
```

Context-Insensitive Analysis

- Merge calling contexts into single abstract context

```
int h(n) {ret n;}\n\nvoid f(a) {\nc1:  x = h(a);\n      assert(x > 0);\nc2:  y = h(input());\n}\n\nc3: void g() {f(8);}\n\nvoid m() {\nc4:  f(4);\nc5:  g();\nc6:  g();\n}
```

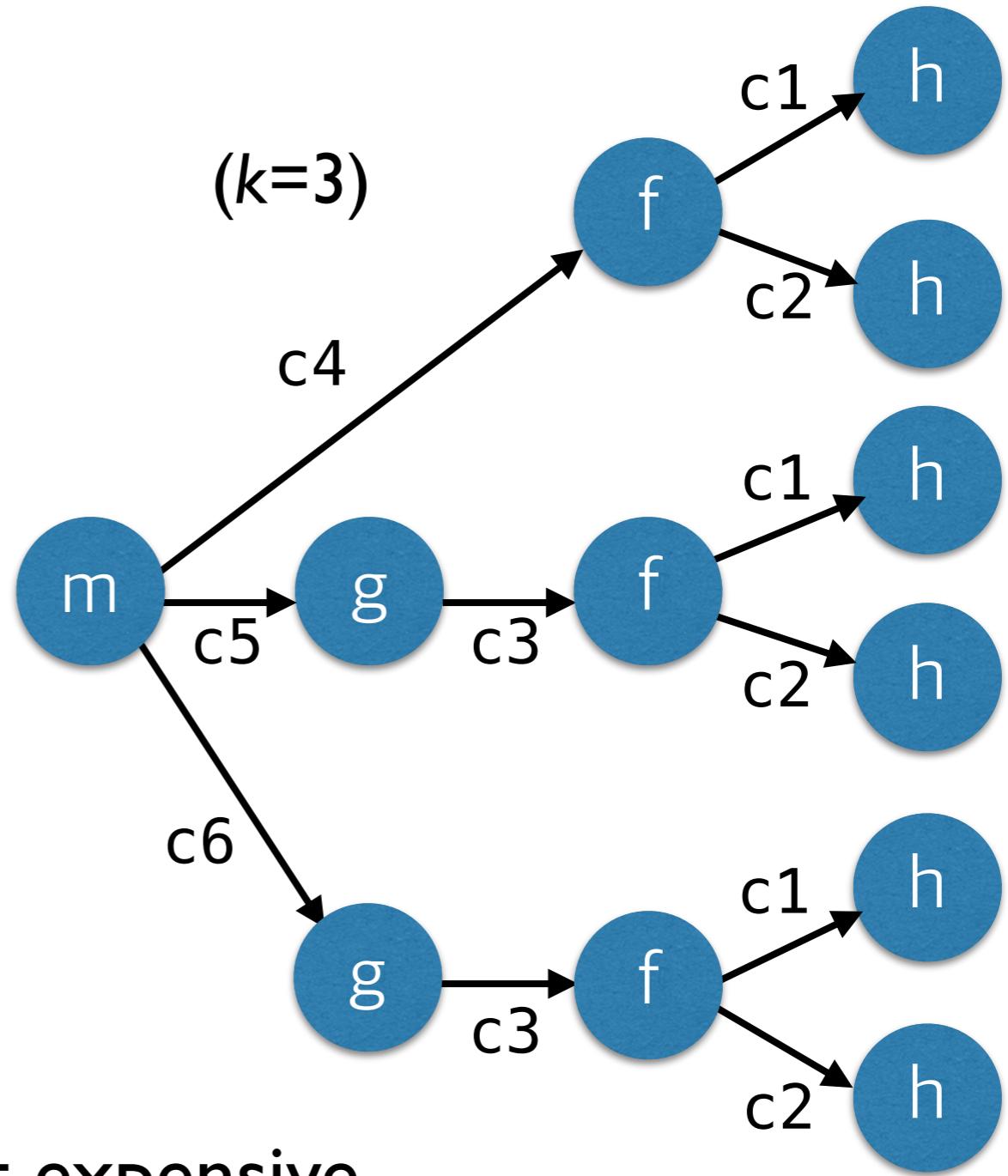


cheap but imprecise

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();  
}
```



