

소프트웨어 증명 기술을 이용한 스마트 컨트랙트 취약점 자동 검증

오학주

고려대학교 정보대학

Dec 13, 2018 @정보보호단기강좌

소개

- 소속: 고려대학교 정보대학 컴퓨터학과
- 전공: 프로그래밍 언어, 소프트웨어 분석, 소프트웨어 보안
- 웹페이지: <http://prl.korea.ac.kr>
- 슬라이드: <http://prl.korea.ac.kr/~pronto/home/talks/sec18.pdf>

강의 내용

- 소프트웨어 증명 기술의 원리 및 응용
 - 필요성: 소프트웨어 결함 문제
 - 소프트웨어 분석 기법 개괄
 - 소프트웨어 증명(Software Verification) 기법
 - 응용: 스마트 컨트랙트 안전성 검증

소프트웨어 결함 문제

- 2017년 소프트웨어 결함으로 인한 사회적 비용은 1.7조 달러로 추정 (Software failure watch, 2017)



소프트웨어 결함 사례

아리안 5 로켓 폭발 (1996). 개발기간 10년, 개발 비용 \$80억.



NASA 화성 탐사선 실종 (1998).



금융 거래 소프트웨어 오류 (2012)

Knight Capital Says Trading Glitch Cost It \$440 Million

BY NATHANIEL POPPER | AUGUST 2, 2012 9:07 AM | 356

Runaway Trades Spread Turmoil Across Wall St.



테슬라 자율주행차 소프트웨어 결함 (2017).

Tesla in fatal California crash was on Autopilot

31 March 2018

f Share



SmartMesh (2018)

BatchOverflow Exploit Creates Trillions of Ethereum Tokens, Major Exchanges Halt ERC20 Deposits

