Introduction to Static Analysis (Static SW Testing)

오학주
고려대학교 컴퓨터학과

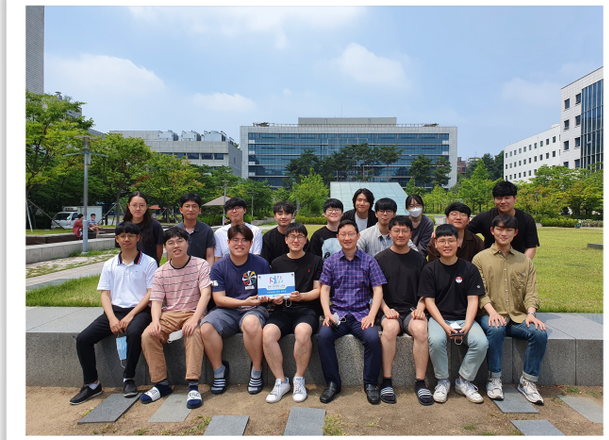
슬라이드

http://prl.korea.ac.kr/~pronto/home/talks/slides_samsung22.pdf

http://shorturl.at/pzPX7

소프트웨어 분석 연구실@Korea Univ.

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



- Publication: top-venues in PL, SE, and Security:
  - PL: POPL('22),PLDI('20),OOPSLA('15,'17a,'17b,'18a,'18b,'19,'20)
  - SE: ICSE('17,'18,'19,'20,'21'22a,'22b), FSE('18,'19,'20,'21), ASE('18), ISSTA('20)
  - Security: IEEE S&P('17,'20), USENIX Security('21)

http://prl.korea.ac.kr

소프트웨어 오류 문제

- 소프트웨어 오류는 사회 모든 영역에서 발생하는 추세

금융거래SW(2012)

자율주행SW(2017)

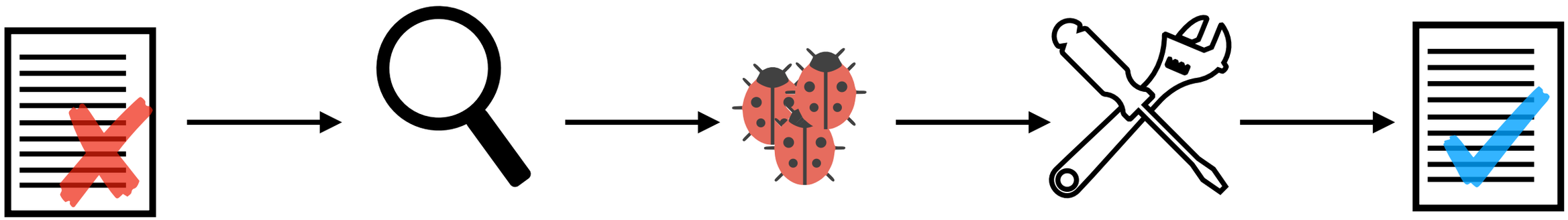
의료SW(2018)

블록체인SW(2020)

- 소프트웨어 결함으로 인한 사회경제적 비용은 연 1.7조 달러로 추정

Software fail watch (5th edition). 2017

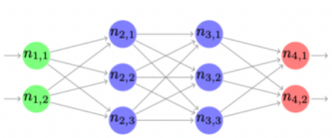
연구 분야: SW 오류 자동 검출 & 수정



오류 검출 기술

오류 수정 기술

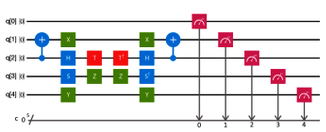
정적 분석



검증(SMT)

콘콜릭

기호 실행



퍼징

⋮

구문 오류

의미 오류

기능 오류

보안 오류

정형 명세

⋮

정적 분석

기호 실행

코드 합성

코드 마이닝

기계 학습

⋮

강의 목표 및 내용

- Part 1 프로그램 분석 개괄
- Part 2: 정적 분석 적용 사례
- Part 3: 정적 분석 이론
- Part 4: 정적 분석 구현

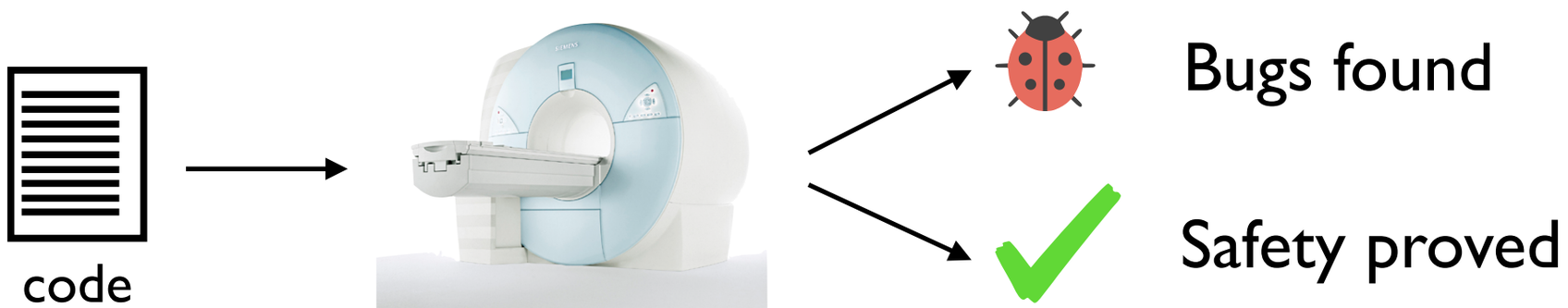
Part 1. 프로그램 분석 개괄

Part 2. 정적 분석 적용 사례

Part 3. 정적 분석 원리

Part 4. 정적 분석 구현

프로그램 분석

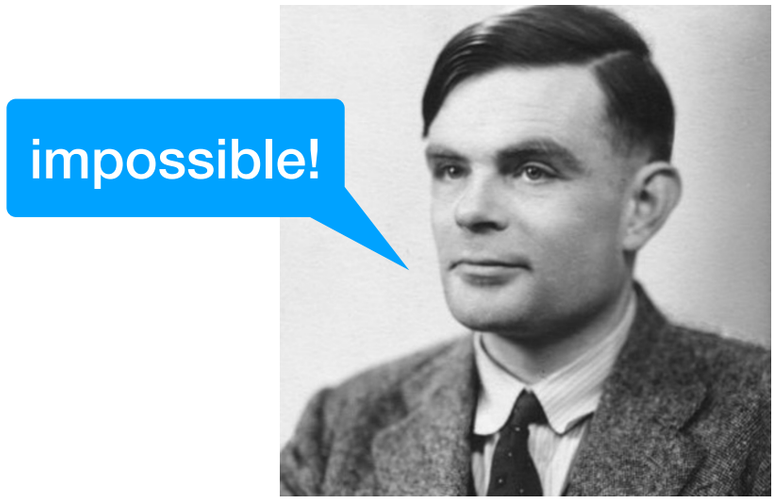


- 소프트웨어의 실행 성질을 엄밀히 확인하는 기술
  - 정적 분석: 실행 전 확인 (요약 해석, 모델 체킹 등)
  - 동적 분석: 실행 중 확인 (퍼징, 기호 실행 등)
- 소프트웨어 산업에서 적극적으로 활용되기 시작

SAMSUNG facebook Google Microsoft

완벽한 프로그램 분석은 불가능

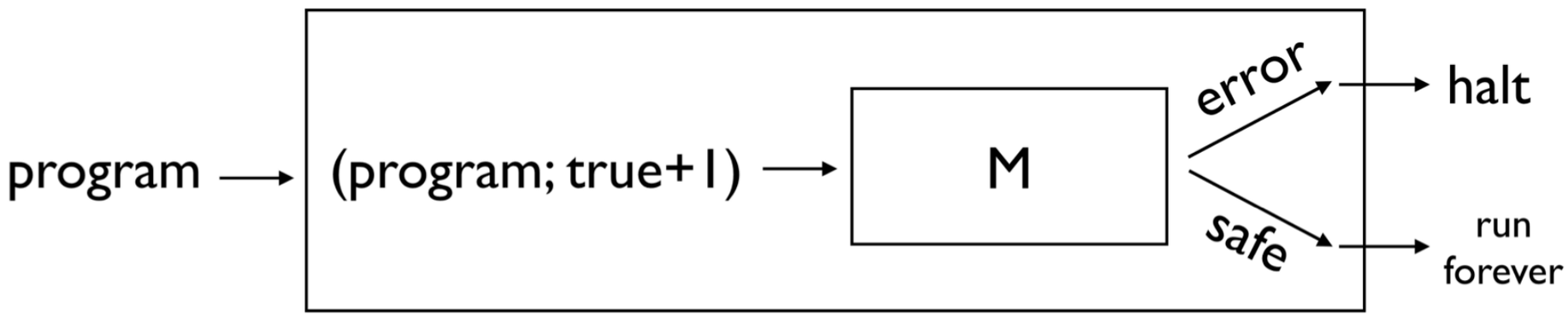
- Halting problem: 주어진 프로그램이 항상 종료하는가?



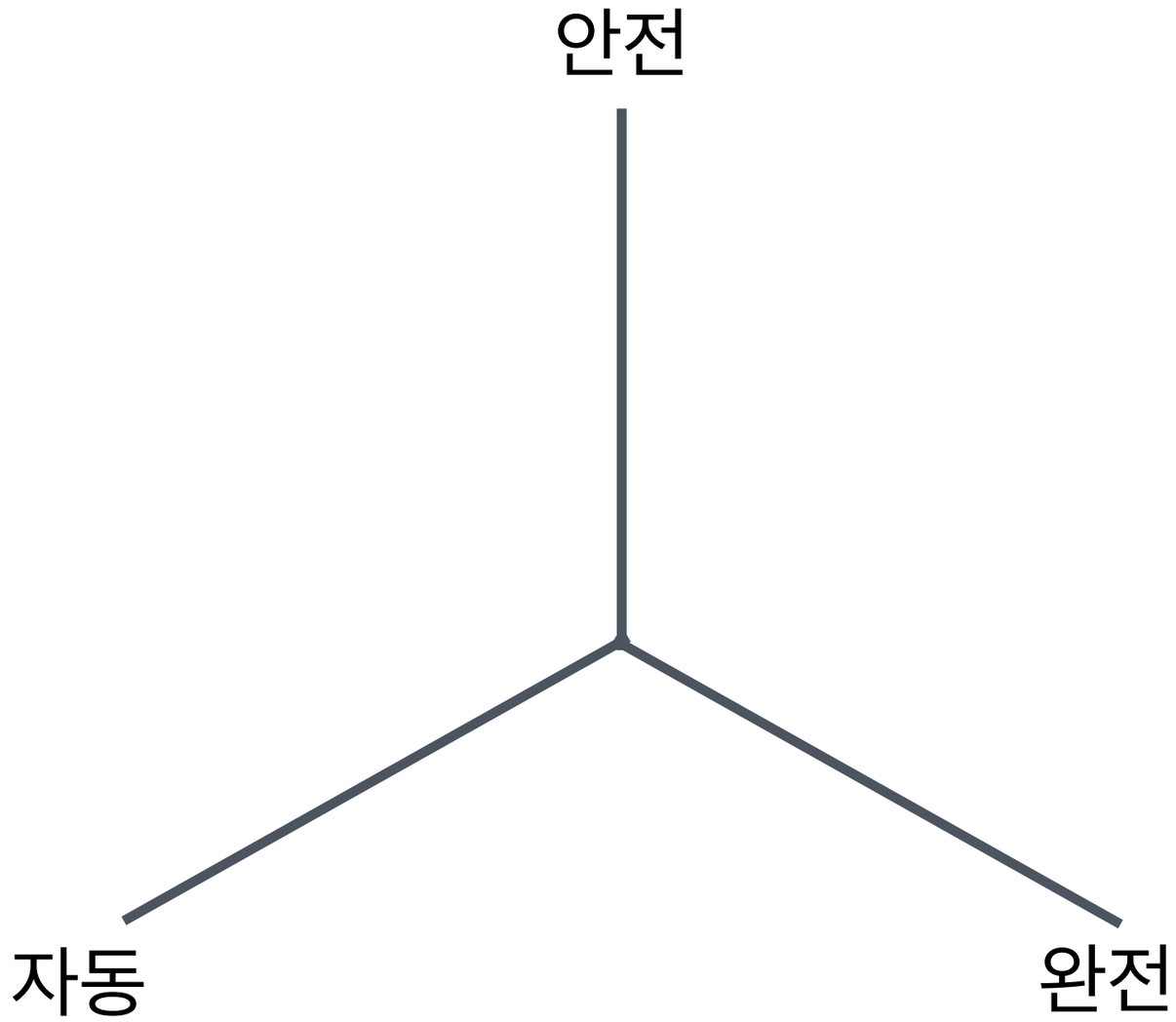
Alan Turing

```
while (symbol->type == Node_array_ref) {
    len += strlen(symbol->vname);
    n++;
    symbol = symbol->prev_array;
}
```

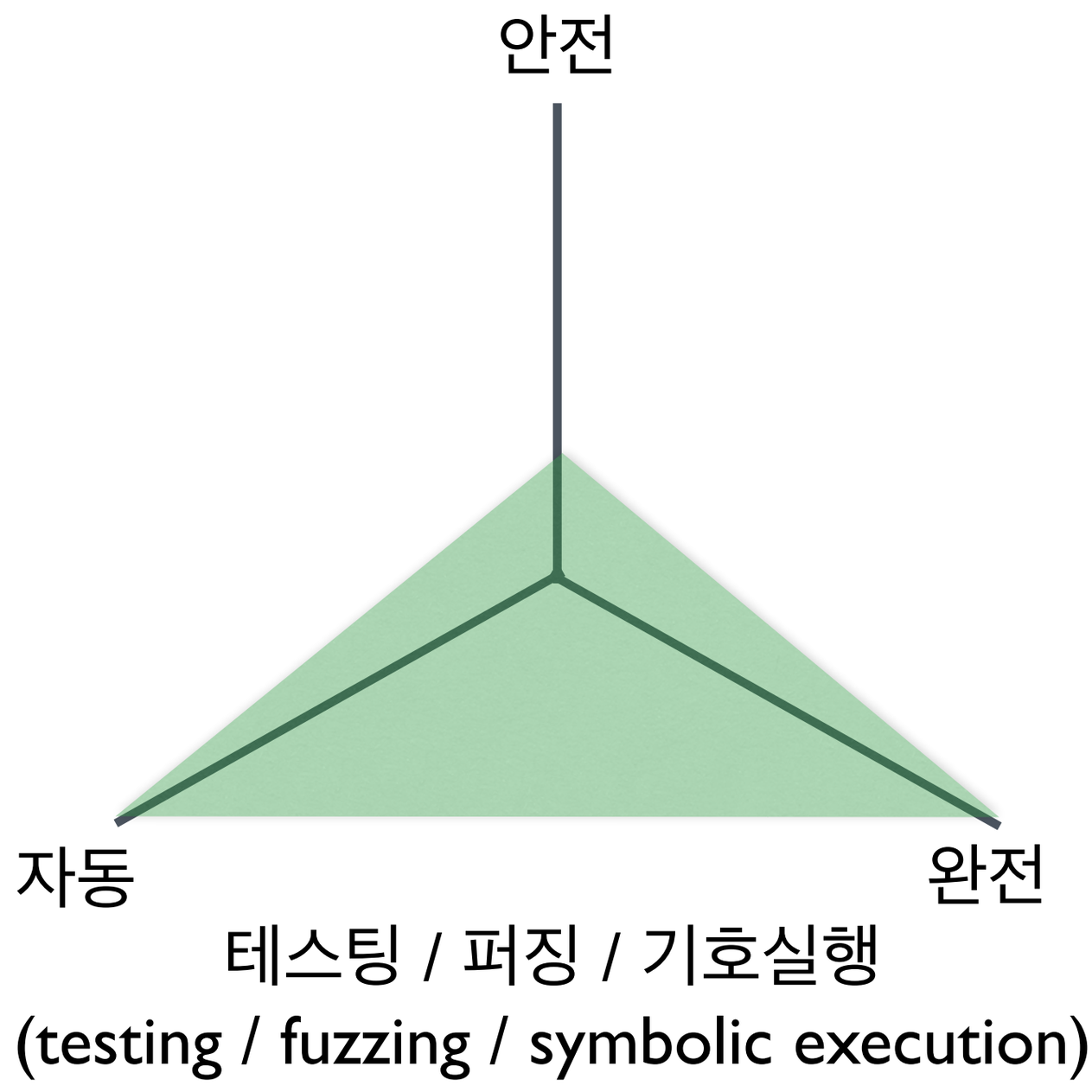
- 완벽한 프로그램 분석기 M이 존재한다면 Halting problem이 풀린다.



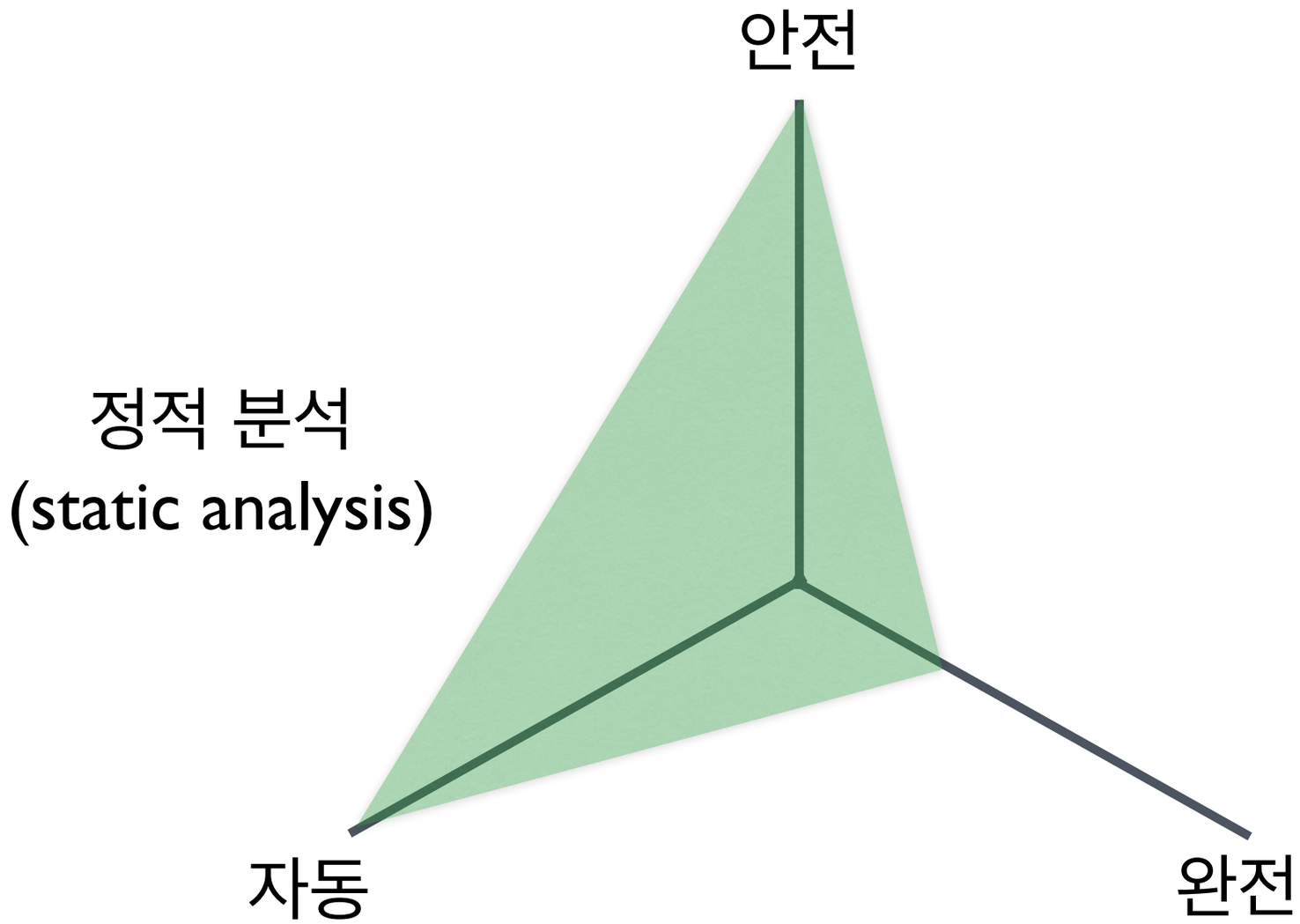
프로그램 분석 기법



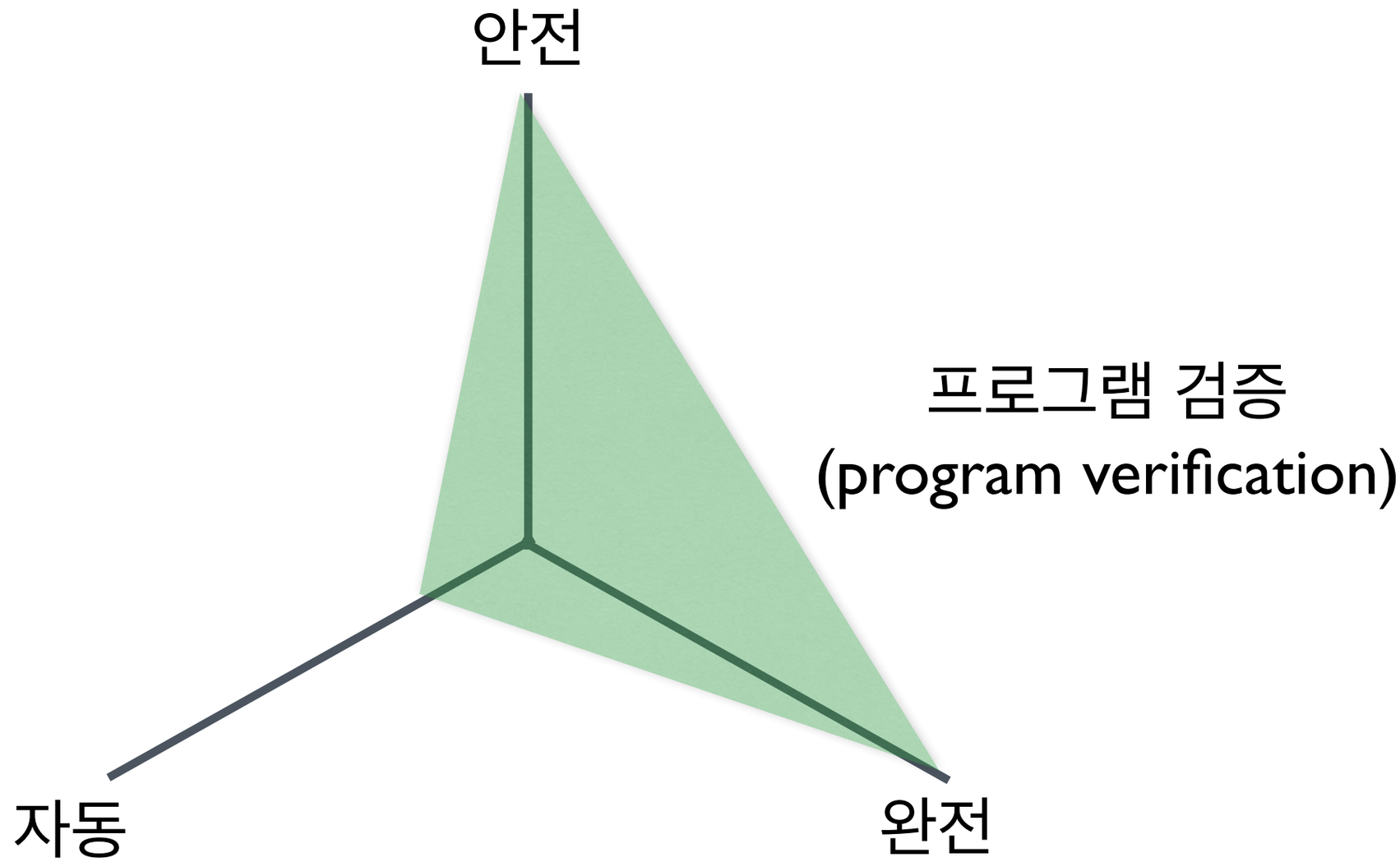
프로그램 분석 기법



프로그램 분석 기법



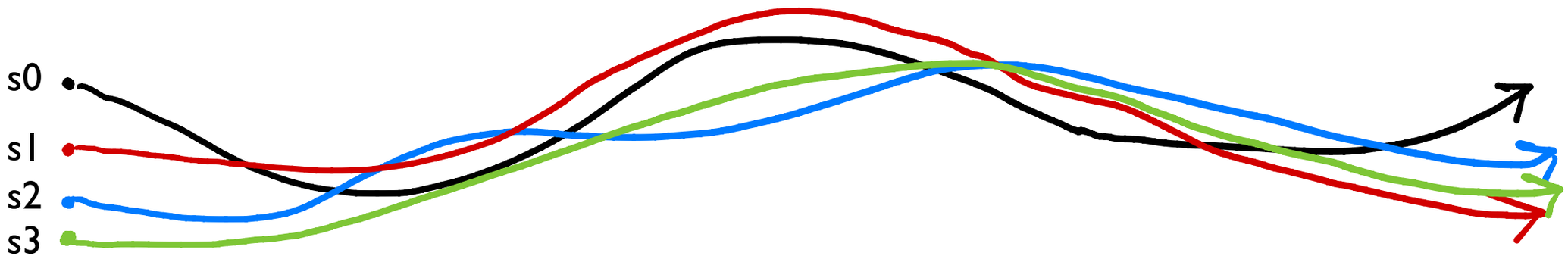
프로그램 분석 기법



테스팅 / 퍼징

- 프로그램의 개별 실행 경로들을 일일이 추적

error zone



error zone

테스팅 / 퍼징

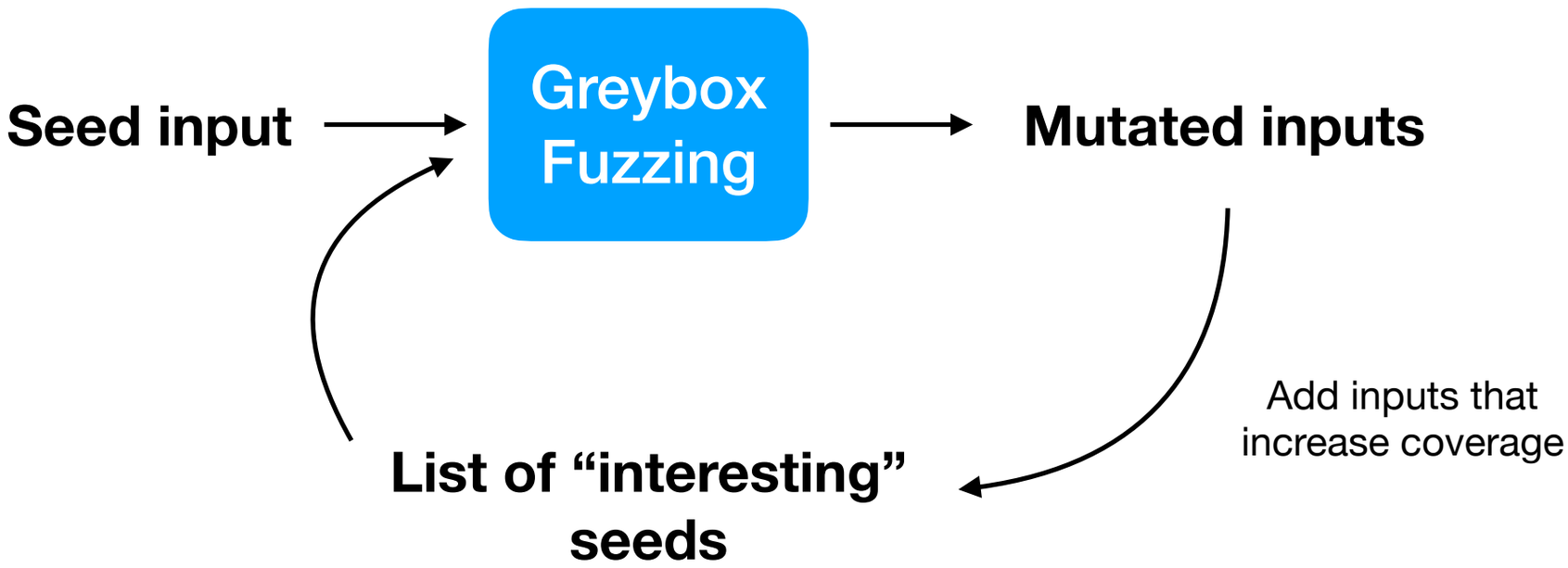
```
int double (int v) {
  return 2*v;
}

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

  if (z==x) {

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

1. Error-triggering test?

2. Probability of the error?
(assume 0 ≤ x,y ≤ 10,000)

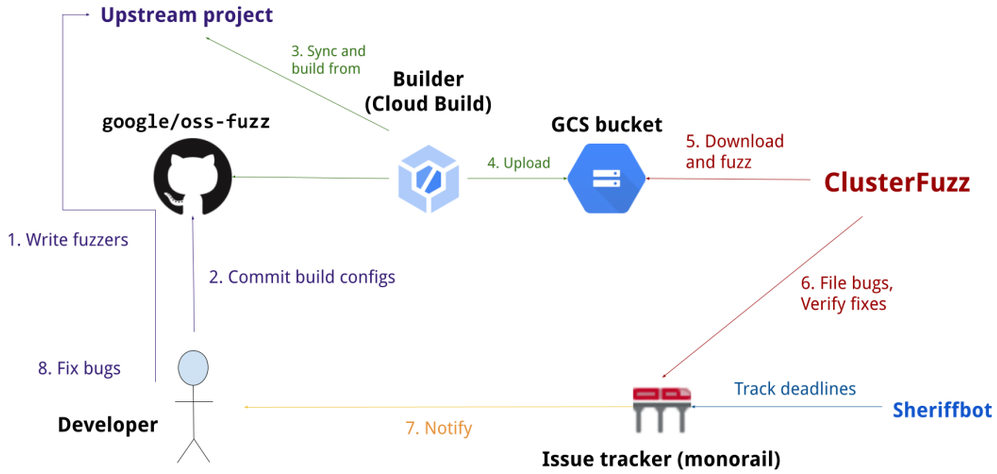
Types of Fuzzing

- Blackbox fuzzing
- Whitebox fuzzing
- Greybox fuzzing



산업체 적용 사례

- AFL (https://github.com/google/AFL)
- OSS-Fuzz (https://github.com/google/oss-fuzz)



Google OSS-Fuzz

DOI:10.1145/3363824

Reviewing software testing techniques for finding security vulnerabilities.

BY PATRICE GODEFROID

Fuzzing: Hack, Art, and Science

FUZZING, OR FUZZ TESTING, is the process of finding security vulnerabilities in input-parsing code by repeatedly testing the parser with modified, or fuzzed, inputs.[35] Since the early 2000s, fuzzing has become a mainstream practice in assessing software security. Thousands of security vulnerabilities have been found while fuzzing all kinds of software applications for processing documents, images, sounds, videos, network packets, Web pages, among others. These applications must deal with untrusted inputs encoded in complex data formats. For example, the Microsoft Windows operating system supports over 360 file formats and includes millions of lines of code just to handle all of these.

Most of the code to process such files and packets evolved over the last 20+ years. It is large, complex, and written in C/C++ for performance reasons. If an attacker could trigger a buffer-overflow bug in one of these applications, s/he could corrupt the memory of the application and possibly hijack its execution to run malicious code (elevation-of-privilege attack), or steal internal information (information-disclosure attack), or simply crash the application (denial-of-service attack).[9] Such attacks might be launched by tricking the victim into opening a single malicious document, image, or Web page. If you are reading this article on an electronic device, you are using a PDF and JPEG parser in order to see Figure 1.

Buffer-overflows are examples of security vulnerabilities: they are programming errors, or bugs, and typically triggered only in specific hard-to-find corner cases. In contrast, an exploit is a piece of code which triggers a security vulnerability and then takes advantage of it for malicious purposes. When exploitable, a security vulnerability is like an unintended backdoor in a software application that lets an attacker enter the victim's device.

There are approximately three main ways to detect security vulnerabilities in software.

Static program analyzers are tools that automatically inspect code and

» key insights

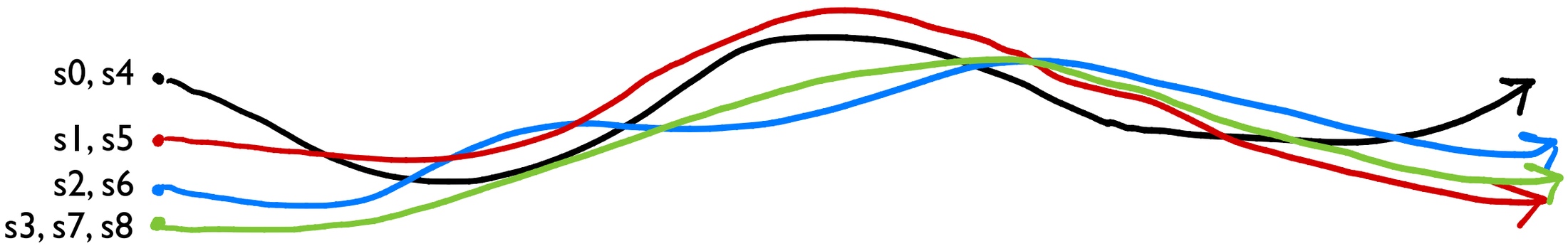
- **Fuzzing means automatic test generation and execution with the goal of finding security vulnerabilities.**
- **Over the last two decades, fuzzing has become a mainstay in software security. Thousands of security vulnerabilities in all kinds of software have been found using fuzzing.**
- **If you develop software that may process untrusted inputs and have never used fuzzing, you probably should.**

Microsoft

기호 실행 (Symbolic Execution)

- 동일한 실행 경로를 가지는 입력들을 한번에 실행

error zone



error zone

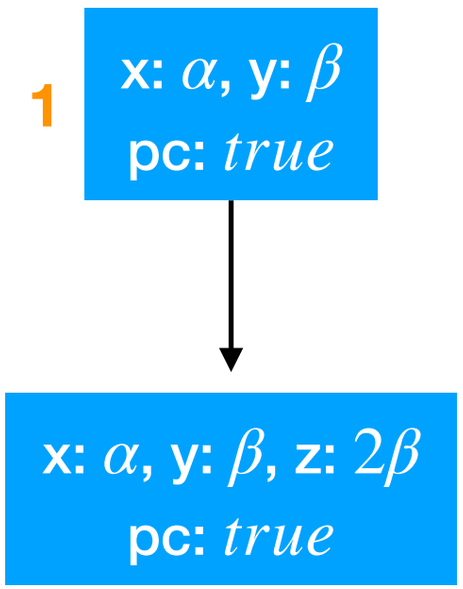
기호 실행 (Symbolic Execution)

1 x: α, y: β
pc: $true$

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

void testme(int x, int y) {
1
   z := double (y);
2
   if (z==x) {
    3
     if (x>y+10) {
      4 Error;
     } else { 5 …}
   }
6 }
```

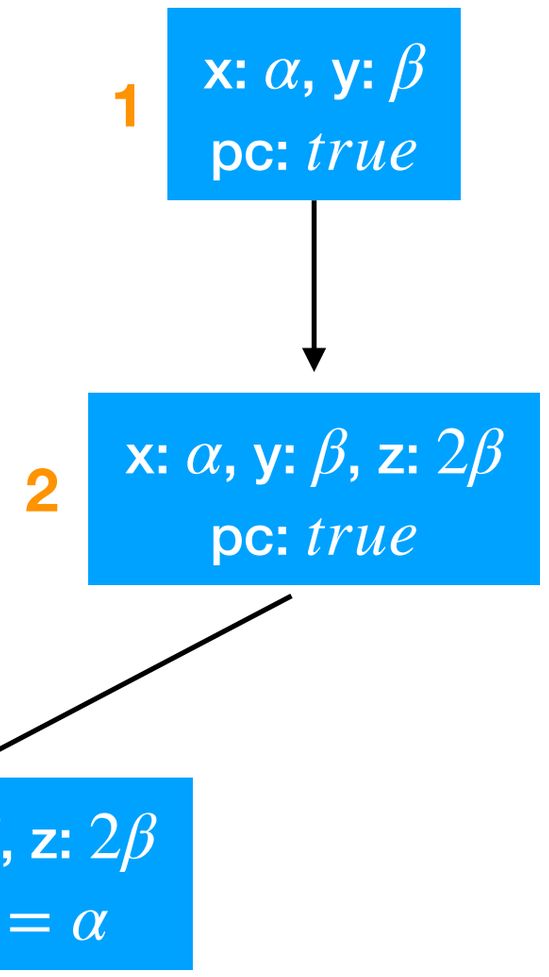
기호 실행 (Symbolic Execution)