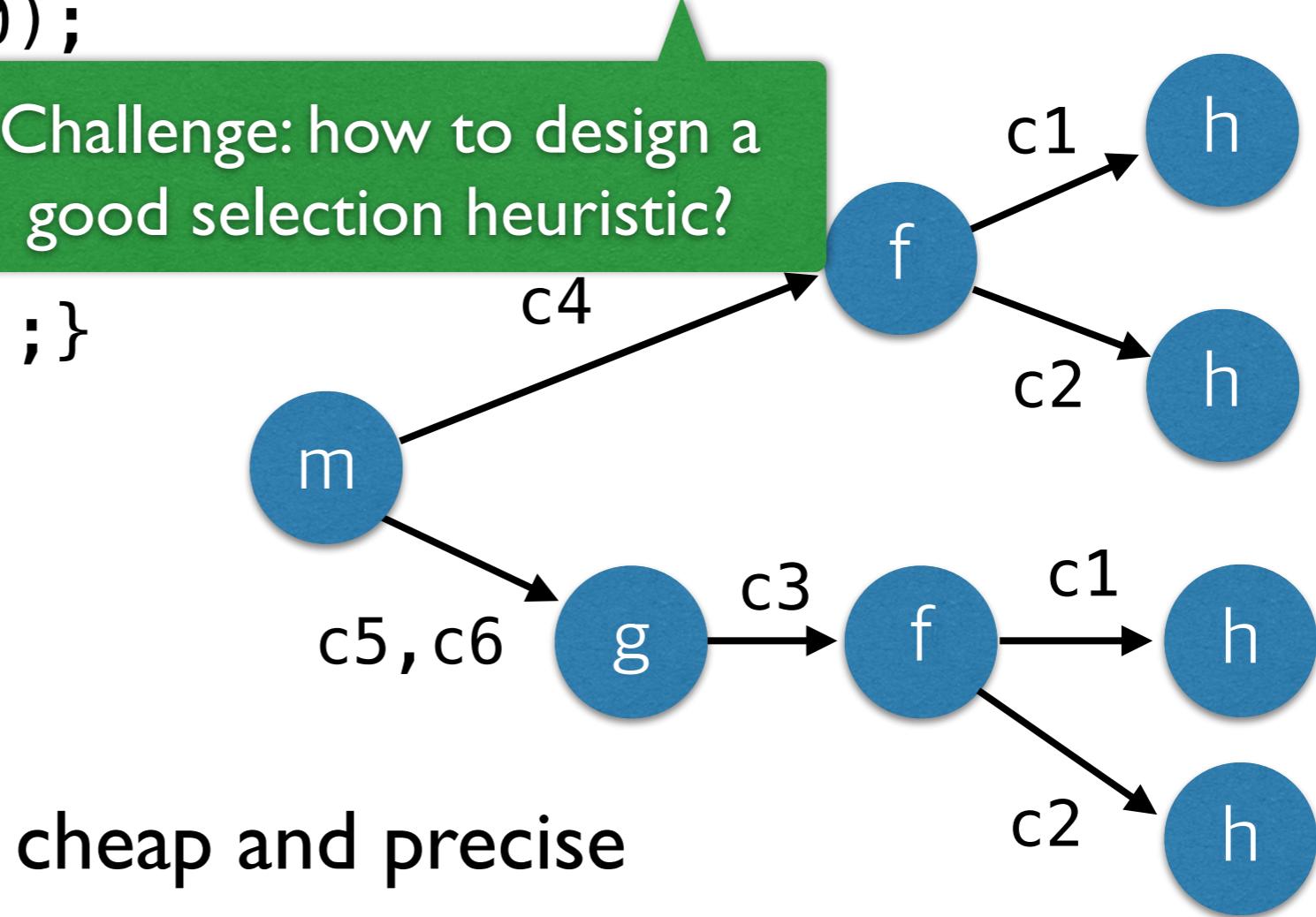
Selective Context-Sensitivity

- 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();  
}
```

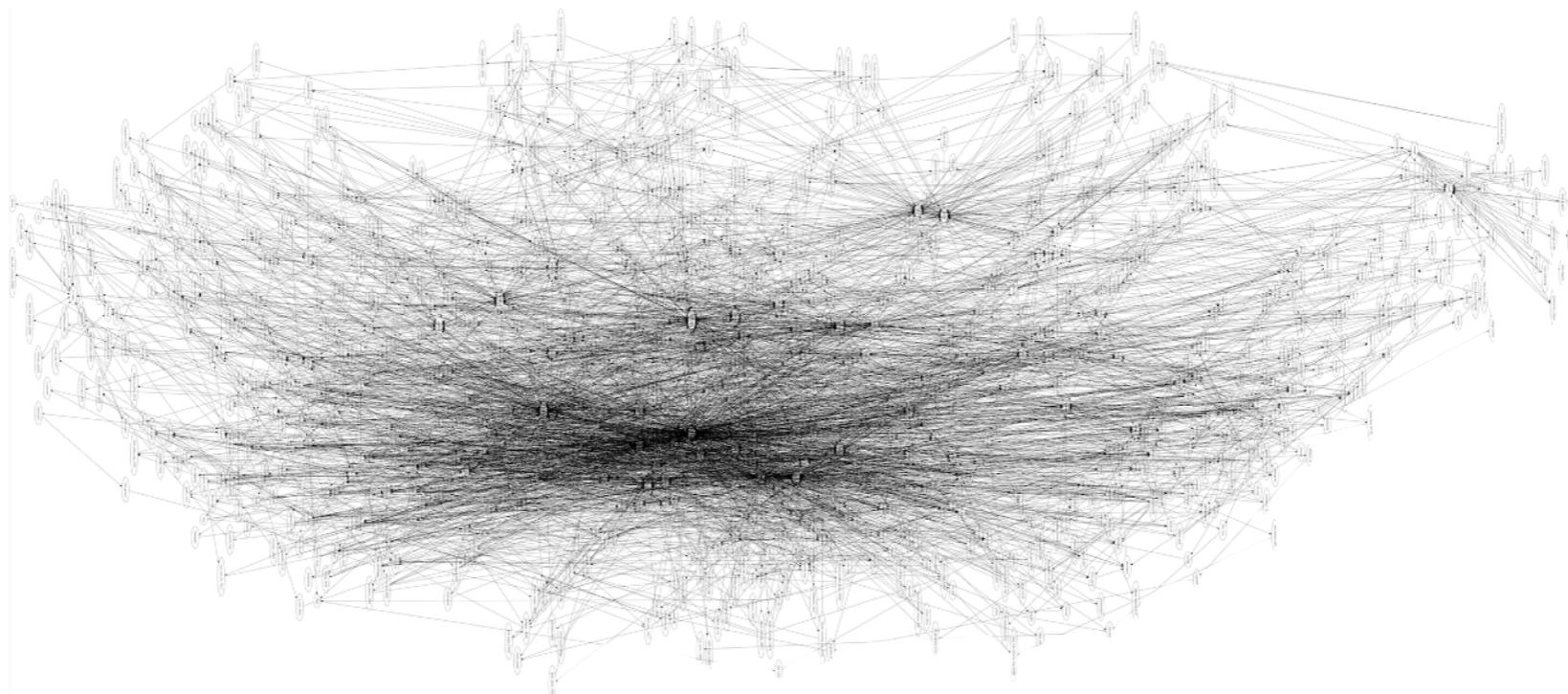
Apply 2-ctx-sens: {h}
Apply 1-ctx-sens: {f}
Apply 0-ctx-sens: {g, m}

Challenge: how to design a
good selection heuristic?



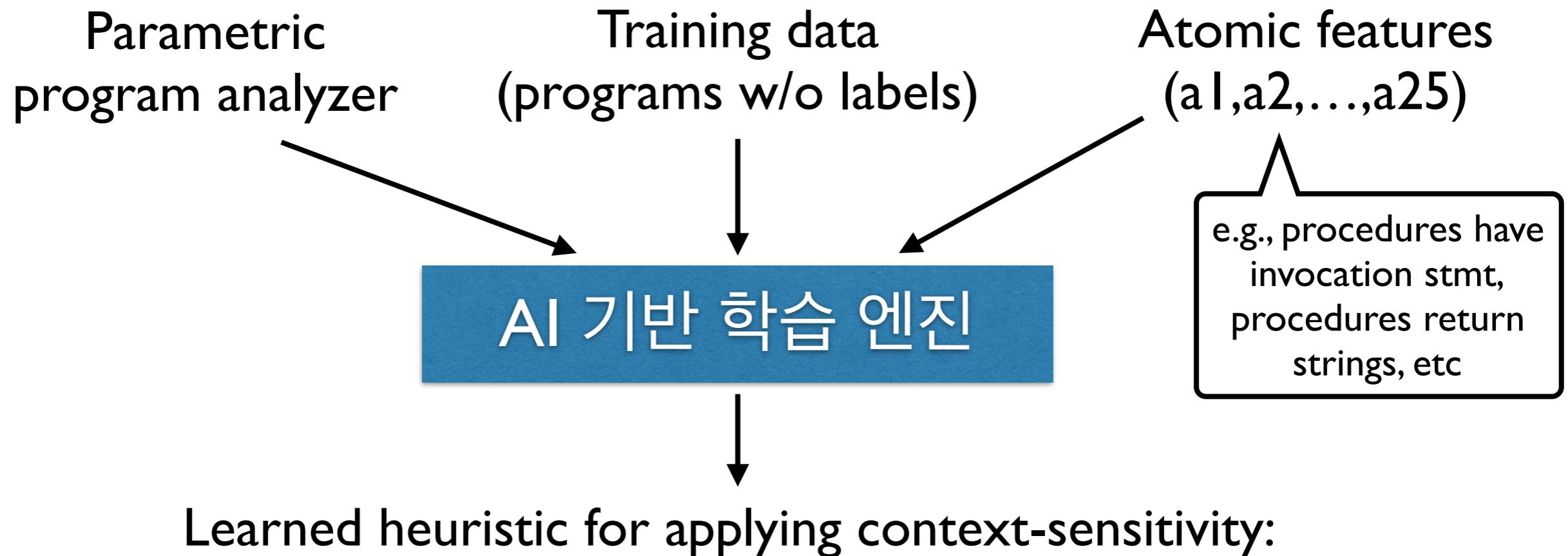
Hard Search Problem

- Intractably large and sparse search space, if not infinite
 - e.g., S^k choices where $S = 2^{|\text{Proc}|}$ 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

AI 기반 정적 분석



f2: procedures to apply 2-context-sensitivity

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

f1: procedures to apply 1-context-sensitivity

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

연구 내용

- 고성능 정적 분석을 위한 AI 기반 엔진
 - 학습 모델
 - 학습 알고리즘
 - 특질(feature) 엔지니어링

SW 오류 자동 수정의 필요성

- 소프트웨어 개발에서 디버깅은 전체 시간의 절반을 차지