Sam Town

April 25, 2018

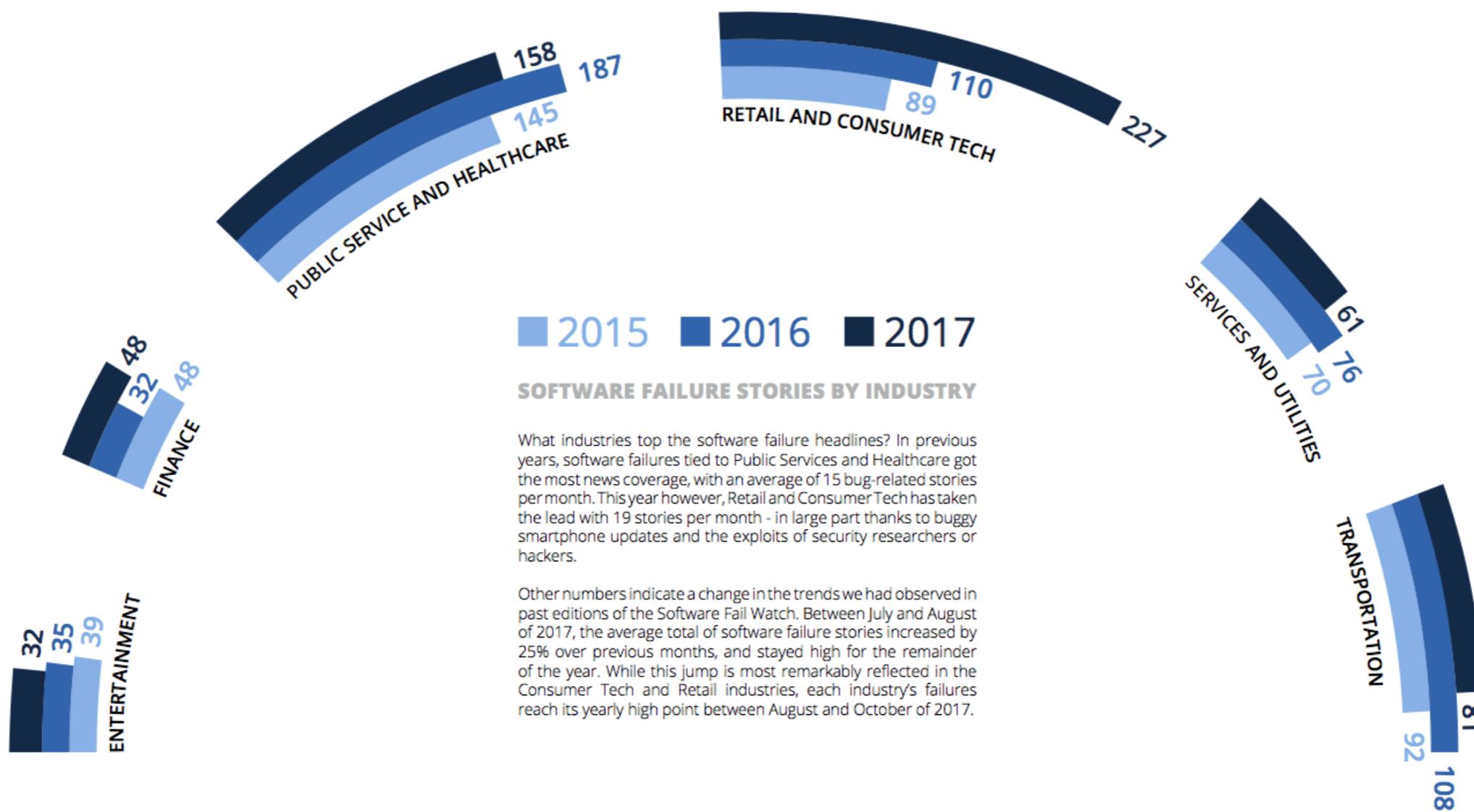
3 min read

6028 Views



사회 모든 영역에서 발생

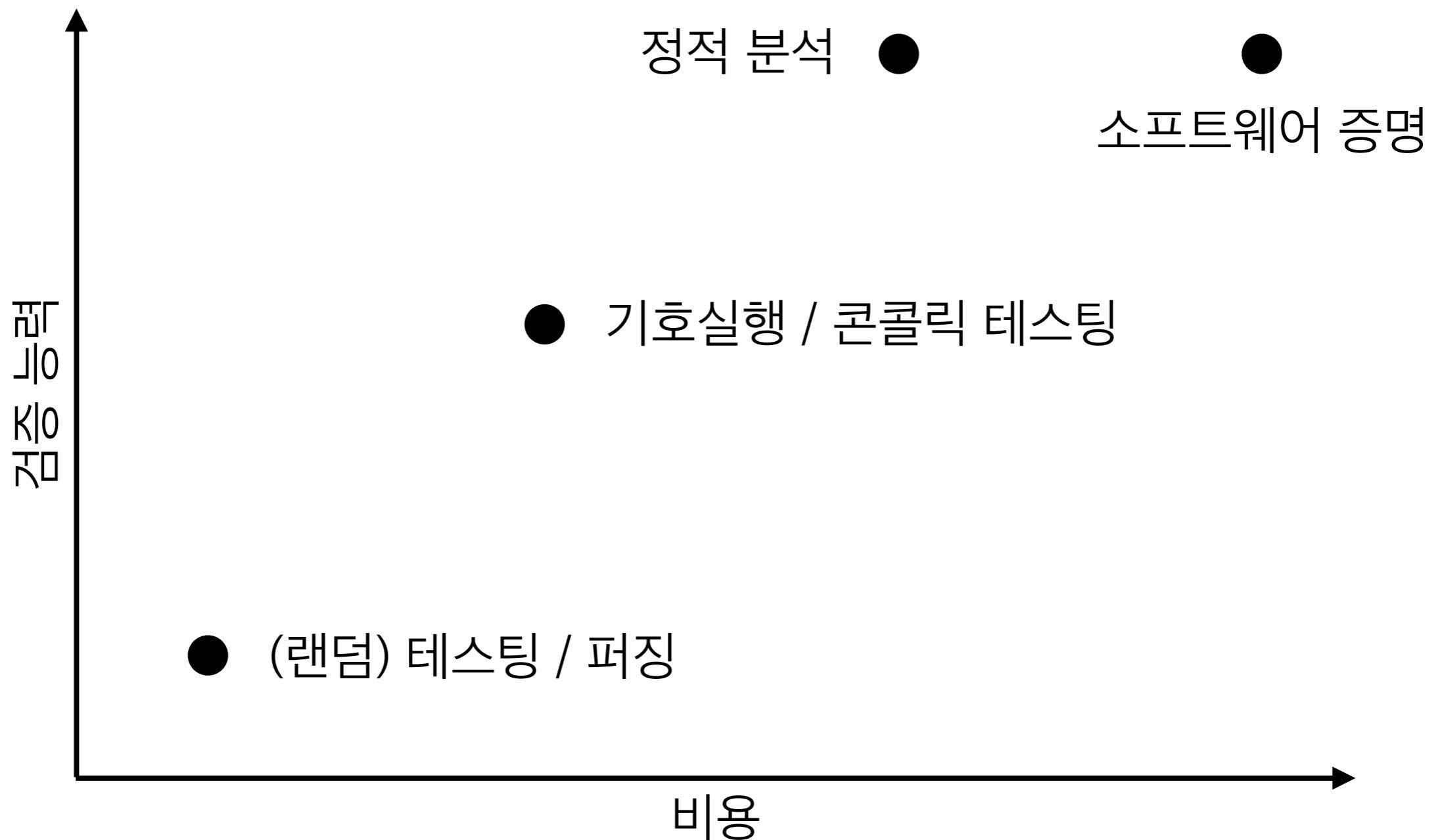
- 금융, 가전, 공공, 교통, 헬스케어, ...



Software fail watch (5th ed) 2017

소프트웨어 분석 기법

- 소프트웨어의 실행 성질을 분석하여 사전에 오류를 탐지하는 기술
- 검증 능력과 비용의 trade-off에 따라 다양한 기법이 존재



랜덤 테스팅

- 무작위로 입력을 생성하여 테스팅

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

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

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

랜덤 테스팅

- 무작위로 입력을 생성하여 테스팅

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

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

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

< 0.4%

기호 실행 (Symbolic Execution)

- 프로그램을 실제값이 아닌 기호를 이용하여 실행

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

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

$x=a, y=\beta$

true

기호 실행 (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;

```
    }
```

```
}
```

```
}
```

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

true

기호 실행 (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;  
        }  
    }  
}
```

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

기호 실행 (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; ←—————  
        }  
    }  
}
```