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

void testme(int x, int y) {
1
  z := double (y);
2
  if (z==x) {
   3
    if (x>y+10) {
     4 Error;
    } else { 5 …}
  }
6 }
```



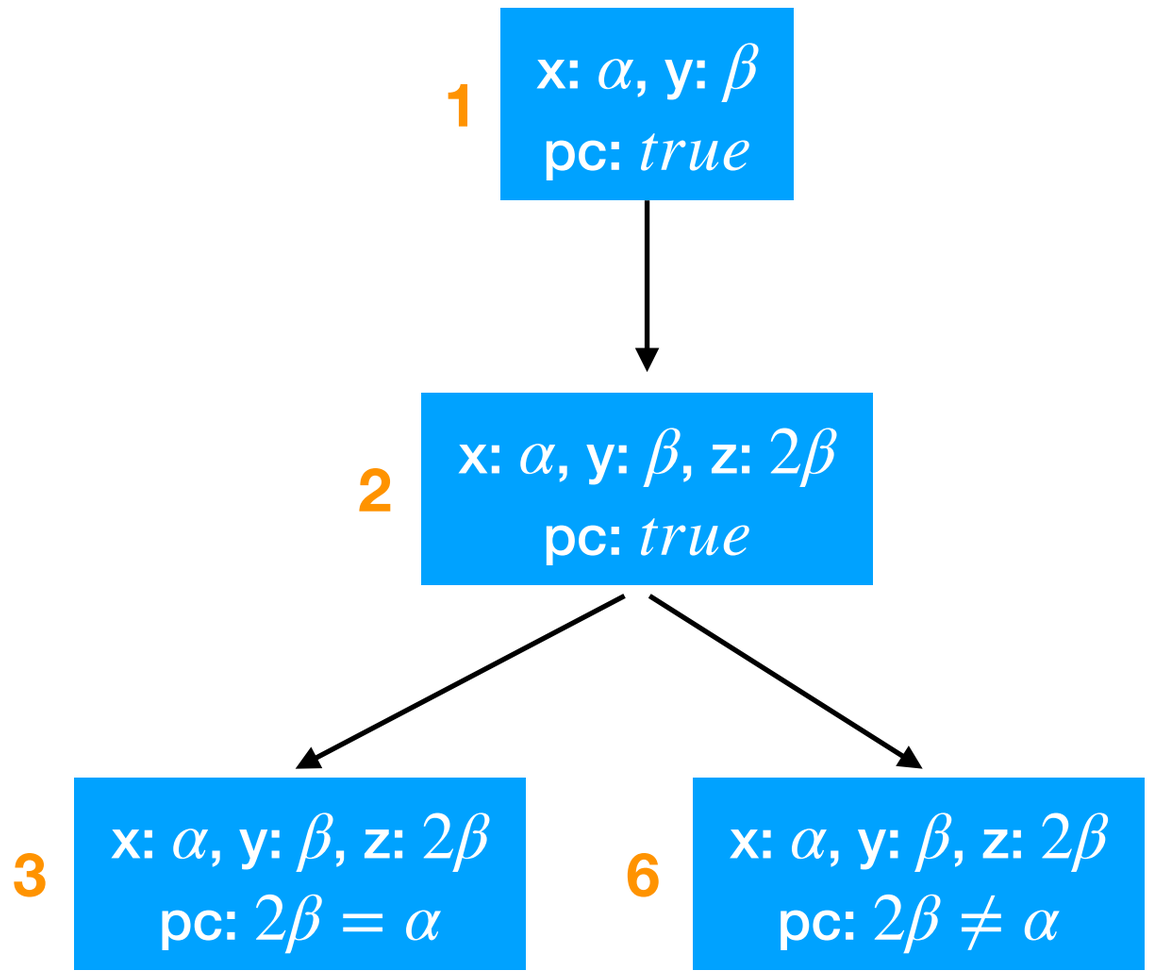
기호 실행 (Symbolic Execution)

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

void testme(int x, int y) {
1
  z := double (y);
2
  if (z==x) {
   3
    if (x>y+10) {
     4 Error;
    } else { 5 …}
  }
6 }
```



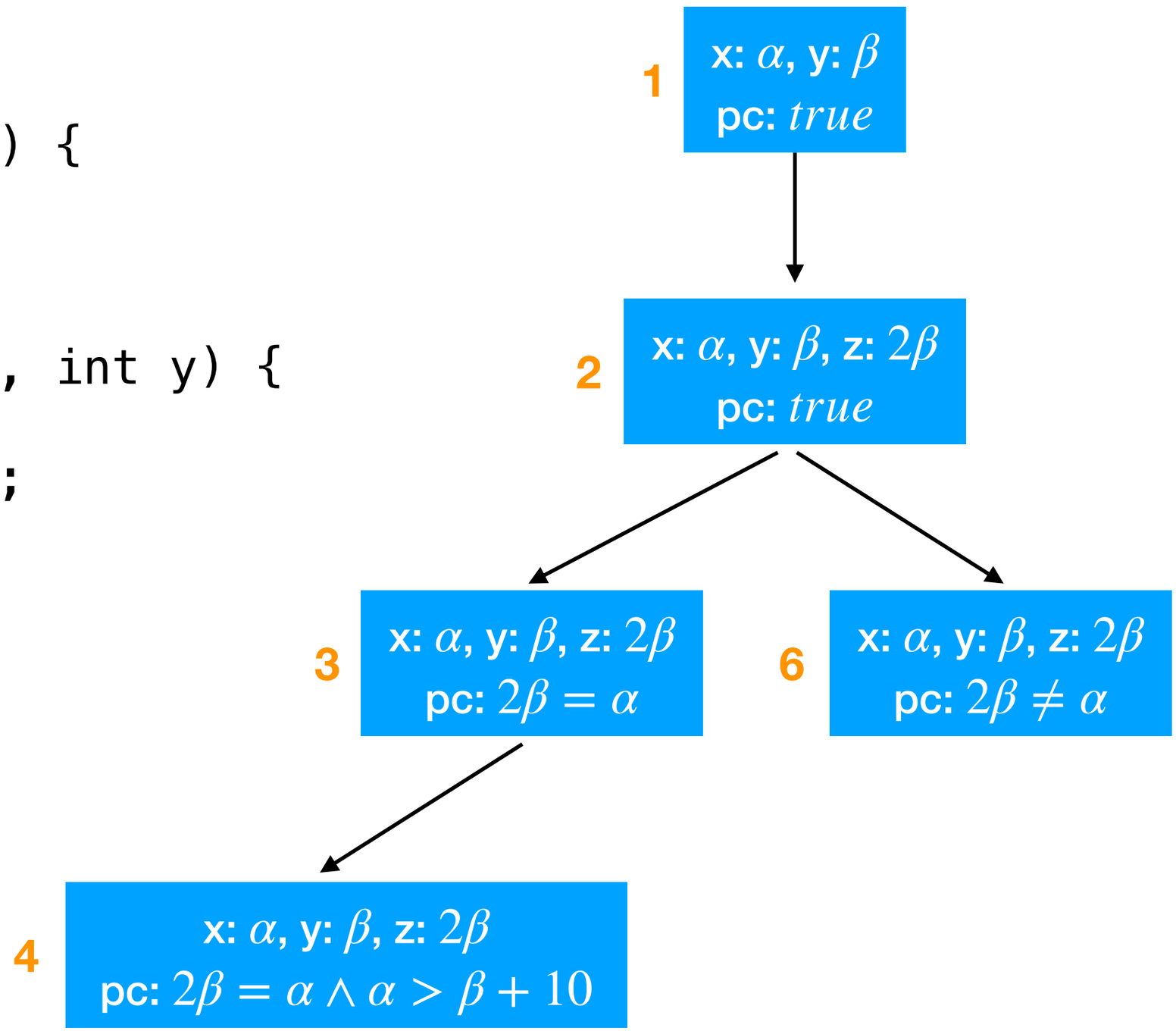
기호 실행 (Symbolic Execution)

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

void testme(int x, int y) {
1
  z := double (y);
2
  if (z==x) {
   3
    if (x>y+10) {
     4 Error;
    } else { 5 …}
  }
6 }
```



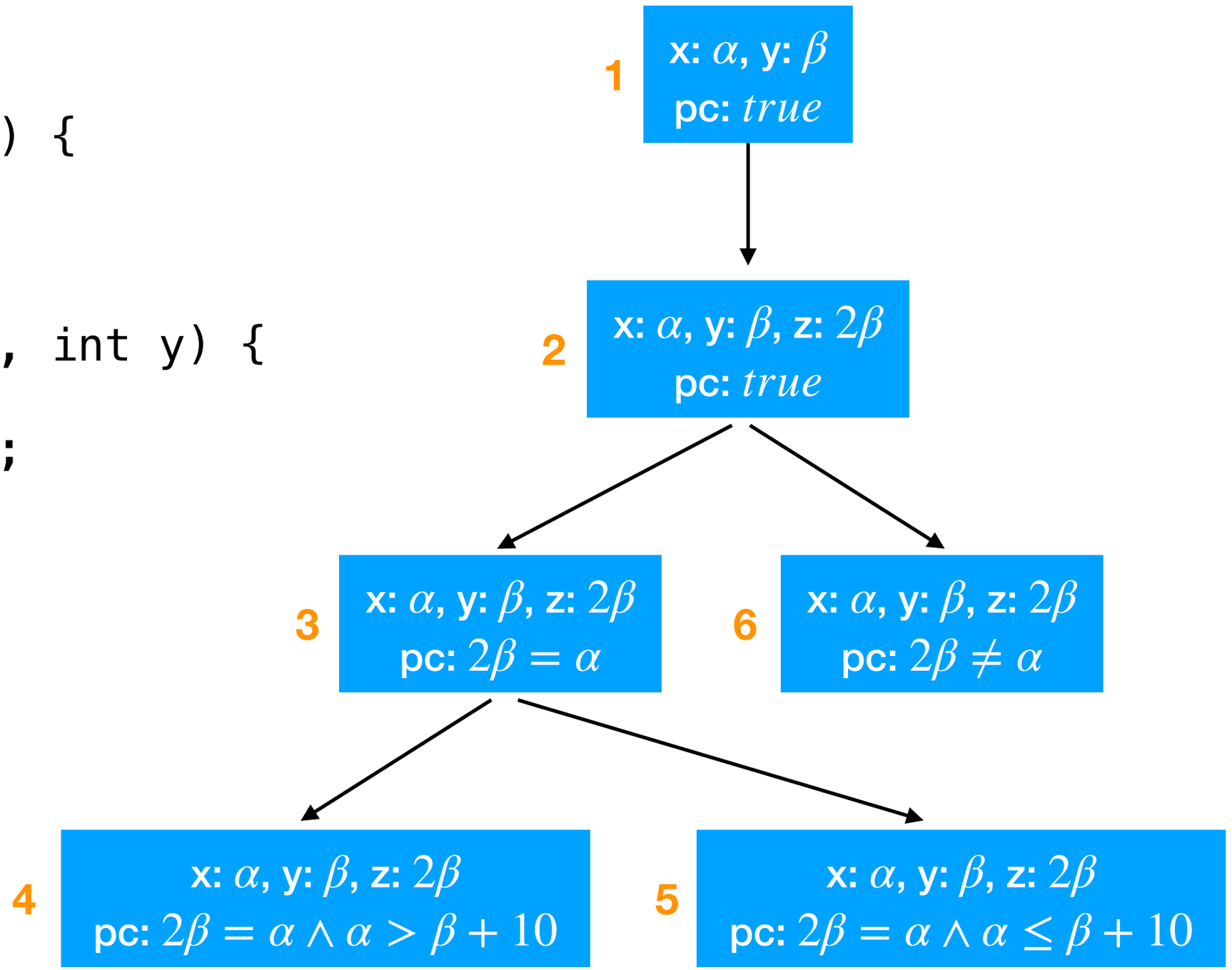
기호 실행 (Symbolic Execution)

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

void testme(int x, int y) {
1
  z := double (y);
2
  if (z==x) {
   3
    if (x>y+10) {
     4 Error;
    } else { 5 …}
  }
6 }
```



기호 실행 (Symbolic Execution)

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

void testme(int x, int y) {
1
  z := double (y);
2
  if (z==x) {
   3
    if (x>y+10) {
     4 Error;
    } else { 5 …}
  }
6 }
```



기호 실행 (Dynamic Symbolic Execution, Whitebox Fuzzing)

```
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

Dynamic Symbolic Execution

```
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

Dynamic Symbolic Execution

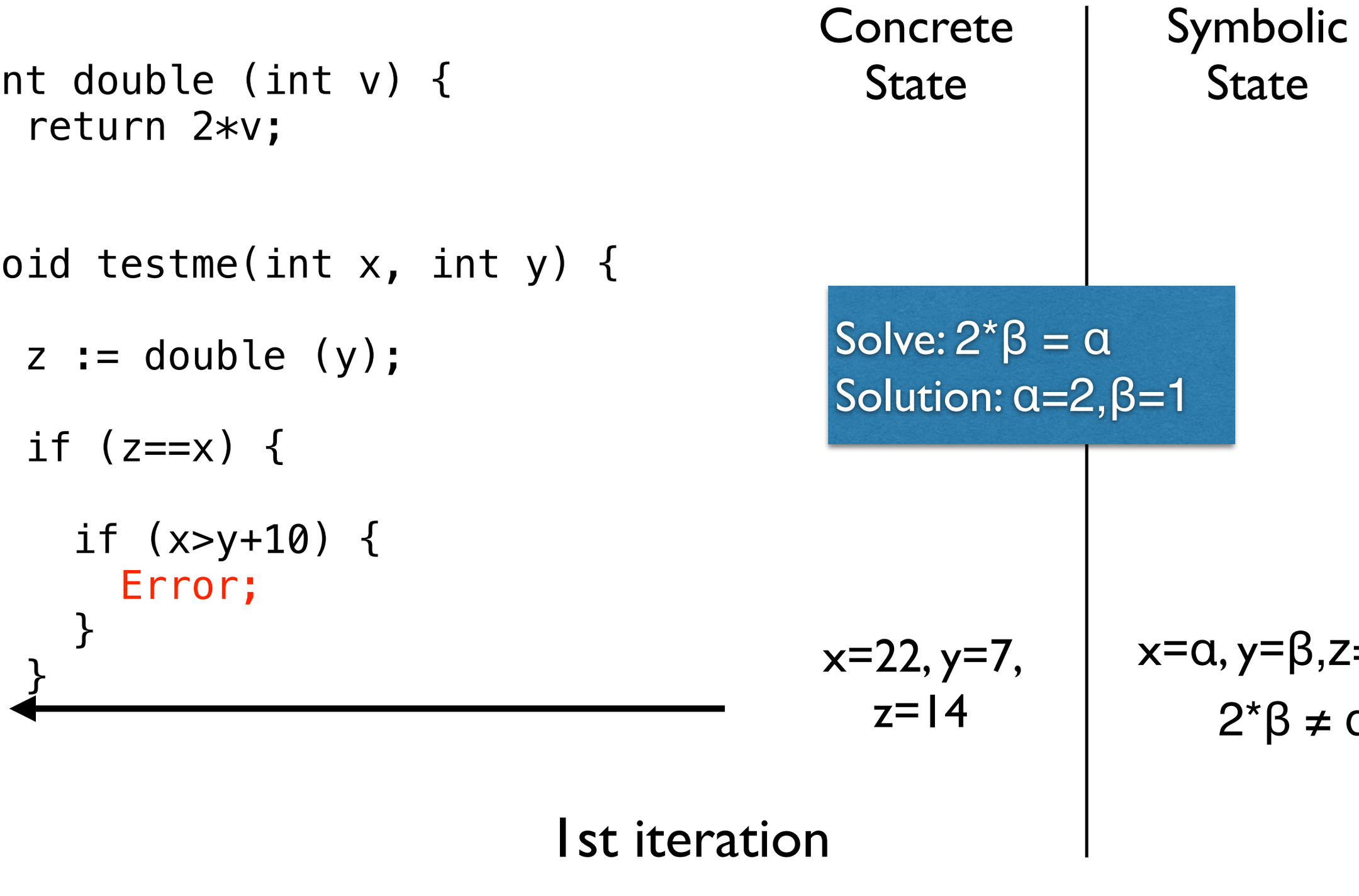
```
int double (int v) {
  return 2*v;
}

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

Dynamic Symbolic Execution

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

void testme(int x, int y) {

  z := double (y);

  if (z==x) {

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



1st iteration

Dynamic Symbolic Execution

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

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

  if (z==x) {

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

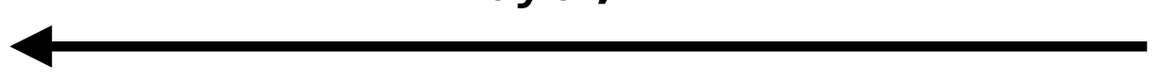
Concrete State	Symbolic State
x=2, y=1	x=α, y=β
	true

2nd iteration

Dynamic Symbolic Execution

```
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

Dynamic Symbolic Execution

```
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

Dynamic Symbolic Execution

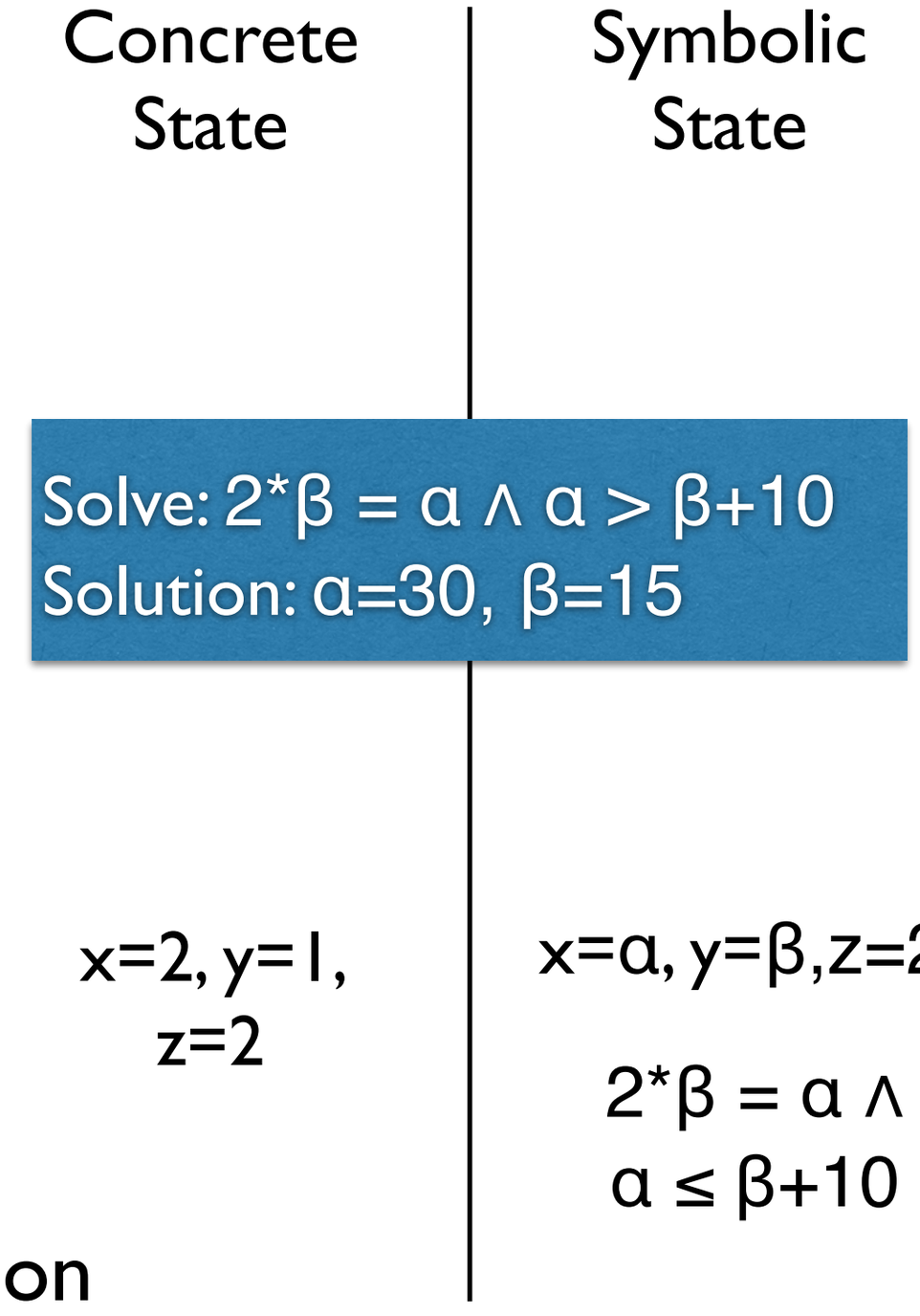
```
int double (int v) {
  return 2*v;
}

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 ∧ α ≤ β+10

2nd iteration

Dynamic Symbolic Execution

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

void testme(int x, int y) {

  z := double (y);

  if (z==x) {

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



2nd iteration

Dynamic Symbolic Execution

```
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

Dynamic Symbolic Execution

```
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

Dynamic Symbolic Execution

```
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

Dynamic Symbolic Execution

```
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+10

error-triggering input

3rd iteration

기호 실행 적용 사례

Benchmarks	Versions	Error Types	Bug-Triggering Inputs
vim	8.1*	Non-termination	`K1!1000100100111110(`
	5.7	Abnormal-termination	`H:w>>`"`\` [press 'Enter']
		Segmentation fault	`=ipI\-9~qOqw`
		Non-termination	`v(ipaprq&T$T`
gawk	4.2.1*	Memory-exhaustion	`'+E_Q$h+w$8==++$6E8#'`
	3.0.3	Abnormal-termination	`'f[][][][][y]^/#['`
		Non-termination	`'$g?E2^=-E-2"?^+$=":/?/#["'`
grep	3.1*	Abnormal-termination	`'\(\)\1*?*?|\W*\1W*'`
		Segmentation fault	`'\(\)\1^*@*\?\1*\+*\?'`
	2.2	Segmentation fault	`"_^^*9|^\(\)\'\1*$"`
		Non-termination	`'\({*****\)*\+++*\1*\+'`
sed	1.17	Segmentation fault	`'{:};:C;b'`

See "Concolic Testing with Adaptively Changing Search Heuristics. FSE 2019"

산업체 적용 사례

DOI:10.1145/2093548.2093564

Article development led by acmqueue
queue.acm.org

SAGE has had a remarkable impact at Microsoft.

BY PATRICE GODEFROID, MICHAEL Y. LEVIN, AND DAVID MOLNAR

SAGE: Whitebox Fuzzing for Security Testing

MOST *COMMUNICATIONS* READERS might think of "program verification research" as mostly theoretical with little impact on the world at large. Think again. If you are reading these lines on a PC running some form of Windows (like over 93% of PC users—that is, more than one billion people), then you have been affected by this line of work—without knowing it, which is precisely the way we want it to be.

Every second Tuesday of every month, also known as "Patch Tuesday," Microsoft releases a list of security bulletins and associated security patches to be deployed on hundreds of millions of machines worldwide. Each security bulletin costs Microsoft and its users millions of dollars. If a monthly security update costs you $0.001 (one tenth of one cent) in just electricity or loss of productivity, then this number multiplied by one billion people is $1 million. Of course, if malware were spreading on your machine, possibly leaking some of your private data, then that might cost you much more than $0.001. This is why we strongly encourage you to apply those pesky security updates.

Many security vulnerabilities are a result of programming errors in code for parsing files and packets that are transmitted over the Internet. For example, Microsoft Windows includes parsers for hundreds of file formats.

If you are reading this article on a computer, then the picture shown in Figure 1 is displayed on your screen after a jpg parser (typically part of your operating system) has read the image data, decoded it, created new data structures with the decoded data, and passed those to the graphics card in your computer. If the code implementing that jpg parser contains a bug such as a buffer overflow that can be triggered by a corrupted jpg image, then the execution of this jpg parser on your computer could potentially be hijacked to execute some other code, possibly malicious and hidden in the jpg data itself.

This is just one example of a possible security vulnerability and attack scenario. The security bugs discussed throughout the rest of this article are mostly buffer overflows.

Hunting for "Million-Dollar" Bugs

Today, hackers find security vulnerabilities in software products using two primary methods. The first is code inspection of binaries (with a good disassembler, binary code is like source code).

The second is *blackbox fuzzing*, a form of blackbox random testing, which randomly mutates well-formed program inputs and then tests the program with those modified inputs,[3] hoping to trigger a bug such as a buf-

Symbolic Execution for Software Testing in Practice – Preliminary Assessment

Cristian Cadar
Imperial College London
c.cadar@imperial.ac.uk

Patrice Godefroid
Microsoft Research
pg@microsoft.com

Sarfraz Khurshid
U. Texas at Austin
khurshid@ece.utexas.edu

Corina S. Păsăreanu*
CMU/NASA Ames
corina.s.pasareanu@nasa.gov

Koushik Sen
U.C. Berkeley
ksen@eecs.berkeley.edu

Nikolai Tillmann
Microsoft Research
nikolait@microsoft.com

Willem Visser
Stellenbosch University
visserw@sun.ac.za

ABSTRACT

We present results for the "Impact Project Focus Area" on the topic of symbolic execution as used in software testing. Symbolic execution is a program analysis technique introduced in the 70s that has received renewed interest in recent years, due to algorithmic advances and increased availability of computational power and constraint solving technology. We review classical symbolic execution and some modern extensions such as generalized symbolic execution and dynamic test generation. We also give a preliminary assessment of the use in academia, research labs, and industry.

Categories and Subject Descriptors

D.2.5 [**Testing and Debugging**]: Symbolic execution

General Terms

Reliability

Keywords

Generalized symbolic execution, dynamic test generation

1. INTRODUCTION

The ACM-SIGSOFT Impact Project is documenting the impact that software engineering research has had on software development practice. In this paper, we present preliminary results for documenting the impact of research in symbolic execution for automated software testing. Symbolic execution is a program analysis technique that was introduced in the 70s [8, 15, 31, 35, 46], and that has found renewed interest in recent years [9, 12, 13, 28, 29, 32, 33, 40, 42, 43, 50–52, 56, 57].

*We thank Matt Dwyer for his advice

ICSE'11, May 21–28, 2011, Waikiki, Honolulu, HI, USA

Symbolic execution is now the underlying technique of several popular testing tools, many of them open-source: NASA's Symbolic (Java) PathFinder[1], UIUC's CUTE and jCUTE[2], Stanford's KLEE[3], UC Berkeley's CREST[4] and BitBlaze[5], etc. Symbolic execution tools are now used in industrial practice at Microsoft (Pex[6], SAGE [29], YOGI[7] and PREfix [10]), IBM (Apollo [2]), NASA and Fujitsu (Symbolic PathFinder), and also form a key part of the commercial testing tool suites from Parasoft and other companies [60].

Although we acknowledge that the impact of symbolic execution in software practice is still limited, we believe that the explosion of work in this area over the past years makes for an interesting story about the increasing impact of symbolic execution since it was first introduced in the 1970s. Note that this paper is not meant to provide a comprehensive survey of symbolic execution techniques; such surveys can be found elsewhere [19, 44, 49]. Instead, we focus here on a few modern symbolic execution techniques that have shown promise to impact software testing in practice.

Software testing is the most commonly used technique for validating the quality of software, but it is typically a mostly manual process that accounts for a large fraction of software development and maintenance. Symbolic execution is one of the many techniques that can be used to automate software testing by automatically generating test cases that achieve high coverage of program executions.

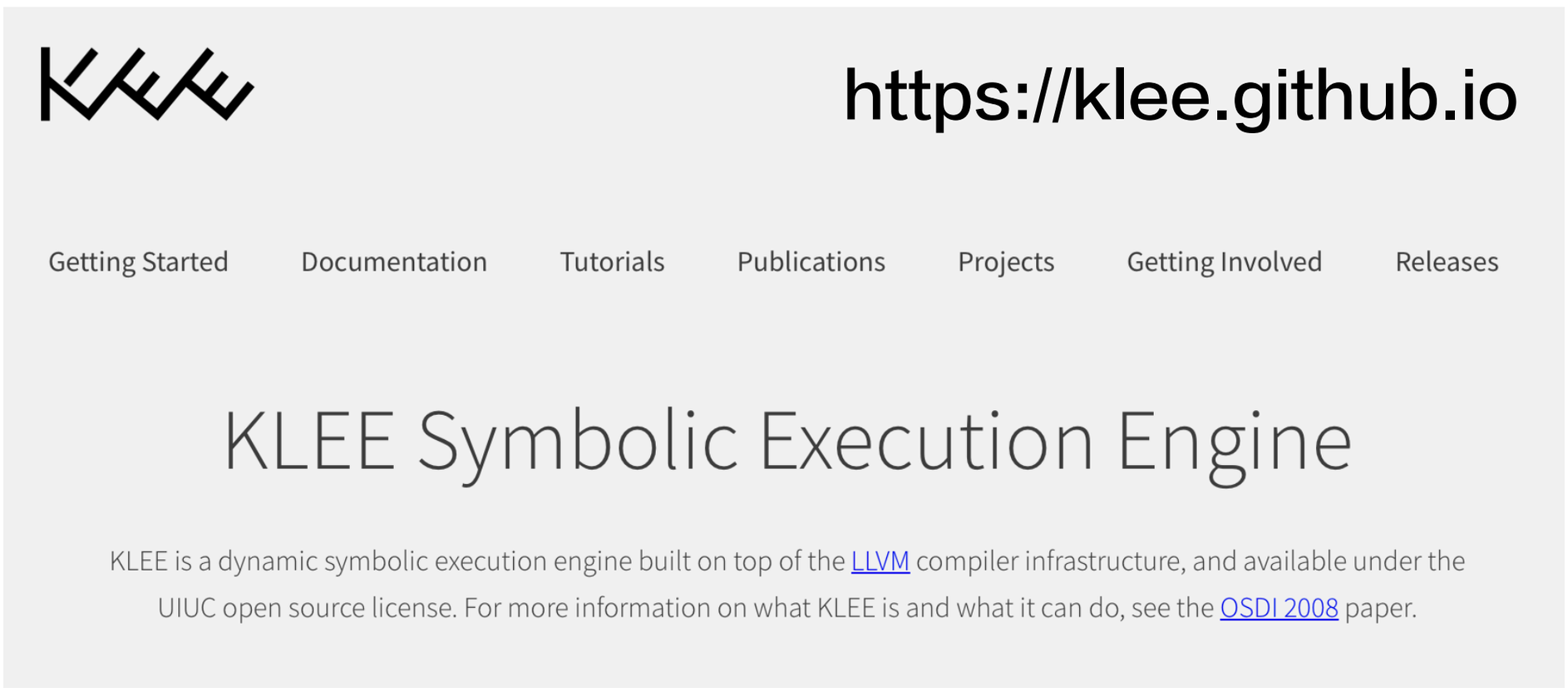
Symbolic execution is a program analysis technique that executes programs with symbolic rather than concrete inputs and maintains a *path condition* that is updated whenever a branch instruction is executed, to encode the constraints on the inputs that reach that program point. Test generation is performed by solving the collected constraints using a constraint solver. Symbolic execution can also be used for bug finding, where it checks for run-time errors or assertion violations and it generates test inputs that trigger those errors.

The original approaches to symbolic execution [8,15,31,35,

[1] http://babelfish.arc.nasa.gov/trac/jpf/wiki/projects/jpf-symbc
[2] http://osl.cs.uiuc.edu/~ksen/cute/
[3] http://klee.llvm.org/
[4] http://code.google.com/p/crest/
[5] http://bitblaze.cs.berkeley.edu/
[6] http://research.microsoft.com/en-us/projects/pex/
[7] http://research.microsoft.com/en-us/projects/yogi/

기호 실행 데모: KLEE

- Open-source symbolic execution engine
- Demo: http://klee.doc.ic.ac.uk



```c
#include <stdio.h>
#include <stdlib.h>

void crash() {
  char* x = (char*)malloc(1);
  free(x);
  free(x);
}

void testme(int x, int y) {
  int z;
  z = y * 2;

  if (z==x) {

    if (x>y+10) {
       crash();
    }
  }
}

int main(int argc, char** argv)
{
   int a, b;
   klee_make_symbolic(&a, sizeof(a), "a");
   klee_make_symbolic(&b, sizeof(b), "b");
   testme(a, b);
   return 0;
}
```

KLEE RESULTS

OUTPUT STATS

```
Job queued!

Executing KLEE

Executing KLEE

Uploading KLEE output directory

Done!


Ran command "/home/klee/klee_build/bin/klee /tmp/code/code.o".


KLEE: output directory is "/tmp/code/klee-out-0"
KLEE: Using STP solver backend
KLEE: WARNING ONCE: Alignment of memory from call "malloc" is not modelled. Using alignment of 8.
KLEE: ERROR: /tmp/code/code.c:7: memory error: invalid pointer: free
KLEE: NOTE: now ignoring this error at this location

KLEE: done: total instructions = 51
KLEE: done: completed paths = 3
KLEE: done: generated tests = 3


Failed tests:

  Memory error on line 7.
   free(x);
```

정적 분석

- 프로그램 실행을 요약(abstraction)하여 실행

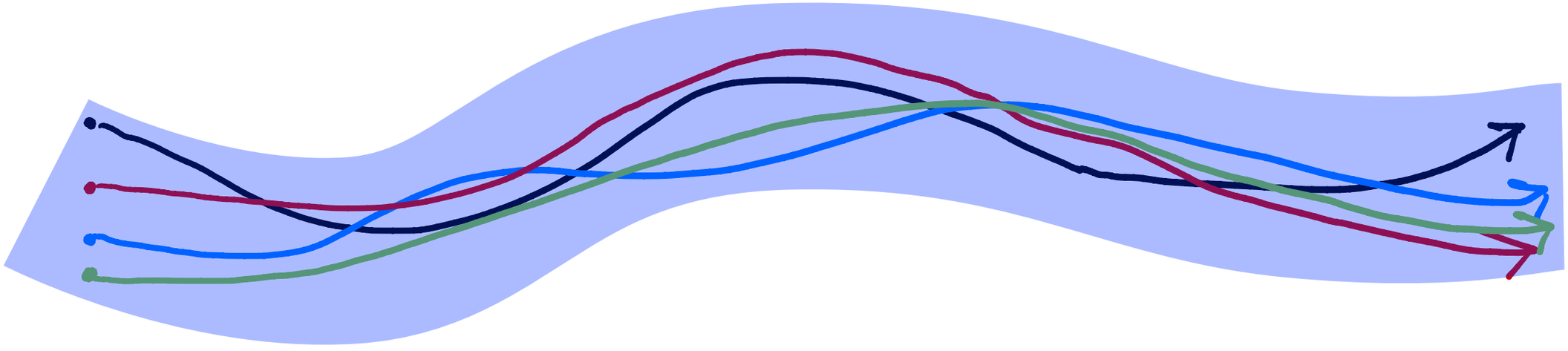
error zone

error zone

정적 분석

- 프로그램 실행을 요약(abstraction)하여 실행

error zone

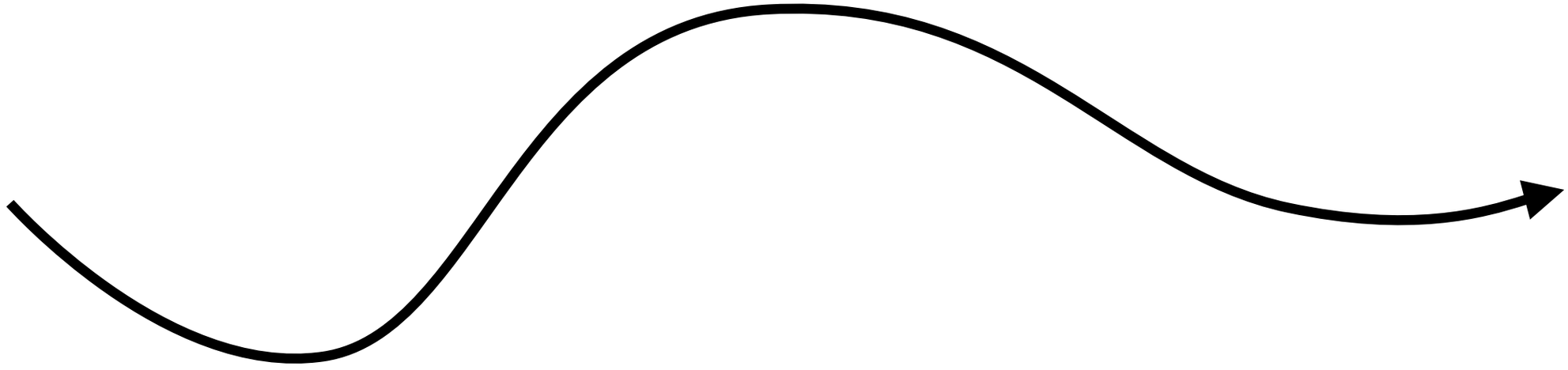


error zone

정적 분석

- Sound (Conservative) static analysis

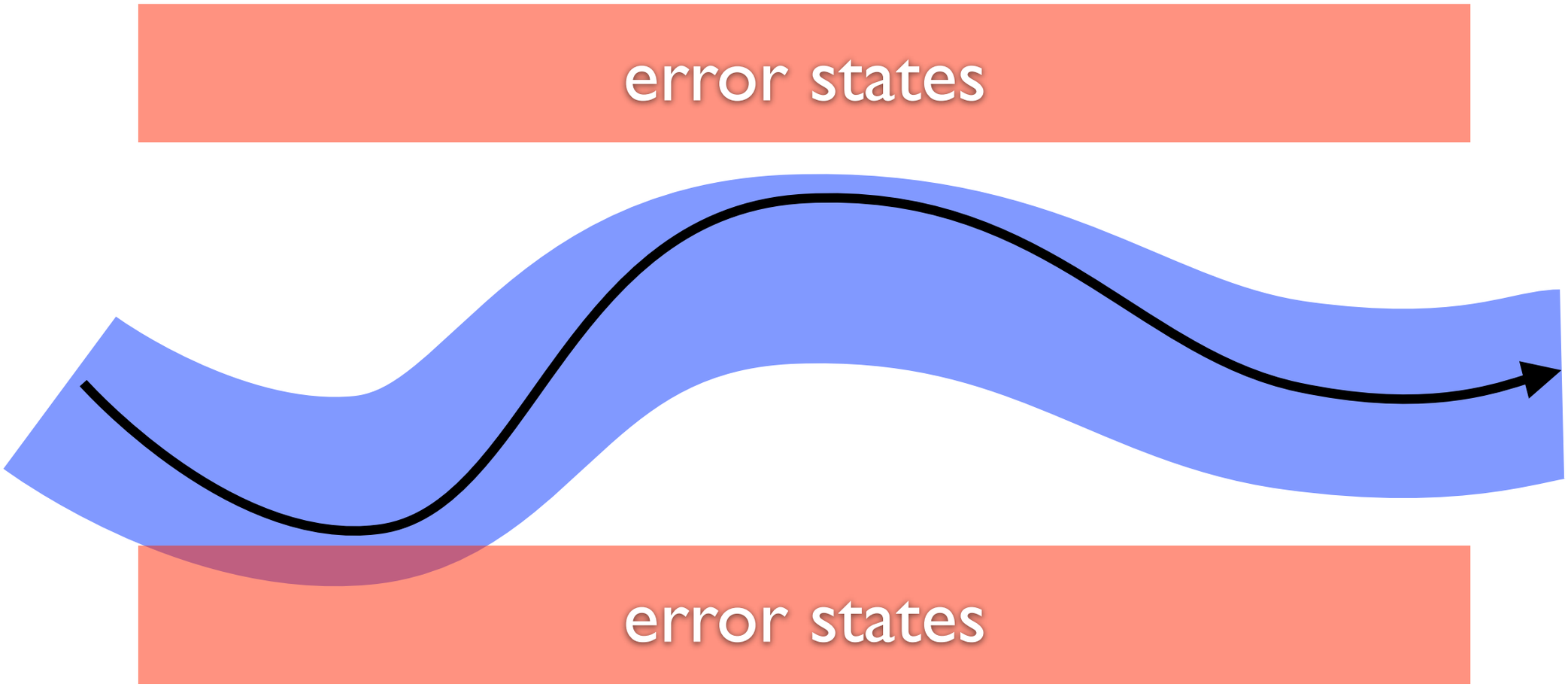
error states



error states

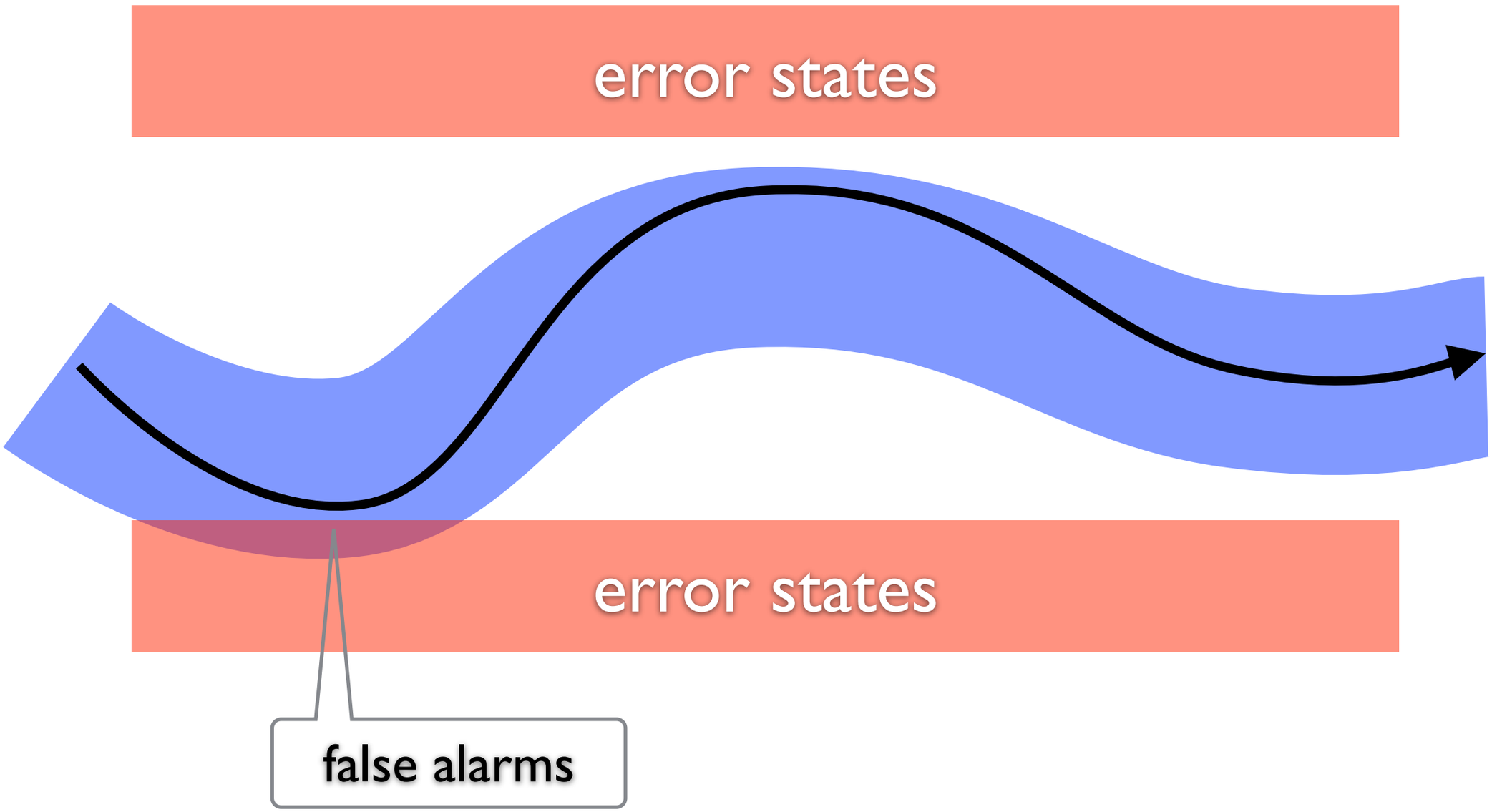
정적 분석

- Sound (Conservative) static analysis



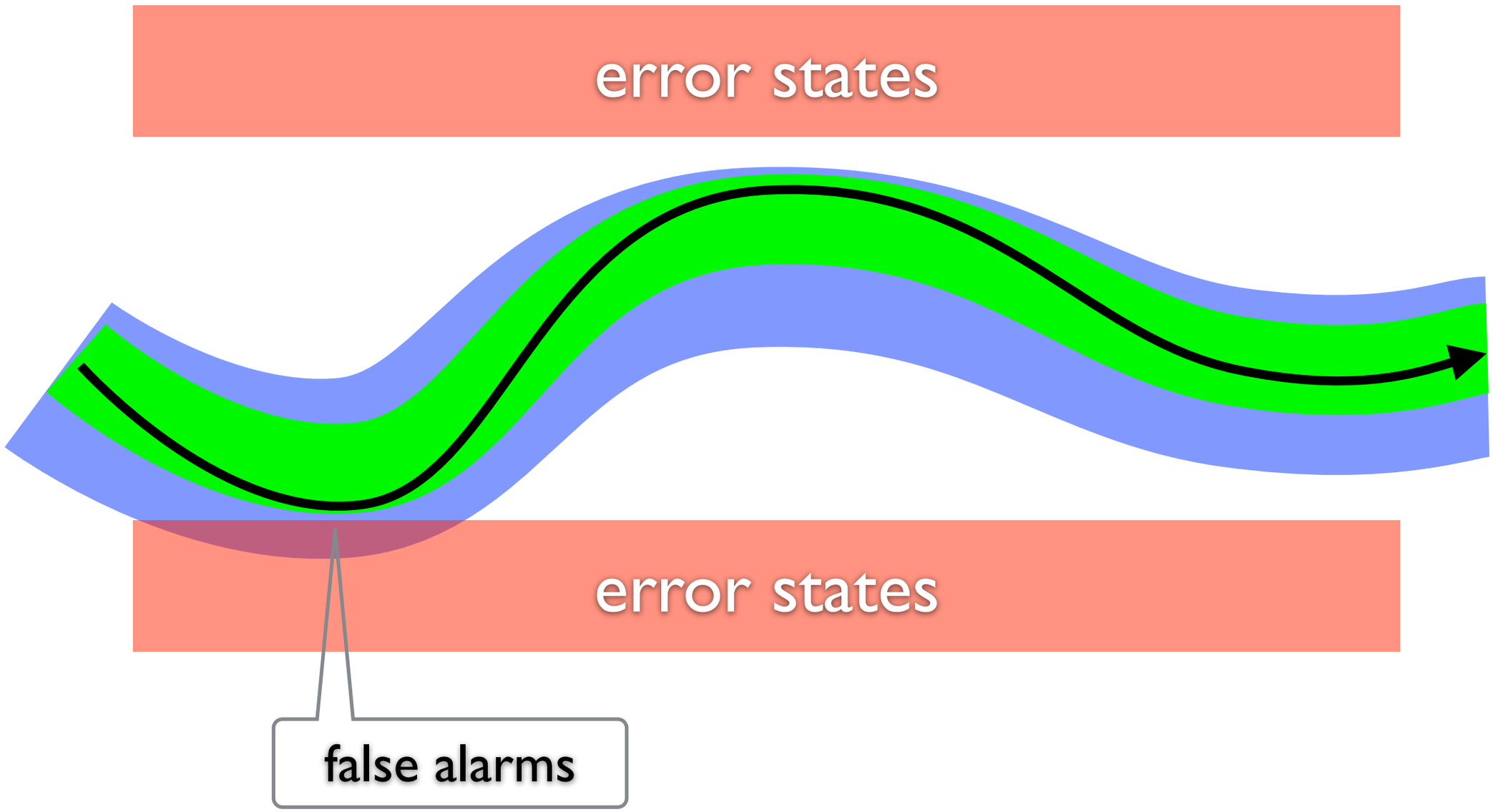
정적 분석

- Sound (Conservative) static analysis



정적 분석

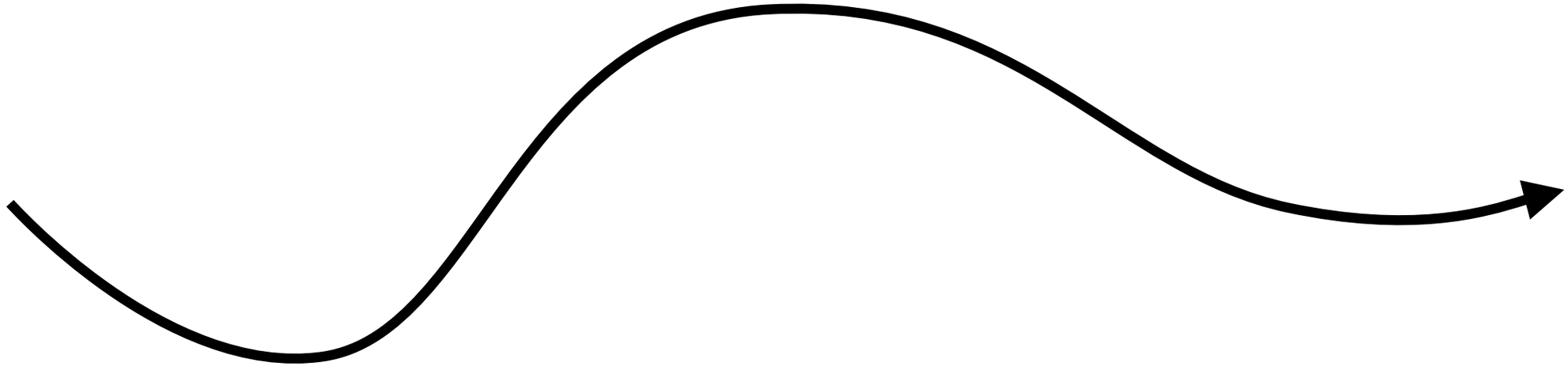
- Sound (Conservative) static analysis



정적 분석

- Bug-finding static analysis

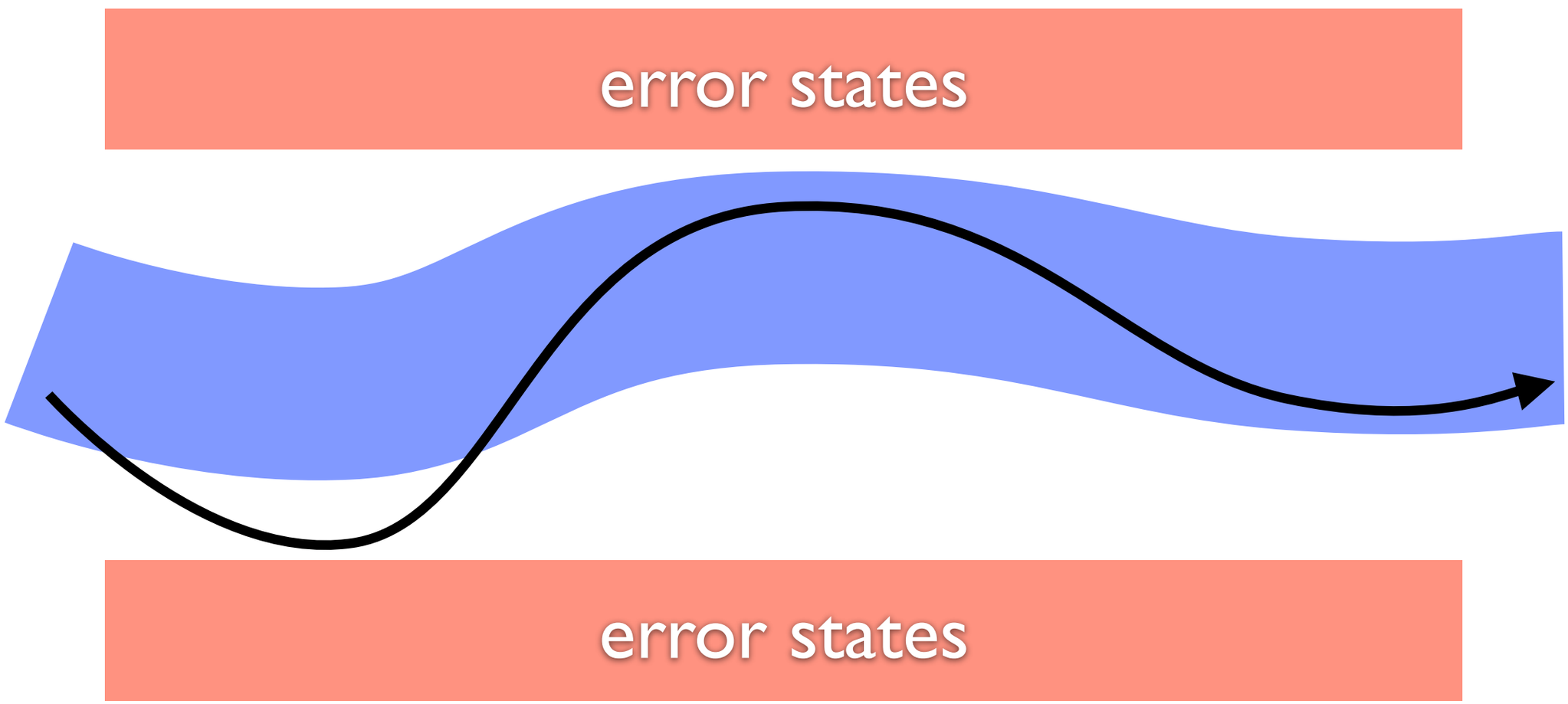
error states



error states

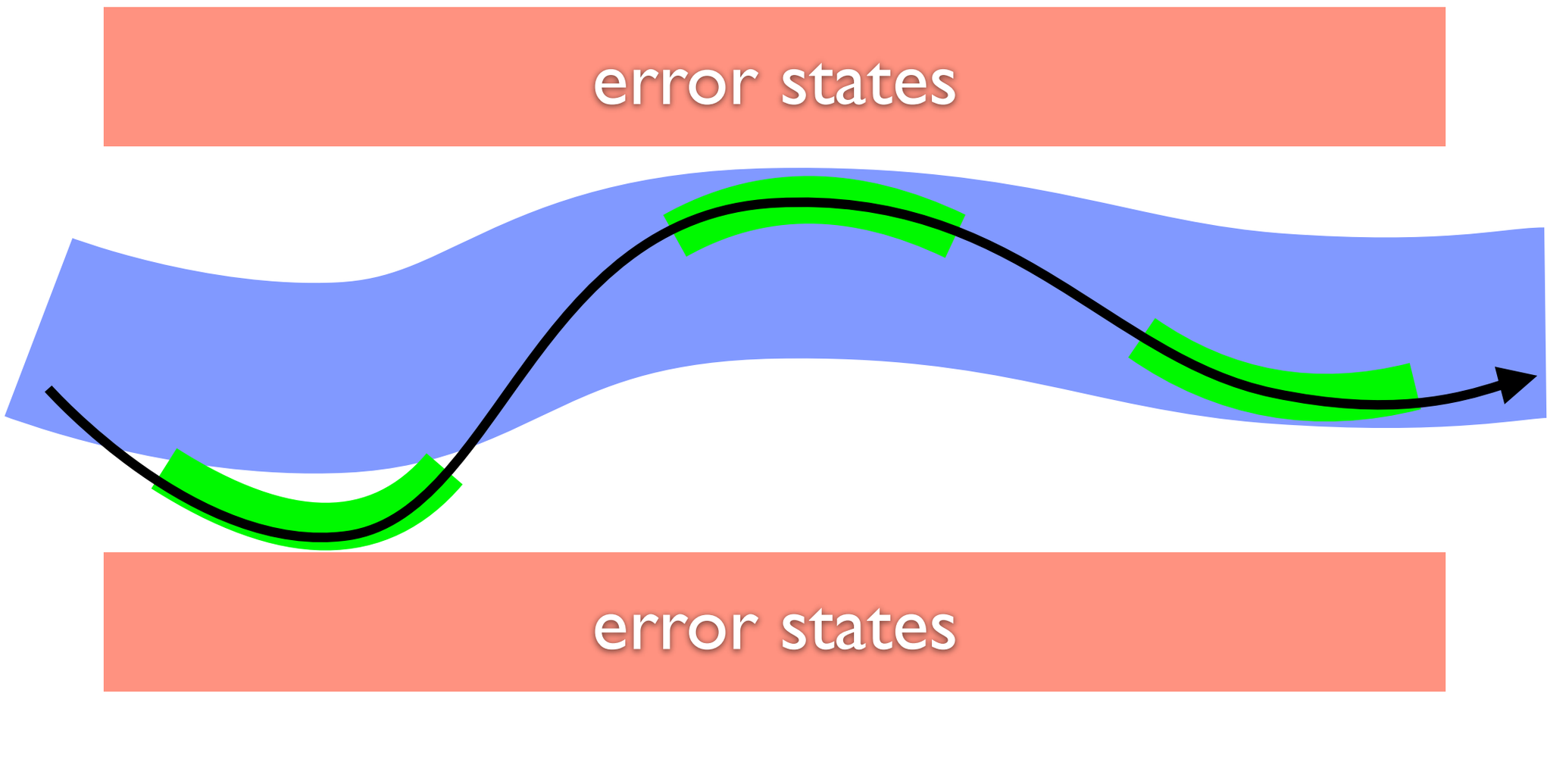
정적 분석

- Bug-finding static analysis



정적 분석

- Bug-finding static analysis



산업체 적용 사례

DOI:10.1145/3338112

Key lessons for designing static analyses tools deployed to find bugs in hundreds of millions of lines of code.

BY DINO DISTEFANO, MANUEL FÄHNDRICH, FRANCESCO LOGOZZO, AND PETER W. O'HEARN

Scaling Static Analyses at Facebook

Infer

STATIC ANALYSIS TOOLS are programs that examine, and attempt to draw conclusions about, the source of other programs without running them. At Facebook, we have been investing in advanced static analysis tools that employ reasoning techniques similar to those from program verification. The tools we describe in this article (Infer and Zoncolan) target issues related to crashes and to the security of our services, they perform sometimes complex reasoning spanning many procedures or files, and they are integrated into engineering workflows in a way that attempts to bring value while minimizing friction.

These tools run on code modifications, participating as bots during the code review process. Infer targets our mobile apps as well as our backend C++ code, codebases with 10s of millions of lines; it has seen over 100 thousand reported issues fixed by developers before code reaches production. Zoncolan targets the 100-million lines of Hack code, and is additionally integrated in the workflow used by security engineers. It has led to thousands of fixes of security and privacy bugs, outperforming any other detection method used at Facebook for such vulnerabilities. We will describe the human and technical challenges encountered and lessons we have learned in developing and deploying these analyses.

There has been a tremendous amount of work on static analysis, both in industry and academia, and we will not attempt to survey that material here. Rather, we present our rationale for, and results from, using techniques similar to ones that might be encountered at the edge of the research literature, not only simple techniques that are much easier to make scale. Our goal is to complement other reports on industrial static analysis and formal methods,[1,6,13,17] and we hope that such perspectives can provide input both to future research and to further industrial use of static analysis.

Next, we discuss the three dimensions that drive our work: bugs that matter, people, and actioned/missed bugs. The remainder of the article describes our experience developing and deploying the analyses, their impact, and the techniques that underpin our tools.

Context for Static Analysis at Facebook

Bugs that Matter. We use static analysis to prevent bugs that would affect our products, and we rely on our engineers' judgment as well as data from production to tell us the bugs that matter the most.

» key insights

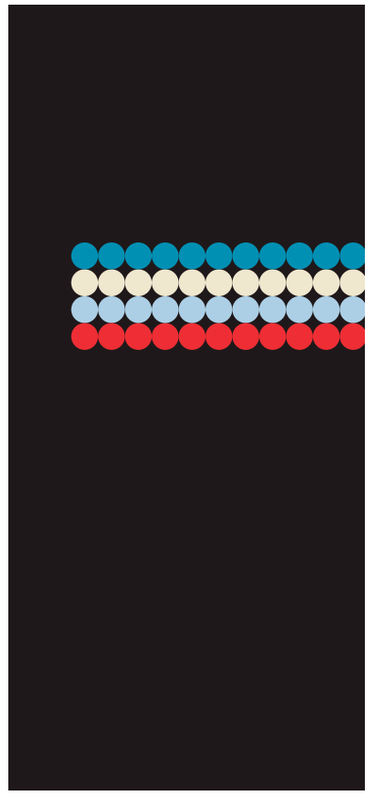
- **Advanced static analysis techniques performing deep reasoning about source code can scale to large industrial codebases, for example, with 100-million LOC.**
- **Static analyses should strike a balance between missed bugs (false negatives) and un-actioned reports (false positives).**
- **A "diff time" deployment, where issues are given to developers promptly as part of code review, is important to catching bugs early and getting high fix rates.**

DOI:10.1145/3188720

For a static analysis project to succeed, developers must feel they benefit from and enjoy using it.

BY CAITLIN SADOWSKI, EDWARD AFTANDILIAN, ALEX EAGLE, LIAM MILLER-CUSHON, AND CIERA JASPAN

Lessons from Building Static Analysis Tools at Google



SOFTWARE BUGS COST developers and software companies a great deal of time and money. For example, in 2014, a bug in a widely used SSL implementation ("goto fail") caused it to accept invalid SSL certificates,[36] and a bug related to date formatting caused a large-scale Twitter outage.[23] Such bugs are often statically detectable and are, in fact, obvious upon reading the code or documentation yet still make it into production software.

Previous work has reported on experience applying bug-detection tools to production software.[6,3,7,29] Although there are many such success stories for developers using static analysis tools, there are also reasons engineers do not always use static analysis tools or ignore their warnings,[6,7,26,30] including:

Not integrated. The tool is not integrated into the developer's workflow or takes too long to run;

Not actionable. The warnings are not actionable;

Not trustworthy. Users do not trust the results due to, say, false positives;

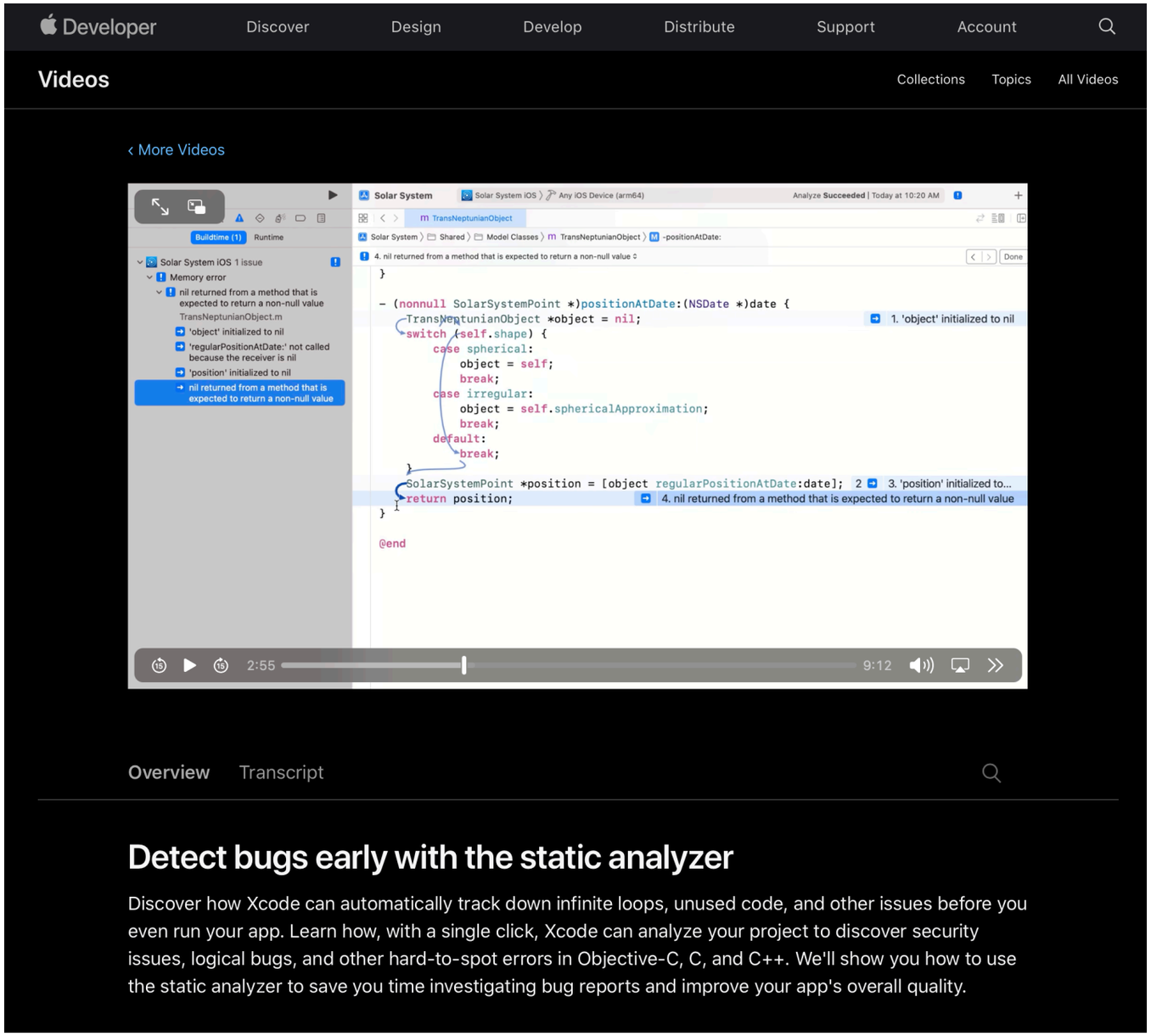
Not manifest in practice. The reported bug is theoretically possible, but the problem does not actually manifest in practice;

» key insights

- **Static analysis authors should focus on the developer and listen to their feedback.**
- **Careful developer workflow integration is key for static analysis tool adoption.**
- **Static analysis tools can scale by crowdsourcing analysis development.**

산업체 적용 사례

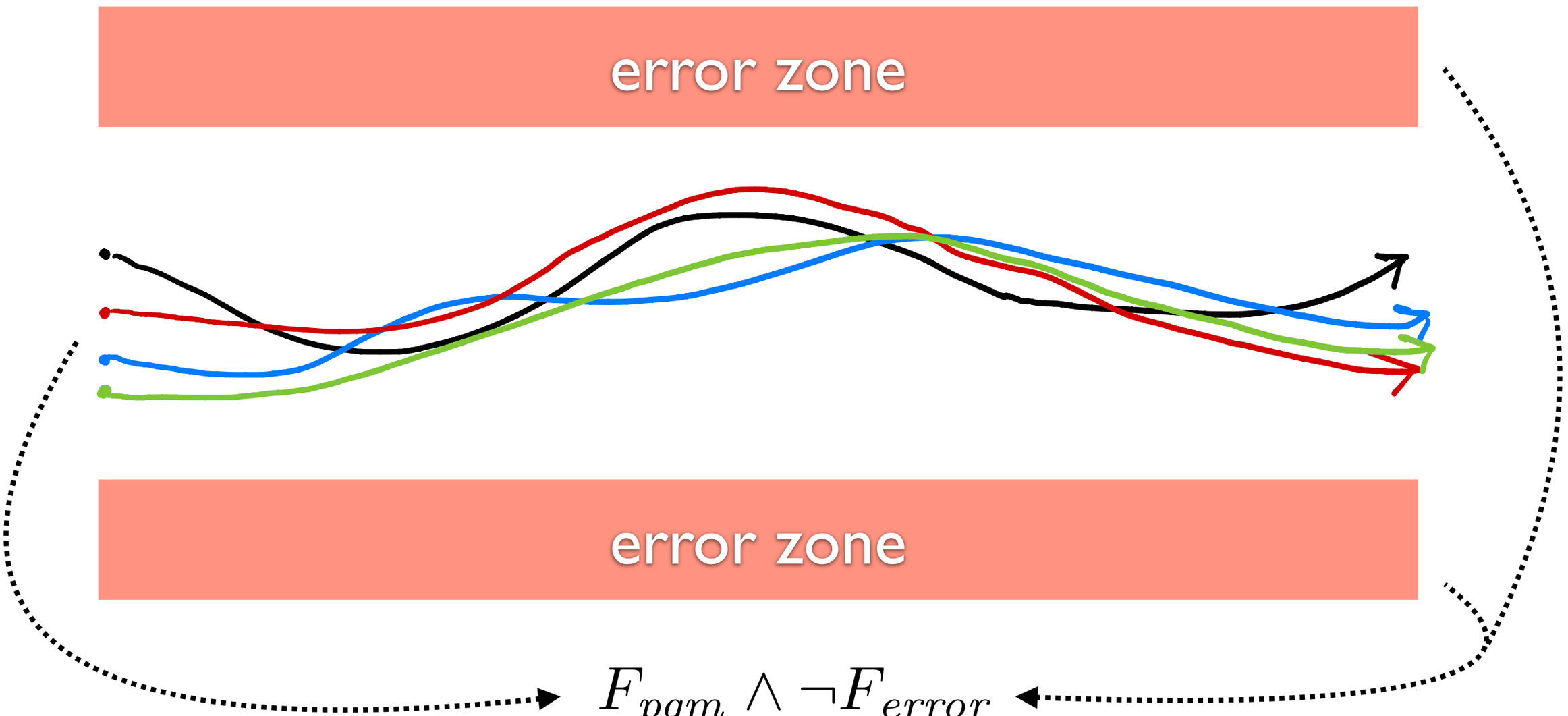
WWDC (Apple Worldwide Developers Conferece) 2021



https://developer.apple.com/videos/play/wwdc2021/10202/

프로그램 검증

- 프로그램 실행 의미와 오류 조건을 논리식으로 변환



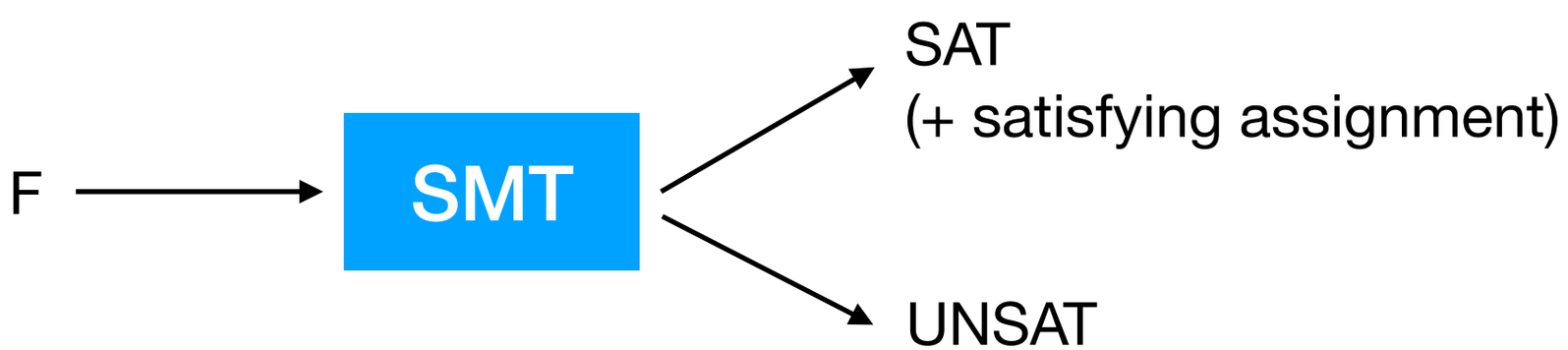
프로그램 검증

- 논리식의 satisfiability

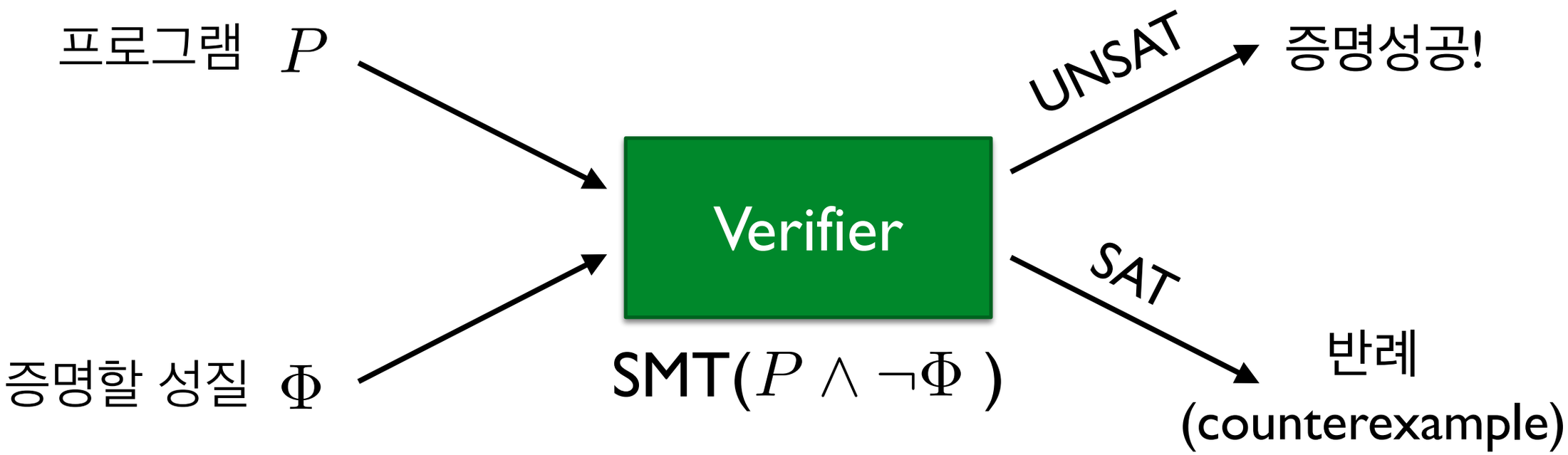
$F : P \wedge Q \rightarrow P \vee \neg Q.$

P	Q	$P \wedge Q$	$\neg Q$	$P \vee \neg Q$	F
0	0	0	1	1	1
0	1	0	0	0	1
1	0	0	1	1	1
1	1	1	0	1	1

- SMT (Satisfiability Modulo Theory) Solver



프로그램 검증



- 프로그램과 증명할 성질을 논리식으로 표현
- 논리식의 satisfiability 여부를 판별

예제

```
int f(bool a) {
  x = 0; y = 0;
  if (a) {
    x = 1;
  }
  if (a) {
    y = 1;
  }
  assert (x == y)
}
```

예제

```
int f(a, b) {
  x = 0; y = 0;
  if (a) {
    x = 1;
  }
  if (b) {
    y = 1;
  }
  assert (x == y)
}
```

반복문 불변 성질 (Invariant)

```
i = 0;
j = 0;
while
(i < 10) {
   i++;
   j++;
}
assert (i-j==0)
```

반복문 불변 성질 (Invariant)

```
i = 0;
j = 0;
while @(i==j)
(i < 10) {
   i++;
   j++;
}
assert (i-j==0)
```

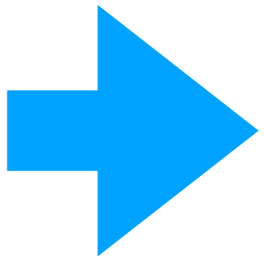
반복문 불변 성질 (Invariant)

```
i = 0;
j = 0;
while @(i==j)
(i < 10) {
  i++;
  j++;
}
assert (i-j==0)
```

무한히 많은 불변 성질들 (i >= 0,j >= 0,i == j,true,…) 가운데 증명에 성공하는 것이 필요

프로그램 검증 기술의 장단점

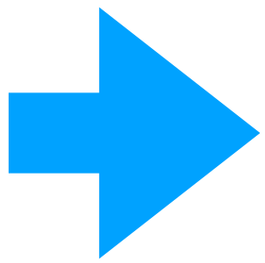
```
bool LinearSearch (int a[], int l, int u, int e) {
  int i := l;
  while (i ≤ u) {
    if (a[i] = e) return true
    i := i + 1;
  }
  return false
}
```



```
@pre : 0 ≤ l ∧ u < |a|
@post : rv ↔ ∃i.l ≤ i ≤ u ∧ a[i] = e
bool LinearSearch (int a[], int l, int u, int e) {
  int i := l;
  while
  @L : l ≤ i ∧ (∀j. l ≤ j < i → a[j] ≠ e)
  (i ≤ u) {
    if (a[i] = e) return true
    i := i + 1;
  }
  return false
}
```

프로그램 검증 기술의 장단점

```
bool BubbleSort (int a[]) {
  int[] a := a_0
  for (int i := |a| − 1; i > 0; i := i − 1) {
    for (int j := 0; j < i; j := j + 1) {
      if (a[j] > a[j + 1]) {
        int t := a[j];
        int a[j] := a[j + 1];
        int a[j + 1] := t;
      }
    }
  }
  return a;
}
```