 - 상용 소프트웨어 오류 수정에 평균 200일 소요¹⁾
 - 오류/취약점은 해마다 증가 개수: e.g., CVE 등록수 4,000('10년), 6,000('15년)
- 다른 개발 단계에 비해 자동화된 도구 지원이 가장 적음
 - cf) 소프트웨어 오류 탐지 분야는 지난 30여년간 눈부신 발전
 - 개발자에 전적으로 의존할수 밖에 없지만 가장 어렵고 부담스러운 단계

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

실제 사례 (Linux Kernel)

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

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

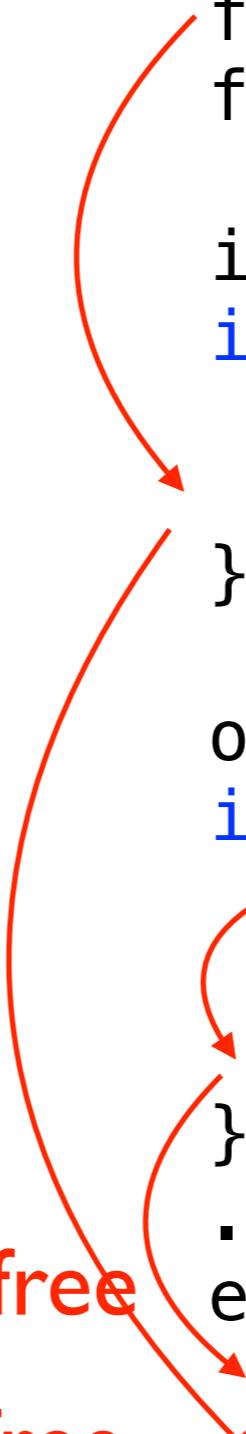
    goto err;
}

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

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

double-free

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>

master ↗ v4.15-rc1 ... v2.6.24-rc1

 Oliver Neukum committed with gregkh on 18 Sep 2007

1 par

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)

수동 디버깅의 문제 2:
고치는 과정에서 새로운 오류가 발생

memory leak

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;
```