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

$2^*\beta = a \wedge$
 $a > \beta + 10$

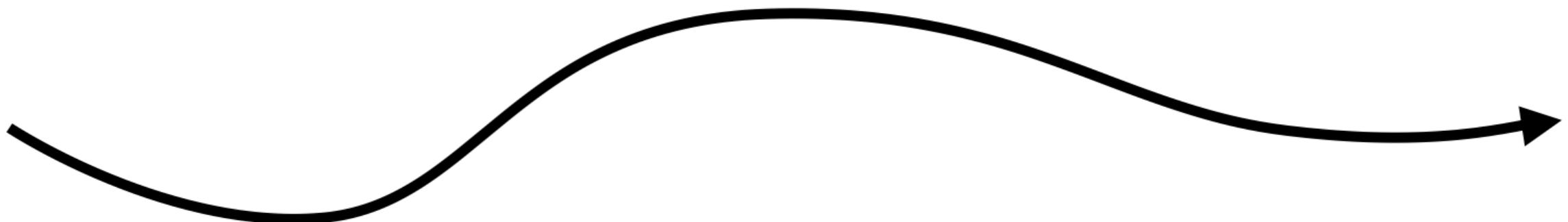
error-triggering
input

→ $x=30, y=15$
SMT solver

정적 분석

- 프로그램의 실제실행을 요약(abstraction)하여 분석

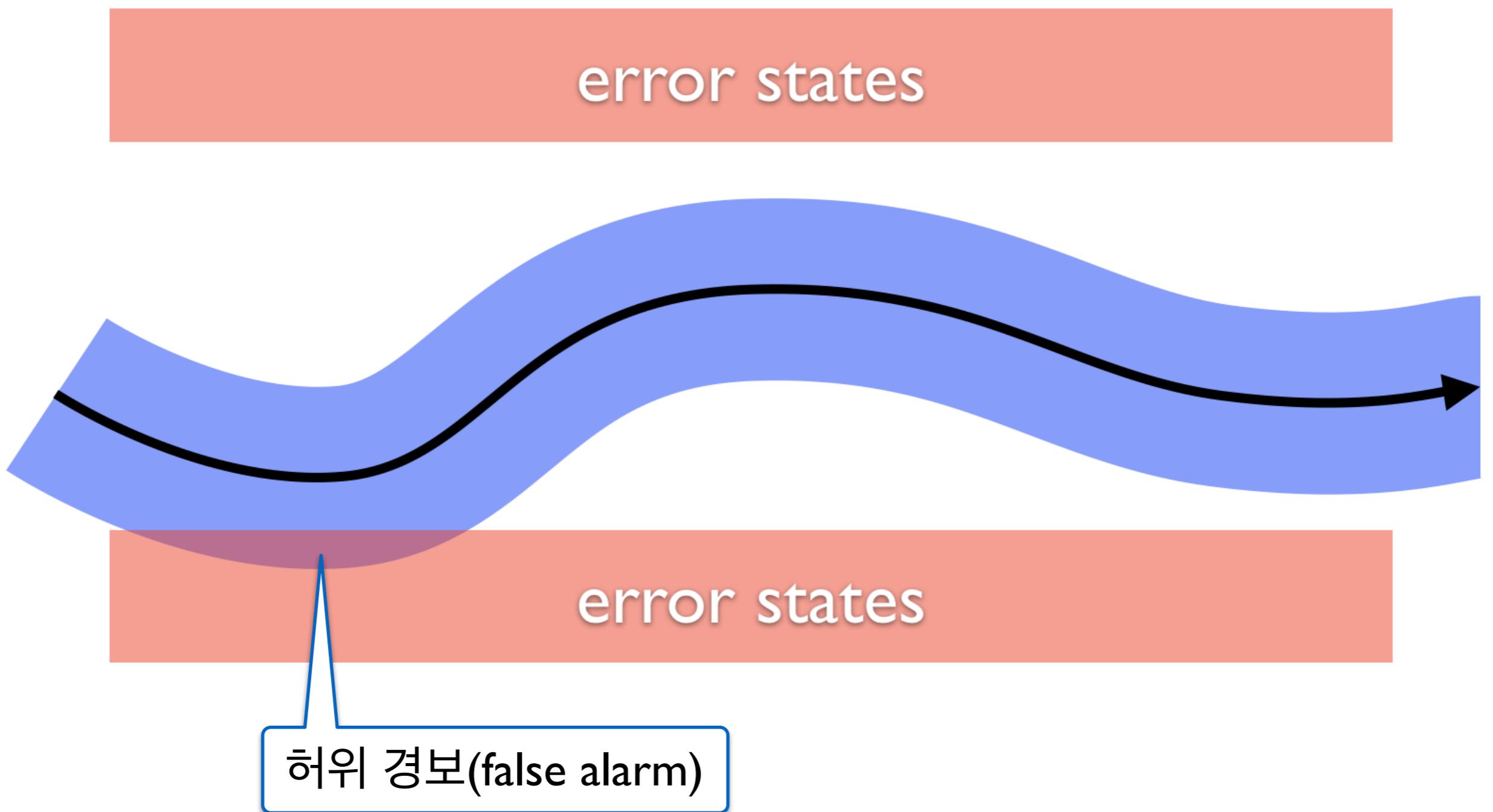
error states



error states

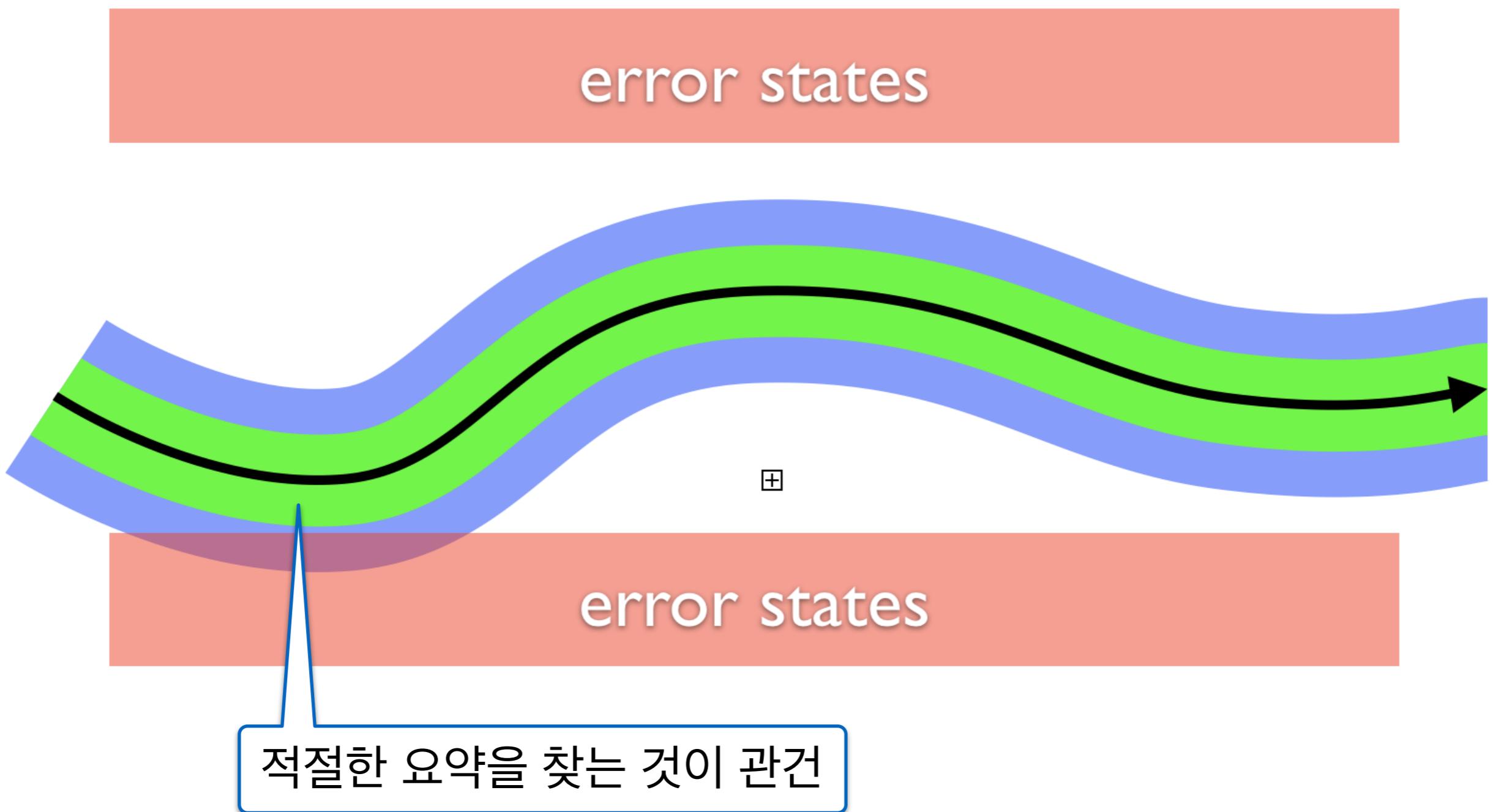
정적 분석

- 프로그램의 실제실행을 요약(abstraction)하여 분석

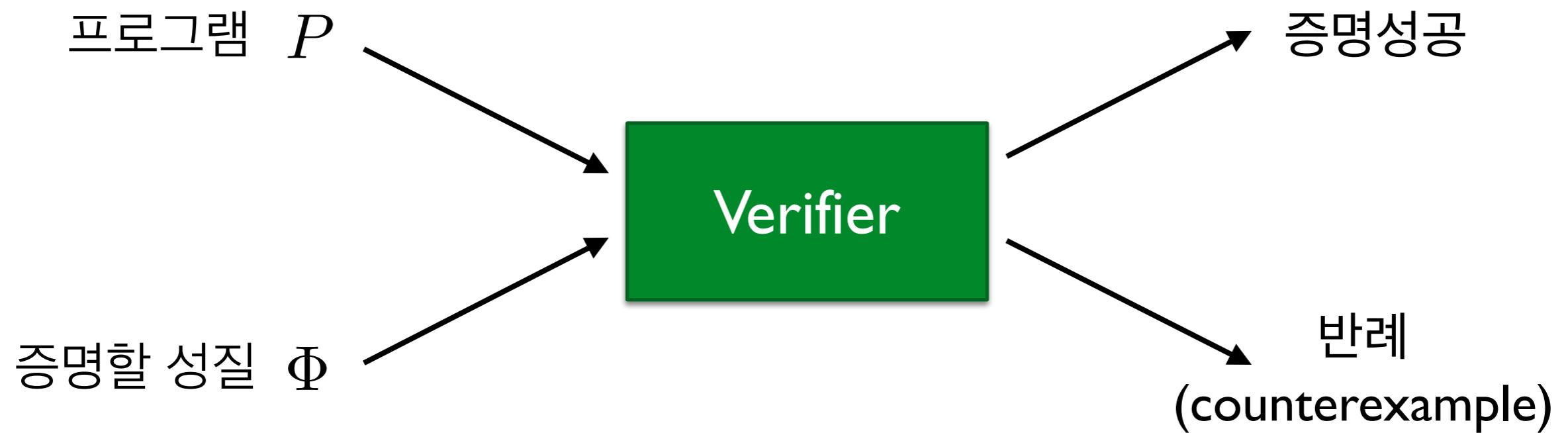


정적 분석

- 프로그램의 실제실행을 요약(abstraction)하여 분석



소프트웨어 증명 (Software Verification)