```
@pre : ⊤
@post : sorted(rv, 0, |rv| − 1)
bool BubbleSort (int a[]) {
  int[] a := a_0
  @L1 [ −1 ≤ i < |a|
      ∧ partitioned(a, 0, i, i + 1, |a| − 1)
      ∧ sorted(a, i, |a| − 1) ]
  for (int i := |a| − 1; i > 0; i := i − 1) {
    @L2 [ 1 ≤ i < |a| ∧ 0 ≤ j ≤ i
        ∧ partitioned(a, 0, i, i + 1, |a| − 1)
        ∧ partitioned(a, 0, j − 1, j, j)
        ∧ sorted(a, i, |a| − 1) ]
    for (int j := 0; j < i; j := j + 1) {
      if (a[j] > a[j + 1]) {
        int t := a[j];
        int a[j] := a[j + 1];
        int a[j + 1] := t;
      }
    }
  }
  return a;
}
```

$$\textbf{sorted}(a, l, u) \iff \forall i, j. l \leq i \leq j \leq u \rightarrow a[i] \leq a[j]$$

산업체 적용 사례

Code-Level Model Checking in the Software Development Workflow

Nathan Chong
Amazon

Byron Cook
Amazon
UCL

Konstantinos Kallas
University of Pennsylvania

Kareem Khazem
Amazon

Felipe R. Monteiro
Amazon

Daniel Schwartz-Narbonne
Amazon

Serdar Tasiran
Amazon

Michael Tautschnig
Amazon
Queen Mary University of London

Mark R. Tuttle
Amazon

ABSTRACT

This experience report describes a style of applying symbolic model checking developed over the course of four years at Amazon Web Services (AWS). Lessons learned are drawn from proving properties of numerous C-based systems, e.g., custom hypervisors, encryption code, boot loaders, and an IoT operating system. Using our methodology, we find that we can prove the correctness of industrial low-level C-based systems with reasonable effort and predictability. Furthermore, AWS developers are increasingly writing their own formal specifications. All proofs discussed in this paper are publicly available on GitHub.

CCS CONCEPTS

• **Software and its engineering** → **Formal software verification**; *Model checking*; Correctness; • **Theory of computation** → Program reasoning.

KEYWORDS

Continuous Integration, Model Checking, Memory Safety.

ACM Reference Format:
Nathan Chong, Byron Cook, Konstantinos Kallas, Kareem Khazem, Felipe R. Monteiro, Daniel Schwartz-Narbonne, Serdar Tasiran, Michael Tautschnig, and Mark R. Tuttle. 2020. Code-Level Model Checking in the Software Development Workflow. In *Software Engineering in Practice (ICSE-SEIP '20), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3377813.3381347

1 INTRODUCTION

This is a report on making code-level proof via model checking a routine part of the software development workflow in a large industrial organization. Formal verification of source code can have a significant positive impact on the quality of industrial code. In particular, formal specification of code provides precise, machine-checked documentation for developers and consumers of a code base. They improve code quality by ensuring that the program's *implementation* reflects the developer's *intent*. Unlike testing, which can only validate code against a set of concrete inputs, formal proof can assure that the code is both secure and correct for all possible inputs.

Unfortunately, rapid proof development is difficult in cases where proofs are written by a separate specialized team and *not* the software developers themselves. The developer writing a piece of code has an internal mental model of their code that explains why, and under what conditions, it is correct. However, this model typically remains known only to the developer. At best, it may be partially captured through informal code comments and design documents. As a result, the proof team must spend significant effort to reconstruct the formal specification of the code they are verifying. This slows the process of developing proofs.

Over the course of four years developing code-level proofs in Amazon Web Services (AWS), we have developed a proof methodology that allows us to produce proofs with reasonable and predictable effort. For example, using these techniques, one full-time verification engineer and two interns were able to specify and verify 171 entry points over 9 key modules in the AWS C Common[1] library over a period of 24 weeks (see Sec. 3.2 for a more detailed description of this library). All specifications, proofs, and related artifacts (such as continuous integration reports), described in this paper have been integrated into the main AWS C Common repository on GitHub, and are publicly available at https://github.com/awslabs/aws-c-common/.

1.1 Methodology

Our methodology has four key elements, all of which focus on communicating with the development team using artifacts that fit their existing development practices. We find that of the many different ways we have approached verification engagements, this combination of techniques has most deeply involved software developers in the proof creation and maintenance process. In particular, developers have begun to write formal functional specifications for code as they develop it. Initially, this involved the development team asking the verification team to assist them in writing specifications for new

ICSE-SEIP '20, May 23–29, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7123-0/20/05.
https://doi.org/10.1145/3377813.3381347

[1] https://github.com/awslabs/aws-c-common

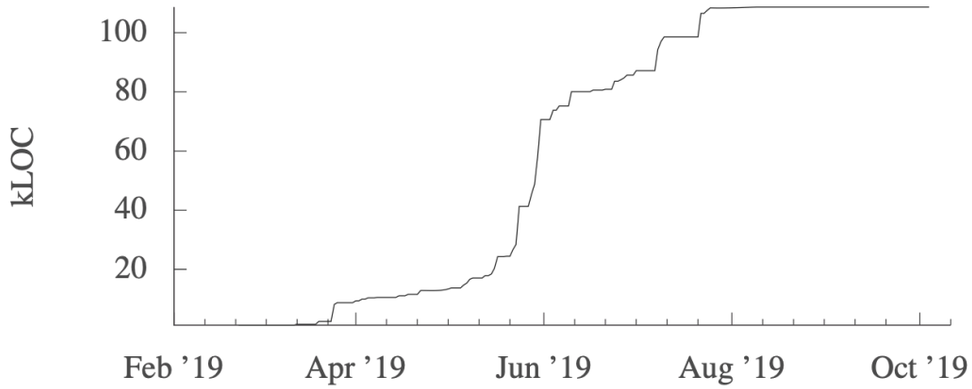


Figure 1: Cumulative number of LOC proven.

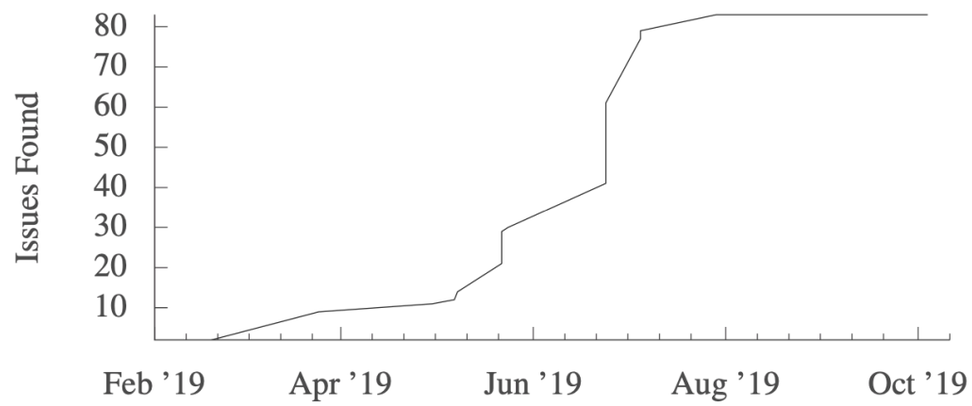


Figure 2: Cumulative number of issues found.

Table 1: Severity and root cause of issues found.

Root cause	# issues	Severity: High	Severity: Medium	Severity: Low
Integer overflow	10 (12%)	2	8	0
Null-pointer deref.	57 (69%)	0	14	43
Functional	11 (13%)	0	4	7
Memory safety	5 (6%)	0	5	0
Total	83	2 (3%)	31 (37%)	50 (60%)

프로그램 분석 기술 비교

	Automatic	Sound	Complete	When
Testing/ Fuzzing				
Symbolic Execution				
Sound Static Analysis				
Bug-Finding Static Analysis				
Program Verification				

Part 1. 프로그램 분석 개괄

Part 2. 정적 분석 적용 사례

Part 3. 정적 분석 원리

Part 4. 정적 분석 구현

상용 정적 분석 도구들

Infer

COVERITY® BY SYNOPSYS

PolySpace TECHNOLOGIES

...

Clang Static Analyzer

sonarqube

Sparrow The Early Bird

https://github.com/facebook/infer

Infer 사용법

- Install (https://github.com/facebook/infer/blob/main/INSTALL.md)

```
# Checkout Infer
git clone https://github.com/facebook/infer.git
cd infer
# Compile Infer
./build-infer.sh java
# install Infer system-wide...
sudo make install
# ...or, alternatively, install Infer into your PATH
export PATH=`pwd`/infer/bin:$PATH
```

- Running Infer: e.g.,
  - `infer capture -- make`
  - `infer analyze`

정적 분석 활용 사례

- Safety-citical software: e.g., smart contract
- Corner case bugs: e.g., memory bugs in C/C++
- Complex buggy scenario: e.g., Java NPEs
- …

사례 1: 스마트 컨트랙트

블록체인 1.0

코인 거래만 가능

vs.

블록체인 2.0

임의의 거래가 가능

Key: 스마트 컨트랙트

스마트 컨트랙트 생김새

```
1   contract Netkoin {
2     mapping (address => uint) public balance;
3     uint public totalSupply;
4
5     constructor (uint initialSupply) {
6       totalSupply = initialSupply;
7       balance[msg.sender] = totalSupply;
8     }
9
10    function transfer (address to, uint value) public
11    returns (bool) {
12      require (balance[msg.sender] >= value);
13      balance[msg.sender] -= value;
14      balance[to] += value;
15      return true;
16    }
17
18    function burn (uint value) public returns (bool) {
19      require (balance[msg.sender] >= value);
20      balance[msg.sender] -= value;
21      totalSupply -= value;
22      return true;
23    }
24  }
```

데이터

생성자

함수

함수

스마트 컨트랙트 생김새

```
1   contract Netkoin {
2     mapping (address => uint) public balance;
3     uint public totalSupply;
4
5     constructor (uint initialSupply) {
6       totalSupply = initialSupply;
7       balance[msg.sender] = totalSupply;
8     }
9
10    function transfer (address to, uint value) public
11    returns (bool) {
12      require (balance[msg.sender] >= value);
13      balance[msg.sender] -= value;
14      balance[to] += value;
15      return true;
16    }
17
18    function burn (uint value) public returns (bool) {
19      require (balance[msg.sender] >= value);
20      balance[msg.sender] -= value;
21      totalSupply -= value;
22      return true;
23    }
24  }
```

사용자의 계좌 정보

송금

잔고가 충분하면
거래를 실행

데이터

생성자

함수

함수

스마트 컨트랙트 특징

- 스마트 컨트랙트는 매우 엄밀한 수준의 안전성 검증이 필요
  - 공격에 성공하면 막대한 금전적 피해가 발생
  - 누구나 온라인에서 소스코드 열람 가능하지만 수정 불가

스마트 컨트랙트 특징

- 스마트 컨트랙트는 매우 엄밀한 수준의 안전성 검증이 필요
  - 공격에 성공하면 막대한 금전적 피해가 발생
  - 누구나 온라인에서 소스코드 열람 가능하지만 수정 불가


KLINT FINLEY 06.18.16 04:30 AM

A $50 Million Hack Just Showed That the DAO Was All Too Human

750억원 (2016)

스마트 컨트랙트 특징

- 스마트 컨트랙트는 매우 엄밀한 수준의 안전성 검증이 필요
  - 공격에 성공하면 막대한 금전적 피해가 발생
  - 누구나 온라인에서 소스코드 열람 가능하지만 수정 불가

KLINT FINLEY 06.18.16 04:30 AM

A $50 M
That the

750억원 (2

WILLIAM SUBERG NOV 08, 2017

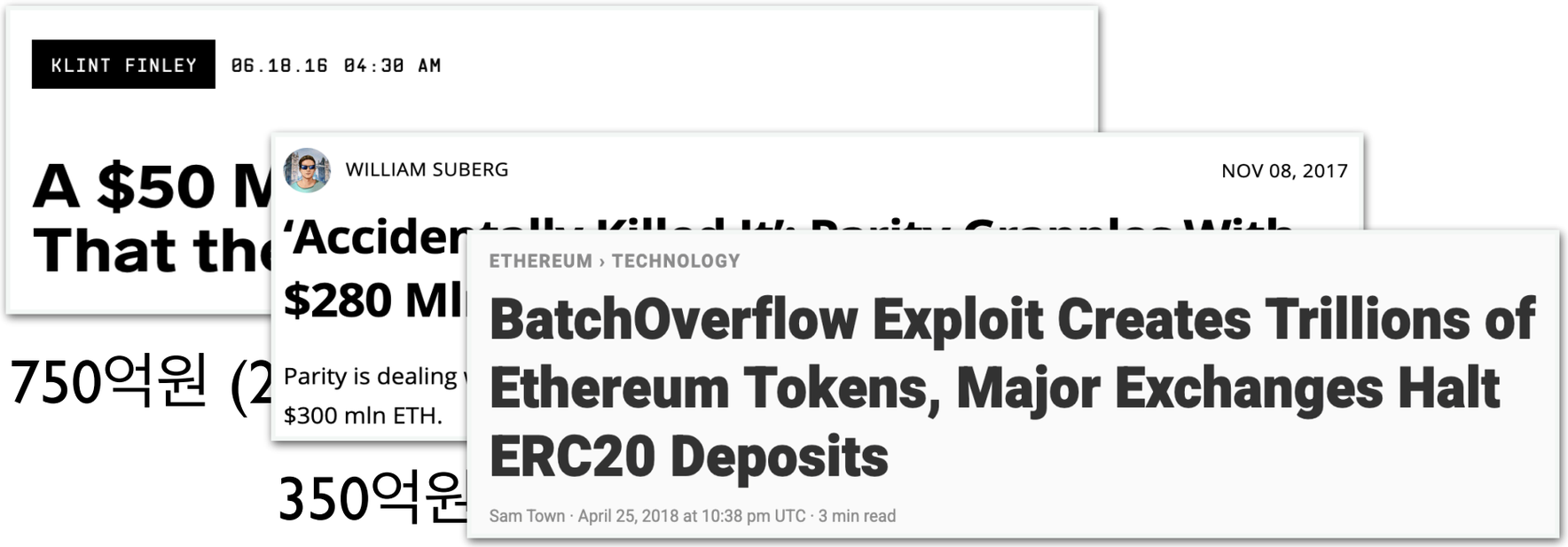
'Accidentally Killed It': Parity Grapples With $280 Mln Locked ETH

Parity is dealing with another code vulnerability which allowed a user to block access to almost $300 mln ETH.

350억원 (2017)

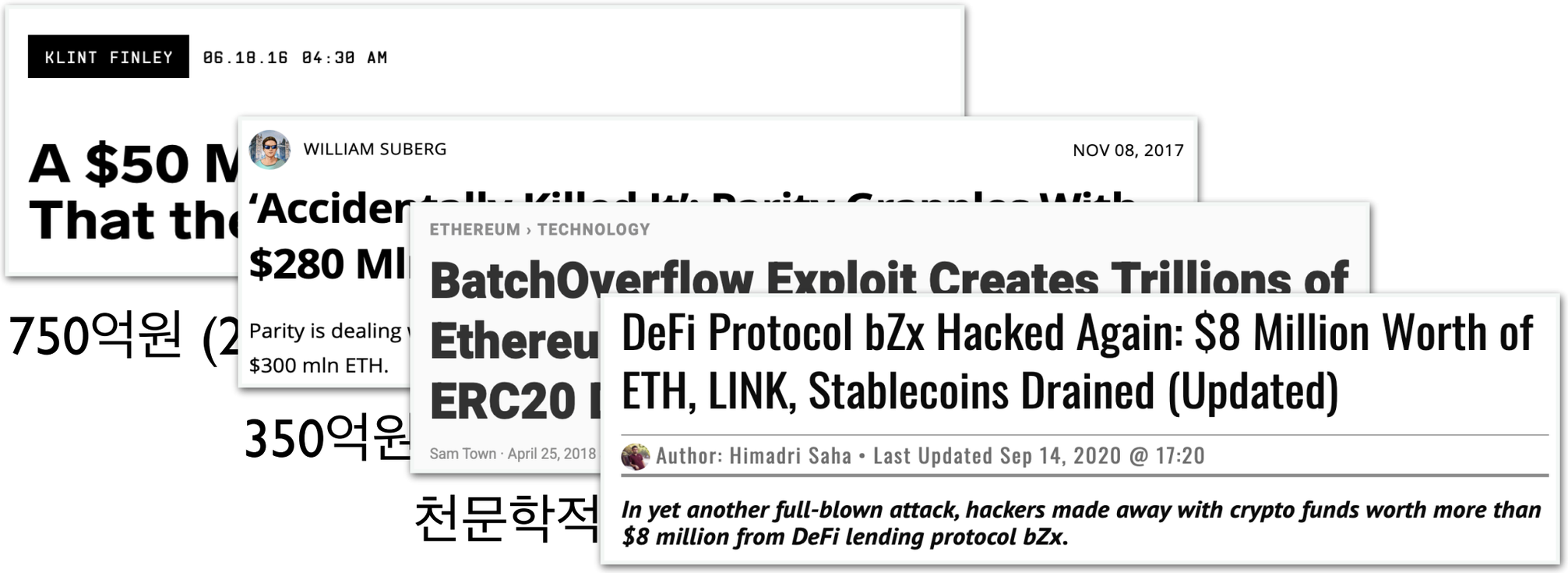
스마트 컨트랙트 특징

- 스마트 컨트랙트는 매우 엄밀한 수준의 안전성 검증이 필요
  - 공격에 성공하면 막대한 금전적 피해가 발생
  - 누구나 온라인에서 소스코드 열람 가능하지만 수정 불가



스마트 컨트랙트 특징

- 스마트 컨트랙트는 매우 엄밀한 수준의 안전성 검증이 필요
  - 공격에 성공하면 막대한 금전적 피해가 발생
  - 누구나 온라인에서 소스코드 열람 가능하지만 수정 불가



SmartMesh (2018)

- SmartMesh 토큰 스마트 컨트랙트의 정수 오버플로우 취약점 (CVE-2018-10376)을 이용하여 천문학적 금액의 토큰을 생성

Timestamp:	592 days 17 hrs ago (Apr-24-2018 07:16:19 PM +UTC)
From:	0xd6a09bdb29e1eafa92a30373c44b09e2e2e0651e
To:	Contract 0x55f93985431fc9304077687a35a1ba103dc1e081 (SmartMesh: Token Sale)
Tokens Transferred: (2 ERC-20 Transfers found)	From 0xdf31a499a5a8358... To 0xdf31a499a5a8358...For 65,133,050,195,990,400,000,000,000,000,000,000,000,000,000,000,000,000,000,000.891004451135422463 ($223,066,504,429,629,000,000,000,000,000,000,000,000,000,000,000,000,000,000.00) SmartMesh (SMT) From 0xdf31a499a5a8358... To 0xd6a09bdb29e1ea... For 50,659,039,041,325,800,000,000,000,000,000,000,000,000,000,000,000,000,000,000.693003461994217473 ($173,496,170,111,934,000,000,000,000,000,000,000,000,000,000,000,000,000,000.00) SmartMesh (SMT)

https://etherscan.io/tx/0x1abab4c8db9a30e703114528e31dee129a3a758f7f8abc3b6494aad3d304e43f

SmartMesh (2018)

```
1   function transferProxy (address from, address to, uint
        value, uint fee) public returns (bool) {
2     if (balance[from] < fee + value)
3       revert();
4     if (balance[to] + value < balance[to] ||
5         balance[msg.sender] + fee < balance[msg.sender])
6       revert();
7     balance[to] += value;
8     balance[msg.sender] += fee;
9     balance[from] -= value + fee;
10    return true;
11  }
```

SmartMesh (2018)

```
1   function transferProxy (address from, address to, uint
        value, uint fee) public returns (bool) {
2     if (balance[from] < fee + value)
3       revert();
4     if (balance[to] + value < balance[to] ||
5         balance[msg.sender] + fee < balance[msg.sender])
6       revert();
7     balance[to] += value;
8     balance[msg.sender] += fee;
9     balance[from] -= value + fee;
10    return true;
11  }
```

송금

SmartMesh (2018)

```
1   function transferProxy (address from, address to, uint
        value, uint fee) public returns
2     if (balance[from] < fee + value)
3       revert();
4     if (balance[to] + value < balance[to] ||
5         balance[msg.sender] + fee < balance[msg.sender])
6       revert();
7     balance[to] += value;
8     balance[msg.sender] += fee;
9     balance[from] -= value + fee;
10    return true;
11  }
```

보내는 사람의 잔고가 충분한지 체크

송금

SmartMesh (2018)

```
1   function transferProxy (address from, address to, uint
        value, uint fee) public returns
2     if (balance[from] < fee + value)
3       revert();
4     if (balance[to] + value < balance[to] ||
5         balance[msg.sender] + fee < balance[msg.sender])
6       revert();
7     balance[to] += value;
8     balance[msg.sender] += fee;
9     balance[from] -= value + fee;
10    return true;
11  }
```

보내는 사람의 잔고가 충분한지 체크

송금

오버플로우 체크

SmartMesh (2018)

```
1   function transferProxy (address from, address to, uint
        value, uint fee) public returns
2     if (balance[from] < fee + value)
3       revert();
4     if (balance[to] + value < balance[to] ||
5         balance[msg.sender] + fee < balance[msg.sender])
6       revert();
7     balance[to] += value;
8     balance[msg.sender] += fee;
9     balance[from] -= value + fee;
10    return true;
11  }
```

보내는 사람의 잔고가 충분한지 체크

송금

오버플로우 체크

(실질적) 오버플로우/언더플로우 발생하지 않음

SmartMesh (2018)

```
1   function transferProxy (address from, address to, uint
        value, uint fee) public returns (bool) {
2     if (balance[from] < fee + value)
3       revert();
4     if (balance[to] + value < balance[to] ||
5         balance[msg.sender] + fee < balance[msg.sender])
6       revert();
7     balance[to] += value;
8     balance[msg.sender] += fee;
9     balance[from] -= value + fee;
10    return true;
11  }
```

SmartMesh (2018)

`balance[from] = balance[to] = balance[msg.sender] = 0`

```
1   function transferProxy (address from, address to, uint
        value, uint fee) public returns (bool) {
2     if (balance[from] < fee + value)
3       revert();
4     if (balance[to] + value < balance[to] ||
5         balance[msg.sender] + fee < balance[msg.sender])
6       revert();
7     balance[to] += value;
8     balance[msg.sender] += fee;
9     balance[from] -= value + fee;
10    return true;
11  }
```

SmartMesh (2018)

```
balance[from] = balance[to] = balance[msg.sender] = 0
value: 8fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
fee  : 7000000000000000000000000000000000000000000000000000000000000001
```

```
1   function transferProxy (address from, address to, uint
        value, uint fee) public returns (bool) {
2     if (balance[from] < fee + value)
3       revert();
4     if (balance[to] + value < balance[to] ||
5         balance[msg.sender] + fee < balance[msg.sender])
6       revert();
7     balance[to] += value;
8     balance[msg.sender] += fee;
9     balance[from] -= value + fee;
10    return true;
11  }
```

SmartMesh (2018)

```
balance[from] = balance[to] = balance[msg.sender] = 0
value: 8fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
fee  : 7000000000000000000000000000000000000000000000000000000000000001
```

```
1   function transferProxy (address from, address to, uint
        value, uint fee) public returns (bool) {
2     if (balance[from] < fee + value)
3       revert();
4     if (balance[to] + value < balance[to] ||
5         balance[msg.sender] + fee < balance[msg.sender])
6       revert();
7     balance[to] += value;
8     balance[msg.sender] += fee;
9     balance[from] -= value + fee;
10    return true;
11  }
```

0!

SmartMesh (2018)

```
balance[from] = balance[to] = balance[msg.sender] = 0
value: 8fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
fee  : 7000000000000000000000000000000000000000000000000000000000000001
```

```
1   function transferProxy (address from, address to, uint
          value, uint fee) public returns (bool) {
2     if (balance[from] < fee + value) false
3       revert();
4     if (balance[to] + value < balance[to] ||
5           balance[msg.sender] + fee < balance[msg.sender])
6       revert();
7     balance[to] += value;
8     balance[msg.sender] += fee;
9     balance[from] -= value + fee;
10    return true;
11  }
```

0!

SmartMesh (2018)

```
balance[from] = balance[to] = balance[msg.sender] = 0
value: 8fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
fee  : 7000000000000000000000000000000000000000000000000000000000000001
```

```
1   function transferProxy (address from, address to, uint
          value, uint fee) public returns (bool) {
2       if (balance[from] < fee + value)
3           revert();
4       if (balance[to] + value < balance[to] ||
5           balance[msg.sender] + fee < balance[msg.sender])
6           revert();
7       balance[to] += value;
8       balance[msg.sender] += fee;
9       balance[from] -= value + fee;
10      return true;
11  }
```

false

false

0!

SmartMesh (2018)

```
balance[from] = balance[to] = balance[msg.sender] = 0
value: 8fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
fee  : 7000000000000000000000000000000000000000000000000000000000000001
```

```
1   function transferProxy (address from, address to, uint
          value, uint fee) public returns (bool) {
2       if (balance[from] < fee + value) revert();
3
4       if (balance[to] + value < balance[to] ||
5           balance[msg.sender] + fee < balance[msg.sender])
6           revert();
7       balance[to] += value;
8       balance[msg.sender] += fee;
9       balance[from] -= value + fee;
10      return true;
11  }
```

false

false

0!

8fffff…ff

SmartMesh (2018)

```
balance[from] = balance[to] = balance[msg.sender] = 0
value: 8fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
fee  : 7000000000000000000000000000000000000000000000000000000000000001
```

```
1   function transferProxy (address from, address to, uint
          value, uint fee) public returns (bool) {
2       if (balance[from] < fee + value)
3           revert();
4       if (balance[to] + value < balance[to] ||
5           balance[msg.sender] + fee < balance[msg.sender])
6           revert();
7       balance[to] += value;
8       balance[msg.sender] += fee;
9       balance[from] -= value + fee;
10      return true;
11  }
```

false

false

0!

8fffff…ff

700…00

SmartMesh (2018)

```
balance[from] = balance[to] = balance[msg.sender] = 0
value: 8fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
fee  : 7000000000000000000000000000000000000000000000000000000000000001
```

```
1   function transferProxy (address from, address to, uint
          value, uint fee) public returns (bool) {
2       if (balance[from] < fee + value)
3           revert();
4       if (balance[to] + value < balance[to] ||
5           balance[msg.sender] + fee < balance[msg.sender])
6           revert();
7       balance[to] += value;
8       balance[msg.sender] += fee;
9       balance[from] -= value + fee;
10      return true;
11  }
```

false

0!

false

8fffff…ff

700…00

0!

SmartMesh (2018)

```
balance[from] = balance[to] = balance[msg.sender] = 0
value: 8fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
fee  : 7000000000000000000000000000000000000000000000000000000000000001
```

```
1   function transferProxy (address from, address to, uint
          value, uint fee) public returns (bool) {
2       if (balance[from] < fee + value)
3           revert();
4       if (balance[to] + value < balance[to] ||
5           balance[msg.sender] + fee < balance[msg.sender])
6           revert();
7       balance[to] += value;
8       balance[msg.sender] += fee;
9       balance[from] -= value + fee;
10      return true;
11  }
```

false

false

0!

8fffff…ff

700…00

0!

(See “VeriSmart: A Highly Precise Safety Verifier for Ethereum Smart Contracts. IEEE S&P 2020”)

사례 2: Memory Bugs in C/C++

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

Memory-Leak

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

Use-After-Free

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

Double-Free

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

메모리 관리 오류

- C/C++ 프로그램에서 가장 빈번하게 발생

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

- 소프트웨어 결함의 주요 원인이지만 탐지 및 수정이 까다로움

(I) 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;
```

(1) Linux Kernel

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

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

  goto err;
}
```

메모리 할당

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

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

(1) Linux Kernel

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

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

  goto err;
}
```

메모리 할당

메모리 해제

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

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

(1) 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;
```

메모리 할당

메모리 해제

메모리 중복 해제
(double-free)

(1) 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;
```

메모리 할당

메모리 해제

메모리 중복 해제
(double-free)

(I) Linux Kernel

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

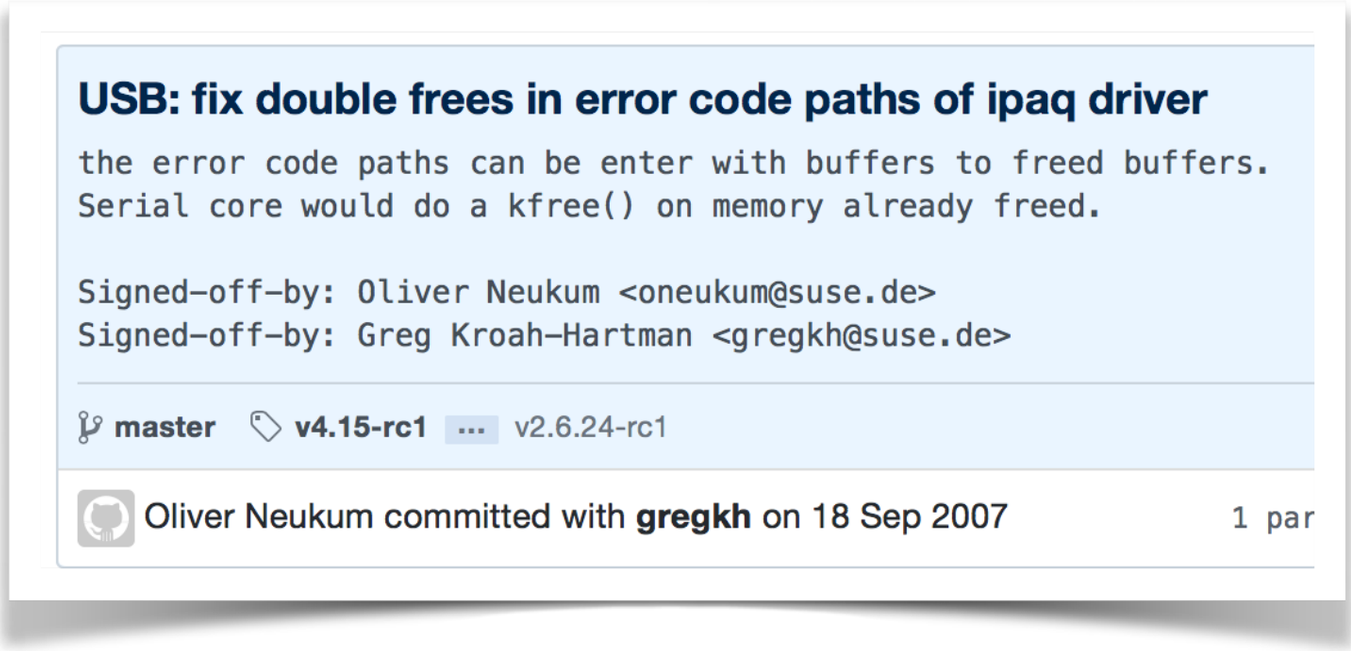
```
the error code paths can be enter with buffers to freed buffers.
Serial core would do a kfree() on memory already freed.

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;
```

(I) Linux Kernel



수동 디버깅의 문제 1:
오류가 제거되었는지 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;
```

(1) 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

9개월 후에 다시 오류 수정을 시도

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

(I) Linux Kernel

memory leak

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

수동 디버깅의 문제 2:
오류 수정 과정에서 새로운 오류가 발생

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.

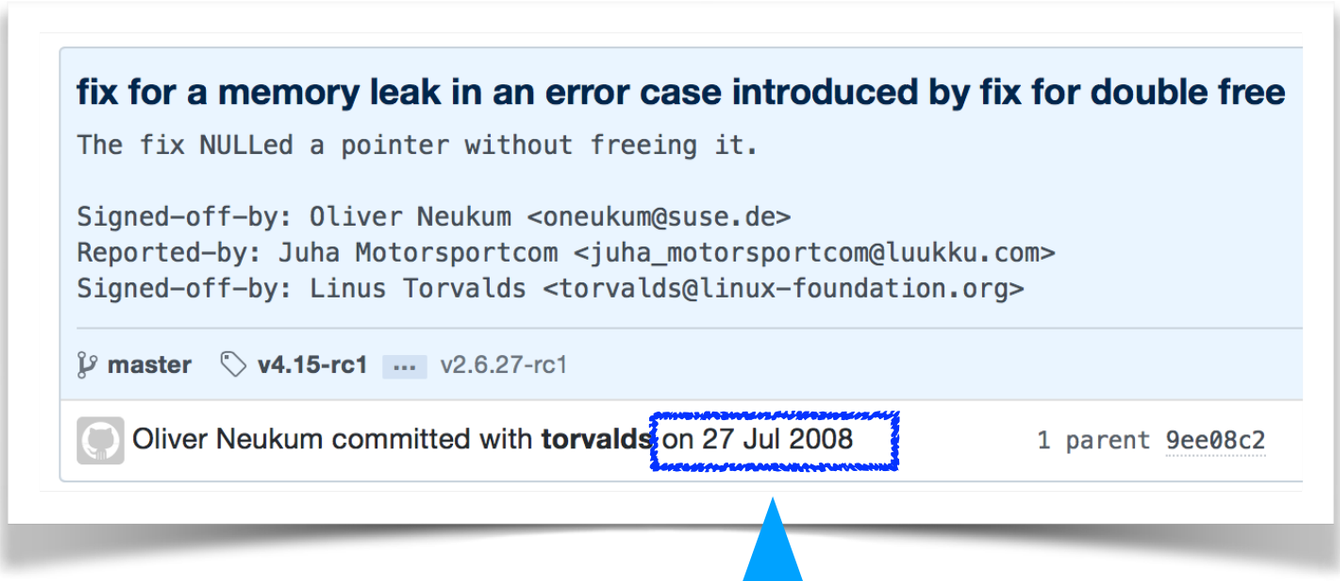
Signed-off-by: Oliver Neukum <oneukum@suse.de>
Signed-off-by: Greg Kroah-Hartman <gregkh@suse.de>

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

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

9개월 후에 다시 오류 수정을 시도

(1) Linux Kernel

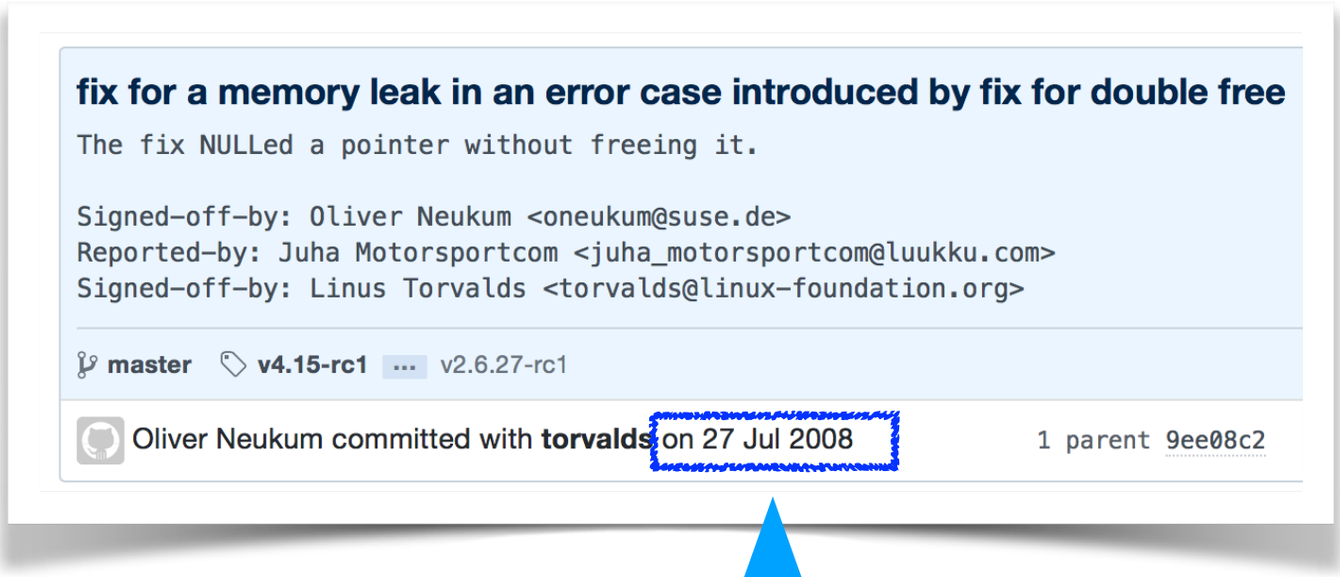


오류 발견에서 수정까지 총 **10개월** 소요

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

(I) Linux Kernel

수동 디버깅의 문제 3:
오류는 제거했지만 코드 품질이 떨어짐



오류 발견에서 수정까지 총 **10개월** 소요

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

정적 분석 기반 오류 탐지 및 수정

```
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); // double-free
  free(out);// double-free
  return;
```

탐지 & 수정

SAVER

✓개발생산성↑
✓SW품질↑
✓안전성 보장

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

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

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

(2) Memory Leak in Swoole

```
1  int swTableColumn_add(swTable *table, ...) {
2    col = sw_malloc(sizeof(swTableColumn));
3    if (type == SW_TABLE_INT)
4        col->size = 1;
5    col->index = table->size;
6    return swHashMap_add(table->columns, ..., col);
7  }
8
9  int swHashMap_add(swHashMap *hmap, ..., void *data) {
10   node = sw_malloc(sizeof(swHashMap_node));
11   if (node == NULL)
12       return SW_ERR;
13   node->data = data;
14   swHashMap_node_add(hmap, ... node);
15   return SW_OK;
16 }
```


```
Memory Leak:
An object allocated at line 2
becomes unreachable after line 7
```

(2) Memory Leak in Swoole

```
1   int swTableColumn_add(swTable *table, ...) {
2     col = sw_malloc(sizeof(swTableColumn));
3     if (type == SW_TABLE_INT)
4         col->size = 1;
5     col->index = table->size;
6     return swHashMap_add(table->columns, ..., col);
7   }
8
9   int swHashMap_add(swHashMap *hmap, ..., void *data) {
10    node = sw_malloc(sizeof(swHashMap_node));
11    if (node == NULL)
12        return SW_ERR;
13    node->data = data;
14    swHashMap_node_add(hmap, ... node);
15    return SW_OK;
16  }
```

정상 실행 경로

Infer

```
Memory Leak:
An object allocated at line 2
becomes unreachable after line 7
```

(2) Memory Leak in Swoole

```
1  int swTableColumn_add(swTable *table, ...) {
2    col = sw_malloc(sizeof(swTableColumn));
3    if (type == SW_TABLE_INT)
4        col->size = 1;
5    col->index = table->size;
6    return swHashMap_add(table->columns, ..., col);
7  }
8
9  int swHashMap_add(swHashMap *hmap, ..., void *data) {
10   node = sw_malloc(sizeof(swHashMap_node));
11   if (node == NULL)
12       return SW_ERR;
13   node->data = data;
14   swHashMap_node_add(hmap, ... node);
15   return SW_OK;
16 }
```


```
Memory Leak:
An object allocated at line 2
becomes unreachable after line 7
```

(2) Memory Leak in Swoole

```
1  int swTableColumn_add(swTable *table, ...) {
2    col = sw_malloc(sizeof(swTableColumn));
3    if (type == SW_TABLE_INT)
4        col->size = 1;
5    col->index = table->size;
6    return swHashMap_add(table->columns, ..., col);
7  }
8
9  int swHashMap_add(swHashMap *hmap, ..., void *data) {
10   node = sw_malloc(sizeof(swHashMap_node));
11   if (node == NULL)
12       return SW_ERR;
13   node->data = data;
14   swHashMap_node_add(hmap, ... node);
15   return SW_OK;
16 }
```

Memory leak

Infer

```
Memory Leak:
An object allocated at line 2
becomes unreachable after line 7
```

(2) Memory Leak in Swoole

```
1  int swTableColumn_add(swTable *table, ...) {
2    if ((int ret = swHashMap_add(table->columns, ..., col)) == SW_ERR)
3        free(p);
4    return ret;
5    col->index = table->size;
6    return swHashMap_add(table->columns, ..., col);
7  }
8
9  int swHashMap_add(swHashMap *hmap, ..., void *data) {
10   node = sw_malloc(sizeof(swHashMap_node));
11   if (node == NULL)
12       return SW_ERR;
13   node->data = data;
14   swHashMap_node_add(hmap, ... node);
15   return SW_OK;
16 }
```

SAVER

Infer