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>

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

 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;
```

메모리 오류 자동 수정 기술

```
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)
=> 추가적인 리뷰 불필요.

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

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

```
    goto err;
}
```

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

```
    goto err;
}
```

```
... // use in, out
err:
```

```
    free(in);
    free(out);
    return;
```

연구 내용

- AI기반 메모리 오류 자동 수정 기술
 - 오류 자동 수정 문제를 탐색 문제로 인코딩
 - 효과적인 인코딩을 위한 프로그램 분석 기법

연구 보고서 목차(안)

1. 연구개요
2. 연구 동향
 - (1) AI기반 SW 결함 검출 기술 동향
 - (2) AI기반 SW 결함 수정 기술 동향
3. SW 결함 검출을 위한 AI 기법
 - (1) 정적 분석 성능 향상을 위한 기법
 - (2) 메모리 오류 자동 수정을 위한 기법
4. 결론

연구 일정 및 결과물

- 기간: 2019.04.01 ~ 2019.10.31 (7개월)
- 연구결과물

결과물 유형	결과물 명칭	규격	수량	제출예정일
중간 보고서	AI 기반 소프트웨어 결합 검출 기술 중간 보고서	20 ~ 30 pages	5부	2019.07.31
최종 보고서	AI 기반 소프트웨어 결합 검출 기술 최종 보고서	30 ~ 40 pages	5부	2019.10.31

Thank You!

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



<http://prl.korea.ac.kr>