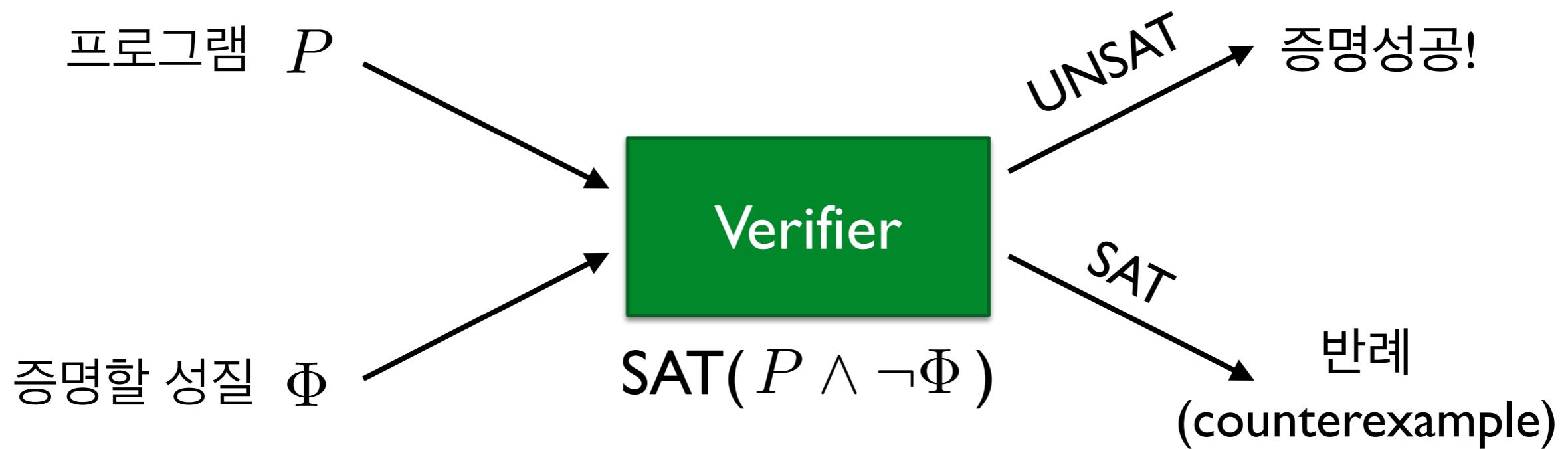
Satisfiability (SAT) Problem

- 명제(Proposition): 참 또는 거짓인 문장
 - true, false
 - 비가 온다 (P)
 - 날이 흐리다 (Q)
 - 비가 오지 않는다 ($\neg P$)
 - 비가 오고 날이 흐리다 ($P \wedge Q$)
 - 비가 오거나 날이 흐리다 ($P \vee Q$)
 - 비가 오면 날이 흐리다 ($P \rightarrow Q$)
- Satisfiability (SAT): 주어진 명제가 참이 될 수 있는지 여부
 - true, false
 - $P \wedge Q$
 - $P \wedge \neg P$
 - $P \vee \neg P$
 - $P \rightarrow Q$

Satisfiability Modulo Theory (SMT) Problem

- SAT을 일차논리(First-order logic)로 확장: e.g.,
 - Theory of Equality $a = b \wedge b = c \rightarrow a = c$
 - Theory of Integers $\exists x, y. x = y + 1$
 - Theory of arrays $\forall i, j. i = j \rightarrow a[i] = a[j]$
- SMT solvers (e.g., Z3, Yices, CVC4)
 - 일차 논리식의 참/거짓 여부를 주어진 theory 내에서 판단

소프트웨어 증명 (Software Verification)



- 프로그램과 증명할 성질을 일차 논리식으로 표현
- 논리식의 satisfiability 여부를 판별

예제

- 증명할 성질을 assert 문으로 표현

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

예제

- 프로그램과 증명할 성질의 반대(negation)를 논리식으로 표현

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

((a ∧ x) ∨ (¬a ∧ ¬x)) ∧
((a ∧ y) ∨ (¬a ∧ ¬y)) ∧
¬(x == y)

예제

- 프로그램과 증명할 성질의 반대(negation)를 논리식으로 표현

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

$$\begin{aligned} & ((a \wedge x) \vee (\neg a \wedge \neg x)) \wedge \\ & ((a \wedge y) \vee (\neg a \wedge \neg y)) \wedge \\ & \neg(x == y) \end{aligned}$$

SAT/SMT solver: unsatisfiable!

예제

- 증명이 불가능한 경우에는 반례를 제공

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

$$\begin{aligned} & ((a \wedge x) \vee (\neg a \wedge \neg x)) \wedge \\ & ((b \wedge y) \vee (\neg b \wedge \neg y)) \wedge \\ & \neg(x == y) \end{aligned}$$

예제

- 증명이 불가능한 경우에는 반례를 제공

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

$$\begin{aligned} & ((a \wedge x) \vee (\neg a \wedge \neg x)) \wedge \\ & ((b \wedge y) \vee (\neg b \wedge \neg y)) \wedge \\ & \neg(x == y) \end{aligned}$$

SAT/SMT solver:
satisfiable when $a=1$ and $b=0$

임의의 성질을 증명 가능

- 일차 논리식으로 표현가능한 모든 성질: e.g., 정렬여부

```
bool BubbleSort (int a[]) {  
    int[] a := a0  
    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;  
}
```

임의의 성질을 증명 가능

- 일차 논리식으로 표현가능한 모든 성질: e.g., 정렬여부

```
bool BubbleSort (int a[]) {  
    int[] a := a0  
    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;  
            }  
        }  
    }  
    assert( $\forall i, j. 0 \leq i \leq j < |a| \rightarrow a[i] \leq a[j]$ );  
}
```

반복문의 불변 성질 (Loop Invariant)

- 프로그램을 논리식으로 변환하려면 반복문마다 불변 성질을 기술해 주어야 함
- 반복문의 불변성질: 반복횟수와 상관없이 항상 성립하는 성질

- 예:

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

반복문의 불변 성질 (Loop Invariant)

- 프로그램을 논리식으로 변환하려면 반복문마다 불변 성질을 기술해 주어야 함
- 반복문의 불변성질: 반복횟수와 상관없이 항상 성립하는 성질

• 예:

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

i ≥ 0 ,
j ≥ 0 ,
i == j,
...



반복문의 불변 성질 (Loop Invariant)

- 프로그램을 논리식으로 변환하려면 반복문마다 불변 성질을 기술해 주어야 함
- 반복문의 불변성질: 반복횟수와 상관없이 항상 성립하는 성질

예: $i = 0;$
 $j = 0;$
while \leftarrow
 $(i < 10)$
{
 $i++;$
 $j++;$
}

$i \geq 0,$
 $j \geq 0,$
 $i == j,$
...