```
Memory Leak:
An object allocated at line 2
becomes unreachable after line 7
```

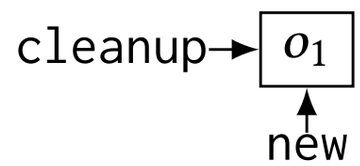
(3) Use-After-Free in Binutils

```
1   struct node *cleanup; // list of objects to be deallocated
2   struct node *first = NULL;
3   for (...) {
4       struct node *new = xmalloc(sizeof(*new));
5       make_cleanup(new); // add new to the cleanup list
6       new->name = ...;
7       ...
8       if (...) {
9           first = new;
10
11          continue;
12      }
13    /* potential use-after-free: `first->name` */
14  (-)   if (first == NULL || new->name != first->name)
15
16          continue;
17    do_cleanups(); // deallocate all objects in cleanup
18  }
```

use-after-free

(3) Use-After-Free in Binutils

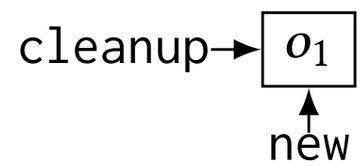
```
1  struct node *cleanup; // list of objects to be deallocated
2  struct node *first = NULL;
3  for (...) {
4      struct node *new = xmalloc(sizeof(*new));
5      make_cleanup(new); // add new to the cleanup list
6      new->name = ...;
7      ...
8      if (...) {
9          first = new;
10
11         continue;
12     }
13   /* potential use-after-free: `first->name` */
14 (-)  if (first == NULL || new->name != first->name)
15
16         continue;
17   do_cleanups(); // deallocate all objects in cleanup
18 }
```

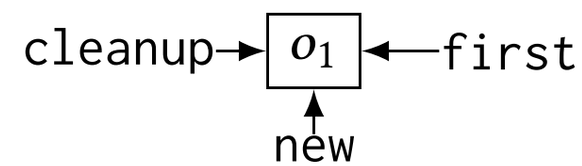


use-after-free

(3) Use-After-Free in Binutils

```
1  struct node *cleanup; // list of objects to be deallocated
2  struct node *first = NULL;
3  for (...) {
4      struct node *new = xmalloc(sizeof(*new));
5      make_cleanup(new); // add new to the cleanup list
6      new->name = ...;
7      ...
8      if (...) {
9          first = new;
10
11         continue;
12     }
13   /* potential use-after-free: `first->name` */
14 (-)  if (first == NULL || new->name != first->name)
15
16         continue;
17   do_cleanups(); // deallocate all objects in cleanup
18 }
```

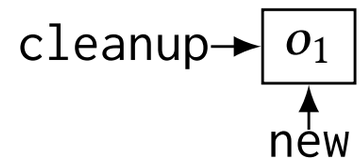


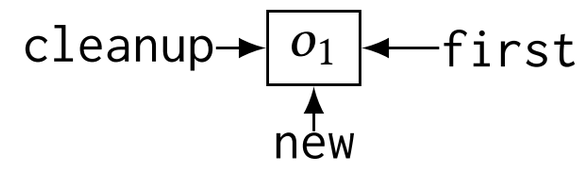


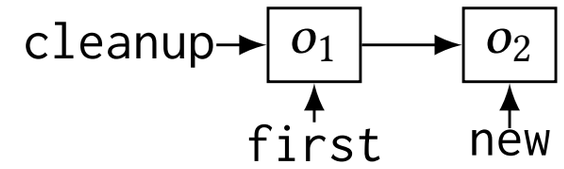
use-after-free

(3) Use-After-Free in Binutils

```
1  struct node *cleanup; // list of objects to be deallocated
2  struct node *first = NULL;
3  for (...) {
4      struct node *new = xmalloc(sizeof(*new));
5      make_cleanup(new); // add new to the cleanup list
6      new->name = ...;
7      ...
8      if (...) {
9          first = new;
10
11         continue;
12     }
13   /* potential use-after-free: `first->name` */
14 (-)  if (first == NULL || new->name != first->name)
15
16         continue;
17   do_cleanups(); // deallocate all objects in cleanup
18 }
```





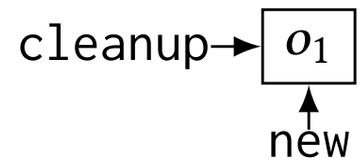


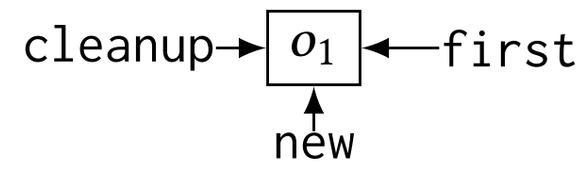
use-after-free

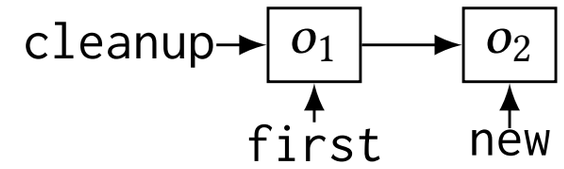
(3) Use-After-Free in Binutils

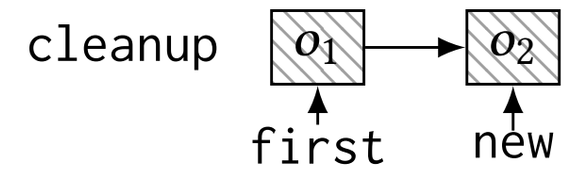
```
1   struct node *cleanup; // list of objects to be deallocated
2   struct node *first = NULL;
3   for (...) {
4       struct node *new = xmalloc(sizeof(*new));
5       make_cleanup(new); // add new to the cleanup list
6       new->name = ...;
7       ...
8       if (...) {
9           first = new;
10
11          continue;
12      }
13    /* potential use-after-free: `first->name` */
14  (-)   if (first == NULL || new->name != first->name)
15
16          continue;
17    do_cleanups(); // deallocate all objects in cleanup
18  }
```

use-after-free





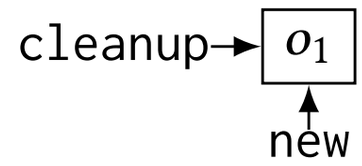


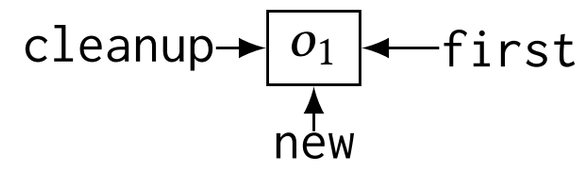


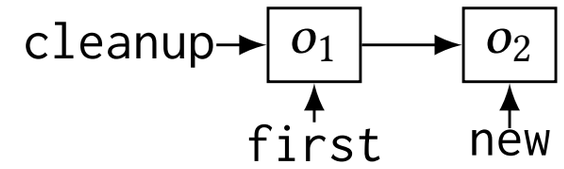
(3) Use-After-Free in Binutils

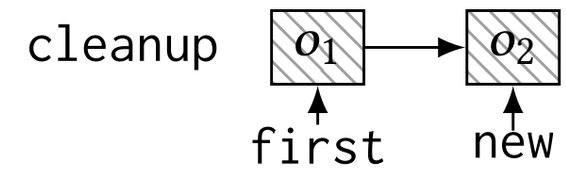
```
1   struct node *cleanup; // list of objects to be deallocated
2   struct node *first = NULL;
3   for (...) {
4       struct node *new = xmalloc(sizeof(*new));
5       make_cleanup(new); // add new to the cleanup list
6       new->name = ...;
7       ...
8       if (...) {
9           first = new;
10
11          continue;
12      }
13    /* potential use-after-free: `first->name */
14  (-)   if (first == NULL || new->name != first->name)
15
16          continue;
17    do_cleanups(); // deallocate all objects in cleanup
18  }
```

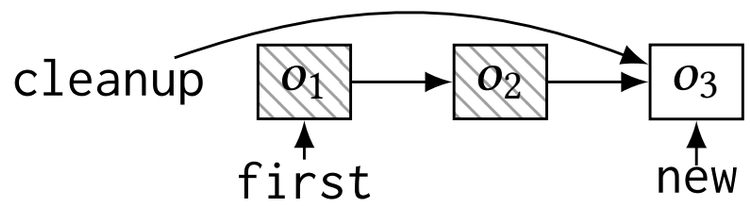
use-after-free











정적 분석 기반 탐지 및 수정

```
1   struct node *cleanup; // list of objects to be deallocated
2   struct node *first = NULL;
3   for (...) {
4       struct node *new = xmalloc(sizeof(*new));
5       make_cleanup(new); // add new to the cleanup list
6       new->name = ...;
7       ...
8       if (...) {
9           first = new;
10  (+)     tmp = first->name;
11          continue;
12      }
13    /* potential use-after-free: `first->name` */
14  (-)   if (first == NULL || new->name != first->name)
15  (+)   if (first == NULL || new->name != tmp)
16          continue;
17    do_cleanups(); // deallocate all objects in cleanup
18  }
```

cf) Rust

- A memory-safe langauge for systems programming
- Heavy use of static analysis technology (type system)

Install Learn Playground Tools Governance Community Blog English (en-US)

Rust

A language empowering everyone
to build reliable and efficient software.

GET STARTED

Version 1.58.1

Why Rust?

Performance

Rust is blazingly fast and memory-efficient: with no runtime or garbage collector, it can power performance-critical services, run on embedded devices, and easily integrate with other languages.

Reliability

Rust's rich type system and ownership model guarantee memory-safety and thread-safety — enabling you to eliminate many classes of bugs at compile-time.

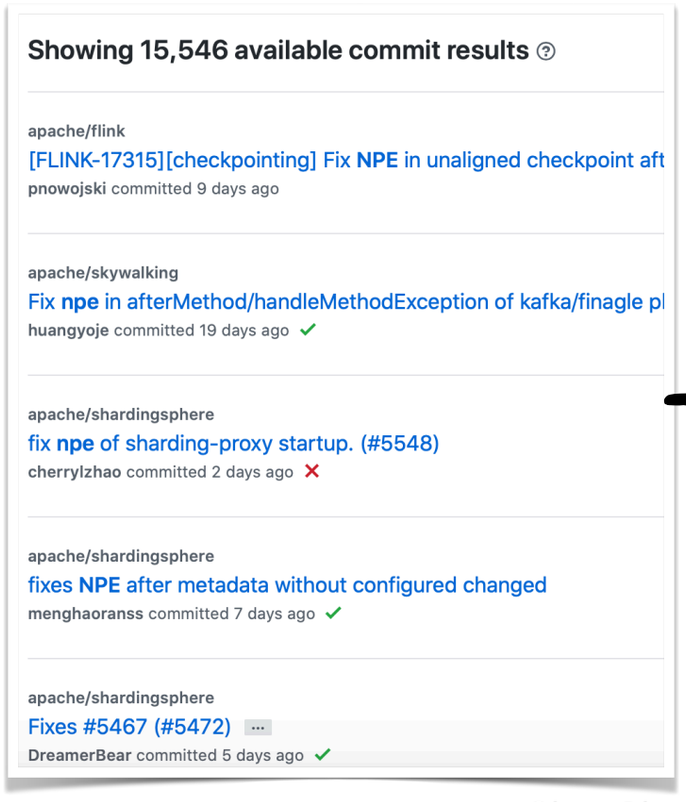
Productivity

Rust has great documentation, a friendly compiler with useful error messages, and top-notch tooling — an integrated package manager and build tool, smart multi-editor support with auto-completion and type inspections, an auto-formatter, and more.

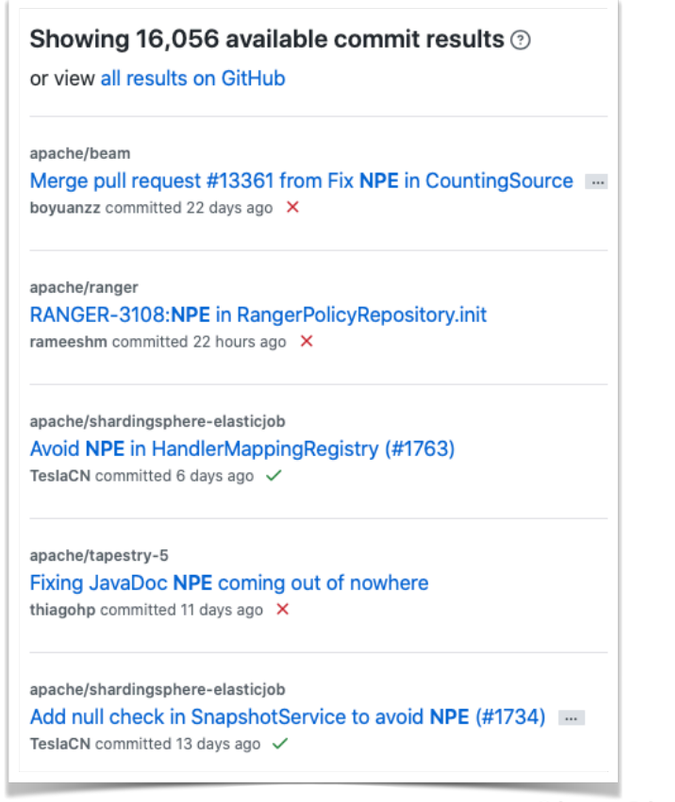
사례 3: Null Pointer Exceptions

- Java에서 가장 흔히 발생하는 오류
  - 안드로이드 exception 가운데 약 40%
  - Mozila, Apache 메모리 오류 중 약 37%

- **Top 10 Java Errors by Frequency** were
  - NullPointerException
  - NumberFormatException
  - IllegalArgumentException
  - RuntimeException



Apache NPE Fix Commits (5월 15일)



Apache NPE Fix Commits (12월 10일)

Apache 재단에서 1년동안 약 1000건의 NPE 발생

Null Pointer Exceptions

- 테스팅, 퍼징으로 탐지가 어려움

		EvoSuite			
Project	**Bug ID**	**#** tests (**#** crashes)	**#** NPEs	`coverage`(%)	`reproduction`(%)
ActiveMQ Artemis	586abba9	-	-	-	-
Aries JPA	97cb979d	3(2)	0	50.0	0.0
Avro	a3e05bee	122(51)	9	55.8	60.0
Commons Collections	796114ea	83(10)	2	51.5	0.0
Commons Configuration	a76a3a65	21(3)	3	41.1	100.0
Commons DBCP	2ee3c53e	27(0)	0	70.8	0.0
Commons IO	89bfb64c	16(1)	0	95.8	0.0
Commons Pool	11521c1f	19(0)	1	29.0	0.0
CXF	209407e0	79(22)	0	24.6	0.0
HttpComponents Client	1026a1e5	6(1)	1	76.0	0.0
Johnzon	847a4268	5(1)	0	100.0	0.0
Log4j2	6a233016	30(7)	7	91.8	0.0
OpenNLP	60792b8f	6(2)	0	50.0	0.0
PDFBox	55586aad	4(1)	1	100.0	100.0
Qpid Proton-J	02998b38	6(1)	0	37.7	0.0
RocketMQ	f5a119f1	-	-	-	-
XML Graphics FOP	10e0d1c2	43(11)	6	89.5	0.0

Null Pointer Exceptions

- 특정 시나리오에서 발생 (테스팅으로 탐지가 어려운 이유)

(a) Buggy program

```
1  boolean compare(int row, Column<?> temp, Column<?> org) {
2      Object o1 = org.get(row);
3      Object o2 = temp.get(temp.size() - 1);
4      return o1.equals(o2); // NPE
5  }
```

(b) NPE-triggering input

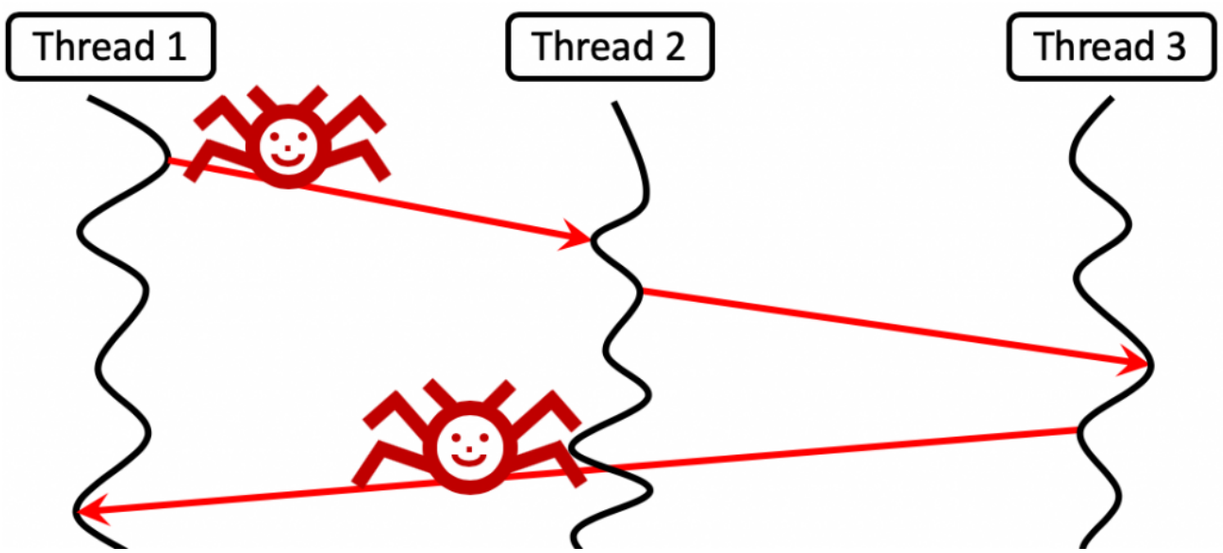
```
int missing = IntColumnType.missingValueIndicator();
Table t1 = Table.create("T1",
    IntColumn.create("Id", 0, 0),
    IntColumn.create("ChildId", missing, missing));
t1.dropDuplicateRows(); // compare is invoked inside
```

기타 사례들

- Type errors in dynamic languages (e.g., Python, JavaScript)

	Type	Attribute	Value	Key	Import
stackoverflow	**31.5%**	19.4%	27.8%	8.3%	13.0%
repos	**29.2%**	19.4%	28.2%	12.9%	10.3%

- Concurrency bugs (races, deadlocks, etc)



RacerD: Compositional Static Race Detection

SAM BLACKSHEAR, Facebook, USA
NIKOS GOROGIANNIS, Facebook and Middlesex University London, UK
PETER W. O'HEARN, Facebook and University College London, UK
ILYA SERGEY*, Yale-NUS College and University College London, Singapore and UK

Automatic static detection of data races is one of the most basic problems in reasoning about concurrency. We present RacerD—a static program analysis for detecting data races in Java programs which is fast, can scale to large code, and has proven effective in an industrial software engineering scenario. To our knowledge, RacerD is the first inter-procedural, compositional data race detector which has been empirically shown to have non-trivial precision and impact. Due to its compositionality, it can analyze code changes quickly, and this allows it to perform *continuous reasoning* about a large, rapidly changing codebase as part of deployment within a continuous integration ecosystem. In contrast to previous static race detectors, its design favors reporting high-confidence bugs over ensuring their absence. RacerD has been in deployment for over a year at Facebook, where it has flagged over 2500 issues that have been fixed by developers before reaching production. It has been important in enabling the development of new code as well as fixing old code: it helped support the conversion of part of the main Facebook Android app from a single-threaded to a multi-threaded architecture. In this paper we describe RacerD's design, implementation, deployment and impact.

CCS Concepts: • **Theory of computation → Program analysis**; • **Software and its engineering → Concurrent programming structures**;

Additional Key Words and Phrases: Concurrency, Static Analysis, Race Freedom

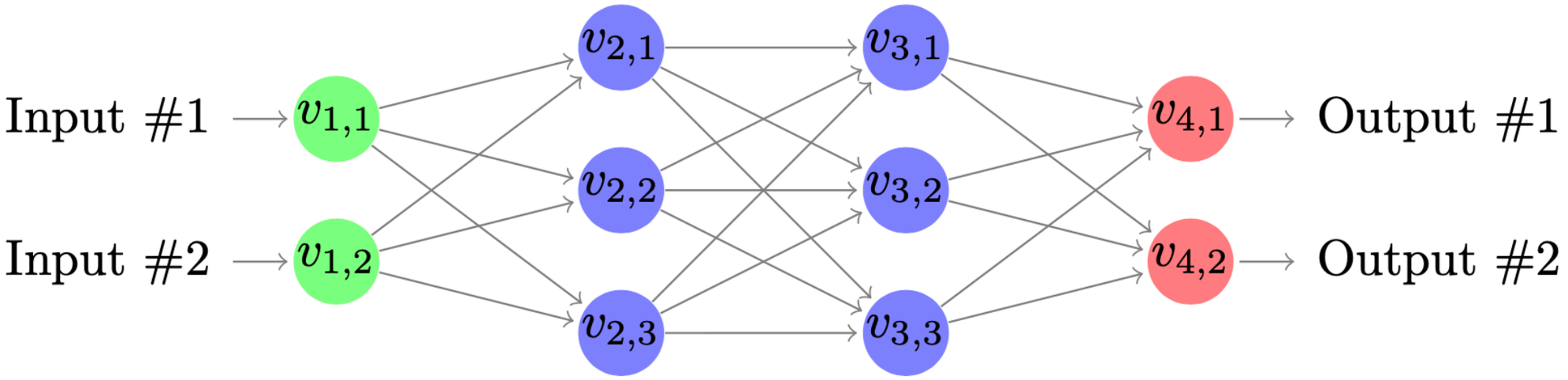
ACM Reference Format:
Sam Blackshear, Nikos Gorogiannis, Peter W. O'Hearn, and Ilya Sergey. 2018. RacerD: Compositional Static Race Detection. *Proc. ACM Program. Lang.* 2, OOPSLA, Article 144 (November 2018), 28 pages. https://doi.org/10.1145/3276514

1 INTRODUCTION

Concurrent programming is hard. It is difficult for humans to think about the vast number of potential interactions between processes, and this makes concurrent programs hard to get right in

기타 사례들

- 인공신경망 안전성 검증



$$\{\ |\boldsymbol{x}-\boldsymbol{c}| \leqslant \mathbf{0.1}\ \}$$

$$\boldsymbol{r}_1 \leftarrow f(\boldsymbol{x})$$

$$\boldsymbol{r}_2 \leftarrow f(\boldsymbol{c})$$

$$\{\ \mathsf{class}(\boldsymbol{r}_1) = \mathsf{class}(\boldsymbol{r}_2)\ \}$$

Summary

- Strengths of static analysis?
- Weaknesses?

Part 1. 프로그램 분석 개괄

Part 2. 정적 분석 적용 사례

Part 3. 정적 분석 원리

Part 4. 정적 분석 구현

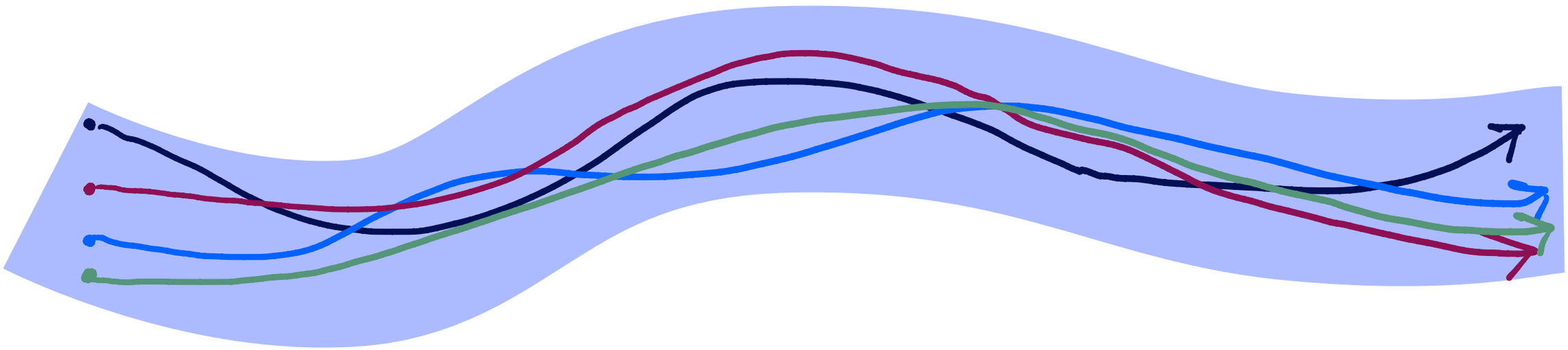
Static Program Analysis

A general method for
automatic and sound approximation of
sw run-time behaviors
before the execution

- "before": statically, without running sw
- "automatic": sw analyzes sw
- "sound": all possibilities into account
- "approximation": cannot be exact
- "general": for any source language and property
  - C, C++, C#, F#, Java, JavaScript, ML, Scala, Python, JVM, Dalvik, x86, Excel, etc
  - "buffer-overrun?", "memory leak?", "type errors?", "x = y at line 2?", "memory use $\leq \boldsymbol{2K}$?", etc

정적 분석 원리

error zone



error zone

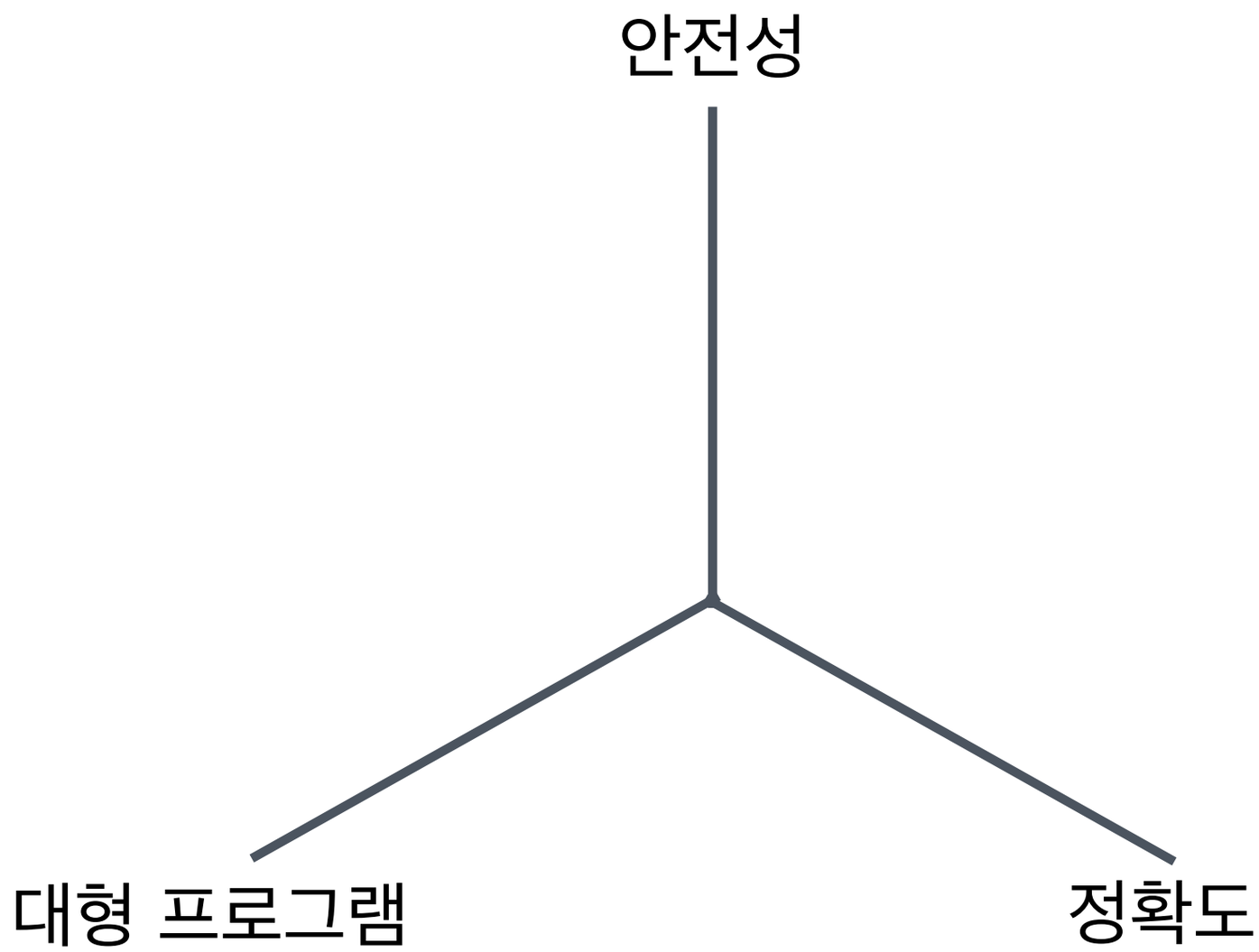
정적 분석 원리

$$30 \times 12 + 11 \times 9 = ?$$

정적 분석 원리

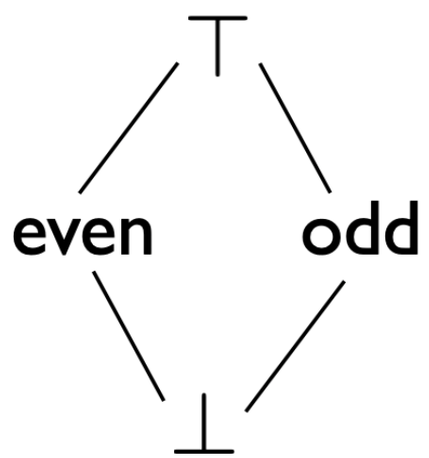
```
void f (int x) {
    y = x * 12 + 9 * 11;
    assert (y % 2 == 0);
}
```

정적 분석의 세 가지 성능



Static Analysis Example 1

- Simple parity domain:



- Abstraction (α) and concretization (γ) functions:

- Join operation:

cf) Partially Ordered Set

Definition (Partial Order)

We say a binary relation $\sqsubseteq$ is a partial order on a set D iff $\sqsubseteq$ is

- reflexive: $\forall p \in D.\ p \sqsubseteq p$
- transitive: $\forall p, q, r \in D.\ p \sqsubseteq q \ \wedge \ q \sqsubseteq r \implies p \sqsubseteq r$
- anti-symmetric: $\forall p, q \in D.\ p \sqsubseteq q \ \wedge \ q \sqsubseteq p \implies p = q$

We call such a pair $(D, \sqsubseteq)$ partially ordered set, or poset.

Definition (Least Upper Bound)

Let $(D, \sqsubseteq)$ be a partially ordered set and let Y be a subset of D. An upper bound of Y is an element d of D such that

$$\forall d' \in Y.\ d' \sqsubseteq d.$$

An upper bound d of Y is a least upper bound if and only if $d \sqsubseteq d'$ for every upper bound d' of Y. The least upper bound of Y is denoted by $\bigsqcup Y$. The least upper bound (lub, join) of a and b is written as $a \sqcup b$.

cf) Complete Partial Order

Definition (Chain)

Let $(\boldsymbol{D}, \sqsubseteq)$ be a poset and $\boldsymbol{Y}$ a subset of $\boldsymbol{D}$. $\boldsymbol{Y}$ is called a chain if $\boldsymbol{Y}$ is totally ordered:

$$\forall \boldsymbol{y_1}, \boldsymbol{y_2} \in \boldsymbol{Y}. \boldsymbol{y_1} \sqsubseteq \boldsymbol{y_2} \text{ or } \boldsymbol{y_2} \sqsubseteq \boldsymbol{y_1}.$$

Example

Consider the poset $(\mathcal{P}(\{\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}\}), \subseteq)$.

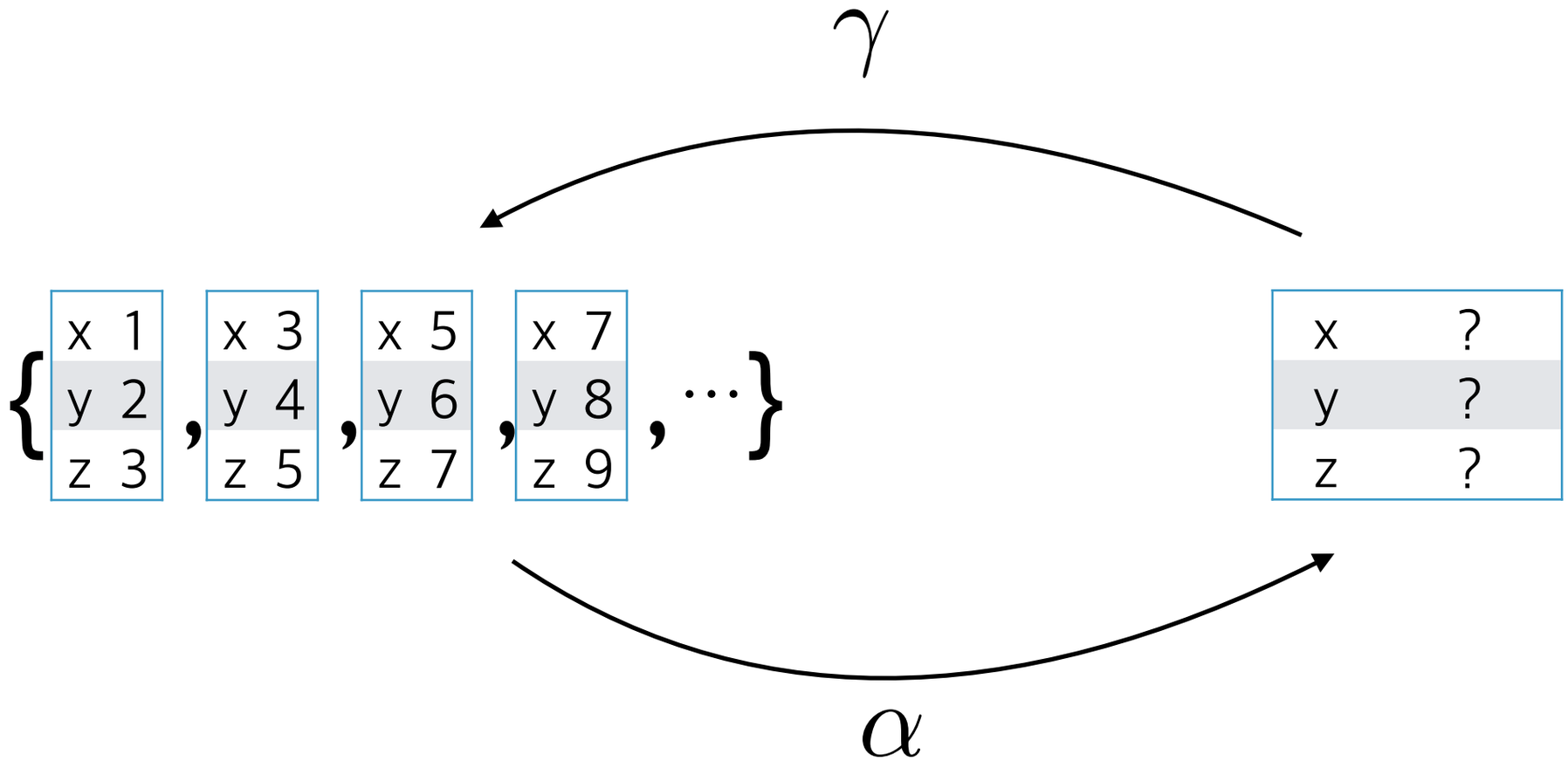
- $\boldsymbol{Y_1} = \{\emptyset, \{\boldsymbol{a}\}, \{\boldsymbol{a}, \boldsymbol{c}\}\}$
- $\boldsymbol{Y_2} = \{\emptyset, \{\boldsymbol{a}\}, \{\boldsymbol{c}\}, \{\boldsymbol{a}, \boldsymbol{c}\}\}$

Definition (CPO)

A poset $(\boldsymbol{D}, \sqsubseteq)$ is a CPO, if every chain $\boldsymbol{Y} \subseteq \boldsymbol{D}$ has $\bigsqcup \boldsymbol{Y} \in \boldsymbol{D}$.

Static Analysis Example 1

- Memory state based on simple parity domain:



- Join operation:

요약 의미 (Abstract Semantics)

- 홀짝 공간에서의 덧셈, 뺄셈, 곱셈:

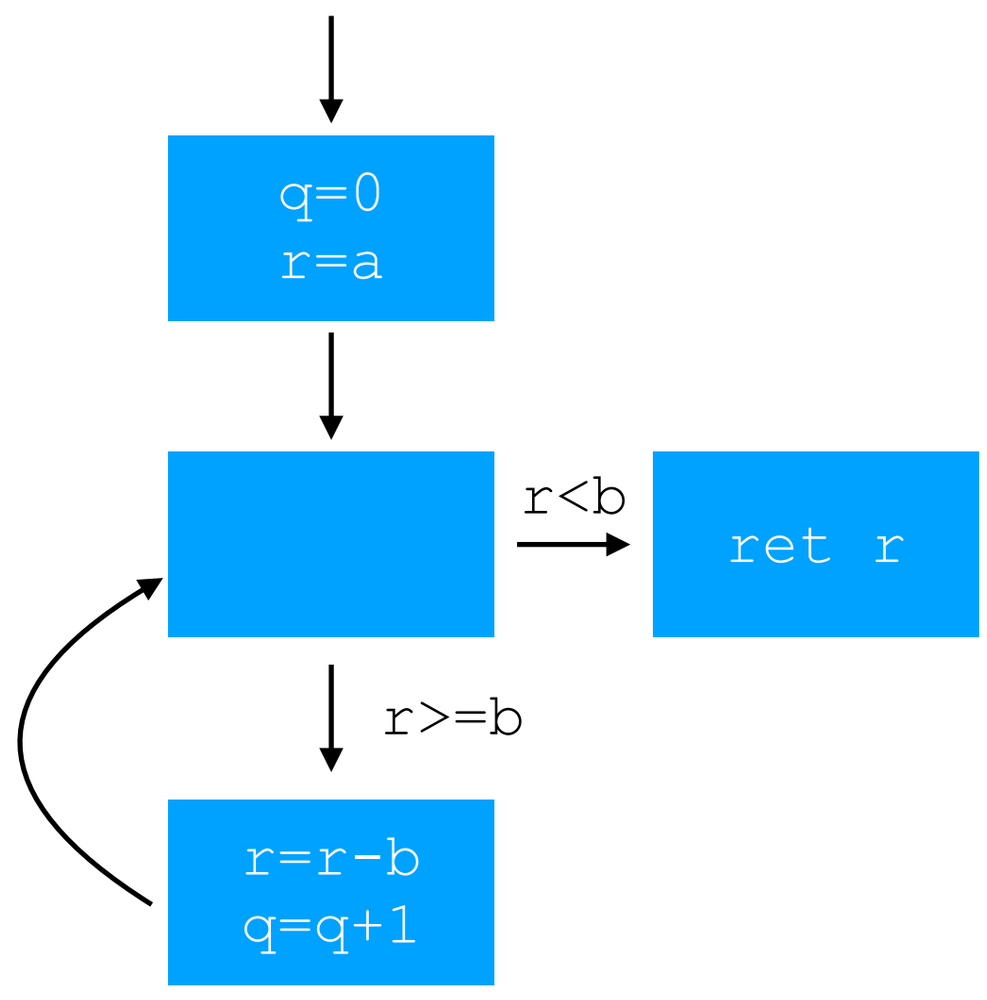
+	top	even	odd	bottom
top				
even				
odd				
bottom				

-	top	even	odd	bottom
top				
even				
odd				
bottom				

*	top	even	odd	bottom
top				
even				
odd				
bottom				

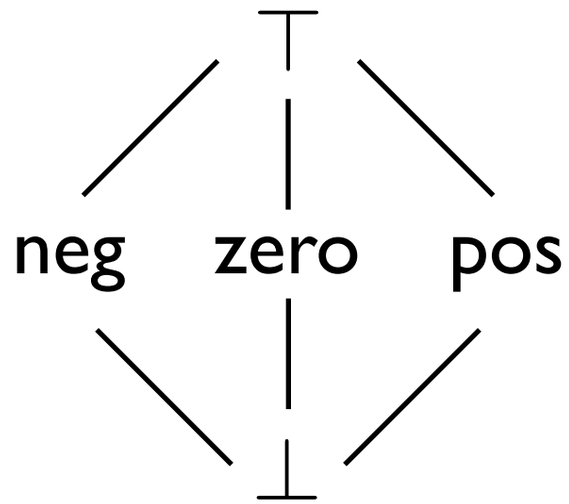
고정점 계산 예제