대상 성질을 증명하는데
필요한 불변식을 선택

프로그램 증명의 핵심 문제

- 대상 성질을 증명하는데 필요한 불변식을 찾는 것이 핵심
- 일반적으로 그러한 불변식을 자동으로 알아내는 것은 불가능

```
bool BubbleSort (int a[]) {  
    int[] a := a0  
    @L1 [  $-1 \leq i < |a|$   
         $\wedge \text{partitioned}(a, 0, i, i + 1, |a| - 1)$   
         $\wedge \text{sorted}(a, i, |a| - 1)$  ]  
    for (int i := |a| - 1; i > 0; i := i - 1) {  
        @L2 [  $1 \leq i < |a| \wedge 0 \leq j \leq i$   
             $\wedge \text{partitioned}(a, 0, i, i + 1, |a| - 1)$   
             $\wedge \text{partitioned}(a, 0, j - 1, j, j)$   
             $\wedge \text{sorted}(a, i, |a| - 1)$  ]  
        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;  
}
```

응용: 스마트 컨트랙트

- 블록체인에서는 중계자 없이 P2P로 계약을 체결
- 프로그래밍 언어로 작성된 계약서. 특정 조건이 만족되면 실행

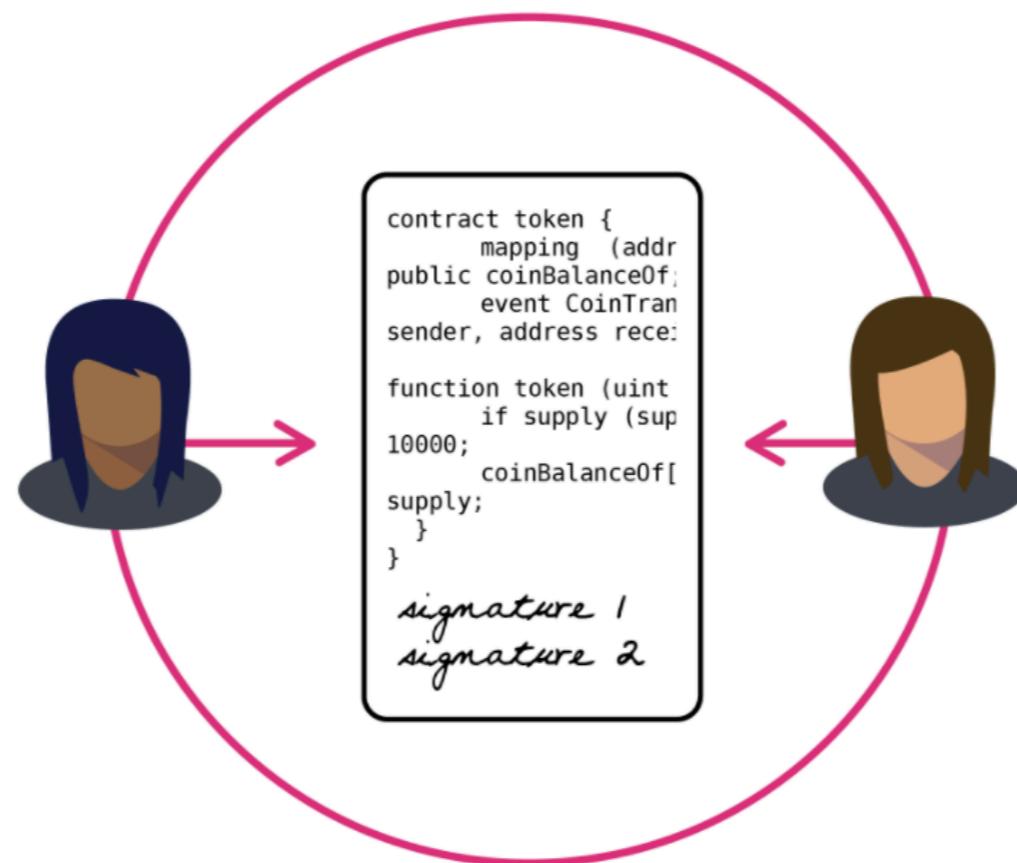


image from <https://www.coindesk.com/information/ethereum-smart-contracts-work>

스마트 컨트랙트 생김새



```
1 contract Netcoin {
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        require (balance[to] + value > balance[to]);
14        balance[msg.sender] -= value;
15        balance[to] += value;
16        return true;
17    }
18
19    function burn (uint value) public returns (bool) {
20        require (balance[msg.sender] >= value);
21        balance[msg.sender] -= value;
22        totalSupply -= value;
23        return true;
24    }
25 }
```

데이터

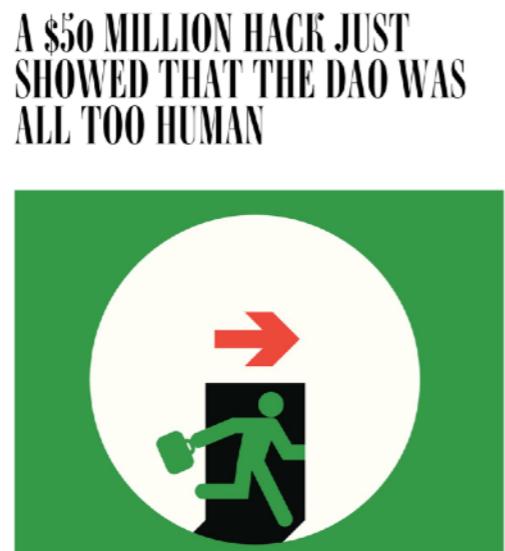
생성자

트랜잭션

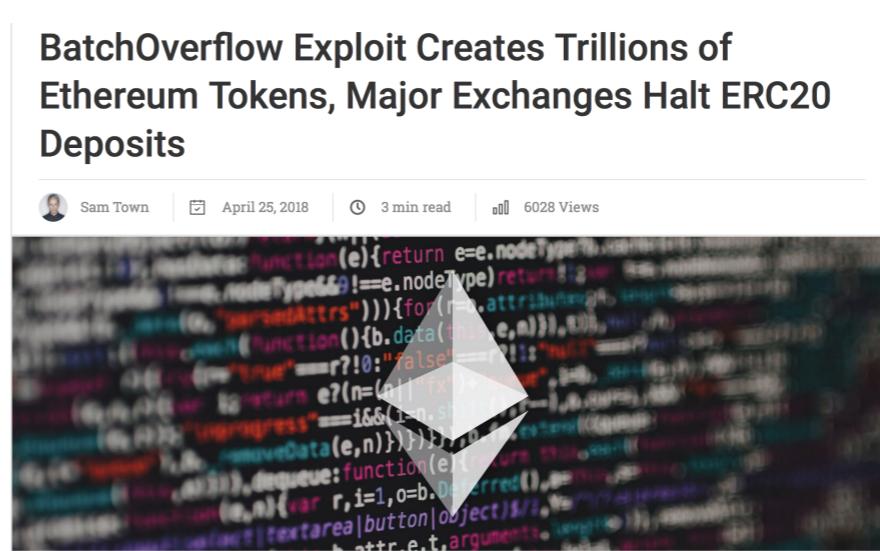
트랜잭션

스마트 컨트랙트의 안전성 문제

- 안전하고 정확하게 동작하는 스마트 컨트랙트 작성은 매우 어려움
 - 제한없는 일반적인 소프트웨어 (Turing-complete)
- 스마트 컨트랙트는 위험에 무방비로 노출
 - 누구나 온라인에서 소스코드 열람 가능하며 수정 불가
 - 공격에 성공하면 막대한 금전적 피해가 발생



The DAO (2016)
750억원



SmartMesh (2018)
천문학적 금액 인출 시도

현재 상황

- 사람이 수작업으로 코드 감사(auditing) 수행
 - 많은 비용 소요, 놓치는 문제들이 존재
- Ex) Parity Wallet 해킹 사례 (2017)
 - 다중 서명 지갑의 취약점으로 인해 350억원 피해
 - 이더리움 개발자들이 코드 감사를 진행했던 코드
- 다른 소프트웨어보다 더욱 엄밀한 검증 기술이 필요



현재 상황

- 사람이 수작업으로 코드 감사(auditing) 수행
 - 많은 비용 소요, 놓치는 문제들이 존재
- Ex) Parity Wallet 해킹 사례 (2017)
 - 다중 서명 지갑의 취약점으로 인해 350억원 피해
 - 이더리움 개발자들이 코드 감사를 진행했던 코드
- 다른 소프트웨어보다 더욱 엄밀한 검증 기술이 필요



VeriSmart: 스마트 컨트랙트 안전성 자동 검증기

정수 오버플로우 취약점

- Solidity에서는 정수를 유한한 비트로 표현

```
uint public totalSupply;
```

- 정수 연산시 표현 가능한 범위를 넘어서는 문제가 발생 가능

```
totalSupply += value;
```

- 오버플로우 유무를 판단하기가 매우 까다로움

- CVE 등록된 스마트 컨트랙트 중 90% (463/493, 2018.08) 이상이 정수 오버플로우에서 비롯됨

SmartMesh 사례 (2018)

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

5499035 (1348012 Block Confirmations)

227 days 10 hrs ago (Apr-24-2018 07:16:19 PM +UTC)

0xd6a09bdb29e1eafa92a30373c44b09e2e2e0651e

Contract [0x55f93985431fc9304077687a35a1ba103dc1e081](#) (SmartMeshICO)

▶ From 0xdf31a499a5a8358... To 0xdf31a499a5a8358... for

▶ From 0xdf31a499a5a8358... To 0xd6a09bdb29e1ea... for

0 Ether (\$0.00)

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

SmartMesh 사례 (2018)

- 정수 오버플로우 (integer overflow) 취약점
- 방어적으로 코드를 작성했음에도 문제가 된 경우

```
1  function transferProxy (address from, address to, uint
2    value, uint fee) public returns (bool) {
3    if (balance[from] < fee + value)
4      revert();
5    if (balance[to] + value < balance[to] ||
6        balance[msg.sender] + fee < balance[msg.sender])
7      revert();
8    balance[to] += value;
9    balance[msg.sender] += fee;
10   balance[from] -= value + fee;
11   return true;
12 }
```

SmartMesh 사례 (2018)

- 정수 오버플로우 (integer overflow) 취약점
- 방어적으로 코드를 작성했음에도 문제가 된 경우

```
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 사례 (2018)

- 정수 오버플로우 (integer overflow) 취약점
- 방어적으로 코드를 작성했음에도 문제가 된 경우

```
1  function transferProxy (address from, address to, uint
2    value, uint fee) public returns
3    if (balance[from] < fee + value)
4      revert();
5    if (balance[to] + value < balance[to] ||
6        balance[msg.sender] + fee < balance[msg.sender])
7      revert();
8    balance[to] += value;
9    balance[msg.sender] += fee;
10   balance[from] -= value + fee;
11   return true;
12 }
```

보내는 사람의 잔고
가 충분한지 체크

송금

SmartMesh 사례 (2018)

- 정수 오버플로우 (integer overflow) 취약점
- 방어적으로 코드를 작성했음에도 문제가 된 경우

```
1  function transferProxy (address from, address to, uint
2    value, uint fee) public returns
3    if (balance[from] < fee + value)
4      revert();
5
6    if (balance[to] + value < balance[to] ||
7      balance[msg.sender] + fee < balance[msg.sender])
8      revert();
9
10   balance[to] += value;
11   balance[msg.sender] += fee;
12   balance[from] -= value + fee;
13
14   return true;
15 }
```

보내는 사람의 잔고
가 충분한지 체크

송금

오버플로우
체크

SmartMesh 사례 (2018)

- 정수 오버플로우 (integer overflow) 취약점
- 방어적으로 코드를 작성했음에도 문제가 된 경우

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

보내는 사람의 잔고
가 충분한지 체크

송금

오버플로우
체크

오버플로우/언더플로우
발생하지 않음

SmartMesh 사례 (2018)

```
1  function transferProxy (address from, address to, uint
2    value, uint fee) public returns (bool) {
3    if (balance[from] < fee + value)
4      revert();
5    if (balance[to] + value < balance[to] ||
6        balance[msg.sender] + fee < balance[msg.sender])
7      revert();
8    balance[to] += value;
9    balance[msg.sender] += fee;
10   balance[from] -= value + fee;
11   return true;
12 }
```

SmartMesh 사례 (2018)

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

```
1  function transferProxy (address from, address to, uint
2    value, uint fee) public returns (bool) {
3    if (balance[from] < fee + value)
4      revert();
5    if (balance[to] + value < balance[to] ||
6        balance[msg.sender] + fee < balance[msg.sender])
7      revert();
8    balance[to] += value;
9    balance[msg.sender] += fee;
10   balance[from] -= value + fee;
11   return true;
12 }
```

SmartMesh 사례 (2018)

```
1 function transferProxy (address from, address to, uint  
2     value, uint fee) public returns (bool) {  
3     if (balance[from] < fee + value)  
4         revert();  
5     if (balance[to] + value < balance[to] ||  
6         balance[msg.sender] + fee < balance[msg.sender])  
7         revert();  
8     balance[to] += value;  
9     balance[msg.sender] += fee;  
10    balance[from] -= value + fee;  
11    return true;  
12 }
```

SmartMesh 사례 (2018)

```
1 function transferProxy (address from, address to, uint  
2     value, uint fee) public returns (bool) {  
3     if (balance[from] < fee + value) 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 사례 (2018)

```
1  function transferProxy (address from, address to, uint  
2   value, uint fee) public returns (bool) {  
3    false → (balance[from] < fee + value) → 0!  
4    revert();  
5  
6    if (balance[to] + value < balance[to] ||  
7      balance[msg.sender] + fee < balance[msg.sender])  
8      revert();  
9  
10   balance[to] += value;  
11   balance[msg.sender] += fee;  
12   balance[from] -= value + fee;  
13  
14   return true;  
15 }
```

SmartMesh 사례 (2018)

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

```
1 function transferProxy (address from, address to, uint  
2     value, uint fee) public returns (bool) {  
3     if (balance[from] < fee + value) 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 사례 (2018)

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

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

SmartMesh 사례 (2018)

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

```
1 function transferProxy (address from, address to, uint  
2     value, uint fee) public returns (bool) {  
3     if (balance[from] < fee + value) revert();  
4     if (balance[to] + value < balance[to] ||  
5         balance[msg.sender] + fee < balance[msg.sender])  
6         revert();  
7     balance[to] += value; // 8fffff...ff  
8     balance[msg.sender] += fee; // 700...00  
9     balance[from] -= value + fee;  
10    return true;  
11 }
```

SmartMesh 사례 (2018)

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