```
// a >= 0, b >= 0
int mod (int a, int b) {
   int q = 0;
   int r = a;
   while (r >= b) {
      r = r - b;
      q = q + 1;
   }
   return r;
}
```



Static Analysis Example 2

- Simple sign domain:

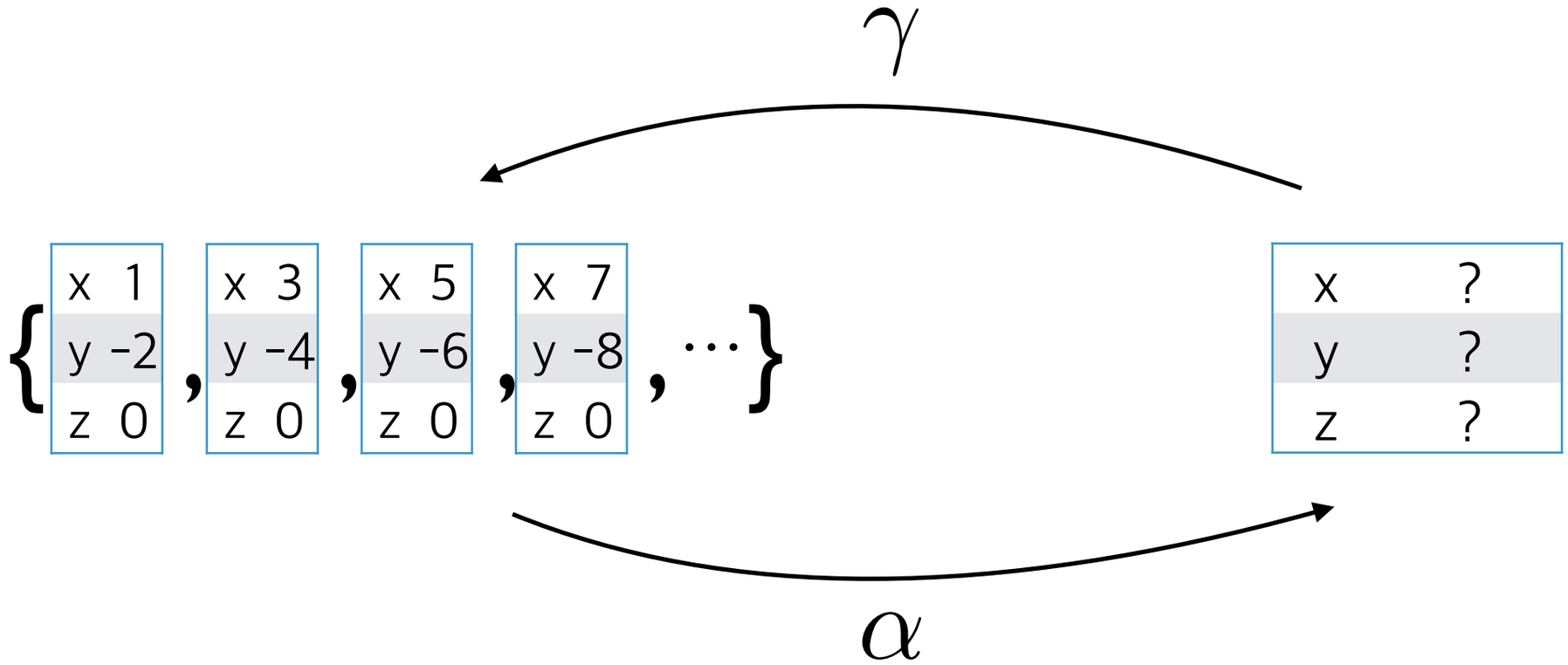


- Abstraction (α) and concretization (γ) functions:

- Join operation:

Static Analysis Example 2

- Memory state abstraction:



- Join operation:

요약 의미 (Abstract Semantics)

- 덧셈, 뺄셈, 곱셈:

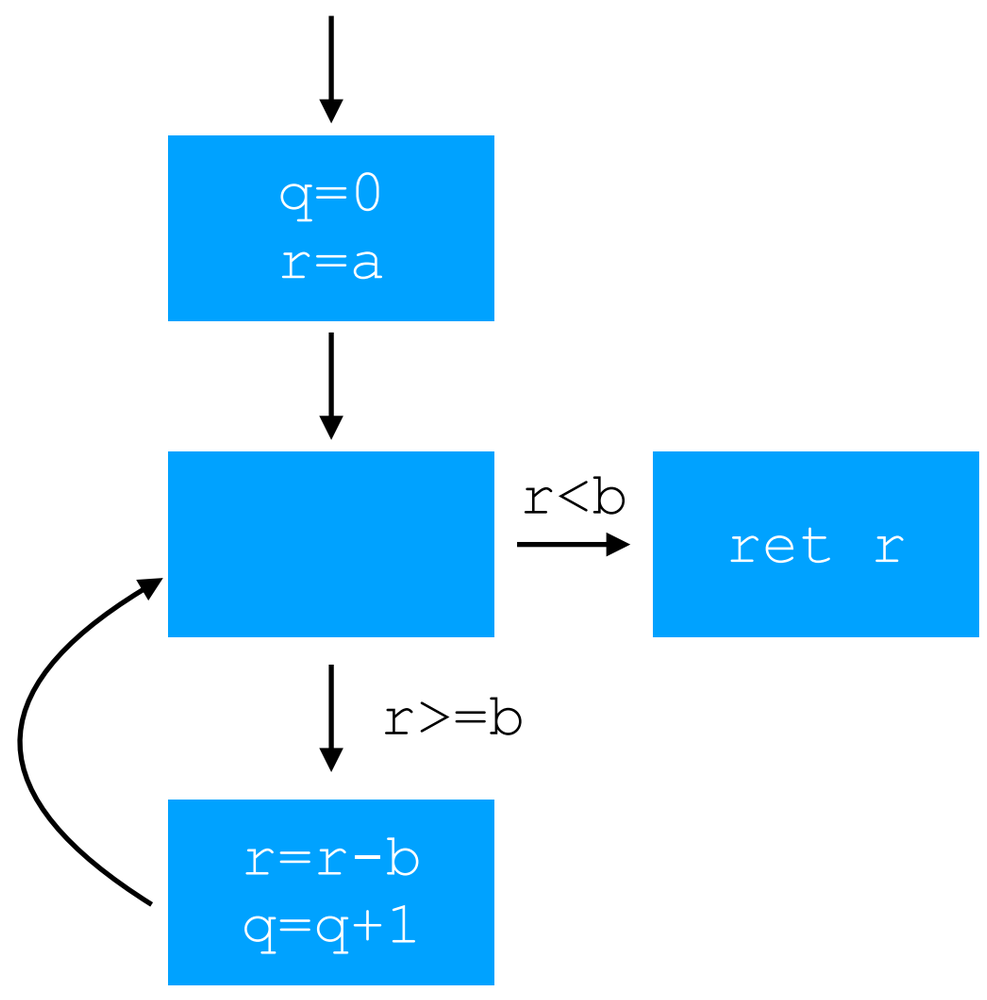
+	top	neg	zero	pos	bot
top					
neg					
zero					
pos					
bot					

-	top	neg	zero	pos	bot
top					
neg					
zero					
pos					
bot					

*	top	neg	zero	pos	bot
top					
neg					
zero					
pos					
bot					

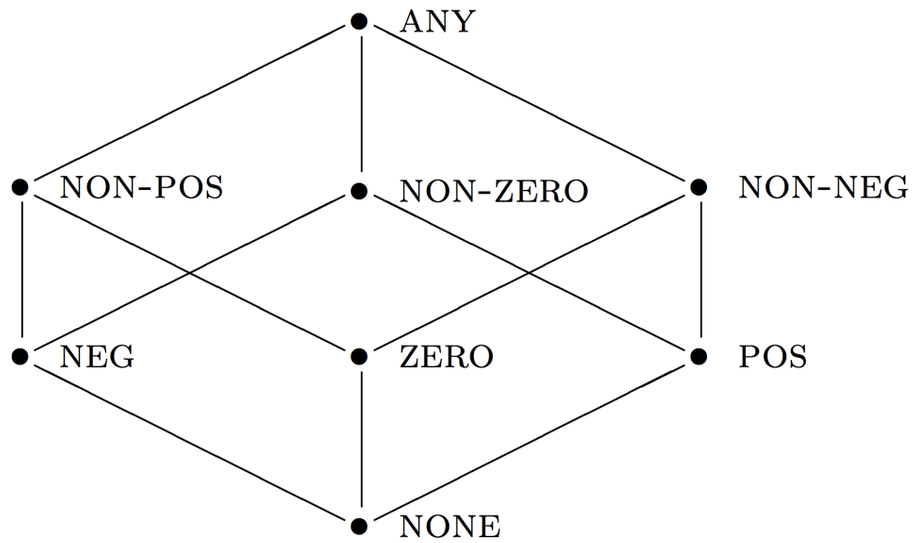
고정점 계산 예제

```
// a >= 0, b >= 0
int mod (int a, int b) {
   int q = 0;
   int r = a;
   while (r >= b) {
      r = r - b;
      q = q + 1;
   }
   return r;
}
```



Static Analysis Example 3

- More precise sign domain:



- Abstraction (α) and concretization (γ) functions:

- Join operation:

요약 의미 (Abstract Semantics)

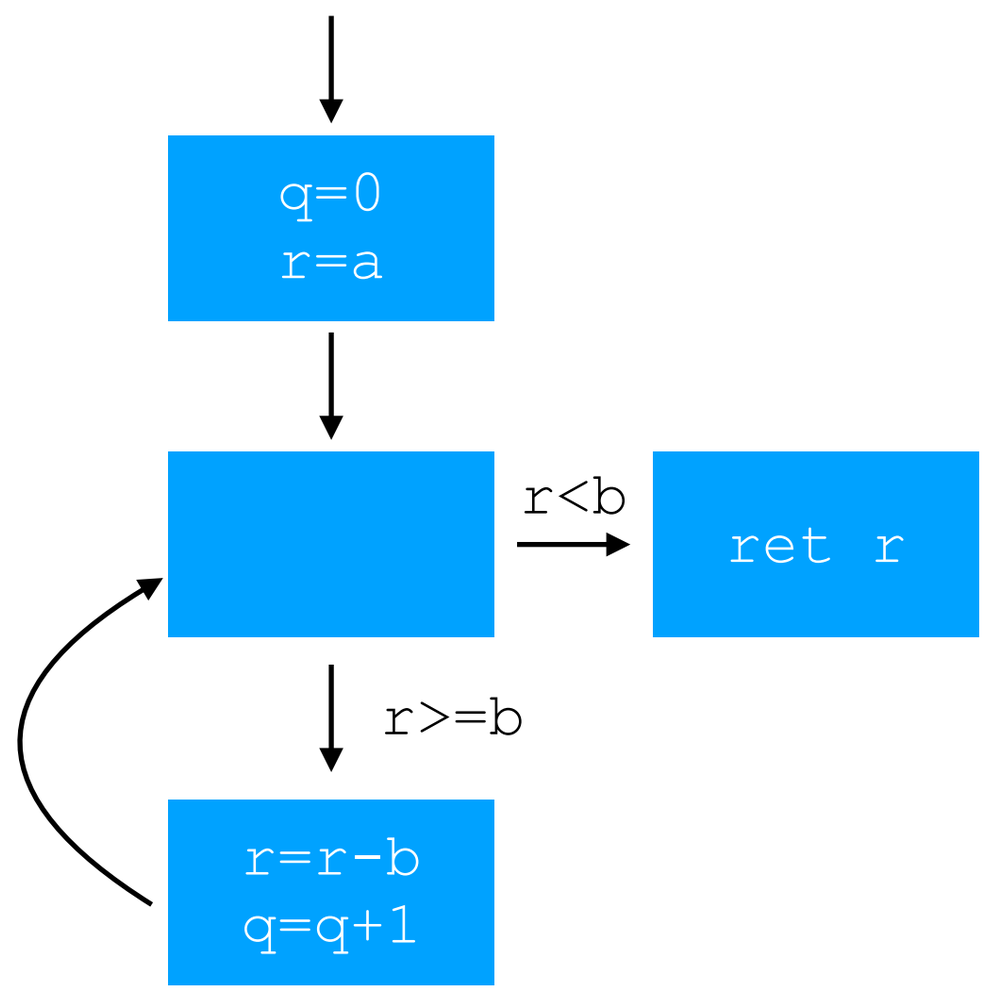
$+_S$	NONE	NEG	ZERO	POS	NON-POS	NON-ZERO	NON-NEG	ANY
NONE								
NEG								
ZERO								
POS								
NON-POS								
NON-ZERO								
NON-NEG								
ANY								

$\star_S$	NEG	ZERO	POS
NEG			
ZERO			
POS			

$-_S$	NEG	ZERO	POS
NEG			
ZERO			
POS			

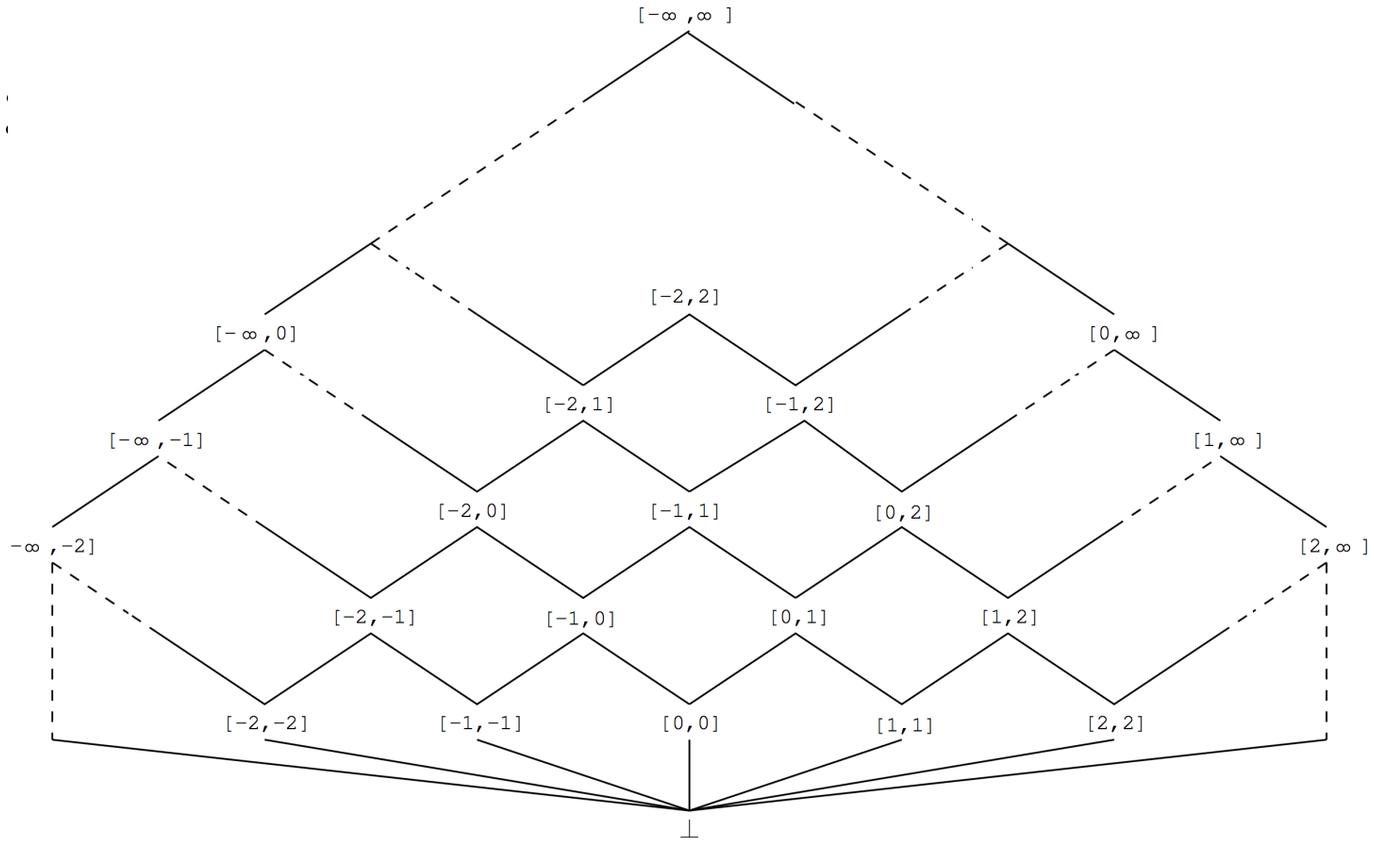
고정점 계산 예제

```
// a >= 0, b >= 0
int mod (int a, int b) {
   int q = 0;
   int r = a;
   while (r >= b) {
      r = r - b;
      q = q + 1;
   }
   return r;
}
```



Static Analysis Example 4

- Interval domain:

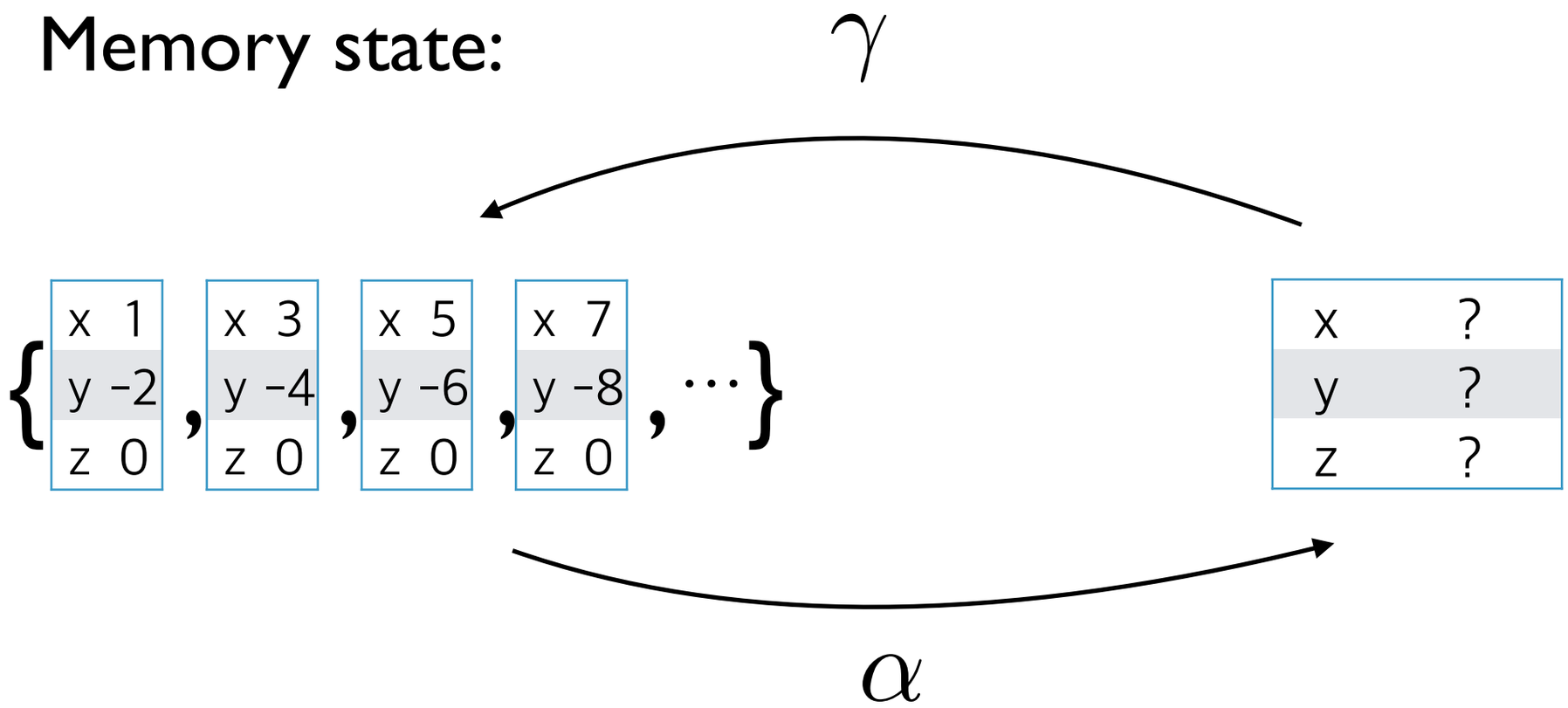


- Abstraction (α) and concretization (γ) functions:

- Join operation:

Static Analysis Example 4

- Memory state:



- Join operation:

요약 의미 (Abstract Semantics)

- 인터벌 공간에서의 덧셈, 뺄셈, 곱셈:

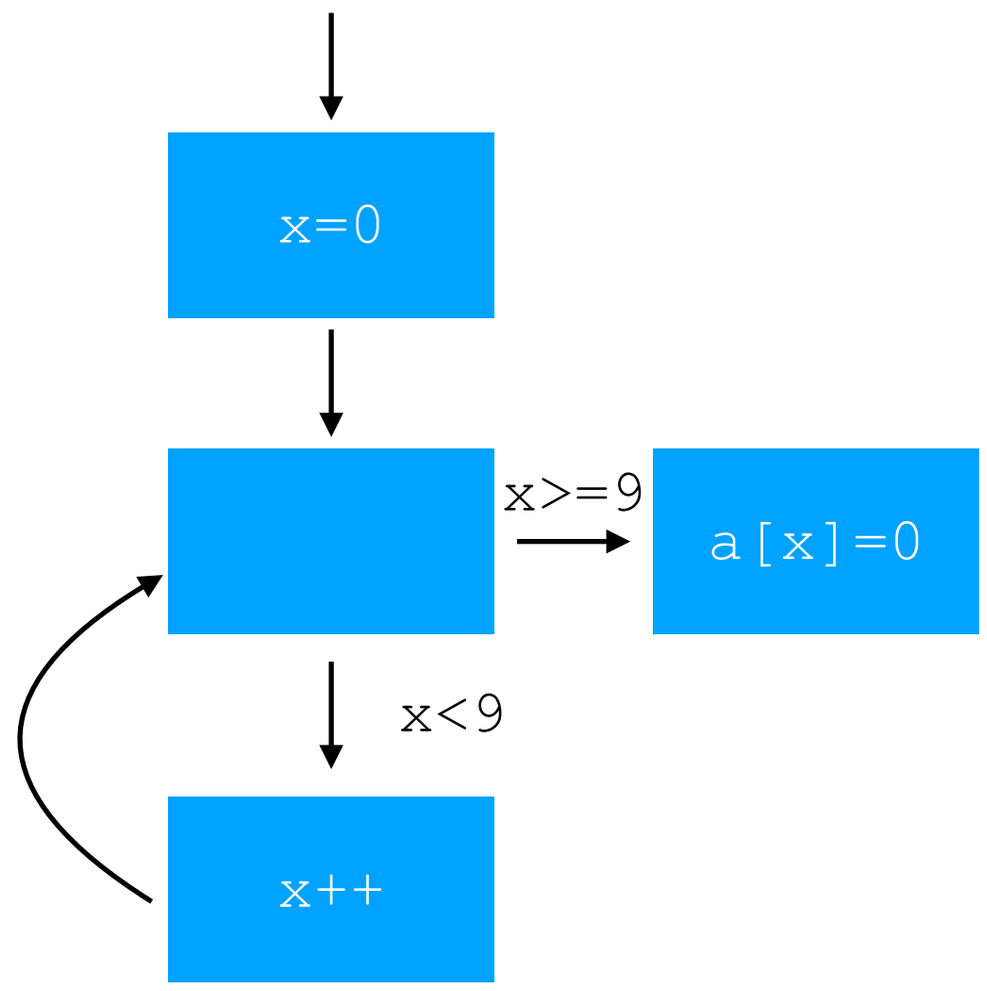
$$[a,b] + [c,d] =$$

$$[a,b] - [c,d] =$$

$$[a,b] \times [c,d] =$$

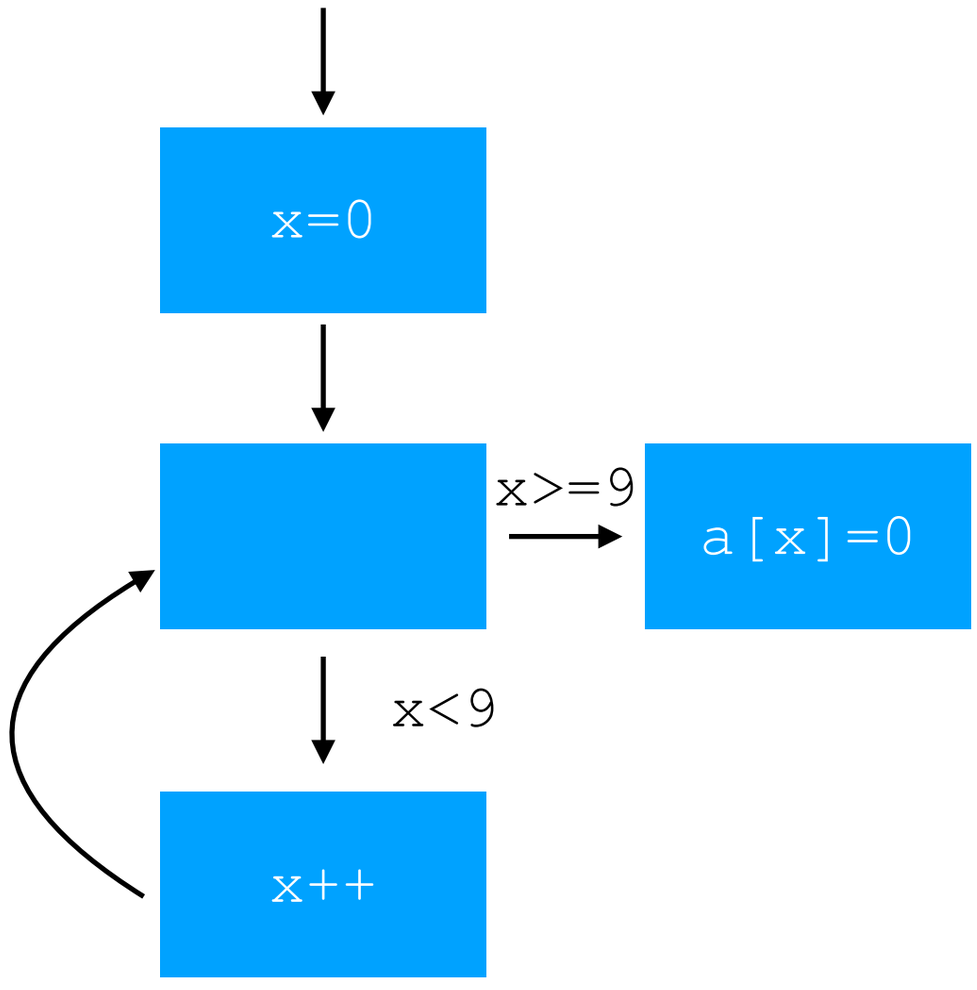
고정점 계산 예제

```
int a[10];
x = 0;

while (x < 9)
     x++;

a[x] = 0;
```



요약: 고정점 알고리즘

```
type Table map[Node]State
type Worklist []Node

func Analyze(cfg Cfg) Table {
    var tbl Table
    var worklist Worklist
    tbl = NewTable()
    worklist = NewWorklist()
    worklist.AddSet(cfg.blocks)

    for !worklist.IsEmpty() {
        here := worklist.Choose()
        state := InputOf(here, cfg, tbl)
        state.TransferBlock(here.Insts)
        old_state := tbl.Find(here)
        if !StateOrder(state, old_state) {
            if NeedWiden(here) {
                tbl.Bind(here, StateWiden(old_state, state))
            } else {
                tbl.Bind(here, StateJoin(old_state, state))
            }
            worklist.AddSet(cfg.Succ(here))
        }
    }
    return tbl
}
```



```
16      static char *curfinal = "HDACB  FE";
17
18      keysym = read_from_input ();
19
20      if ((((KeySym)(keysym) >= 0xFF91) && ((KeySym)(keysym) <= 0xFF94)))
21      {
22         unparseputc((char)(keysym-0xFF91 +'P'), pty);
23         key = 1;
24      }
25      else if (keysym >= 0)
26      {
27          if (keysym < 16)
28          {
29              if (read_from_input())
30              {
31                 if (keysym >= 10) return;
32                 curfinal[keysym] = 1;
33              }
34              else
35              {
36                  curfinal[keysym] = 2;
37              }
38          }
39          if (keysym < 10)
40          {
41              unparseputc(curfinal[keysym], pty);
42          }
43      }
```

```
16      static char *curfinal = "HDACB  FE";
17
18      keysym = read_from_input ();
19
20      if ((((KeySym)(keysym) >= 0xFF91) && ((KeySym)(keysym) <= 0xFF94)))
21      {
22         unparseputc((char)(keysym-0xFF91 +'P'), pty);
23         key = 1;
24      }
25      else if (keysym >= 0)
26      {
27          if (keysym < 16)
28          {
29              if (read_from_input())
30              {
31                  if (keysym >= 10) return;
32                  curfinal[keysym] = 1;
33              }
34              else
35              {
36                  curfinal[keysym] = 2;
37              }
38          }
39          if (keysym < 10)
40          {
41              unparseputc(curfinal[keysym], pty);
42          }
43      }
```

safe

buffer-

safe

curfinal: buffer of size 10

Sparrow The Early Bird

```
16      static char *curfinal = "HDACB   FE";
17
18      keysym = read_from_input ();
19
20      if ((((KeySym)(keysym) >= 0xFF91) && ((KeySym)(keysym) <= 0xFF94)))
21      {
22          unparseputc((char)(keysym-0xFF91 +'P'), pty);
23          key = 1;
24      }
25      else if (keysym >= 0)
26      {
27          if (keysym < 16)
28          {
29              if (read_from_input())
30              {
31                  if (keysym >= 10) return;
32                  curfinal[keysym] = 1;
33              }
34              else
35              {
36                  curfinal[keysym] = 2;
37              }
38          }
39          if (keysym < 10)
40          {
41              unparseputc(curfinal[keysym], pty);
42          }
43      }
```

safe

buffer-

safe

Sparrow The Early Bird

curfinal: buffer of size 10

keysym: any integer

```
16      static char *curfinal = "HDACB   FE";
17
18      keysym = read_from_input ();
19
20      if ((((KeySym)(keysym) >= 0xFF9                ym) <= 0xFF94)))
21      {
22          unparseputc((char)(keysym-0xFF91 +'P'), pty);
23          key = 1;
24      }
25      else if (keysym >= 0)
26      {
27          if (keysym < 16)
28          {
29              if (read_from_input())
30              {
31                  if (keysym >= 10) return;
32                  curfinal[keysym] = 1;
33              }
34              else
35              {
36                  curfinal[keysym] = 2;
37              }
38          }
39          if (keysym < 10)
40          {
41              unparseputc(curfinal[keysym], pty);
42          }
43      }
```

safe

buffer-

safe

Sparrow The Early Bird

```
16      static char *curfinal = "HDACB   FE";
17
18      keysym = read_from_input ();
19
20      if ((((KeySym)(keysym) >= 0xFF9                    ym) <= 0xFF94)))
21      {
22          unparseputc((char)(keysym-0xFF91 +'P'), pty);
23          key = 1;
24      }
25      else if (keysym >= 0)
26      {
27          if (keysym < 16)
28          {
29              if (read_from_input())
30              {
31                  if (keysym >= 10) return;
32                  curfinal[keysym] = 1;
33              }
34              else
35              {
36                  curfinal[keysym] = 2;
37              }
38          }
39          if (keysym < 10)
40          {
41              unparseputc(curfinal[keysym], pty);
42          }
43      }
```

curfinal: buffer of size 10

keysym: any integer

keysym: [0,15]

safe

buffer-

safe

Sparrow The Early Bird