```
1 function transferProxy (address from, address to, uint  
2   value, uint fee) public returns (bool) {  
3   if (balance[from] < fee + value) revert();  
4   if (balance[to] + value < balance[to] ||  
5       balance[msg.sender] + fee < balance[msg.sender])  
6     revert();  
7   balance[to] += value; // 8fffff...ff  
8   balance[msg.sender] += fee; // 700...00  
9   balance[from] -= value + fee; // 0!  
10  return true;  
11 }
```

스마트 컨트랙트 분석 기술의 한계

- 취약점 검출기 (e.g., Mythril, Osiris [ACSAC'18], Oyente [CCS'16])
 - 분석의 정확도를 위하여 안전성을 희생 (“bug-finders”)
 - 취약점이 발견되지 않더라도 안심할 수 없음
- 취약점 검증기 (e.g., Zeus [NDSS'18])
 - 모든 취약점을 탐지 가능 (“verifiers”)
 - 안전성을 위해서 정확도를 희생 (허위 경보 문제)

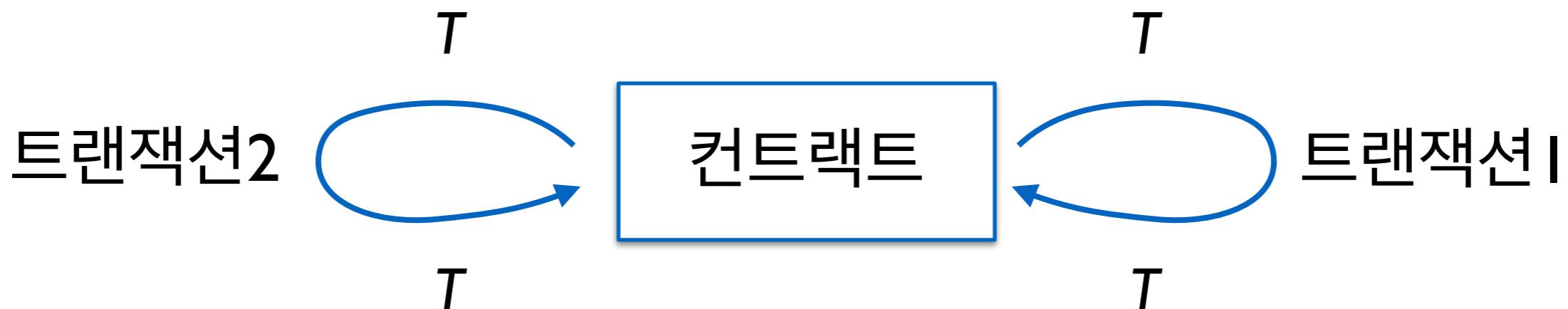
스마트 컨트랙트 분석 기술의 한계

- 취약점 검출기 (e.g., Mythril, Osiris [ACSAC'18], Oyente [CCS'16])
 - 분석의 정확도를 위하여 안전성을 희생 (“bug-finders”)
 - 취약점이 발견되지 않더라도 안심할 수 없음
- 취약점 검증기 (e.g., Zeus [NDSS'18])
 - 모든 취약점을 탐지 가능 (“verifiers”)
 - 안전성을 위해서 정확도를 희생 (허위 경보 문제)

VeriSmart: 안전하면서 정확한 스마트 컨트랙트 검증기

정확한 검증의 핵심

- 트랜잭션의 불변 성질 (Transaction invariant)을 유추하고 이를 검증에 활용하는 것이 필요
- 트랜잭션 불변 성질의 조건:
 - 생성자 실행후 성립
 - 각 트랜잭션의 실행전/후에 변함없이 성립



Netkoin 예제

```
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 }
```

Netkoin 예제

```
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 }
```

오버플로우로 착각하기 쉬움

Netkoin 예제

```
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    }                                오버플로우로 착각하기 쉬움
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    }                                언더플로우로 착각하기 쉬움
24 }
```

Netkoin 예제

```
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 }
```

트랜잭션 불변 성질:
 $\text{totalSupply} = \sum \text{balance}$

오버플로우로 착각하기 쉬움

언더플로우로 착각하기 쉬움

Netkoin 예제

```
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 }
```

트랜잭션 불변 성질:
 $\text{totalSupply} = \sum \text{balance}$

$\text{totalSupply} = \sum \text{balance}$

오버플로우로 착각하기 쉬움

언더플로우로 착각하기 쉬움

Netkoin 예제

```
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 }
```

트랜잭션 불변 성질:
 $\text{totalSupply} = \sum \text{balance}$

$\text{totalSupply} = \sum \text{balance}$

오버플로우로 착각하기 쉬움

언더플로우로 착각하기 쉬움

Netkoin 예제

```
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 }
```

트랜잭션 불변 성질:
 $\text{totalSupply} = \sum \text{balance}$

$\text{totalSupply} = \sum \text{balance}$

$\text{totalSupply} = \sum \text{balance}$

오버플로우로 착각하기 쉬움

언더플로우로 착각하기 쉬움

Netkoin 예제

```
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 }
```

트랜잭션 불변 성질:
 $\text{totalSupply} = \sum \text{balance}$

$\text{totalSupply} = \sum \text{balance}$

$\text{totalSupply} = \sum \text{balance}$

$\text{totalSupply} = \sum \text{balance}$

오버플로우로 착각하기 쉬움

$\text{totalSupply} = \sum \text{balance}$

언더플로우로 착각하기 쉬움

Netkoin 예제

```
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}
```

트랜잭션 불변 성질:
 $\text{totalSupply} = \sum \text{balance}$

$\text{totalSupply} = \sum \text{balance}$

$\text{totalSupply} = \sum \text{balance}$

$\text{totalSupply} = \sum \text{balance}$

오버플로우로 착각하기 쉬움

언더플로우로 착각하기 쉬움

Netkoin 예제

- 트랜잭션의 불변 성질을 이용한 안전성 증명

```
20     require (balance[msg.sender] >= value);  
21     balance[msg.sender] -= value;  
22     totalSupply -= value;
```

totalSupply >= value at line 22?

Supply = \sum balance ⋯ transaction invariant
>= balance[msg.sender] ⋯ def. of \sum balance
>= value ⋯ assumption

Netkoin 예제

- 트랜잭션의 불변 성질을 이용한 안전성 증명