```
16      static char *curfinal = "HDACB   FE";
17
18      keysym = read_from_input ();
19
20      if ((((KeySym)(keysym) >= 0xFF9                    ym) <= 0xFF94)))
21      {
22          unparseputc((char)(keysym-0xFF91 +'P'), pty);
23          key = 1;
24      }
25      else if (keysym >= 0)
26      {
27          if (keysym < 16)
28          {
29              if (read_from_input())
30              {
31                  if (keysym >= 10) return;
32                  curfinal[keysym] = 1;
33              }
34              else
35              {
36                  curfinal[keysym] = 2;
37              }
38          }
39          if (keysym < 10)
40          {
41              unparseputc(curfinal[keysym], pty);
42          }
43      }
```

curfinal: buffer of size 10

keysym: any integer

keysym: [0,15]

safe

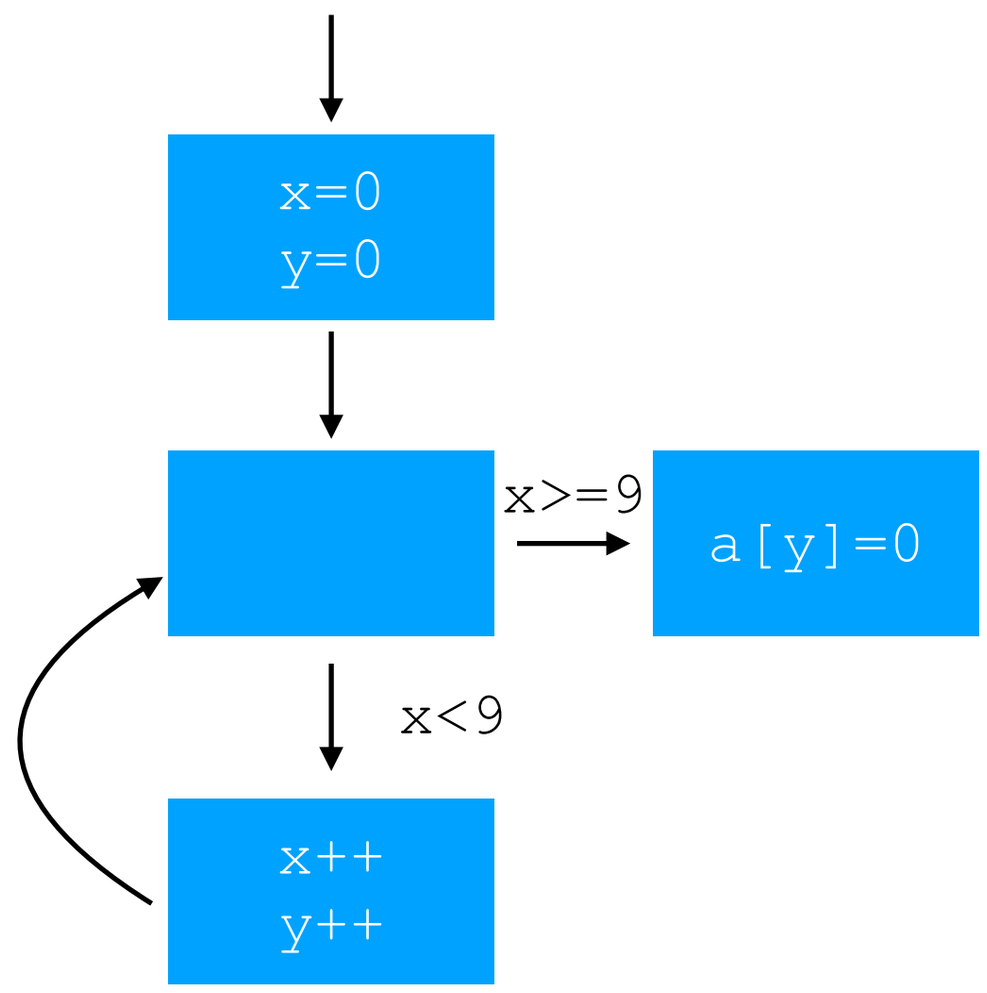
buffer-

curfinal:[10,10]
keysym:[10,15]

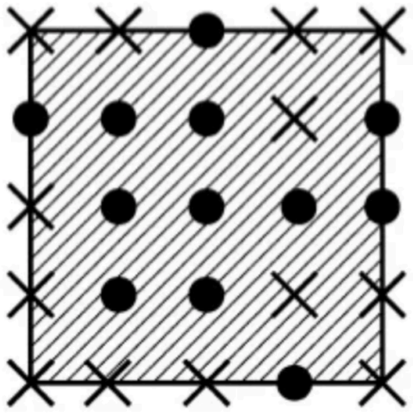
safe

인터벌 분석의 한계

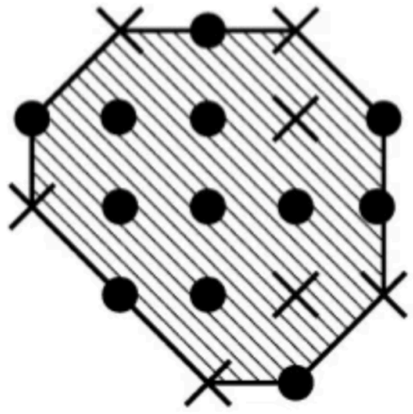
```
int a[10];
x = 0;
y = 0;

while (x < 9) {
      x++;
      y++;
}

a[y] = 0;
```



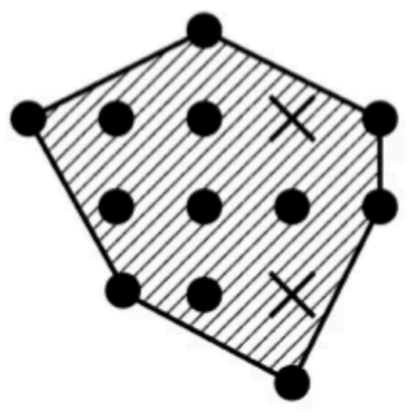
더 정교한 요약도 가능



Intervals



Octagons



Polyhedra

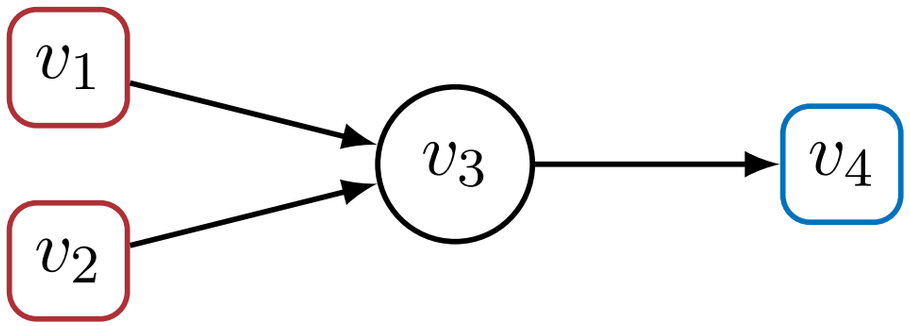
```
int a[10];
x = 0;
y = 0;

while (x < 9) {
     x++;
     y++;
}

a[y] = 0;
```

Octagon analysis

Interval analysis

cf) Neural Network Analysis

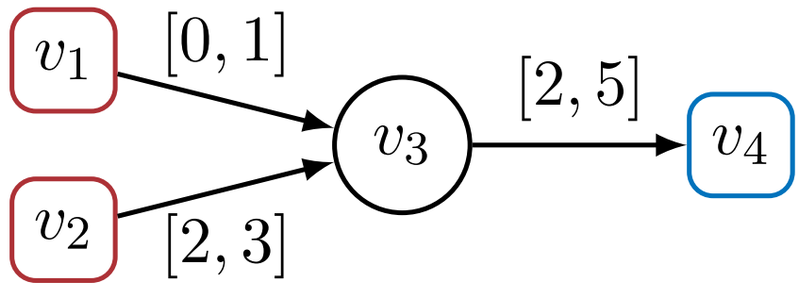
- Example neural network:



$$f_{v_3}(\boldsymbol{x}) = 2x_1 + x_2 \text{ and } f_{v_4}(x) = \text{relu}(x)$$

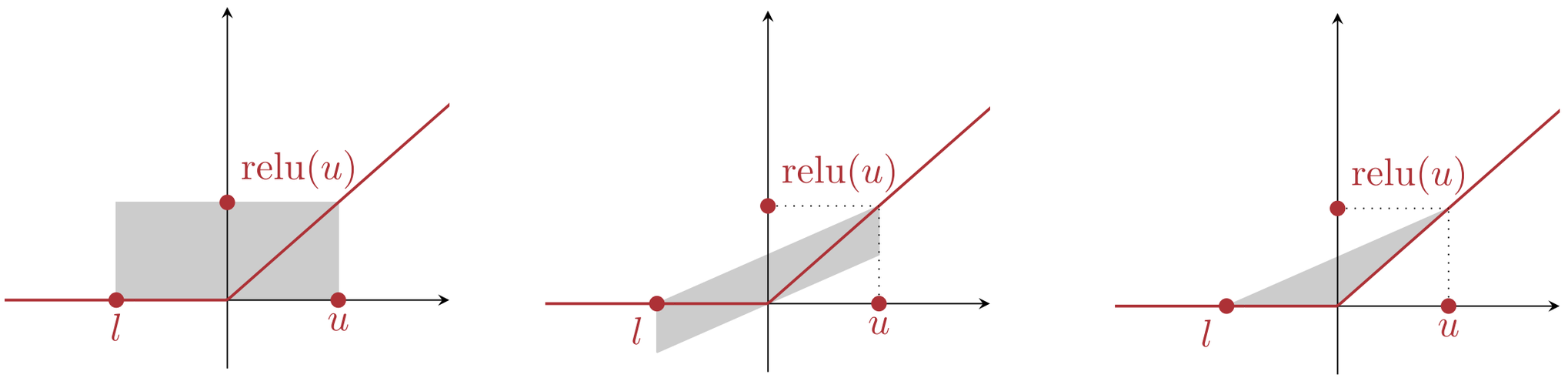
- **Evaluate** $f_G^a([0,1],[2,3])$

$$\text{out}^a(v_1) = [0,1]$$
$$\text{out}^a(v_2) = [2,3]$$
$$\text{out}^a(v_3) = [2*0+2,\ 2*1+3] = [2,5]$$
$$\text{out}^a(v_4) = [\text{relu}(2), \text{relu}(5)] = [2,5]$$



cf) Neural Network Analysis

- Intervals vs. Zonotopes vs. Polyhedra



cf) Abstract Interpretation Theory

Step 1: Define Concrete Semantics

The concrete semantics describes the real executions of the program.
Described by semantic domain and function.

- A *semantic domain* $\boldsymbol{D}$, which is a CPO:
  - $\boldsymbol{D}$ is a partially ordered set with a least element $\bot$.
  - Any increasing chain $\boldsymbol{d_0 \sqsubseteq d_1 \sqsubseteq \ldots}$ in $\boldsymbol{D}$ has a least upper bound $\bigsqcup_{n \geq 0} \boldsymbol{d_n}$ in $\boldsymbol{D}$.
- A *semantic function* $\boldsymbol{F : D \to D}$, which is continuous: for all chains $\boldsymbol{d_0 \sqsubseteq d_1 \sqsubseteq \ldots}$,

$$\boldsymbol{F(\bigsqcup_{n \geq 0} d_i) = \bigsqcup_{n \geq 0} F(d_n).}$$

Then, the concrete semantics (or collecting semantics) is defined as the least fixed point of *semantic function* $\boldsymbol{F : D \to D}$:

$$\boldsymbol{fix F = \bigsqcup_{i \in N} F^i(\bot).}$$

Step 2: Define Abstract Semantics

Define the abstract semantics of the input program.

- Define an *abstract semantic domain* CPO $\hat{\boldsymbol{D}}$.
  - Intuition: $\hat{\boldsymbol{D}}$ is an abstraction of $\boldsymbol{D}$
- Define an *abstract semantic function* $\hat{\boldsymbol{F}} : \hat{\boldsymbol{D}} \to \hat{\boldsymbol{D}}$.
  - Intuition: $\hat{\boldsymbol{F}}$ is an abstraction of $\boldsymbol{F}$.
  - $\hat{\boldsymbol{F}}$ must be monotone:

$$\boldsymbol{\forall \hat{x}, \hat{y} \in \hat{D}.\ \hat{x} \sqsubseteq \hat{y} \implies \hat{F}(\hat{x}) \sqsubseteq \hat{F}(\hat{y})}$$

(or extensive: $\boldsymbol{\forall x \in \hat{D}.\ x \sqsubseteq \hat{F}(x)}$)

Then, static analysis is to compute an upper bound of:

$$\boldsymbol{\bigsqcup_{i \in \mathbb{N}} \hat{F}^i(\bot)}$$

How can we ensure that the result soundly approximate the concrete semantics?

Requirement 1: Galois Connection

$\boldsymbol{D}$ and $\hat{\boldsymbol{D}}$ must be related with Galois-connection:

$$\boldsymbol{D \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \hat{D}}$$

Requirement 2: $\hat{\boldsymbol{F}}$ and $\boldsymbol{F}$

- $\hat{\boldsymbol{F}}$ and $\boldsymbol{F}$ must satisfy

$$\boldsymbol{\alpha \circ F \sqsubseteq \hat{F} \circ \alpha} \quad \text{(i.e., } \boldsymbol{F \circ \gamma \sqsubseteq \gamma \circ \hat{F}}\text{)}$$

- or, alternatively,

$$\boldsymbol{\alpha(x) \sqsubseteq \hat{x} \implies \alpha(F(x)) \sqsubseteq \hat{F}(\hat{x})}$$

Soundness Guarantee

Theorem (Fixpoint Transfer)

Let $\boldsymbol{D}$ *and* $\hat{\boldsymbol{D}}$ *be related by Galois-connection* $\boldsymbol{D \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \hat{D}}$. *Let* $\boldsymbol{F : D \to D}$ *be a continuous function and* $\hat{\boldsymbol{F}} : \hat{\boldsymbol{D}} \to \hat{\boldsymbol{D}}$ *be a monotone function such that* $\boldsymbol{\alpha \circ F \sqsubseteq \hat{F} \circ \alpha}$. *Then,*

$$\boldsymbol{\alpha(fix F) \sqsubseteq \bigsqcup_{i \in \mathbb{N}} \hat{F}^i(\hat{\bot}).}$$

Theorem (Fixpoint Transfer2)

Let $\boldsymbol{D}$ *and* $\hat{\boldsymbol{D}}$ *be related by Galois-connection* $\boldsymbol{D \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \hat{D}}$. *Let* $\boldsymbol{F : D \to D}$ *be a continuous function and* $\hat{\boldsymbol{F}} : \hat{\boldsymbol{D}} \to \hat{\boldsymbol{D}}$ *be a monotone function such that* $\boldsymbol{\alpha(x) \sqsubseteq \hat{x} \implies \alpha(F(x)) \sqsubseteq \hat{F}(\hat{x})}$. *Then,*

$$\boldsymbol{\alpha(fix F) \sqsubseteq \bigsqcup_{i \in \mathbb{N}} \hat{F}^i(\hat{\bot}).}$$

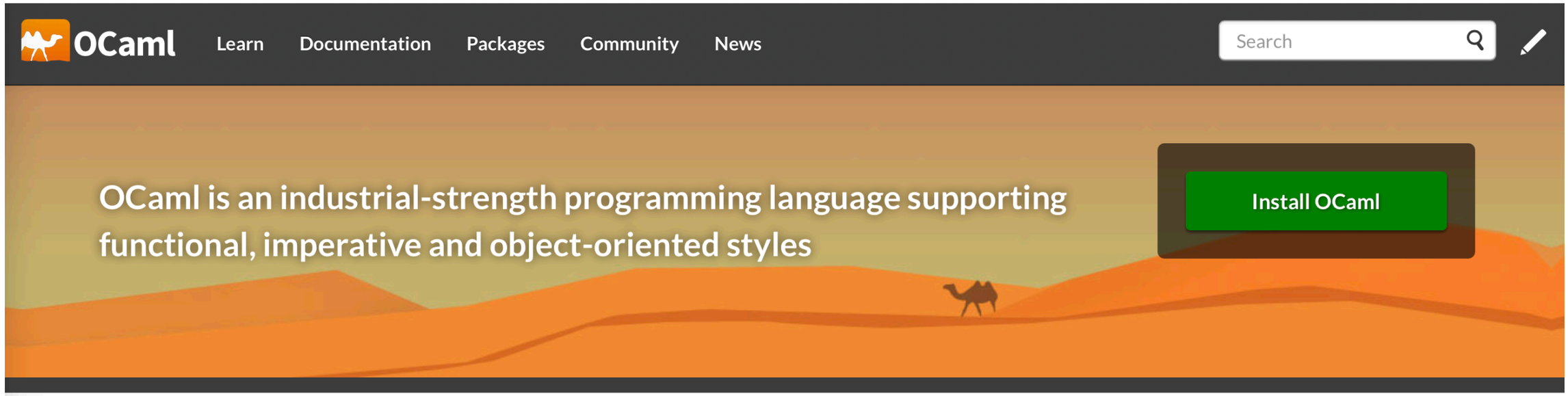
Part 1. 프로그램 분석 개괄

Part 2. 정적 분석 적용 사례

Part 3. 정적 분석 원리

Part 4. 정적 분석 구현

The OCaml Programming Language



[en | fr]

Learn

Find out about OCaml, read about users, see code examples, go through tutorials and more.

Documentation

Install OCaml, look up package docs, access the Manual, get the cheat sheets and more.

Packages

The OCaml Package Manager, gives you access to multiple versions of thousands of packages.

Community

Read the news feed, join the mailing lists, get support, attend meetings, and find OCaml around the web.

Got a question? Ask OCaml experts!

Learn OCaml in your browser with TryOCaml

News

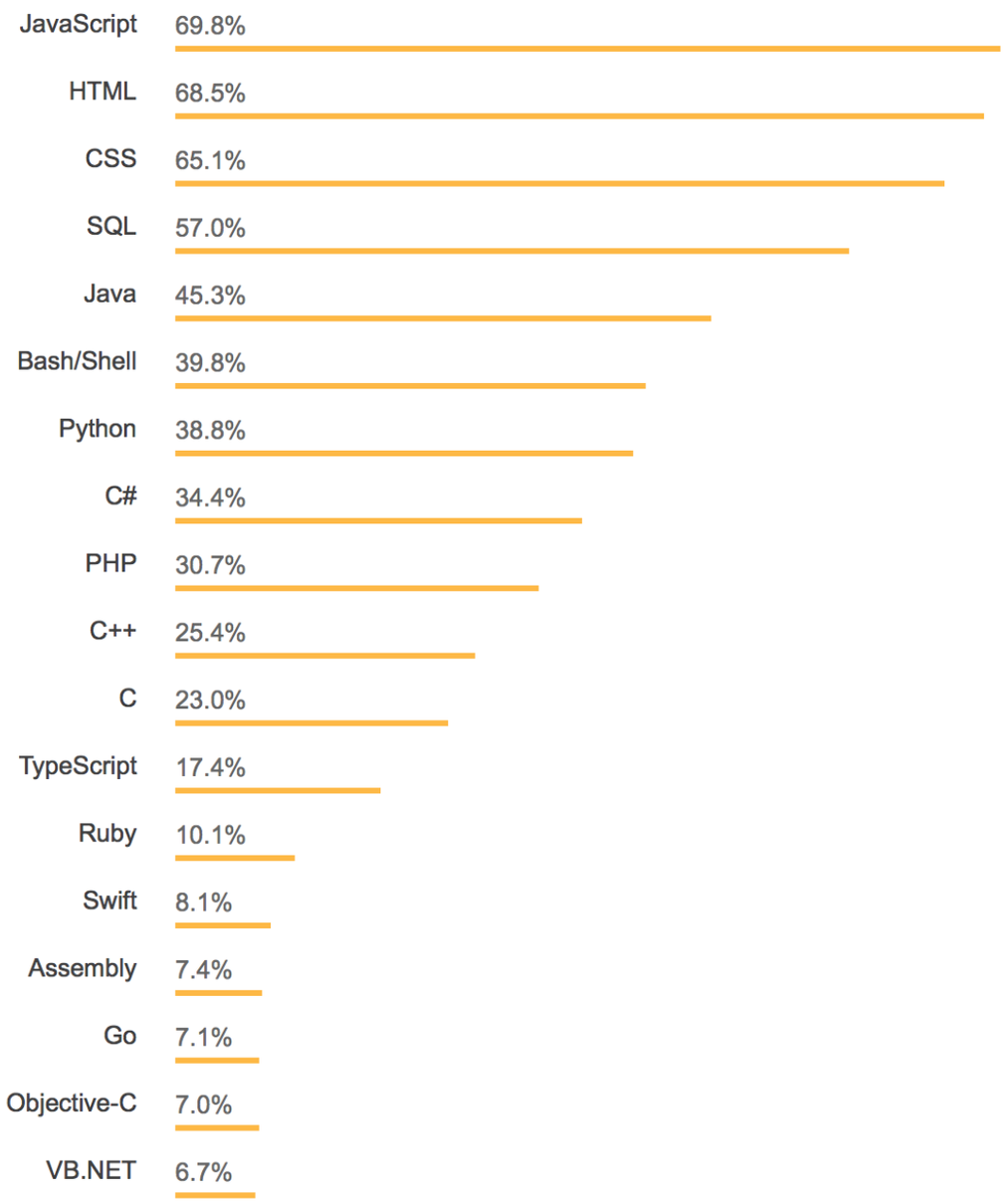
- Release of OCaml 4.14.0 — March 28, 2022
- OCaml Weekly News — March 29, 2022
- MirageOS 4 Released! — March 29, 2022
- PhD Position at CEA LIST - LSL — March 28, 2022
- All your metrics belong to influx — March 8, 2022
- Secure Virtual Messages in a Bottle with SC... — March 8, 2022

More...

https://ocaml.org

Why Functional Programming?

Most Popular Programming Languages (Stackoverflow, 2018)



Why Functional Programming?

Top Paying Programming Languages (Stackoverflow, 2018):

Why Functional Programming?

Functional languages are increasingly used in industry:

Why Functional Programming?

- Functional programming, as opposed to imperative programming, provides a new and important frame of thinking
- Many modern programming languages has been inspired by functional languages
- Learning functional languages helps to understand foundational ideas in programming
- Writing programs in functional languages is concise and pleasurable
- and many more

Why Functional Programming in OCaml?

OCaml is a good programming language:

- functional programming: scala, java8, haskell, python, JavaScript, etc
- static type system: scala, java, haskell, etc
- automatic type inference: scala, haskell, etc
- pattern matching: scala, etc
- algebraic data types, module system, etc

Basics of the Language

- Expressions
- Names
- Functions
- Pattern matching
- Type inference
- Tuples and lists
- Data types
- Exceptions

Write and run all examples in the slides by yourself!

An OCaml Program is an Expression

Statement and expressions:

- A statement *does something*.
- An expression *evaluates to a value*.

Programming languages can be classified into

- statement-oriented: C, C++, Java, Python, JavaScript, etc
  - often called "imperative languages"
- expression-oriented: ML, Haskell, Scala, Lisp, etc
  - often called "functional languages"

...

Basic Structure of OCaml Programs

An OCaml program is a sequence of definitions:

$$\begin{aligned}
&\texttt{let } x_1 = e_1 \\
&\texttt{let } x_2 = e_2 \\
&\qquad \vdots \\
&\texttt{let } x_n = e_n
\end{aligned}$$

- $e_1, e_2, \ldots, e_n$ are evaluated in order
- Variable x_i refers to the value of e_i

Example

- Hello World:

```
let hello = "Hello"
let world = "World"
let helloworld = hello ^ " " ^ world
let _ = print_endline helloworld
```

- Interpreter:

```
$ ocaml helloworld.ml
Hello World
```

- REPL (Read-Eval-Print-Loop)

```
$ ocaml
        OCaml version 4.04.0

# let hello = "Hello";;
val hello : string = "Hello"
# let world = "World";;
val world : string = "World"
# let helloworld = hello ^ " " ^ world;;
val helloworld : string = "Hello World"
```

Arithmetic Expressions

- Arithmetic expressions evaluate to numbers: e.g., `1+2*3`, `1+5`, `7`
- Try to evaluate expressions in the REPL:

```
# 1+2*3;;
- : int = 7
```

- Arithmetic operators on integers:

a + b	addition
a - b	subtraction
a * b	multiplication
a / b	divide a by b, returning the whole part
a `mod` b	divide a by b, returning the remaining part

Boolean Expressions

- Boolean expressions evaluate to boolean values (i.e., true, false).
- Try to evaluate boolean expressions:

```
# true;;
- : bool = true
# false;;
- : bool = false
# 1 > 2;;
- : bool = false
```

- Comparison operators produces boolean values:

a = b	true if a and b are equal
a <> b	true if a and b are not equal
a < b	true if a is less than b
a <= b	true if a is less than or equal to b
a > b	true if a is greater than b
a >= b	true if a is greater than or equal to b

Boolean Operators

- Boolean expressions are combined by boolean operators:

```
# true && false;;
- : bool = false
# true || false;;
- : bool = true
# (2 > 1) && (3 > 2);;
- : bool = true
```

ML is a Statically Typed Language

If you try to evaluate an expression that does not make sense, OCaml rejects and does not evaluate the program: e.g.,

```
# 1 + true;;
Error: This expression has type bool but an expression was
expected of type int
```

Static Types and Dynamic Types

Programming languages are classified into:

- *Statically typed languages*: type checking is done at compile-time.
  - type errors are detected before program executions
  - C, C++, Java, ML, Scala, etc
- *Dynamically typed languages*: type checking is done at run-time.
  - type errors are detected during program executions
  - Python, JavaScript, Ruby, Lisp, etc

Statically typed languages are further classified into:

- *Type-safe languages* guarantee that compiled programs do not have type errors at run-time.
  - All type errors are detected at compile time.
  - Compiled programs do not stuck.
  - ML, Haskell, Scala
- *Unsafe languages* do not provide such a guarantee.
  - Some type errors remain at run-time.
  - C, C++

cf) Which one is better?

Statically typed languages:

- (+) Type errors are caught early in the development cycle.
- (+) Program execution is efficient by omitting runtime checks.
- (−) Less flexible than dynamic languages.

Dynamically typed languages:

- (−) Type errors appear at run-time, often unexpectedly.
- (+) Provide more flexible language features.
- (+) Easy and fast prototyping.

Conversion between Different Types

- In OCaml, different types of values are distinguished:

```
# 3 + 2.0;;
Error: This expression has type float but an expression
was expected of type int
```

- Types must be explicitly converted:

```
# 3 + int_of_float 2.0;;
- : int = 5
```

- Operators for floating point numbers:

```
# 1.2 +. 2.3;;
- : float = 3.5
# 1.5 *. 2.0;;
- : float = 3.
# float_of_int 1 +. 2.2;;
- : float = 3.2
```

Other Primitive Values

- OCaml provides six primitive values: integers, booleans, floating point numbers, characters, strings, and unit.

```
# 'c';;
- : char = 'c'
# "cose212";;
- : string = "cose212"
# ();;
- : unit = ()
```

Conditional Expressions

$$\texttt{if } be \texttt{ then } e_1 \texttt{ else } e_2$$

- If be is true, the value of the conditional expression is the value of e_1.
- If be is false, the value of the expression is the value of e_2.
- ```
  # if 2 > 1 then 0 else 1;;
  - : int = 0
  # if 2 < 1 then 0 else 1;;
  - : int = 1
  ```
- be must be a boolean expression.
- types of e_1 and e_2 must be equivalent.
- ```
  # if 1 then 1 else 2;;
  Error: ...
  # if true then 1 else true;;
  Error: ...
  # if true then true else false;;
  - : bool = true
  ```

Names and Functions

- Create a global variable with the `let` keyword:

```
# let x = 3 + 4;;
val x : int = 7
```

We say a variable `x` is *bound* to value **7**.

```
# let y = x + x;;
val y : int = 14
```

- Create a local variable with `let ... in ...` construct:

$$\texttt{let } x \texttt{ = } e_1 \texttt{ in } e_2$$

  - x is bound to the value of e_1
  - the scope of x is e_2
  - the value of e_2 becomes the value of the entire expression

Examples

- ```
  # let a = 1 in a;;
  - : int = 1
  # let a = 1 in a * 2;;
  - : int = 2
  ```
- ```
  # let a = 1 in
    let b = a + a in
    let c = b + b in
      c + c;;
  - : int = 8
  ```
- ```
  # let d =
      let a = 1 in
      let b = a + a in
      let c = b + b in
        c + c;;
   val d : int = 8
  ```
  ```

Functions

- Define a function with `let`:

```
# let square x = x * x;;
 val square : int -> int = <fun>
```

- Apply the function:

```
# square 2;;
- : int = 4
# square (2 + 5);;
- : int = 49
# square (square 2);;
- : int = 16
```

- The body can be any expression:

```
# let neg x = if x < 0 then true else false;;
val neg : int -> bool = <fun>
# neg 1;;
- : bool = false
# neg (-1);;
- : bool = true
```

Functions

- Functions with multiple arguments:

```
# let sum_of_squares x y = (square x) + (square y);;
val sum_of_squares : int -> int -> int = <fun>
# sum_of_squares 3 4;;
- : int = 25
```

- Recursive functions are defined with `let rec` construct:

```
# let rec factorial a =
    if a = 1 then 1 else a * factorial (a - 1);;
val factorial : int -> int = <fun>
# factorial 5;;
- : int = 120
```

Nameless Functions

- Many modern programming languages provide nameless functions, e.g., ML, Scala, Java8, JavaScript, Python, etc.
- In OCaml, a function can be defined without names:

```
# fun x -> x * x;;
- : int -> int = <fun>
```

  Called *nameless* or *anonymous* functions.
- Apply nameless function as usual:

```
# (fun x -> x * x) 2;;
- : int = 4
```

- A variable can be bound to functions:

```
# let square = fun x -> x * x;;
val square : int -> int = <fun>
```

- The followings are equivalent:

```
let square = fun x -> x * x
```

```
let square x = x * x
```

Functions are First-Class in OCaml

In programming languages, a value is *first-class*, if the value can be

- stored in a variable,
- passed as an argument of a function, and
- returned from other functions.

A language is often called *functional*, if functions are first class values, e.g., ML, Scala, Java8, JavaScript, Python, Lisp, etc.

Functions are First-Class in OCaml

- Functions can be stored in variables:

```
# let square = fun x -> x * x;;
# square 2;;
- : int = 4
```

- Functions can be passed to other functions:

```
# let sum_if_true test first second =
    (if test first then first else 0)
  + (if test second then second else 0);;
val sum_if_true : (int -> bool) -> int -> int -> int = <fun>

# let even x = x mod 2 = 0;;
val even : int -> bool = <fun>
# sum_if_true even 3 4;;
- : int = 4
# sum_if_true even 2 4;;
- : int = 6
```

Functions are First-Class in OCaml

- Functions can be also returned from a procedure:

```
# let plus_a a = fun b -> a + b;;
val plus_a : int -> int -> int = <fun>

# let f = plus_a 3;;
val f : int -> int = <fun>
# f 1;;
- : int = 4
# f 2;;
- : int = 5
```

Functions that manipulate functions are called *higher-order functions*.

- i.e., functions that take as argument functions or return functions
- greatly increase the expressiveness of the language

Pattern Matching

- An elegant way of doing case analysis.
- E.g., using pattern-matching, the factorial function

```
let rec factorial a =
  if a = 1 then 1 else a * factorial (a - 1)
```

can be written as follows:

```
let factorial a =
match a with
 1 -> 1
|_ -> a * factorial (a - 1)
```

Pattern Matching

The nested if-then-else expression

```
let isabc c = if c = 'a' then true
              else if c = 'b' then true
              else if c = 'c' then true
              else false
```

can be written using pattern matching:

```
let isabc c =
  match c with
   'a' -> true
  |'b' -> true
  |'c' -> true
  | _  -> false
```

or simply,

```
 let isabc c =
  match c with
  'a' | 'b' | 'c' -> true
  | _ -> false
```

Type Inference

In C or Java, types must be annotated:

```
public static int f(int n)
{
  int a = 2;
  return a * n;
}
```

In OCaml, type annotations are not mandatory:

```
# let f n =
    let a = 2 in
      a * n;;
val f : int -> int = <fun>
```

Type Inference

OCaml can infer types, no matter how complex the program is:

```
# let sum_if_true test first second =
    (if test first then first else 0)
  + (if test second then second else 0);;
val sum_if_true : (int -> bool) -> int -> int -> int = <fun>
```

OCaml compiler infers the type through the following reasoning steps:

1. the types of `first` and `second` must be `int`, because both branches of a conditional expression must have the same type,
2. the type of `test` is a function type $\alpha \rightarrow \beta$, because `test` is used as a function,
3. α must be of `int`, because `test` is applied to `first`, a value of `int`,
4. β must be of `bool`, because conditions must be boolean expressions,
5. the return value of the function has type `int`, because the two conditional expressions are of `int` and their addition gives `int`.

Type Annotation

Explicit type annotations are possible:

```
# let sum_if_true (test : int -> bool) (x : int) (y : int) : int =
    (if test x then x else 0) + (if test y then y else 0);;
val sum_if_true : (int -> bool) -> int -> int -> int = <fun>
```

If the annotation is wrong, OCaml finds the error and report it:

```
# let sum_if_true (test : int -> int) (x : int) (y : int) : int =
    (if test x then x else 0) + (if test y then y else 0);;
Error: The expression (test x) has type int but an expression
was expected of type bool
```

Polymorphic Types

- What is the type of the program?

  ```
  let id x = x
  ```

  See how OCaml infers its type:

  ```
  # let id x = x;;
  val id : 'a -> 'a = <fun>
  ```

  The function works for values of any type:

  ```
  # id 1;;
  - : int = 1
  # id "abc";;
  - : string = "abc"
  # id true;;
  - : bool = true
  ```

- Such a function is called *polymorphic* and `'a` is a *type variable*.

Tuples

- An ordered collection of values, each of which can be a different types, e.g.,

```
# let x = (1, "one");;
val x : int * string = (1, "one")
# let y = (2, "two", true);;
val y : int * string * bool = (2, "two", true)
```

- Extract each component using pattern-matching:

```
# let fst p = match p with (x,_) -> x;;
val fst : 'a * 'b -> 'a = <fun>
# let snd p = match p with (_,x) -> x;;
val snd : 'a * 'b -> 'b = <fun>
```

or equivalently,

```
# let fst (x,_) = x;;
val fst : 'a * 'b -> 'a = <fun>
# let snd (_,x) = x;;
val snd : 'a * 'b -> 'b = <fun>
```

Tuples

- Patterns can be used in `let`:

```
# let p = (1, true);;
val p : int * bool = (1, true)
# let (x,y) = p;;
val x : int = 1
val y : bool = true
```

Lists

- A finite sequence of elements, each of which has the same type, e.g.,

  `[1; 2; 3]`

  is a list of integers:

  ```
  # [1; 2; 3];;
  - : int list = [1; 2; 3]
  ```

  Note that
  - all elements must have the same type, e.g., `[1; true; 2]` is not a list,
  - the elements are ordered, e.g., `[1; 2; 3]` $\neq$ `[2; 3; 1]`, and
  - the first element is called *head*, the rest *tail*.
- `[]`: the empty list, i.e., nil. What are head and tail of `[]`?
- `[5]`: a list with a single element. What are head and tail of `[5]`?

List Examples

- ```
  # [1;2;3;4;5];;
  - : int list = [1; 2; 3; 4; 5]
  ```
- ```
  # ["OCaml"; "Java"; "C"];;
  - : string list = ["OCaml"; "Java"; "C"]
  ```
- ```
  # [(1,"one"); (2,"two"); (3,"three")];;
  - : (int * string) list = [(1, "one"); (2, "two"); (3, "three"
  ```
- ```
  # [[1;2;3];[2;3;4];[4;5;6]];;
  - : int list list = [[1; 2; 3]; [2; 3; 4]; [4; 5; 6]]
  ```
- ```
  # [1;"OCaml";3] ;;
  Error: This expression has type string but an expression was
   expected of type int
  ```
  ```

List Operators

- `::` (cons): add a single element to the front of a list, e.g.,

```
# 1::[2;3];;
- : int list = [1; 2; 3]
# 1::2::3::[];;
- : int list = [1; 2; 3]
```

(`[1; 2; 3]` is a shorthand for `1::2::3::[]`)

- `@` (append): combine two lists, e.g.,

```
# [1; 2] @ [3; 4; 5];;
- : int list = [1; 2; 3; 4; 5]
```

Patterns for Lists

Pattern matching is useful for manipulating lists.

- A function to check if a list is empty:

```
# let isnil l =
    match l with
      [] -> true
    |_  -> false;;
val isnil : 'a list -> bool = <fun>
# isnil [1];;
- : bool = false
# isnil [];;
- : bool = true
```

Patterns for Lists

- A function that computes the length of lists:

```
# let rec length l =
    match l with
     [] -> 0
    |h::t -> 1 + length t;;
val length : 'a list -> int = <fun>
# length [1;2;3];;
- : int = 3
```

We can replace pattern `h` by `_`:

```
let rec length l =
  match l with
    [] -> 0
  |_::t -> 1 + length t;;
```

Data Types

- If data elements are finite, just enumerate them, e.g., "days":

```
# type days = Mon | Tue | Wed | Thu | Fri | Sat | Sun;;
type days = Mon | Tue | Wed | Thu | Fri | Sat | Sun
```

Construct values of the type:

```
# Mon;;
- : days = Mon
# Tue;;
- : days = Tue
```

A function that manipulates the defined data:

```
# let nextday d =
  match d with
  | Mon -> Tue | Tue -> Wed | Wed -> Thu | Thu -> Fri
  | Fri -> Sa  | Sat -> Sun | Sun -> Mon ;;
val nextday : days -> days = <fun>
# nextday Mon;;
- : days = Tue
```

Data Types

- Constructors can be associated with values, e.g.,

```
# type shape = Rect of int * int | Circle of int;;
type shape = Rect of int * int | Circle of int
```

Construct values of the type:

```
# Rect (2,3);;
- : shape = Rect (2, 3)
# Circle 5;;
- : shape = Circle 5
```

A function that manipulates the data:

```
# let area s =
    match s with
      Rect (w,h) -> w * h
    | Circle r -> r * r * 3;;
val area : shape -> int = <fun>
# area (Rect (2,3));;
- : int = 6
# area (Circle 5);;
- : int = 75
```

Data Types

- Inductive data types, e.g.,

```
# type mylist = Nil | List of int * mylist;;
type mylist = Nil | List of int * mylist
```

Construct values of the type:

```
# Nil;;
- : mylist = Nil
# List (1, Nil);;
- : mylist = List (1, Nil)
# List (1, List (2, Nil));;
- : mylist = List (1, List (2, Nil))
```

A function that manipulates the data:

```
# let rec mylength l =
    match l with
     Nil -> 0
    |List (_,l') -> 1 + mylength l';;
val mylength : mylist -> int = <fun>
# mylength (List (1, List (2, Nil)));;
- : int = 2
```

Exceptions

- An exception means a run-time error: e.g.,

```
# let div a b = a / b;;
val div : int -> int -> int = <fun>
# div 10 5;;
- : int = 2
# div 10 0;;
Exception: Division_by_zero.
```

- The exception can be handled with `try ... with` constructs.