```
20     require (balance[msg.sender] >= value);  
21     balance[msg.sender] -= value;  
22     totalSupply -= value;
```

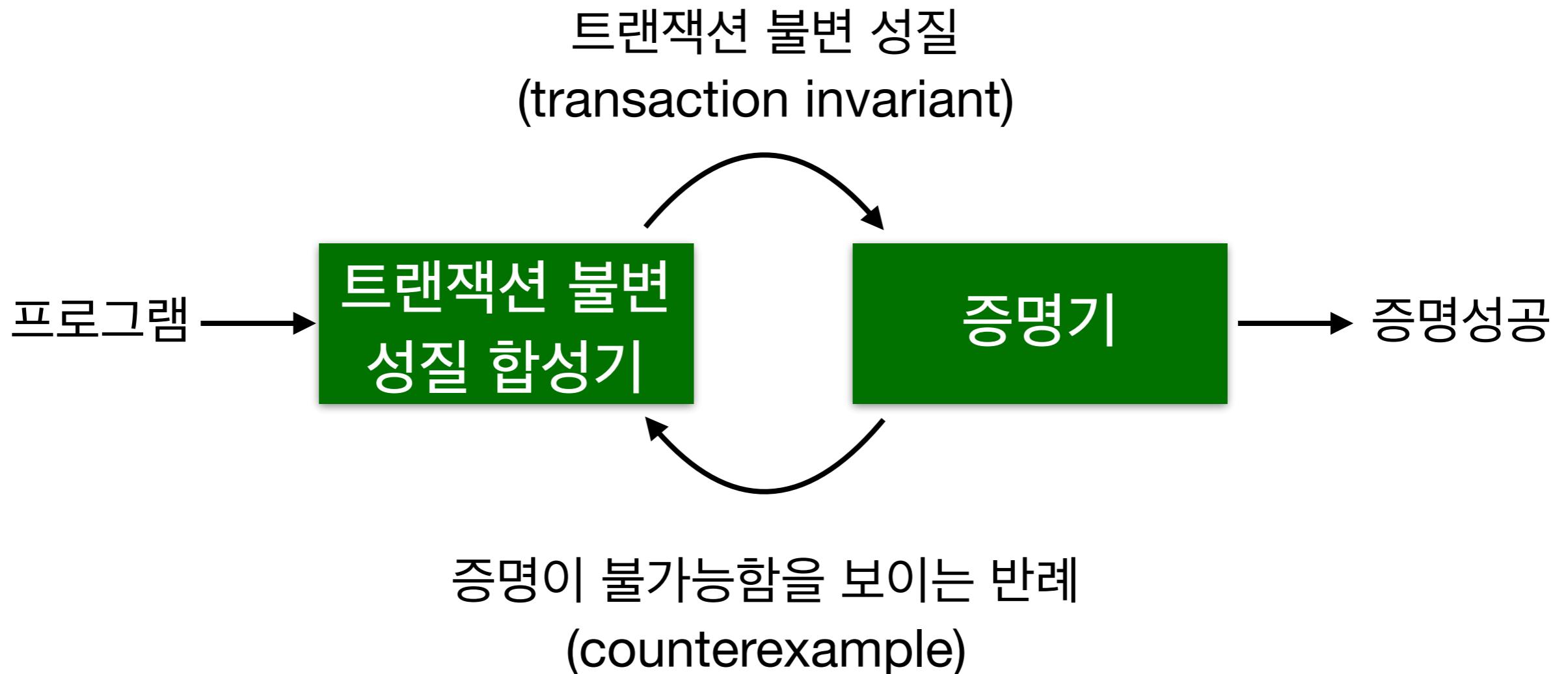
totalSupply >= value at line 22?

Supply = \sum balance ⋯ transaction invariant
>= balance[msg.sender] ⋯ def. of \sum balance
>= value ⋯ assumption

VeriSmart: 트랜잭션 불변성질을 자동 유추하여 정확하게 검증

VeriSmart 검증 알고리즘

- 프로그램 증명과 불변 성질 합성을 동시에 진행



VeriSmart 검증 성능

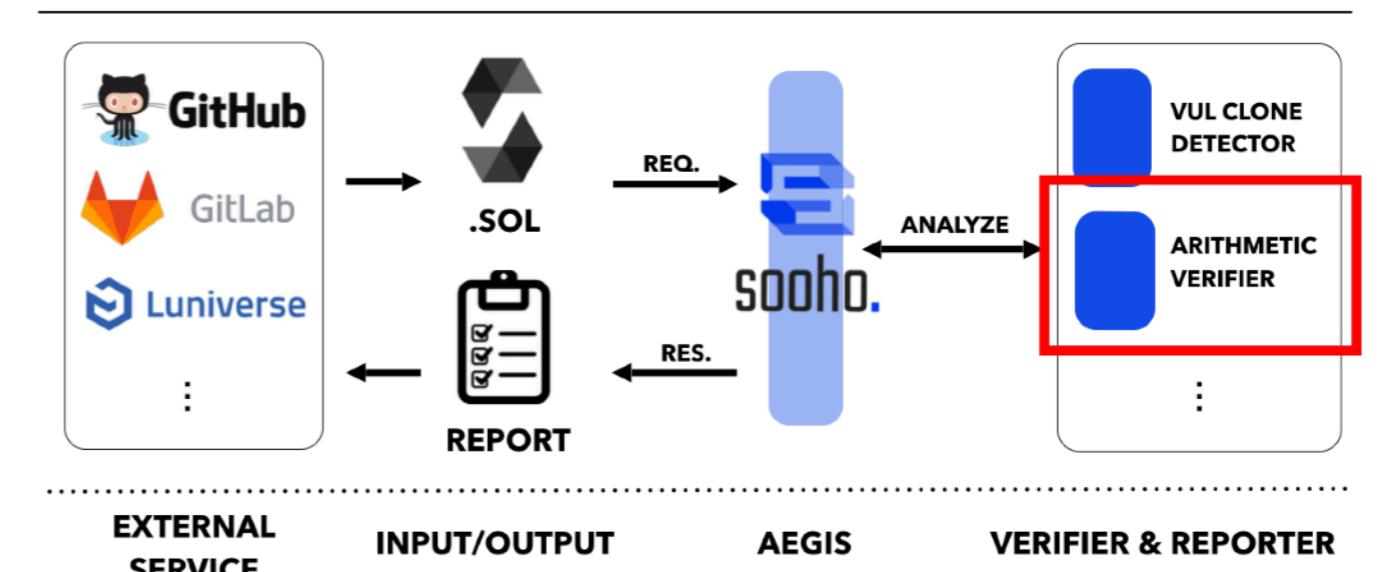
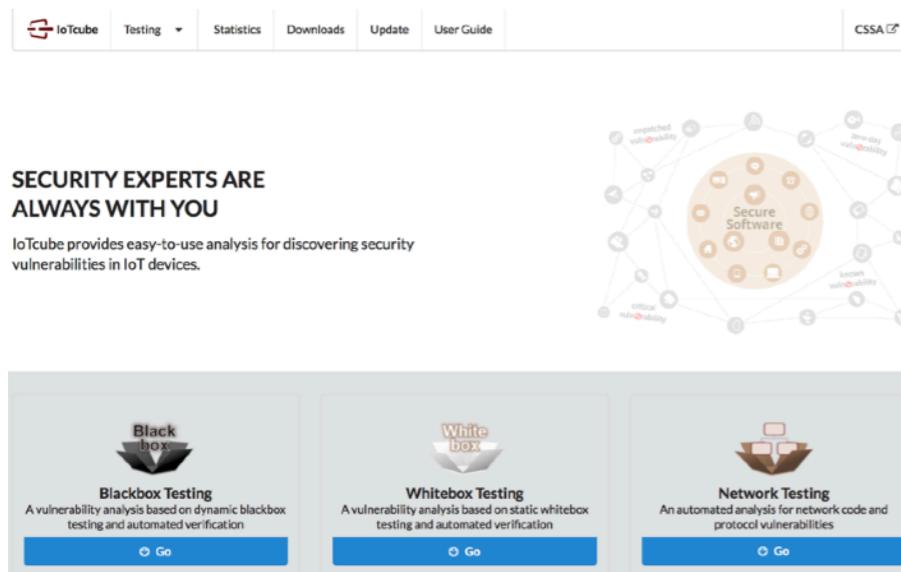
- ZEUS 가 검증에 실패했던 13개 프로그램에 대해 예비 실험

프로그램	증명 대상 개수 (#queries)	Zeus	증명 쿼리 갯수 (트랜잭션 불변식 O)
zeus1	3	2	3
zeus2	3	2	3
zeus3	7	5	7
zeus4	6	3	6
zeus5	7	5	7
zeus6	7	5	7
zeus7	7	5	7
zeus8	7	5	7
zeus9	7	5	7
zeus10	5	2	5
zeus11	7	5	7
zeus12	3	2	3
zeus13	3	2	3
전체	72	48	72

Zeus가 증명에 실패한 13개 프로그램에 대해 모두 증명 성공

마무리

- VeriSmart: 스마트 컨트랙트 안전성 증명기
- IoTCube의 취약점 검증 엔진에 탑재 예정
- 스마트 컨트랙트 취약점 자동 분석 플랫폼 상용화 예정



<http://iotcube.net>

<http://sooho.io>