```
# let div a b =
    try
      a / b
    with Division_by_zero -> 0;;
val div : int -> int -> int = <fun>
# div 10 5;;
- : int = 2
# div 10 0;;
- : int = 0
```

Exceptions

- User-defined exceptions: e.g.,

```
# exception Problem;;
exception Problem
# let div a b =
    if b = 0 then raise Problem
    else a / b;;
val div : int -> int -> int = <fun>
# div 10 5;;
- : int = 2
# div 10 0;;
Exception: Problem.
# try
    div 10 0
  with Problem -> 0;;
- : int = 0
```

Summary

We've gone through the basics of OCaml programming:

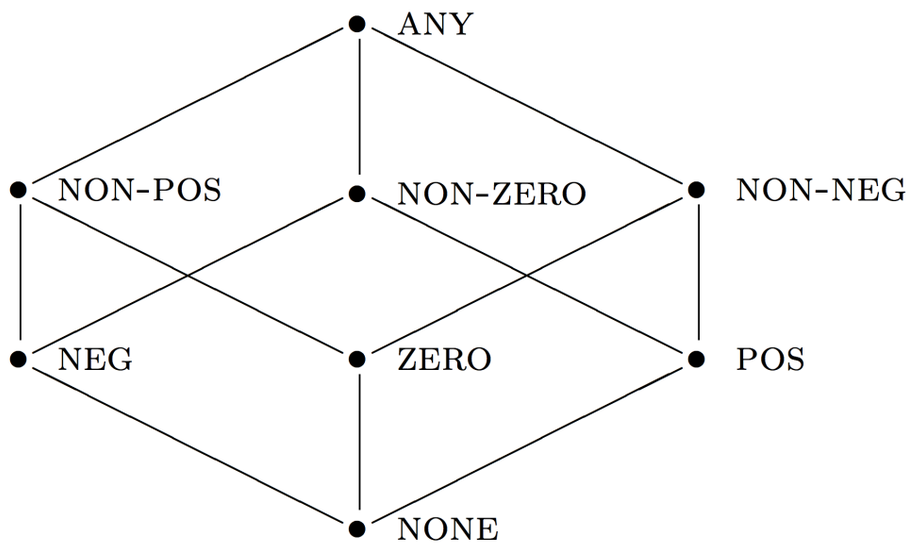
- Expressions
- Names
- Functions
- Pattern matching
- Type inference
- Tuples and lists
- Data types
- Exceptions

Implementation: Simple Sign Analysis

- Language:

$$
\begin{array}{lcl}
a & \to & n \mid x \mid a_1 + a_2 \mid a_1 \star a_2 \mid a_1 - a_2 \\
b & \to & \texttt{true} \mid \texttt{false} \mid a_1 = a_2 \mid a_1 \leq a_2 \mid \neg b \mid b_1 \wedge b_2 \\
c & \to & x := a \mid \texttt{skip} \mid c_1; c_2 \mid \texttt{if } b\ c_1\ c_2 \mid \texttt{while } b\ c
\end{array}
$$

- Abstract domain:



LLVM IR References

- “LLVM IR Tutorial-Phis,GEPs and other things, ohmy!”, V.Bridgers, F. Piovezan, EuroLLVM
    - Slides: https://llvm.org/devmtg/2019-04/slides/Tutorial-Bridgers-LLVM_IR_tutorial.pdf
    - Video: https://www.youtube.com/watch?v=m8G_S5LwlTo&feature=youtu.be
- "Mapping High Level Constructs to LLVM IR", M. Rodler
    - https://mapping-high-level-constructs-to-llvm-ir.readthedocs.io/en/latest/
- LLVM Language Reference Manual
    - https://llvm.org/docs/LangRef.html

LLVM IR References

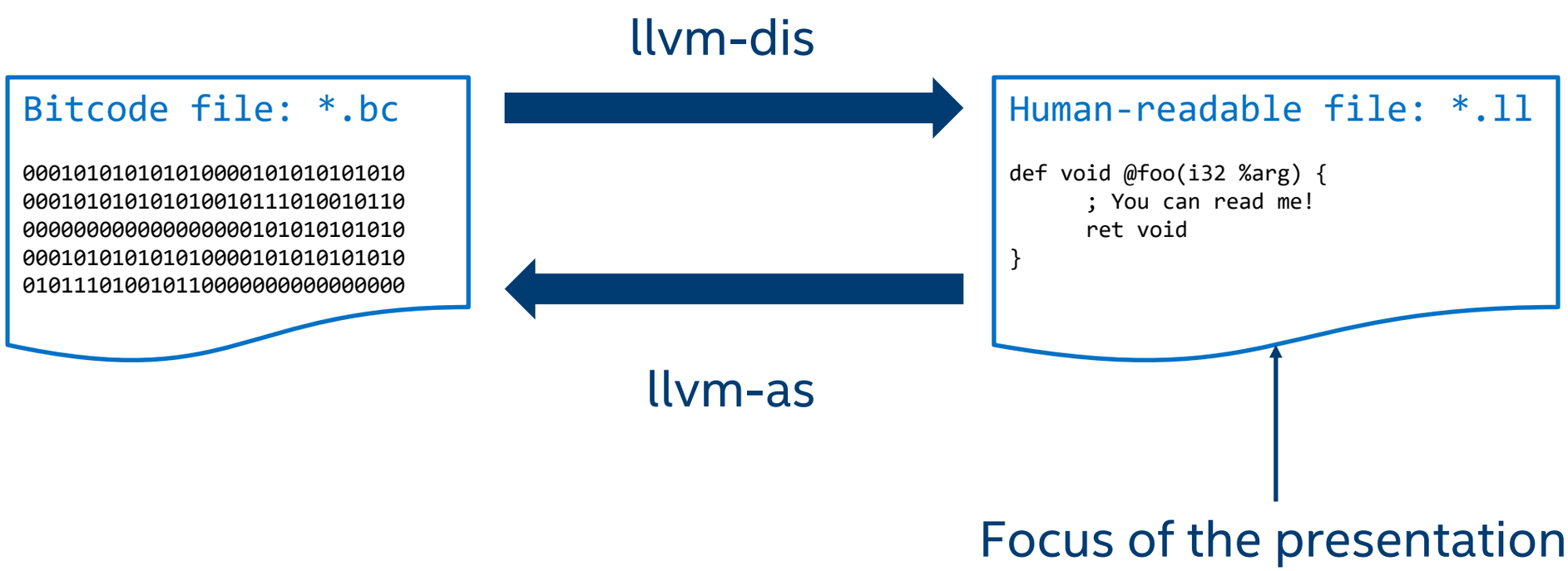
- “LLVM IR Tutorial-Phis,GEPs and other things, ohmy!”, V.Bridgers, F. Piovezan, EuroLLVM
  - Slides: https://llvm.org/devmtg/2019-04/slides/Tutorial-Bridgers-LLVM_IR_tutorial.pdf
  - Video: https://www.youtube.com/watch?v=m8G_S5LwlTo&feature=youtu.be
- "Mapping High Level Constructs to LLVM IR", M. Rodler
  - https://mapping-high-level-constructs-to-llvm-ir.readthedocs.io/en/latest/
- LLVM Language Reference Manual
  - https://llvm.org/docs/LangRef.html

What is the LLVM IR?

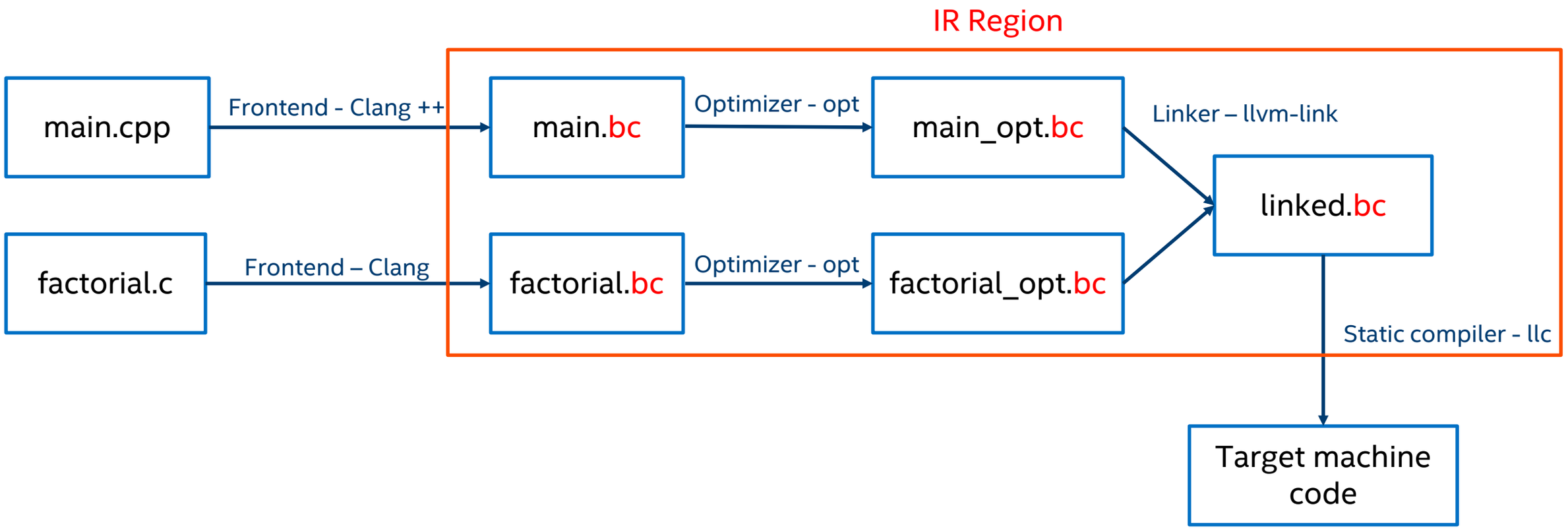
The LLVM **I**ntermediate **R**epresentation:

- Is a low level programming language
  - RISC-like instruction set
- … while being able to represent high-level ideas.
  - i.e. high-level languages can map cleanly to IR.
- Enables efficient code optimization

IR representation



IR & the compilation process



From C to LLVM IR

- Install llvm-10 and clang-10

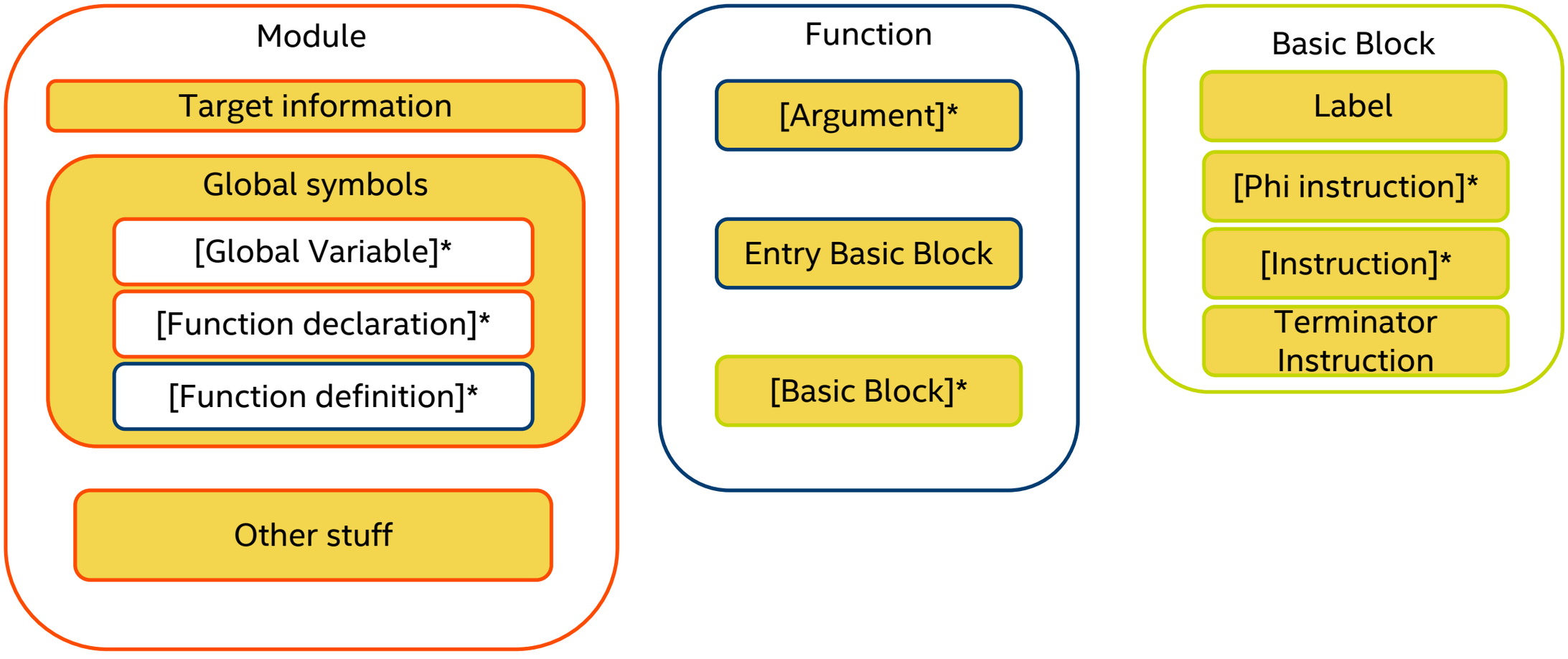
  sudo apt-get install llvm-10 lldb-10 llvm-10-dev libllvm10 llvm-10-runtime

  sudo apt-get install clang-10

- Use Clang to convert C to LLVM IR

  clang-10 -c -emit-llvm -S -fno-discard-value-names -Xclang -disable-O0-optnone -o FILENAME.ll -g FILENAME.c

Simplified IR layout

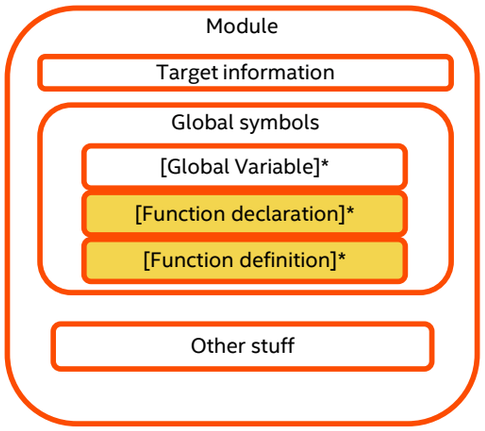


A basic main program

Hand-written IR for this program:

```
int factorial(int val);

int main(int argc, char** argv)
{
  return factorial(2) * 7 == 42;
}
```

```
declare i32 @factorial(i32)

define i32 @main(i32 %argc, i8** %argv) {
  %1 = call i32 @factorial(i32 2)
  %2 = mul i32 %1, 7
  %3 = icmp eq i32 %2, 42
  %result = zext i1 %3 to i32
  ret i32 %result
}
```



% Virtual Registers %

Those are “local” variables.

Two flavors of names:

- Unnamed: %<number>
- Named: %<name>

“LLVM IR has infinite registers”

```
declare i32 @factorial(i32)

define i32 @main(i32 %argc, i8** %argv) {
  %1 = call i32 @factorial(i32 2)
  %2 = mul i32 %1, 7
  %3 = icmp eq i32 %2, 42
  %result = zext i1 %3 to i32
  ret i32 %result
}
```

Types, types everywhere!

Very much a typed language.

```
declare i32 @factorial(i32)

define i32 @main(i32 %argc, i8** %argv) {
  %1 = call i32 @factorial(i32 2)
  %2 = mul i32 %1, 7
  %3 = icmp eq i32 %2, 42
  %result = zext i1 %3 to i32
  ret i32 %result
}
```

Types, types everywhere!

The instructions explicitly dictate the types expected.

Easy to figure out argument types.

```
declare i32 @factorial(i32)

define i32 @main(i32 %argc, i8** %argv) {
  %1 = call i32 @factorial(i32 2)
  %2 = mul i32 %1, 7
  %3 = icmp eq i32 %2, 42
  %result = zext i1 %3 to i32
  ret i32 %result
}
```

Types, types everywhere!

The instructions explicitly dictate the types expected.

Easy to figure out argument types.

Easy to figure out return types (mostly)

```
declare i32 @factorial(i32)

define i32 @main(i32 %argc, i8** %argv) {
  %1 = call i32 @factorial(i32 2)
  %2 = mul i32 %1, 7
  %3 = icmp eq i32 %2, 42
  %result = zext i1 %3 to i32
  ret i32 %result
}
```

Types, types everywhere!

No implicit conversions!

```
declare i32 @factorial(i32)

define i32 @main(i32 %argc, i8** %argv) {
  %1 = call i32 @factorial(i32 2)
  %2 = mul i32 %1, 7
  %3 = icmp eq i32 %2, 42
  %result = zext i1 %3 to i32
  ret i32 %result
}
```

Types, types everywhere!

No implicit conversions!

To check if this is valid IR:

opt –verify input.ll

```
declare i32 @factorial(i32)

define i32 @main(i32 %argc, i8** %argv) {
  %1 = call i32 @factorial(i32 2)
  %2 = mul i32 %1, 7
  %3 = icmp eq i32 %2, 42

  ret i32 %3
}
```

```
$ opt-10 -verify test.ll
opt-10: test.ll:8:13: error: '%3' defined with type 'i1' but expected 'i32'
    ret i32 %3
            ^
```

The LangRef is your friend

Instructions often have **many** variants.

What else could a call instruction possibly need?

```
declare i32 @factorial(i32)

define i32 @main(i32 %argc, i8** %argv) {
  %1 = call i32 @factorial(i32 2)
  %2 = mul i32 %1, 7
  %3 = icmp eq i32 %2, 42
  %result = zext i1 %3 to i32
  ret i32 %result
}
```

- LLVM Language Reference Manual: https://llvm.org/docs/LangRef.html

The LangRef is your friend

'call' Instruction

Syntax: `%1 = call i32 @factorial(i32 2)`

```
<result> = [tail | musttail | notail ] call [fast-math flags] [cconv] [ret attrs] [addrspace(<num>)]
           [<ty>|<fnty> <fnptrval>(<function args>) [fn attrs] [ operand bundles ]
```

Overview:

The 'call' instruction represents a simple function call.

Arguments:

This instruction requires several arguments:

The LangRef is your friend

Semantics:

The '`call`' instruction is used to cause control flow to transfer to a specified function, with its incoming arguments bound to the specified values. Upon a '`ret`' instruction in the called function, control flow continues with the instruction after the function call, and the return value of the function is bound to the result argument.

Example:

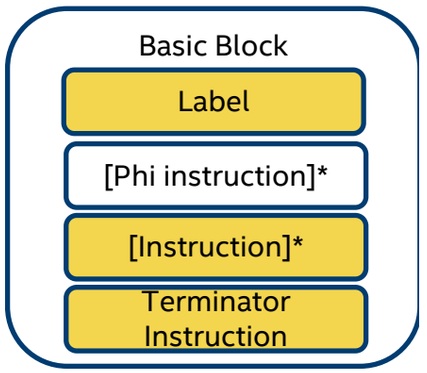
```
%retval = call i32 @test(i32 %argc)
call i32 (i8*, ...)* @printf(i8* %msg, i32 12, i8 42)        ; yields i32
%X = tail call i32 @foo()                                    ; yields i32
%Y = tail call fastcc i32 @foo()  ; yields i32
call void %foo(i8 97 signext)

%struct.A = type { i32, i8 }
%r = call %struct.A @foo()                        ; yields { i32, i8 }
%gr = extractvalue %struct.A %r, 0                ; yields i32
%gr1 = extractvalue %struct.A %r, 1               ; yields i8
%Z = call void @foo() noreturn                    ; indicates that %foo never returns normally
%ZZ = call zeroext i32 @bar()                     ; Return value is %zero extended
```

Recursive factorial

```c
// Precondition: val is non-negative.
int factorial(int val) {
  if (val == 0)
    return 1;
  return val * factorial(val - 1);
}
```

```
; Precondition: %val is non-negative.
define i32 @factorial(i32 %val) {
  %is_base_case = icmp eq i32 %val, 0
  br i1 %is_base_case, label %base_case, label %recursive_case
base_case:
  ret i32 1
recursive_case:
  %1 = add i32 -1, %val
  %2 = call i32 @factorial(i32 %0)
  %3 = mul i32 %val, %1
  ret i32 %2
}
```

Basic Blocks



List of non-terminator instructions ending with a terminator instruction:

- **Branch - “br”**
- **Return - “ret”**
- Switch – “switch”
- Unreachable – “unreachable”
- Exception handling instructions

```
; Precondition: %val is non-negative.
define i32 @factorial(i32 %val) {
  %is_base_case = icmp eq i32 %val, 0
  br i1 %is_base_case, label %base_case, label %recursive_case
base_case:
  ret i32 1
recursive_case:
  %1 = add i32 -1, %val
  %2 = call i32 @factorial(i32 %0)
  %3 = mul i32 %val, %1
  ret i32 %2
}
```

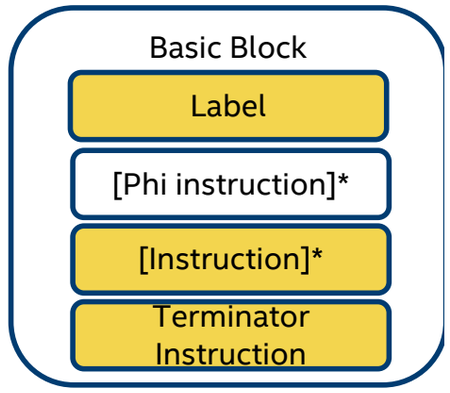
Basic Blocks

List of non-terminator instructions ending with a terminator instruction:

- Return - “ret”

Execution proceeds to:

- calling function




```
; Precondition: %val is non-negative.
define i32 @factorial(i32 %val) {
  %is_base_case = icmp eq i32 %val, 0
  br i1 %is_base_case, label %base_case, label %recursive_case
base_case:
  ret i32 1
recursive_case:
  %1 = add i32 -1, %val
  %2 = call i32 @factorial(i32 %0)
  %3 = mul i32 %val, %1
  ret i32 %2
}
```

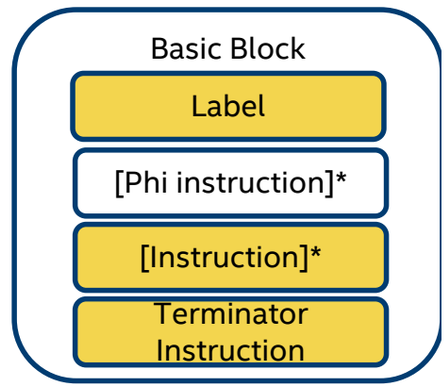
Basic Blocks

List of non-terminator instructions ending with a terminator instruction:

- Branch - “br”

Execution proceeds to:

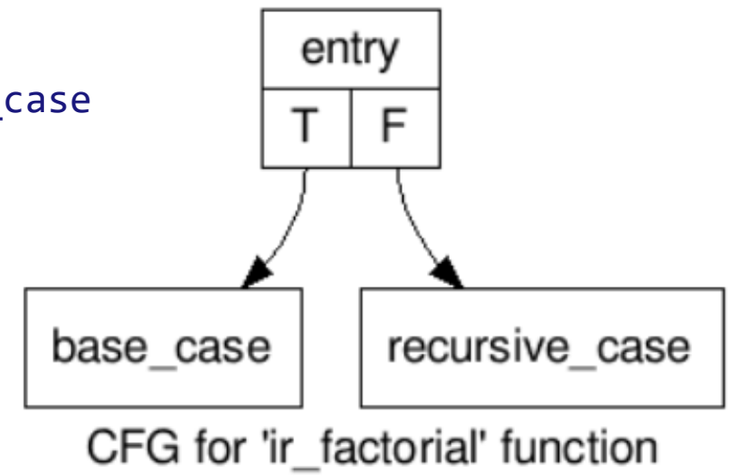
- another Basic Block
  - It’s **successor**!




```
; Precondition: %val is non-negative.
define i32 @factorial(i32 %val) {
  %is_base_case = icmp eq i32 %val, 0
  br i1 %is_base_case, label %base_case, label %recursive_case
base_case:
  ret i32 1
recursive_case:
  %1 = add i32 -1, %val
  %2 = call i32 @factorial(i32 %0)
  %3 = mul i32 %val, %1
  ret i32 %2
}
```

Control Flow Graph (CFG)

```
; Precondition: %val is non-negative.
define i32 @factorial(i32 %val) {
entry:
  %is_base_case = icmp eq i32 %val, 0
  br i1 %is_base_case, label %base_case, label %recursive_case
base_case:                        ; preds = %entry
  ret i32 1
recursive_case:                   ; preds = %entry
  %0 = add i32 -1, %val
  %1 = call i32 @factorial(i32 %0)
  %2 = mul i32 %val, %1
  ret i32 %2
}
```



CFG for 'ir_factorial' function

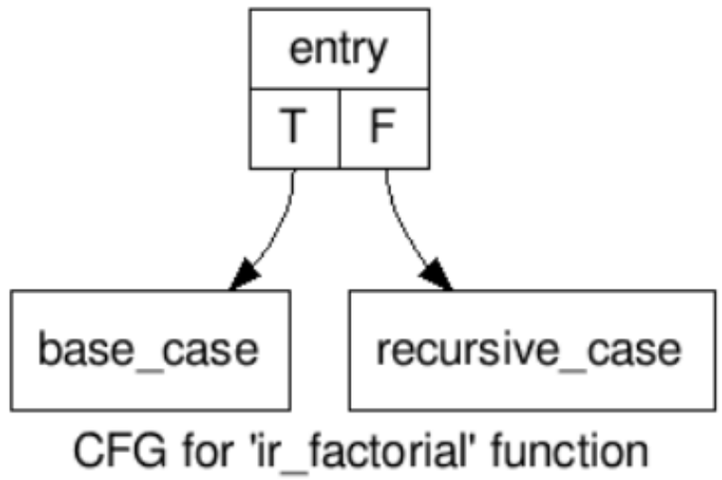
Automatically generated comments

Control Flow Graph (CFG)

The optimizer can generate the CFG in dot format:

opt –analyze –dot-cfg**-only** <input.ll>

-dot-cfg-only = Generate .dot files. Don’t include instructions.



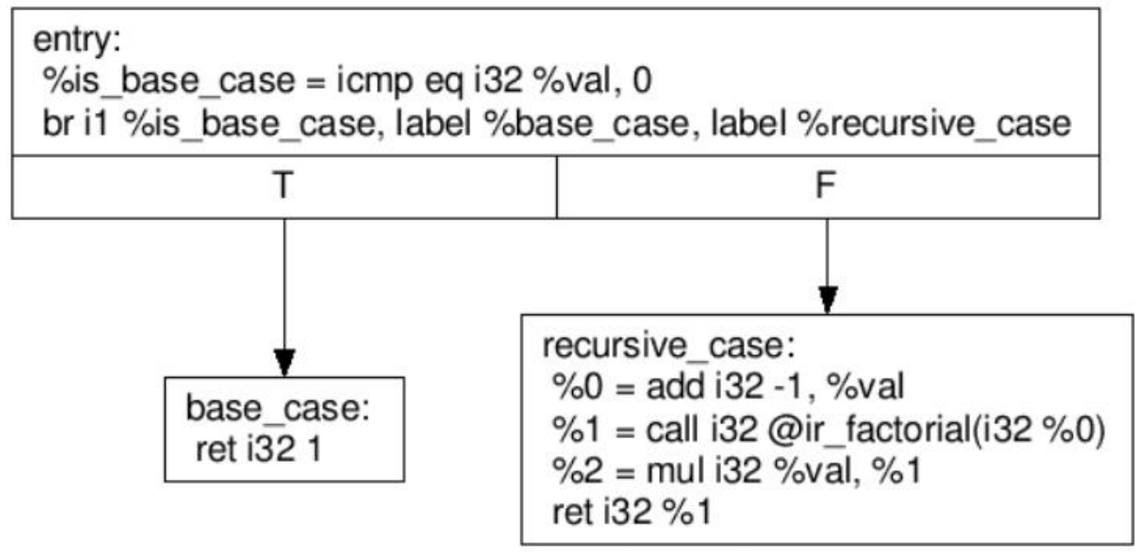
CFG for 'ir_factorial' function

Control Flow Graph (CFG)

The optimizer can generate the CFG in dot format:

opt –analyze –dot-cfg <input.ll>

-dot-cfg = Generate .dot files.



CFG for 'ir_factorial' function

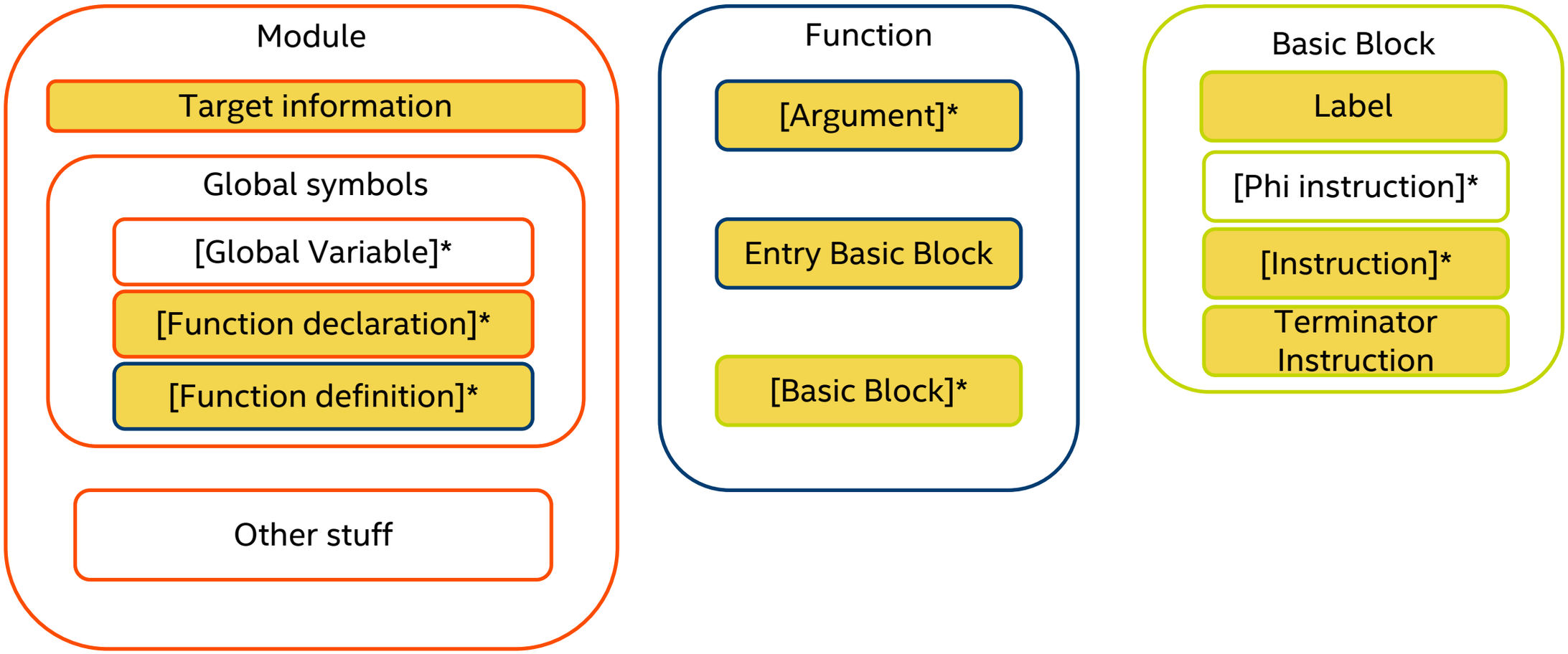
Implicit labels

Every Basic Block has a label...

... even if it's not explicit

```
; Precondition: %val is non-negative.
define i32 @factorial(i32 %val) {
entry:
  %is_base_case = icmp eq i32 %val, 0
  br i1 %is_base_case, label %base_case, label %recursive_case

base_case:
  ret i32 1

recursive_case:
  %0 = add i32 -1, %val
  %1 = call i32 @factorial(i32 %0)
  %2 = mul i32 %val, %1
  ret i32 %2
}
```

Simplified IR layout



Iterative factorial

```c
int factorial(int val) {
  int temp = 1;
  for (int i = 2; i <= val; ++i)
    temp *= i;
  return temp;
}
```

```
define i32 @factorial(i32 %val) {
entry:
  %i = add i32 0, 2
  %temp = add i32 0, 1
  br label %check_for_condition
check_for_condition:

  %i_leq_val = icmp sle i32 %i, %val
  br i1 %i_leq_val, label %for_body, label %end_loop
for_body:

  %temp = mul i32 %temp, %i
  %i = add i32 %i, 1
  br label %check_for_condition
end_loop:
  ret i32 %temp
}
```

You wish you could do this...

Iterative factorial

```
int factorial(int val) {
  int temp = 1;
  for (int i = 2; i <= val; ++i)
    temp *= i;
  return temp;
}
```

```
define i32 @factorial(i32 %val) {
entry:
  %i = add i32 0, 2
  %temp = add i32 0, 1
  br label %check_for_condition
check_for_condition:

  %i_leq_val = icmp sle i32 %i, %val
  br i1 %i_leq_val, label %for_body, label %end_loop
for_body:

  %temp = mul i32 %temp, %i
  %i = add i32 %i, 1
}
```

You wish you could do this...

```
opt: test.ll:12:5: error: multiple definition of local value named 'temp'
    %temp = mul i32 %temp, %i
    ^
```

Static Single Assignment (SSA)

Every variable is assigned *exactly* once.

Every variable is defined before it is used.

Iterative factorial

```
int factorial(int val) {
  int temp = 1;
  for (int i = 2; i <= val; ++i)
    temp *= i;
  return temp;
}
```

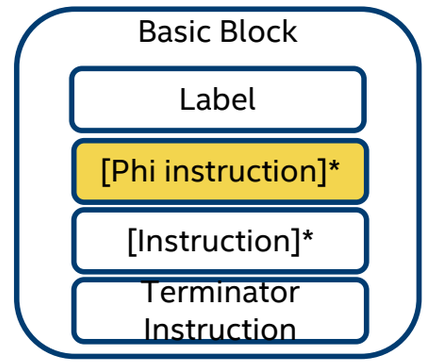
```
define i32 @factorial(i32 %val) {
entry:
  %i = add i32 0, 2
  %temp = add i32 0, 1
  br label %check_for_condition
check_for_condition:

  %i_leq_val = icmp sle i32 %i, %val
  br i1 %i_leq_val, label %for_body, label %end_loop
for_body:

  %new_temp = mul i32 %temp, %i
  %i_plus_one = add i32 %i, 1
  br label %check_for_condition
end_loop:
  ret i32 %temp
}
```

Now *%i* is always *2!*

So you do this:

Now *%temp* is always *1!*

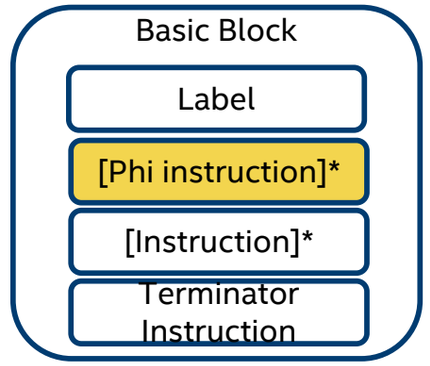
Phis to the rescue!




```
<result> = phi <ty> [<val0>, <label0>], [<val1>, <label1>] …
```

Select a value based on the **BasicBlock** that executed previously!

Phis to the rescue!




```
entry:


 br label %check_for_condition
```

```
check_for_condition:
 %current_i = phi i32 [2, %entry], [%i_plus_one, %for_body]
 %temp      = phi i32 [1, %entry], [%new_temp, %for_ body]
 %i_leq_val = icmp sle i32 %current_i, %val
 br i1 %i_leq_val, label %for_body, label %end_loop
```

True

False

```
for_body:
  %new_temp = mul i32 %temp, %current_i
  %i_plus_one = add i32 %current_i, 1
  br label %check_for_condition
```

```
end_loop:
  ret i32 %temp
```

Another way to cheat SSA

Alloca instruction:

- You give it a type, it gives you a pointer to that type:
  - `%ptr = alloca i32 ; ptr is i32*`
  - `%ptr = alloca <any_type> ; ptr is <any_type>*`
- Allocates memory on the stack frame of the executing function.
- Automatically released.
  - Akin to changing the stack pointer.
- Plays a big part in generating IR in SSA form.

Allocas to the rescue!

```
entry:
  %i.addr = alloca i32
  %temp.addr = alloca i32
  store i32 2, i32* %i.addr
  store i32 1, i32* %temp.addr
  br label %check_for_condition
```

```
check_for_condition:
  %current_i = load i32, i32* %i.addr
  %temp      = load i32, i32* %temp.addr
  %i_leq_val = icmp sle i32 %current_i, %val
  br i1 %i_leq_val, label %for_body, label %end_loop
```

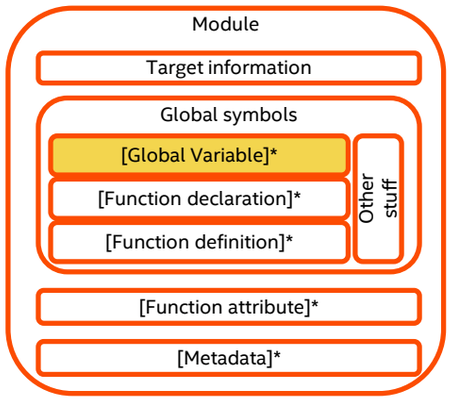
True

False

```
for_body:
  %i_plus_one = add i32 %current_i, 1
  %new_temp = mul i32 %temp, %current_i
  store i32 %i_plus_one, i32* %i.addr
  store i32 %new_temp, i32* %temp.addr
  br label %check_for_condition
```

```
end_loop:
  ret i32 %temp
```

Global variables



Allocas allocate memory for function scopes.

Global variables fill that role for the module in a static way.

- They are always pointers, like the values returned by Allocas.

Global variables

- Name prefixed with “@”. `@gv =`

Are always pointers

Always **constant pointers!**

```
@gv = global i16 46
; …
; … Inside some function:
%load = load i16 , i16* @gv
store i16 0, i16* @gv
```

Building a Static Analyzer for LLVM IR

- We use llir, a library for LLVM IR in Go
  - Code: https://github.com/llir/llvm
  - Documentation: https://godoc.org/github.com/llir/llvm
- Infrastructure
  - https://github.com/kupl/AAA616
- Sign Analysis
  - https://github.com/kupl/AAA616/tree/main/sign

main.go

```go
package main

import (
    "flag"
    "fmt"

    "github.com/llir/llvm/asm"
)

func main() {
    flag.Parse()
    args := flag.Args()
    filename := args[0]
    m, err := asm.ParseFile(filename)
    if err != nil {
        panic(err)
    }
    md := NewModule(m)
    for _, cfg := range md.cfgs {
        fmt.Printf("Analysis of %s begins...\n", cfg.GetFid())
        tbl := Analyze(cfg)
        fmt.Println()
        fmt.Println("Analysis Results for:", cfg.GetFid())
        fmt.Println(tbl.String())
        fmt.Println()
    }
}
```

```go
type Module struct {
    typeDefs []types.Type
    globals []*ir.Global
    cfgs []Cfg
}


type Cfg struct {
    fid string
    sig *types.FuncType
    params []*ir.Param
    blocks []Node
    succ map[Node][]Node
    pred map[Node][]Node
}
```

program.go

fixpoint.go

```
type Table map[Node]State
type Worklist []Node

func Analyze(cfg Cfg) Table {
    var tbl Table
    var worklist Worklist
    tbl = NewTable()
    worklist = NewWorklist()
    worklist.AddSet(cfg.blocks)

    for !worklist.IsEmpty() {
        here := worklist.Choose()
        state := InputOf(here, cfg, tbl)
        state.TransferBlock(here.Insts)
        old_state := tbl.Find(here)
        if !StateOrder(state, old_state) {
            if NeedWiden(here) {
                tbl.Bind(here, StateWiden(old_state, state))
            } else {
                tbl.Bind(here, StateJoin(old_state, state))
            }
            worklist.AddSet(cfg.Succ(here))
        }
    }
    return tbl
}
```

domain.go

```go
package main

import (
    "github.com/llir/llvm/ir"
)

//////////////////////
// Sign Domain //
//////////////////////
type Sign interface {
    String() string
}

type Top struct { }
type Bot struct { }
type Pos struct { }
type Neg struct { }
type Zero struct { }

func (b Bot) String() string { return "Bot" }
func (b Top) String() string { return "Top" }
func (b Pos) String() string { return "Pos" }
func (b Neg) String() string { return "Neg" }
func (b Zero) String() string { return "Zero" }

func SignTop() Sign { return Top{} }
func SignBot() Sign { return Bot{} }

func SignFromInt (l int) Sign {
    if l > 0 {
        return Pos{}
    } else if l < 0 {
```

semantics.go

```
func (s *State) transferInst(inst ir.Instruction) {
    switch inst := inst.(type) {
    case *ir.InstAdd: s.transferInstAdd(inst)
    case *ir.InstSub: s.transferInstSub(inst)
    case *ir.InstMul: s.transferInstMul(inst)
    case *ir.InstICmp: s.transferInstICmp(inst)
    case *ir.InstPhi: s.transferInstPhi(inst)
    case *ir.InstCall: s.transferInstCall(inst)
    default: fmt.Printf("Unsupported instruction: %T\n", inst)
    }
}

func (s *State) TransferBlock(insts []ir.Instruction) {
    for _, inst := range insts {
        s.transferInst(inst)
    }
}
```

Exercise 1: Interval Analysis for LLVM IR

- https://github.com/kupl/AAA616/tree/main/interval

main ▾ AAA616 / interval / Go to file Add file ···

오학주 and 오학주 readme 5b130c2 on 15 Oct 2020 History

..		
README.md	readme	16 months ago
domain.go	add initial files	16 months ago
fixpoint.go	add initial files	16 months ago
main.go	add initial files	16 months ago
program.go	add initial files	16 months ago
semantics.go	add initial files	16 months ago
worklist.go	add initial files	16 months ago

README.md

Project: Building an Interval Analysis for LLVM IR

Running the analysis

```
$ go build
$ ./interval ../test/toy/ll/fact.ll
```

Exercise 2: Taint Analysis for LLVM IR

Can the information from the untrustworthy source be transferred to the sink?

```
x:=source(); ...; sink(y)
```

Applications to sw security:

- privacy leak
- SQL injection
- buffer overflow
- integer overflow
- XSS
- ...

- Very simple but useful analysis:

Summary

- Introduction to program analysis
- Static analysis theory / implementation / applications
- Core principles behind modern static analysis tools

Infer

...

Clang Static Analyzer

Summary

- Introduction to program analysis
- Static analysis theory / implementation / applications
- Core principles behind modern static analysis tools

Infer

Clang Static Analyzer

COVERITY® BY SYNOPSYS

PolySpace TECHNOLOGIES

sonarqube

Sparrow The Early Bird

...

Thank you!