CS 190I Program Synthesis for the Masses

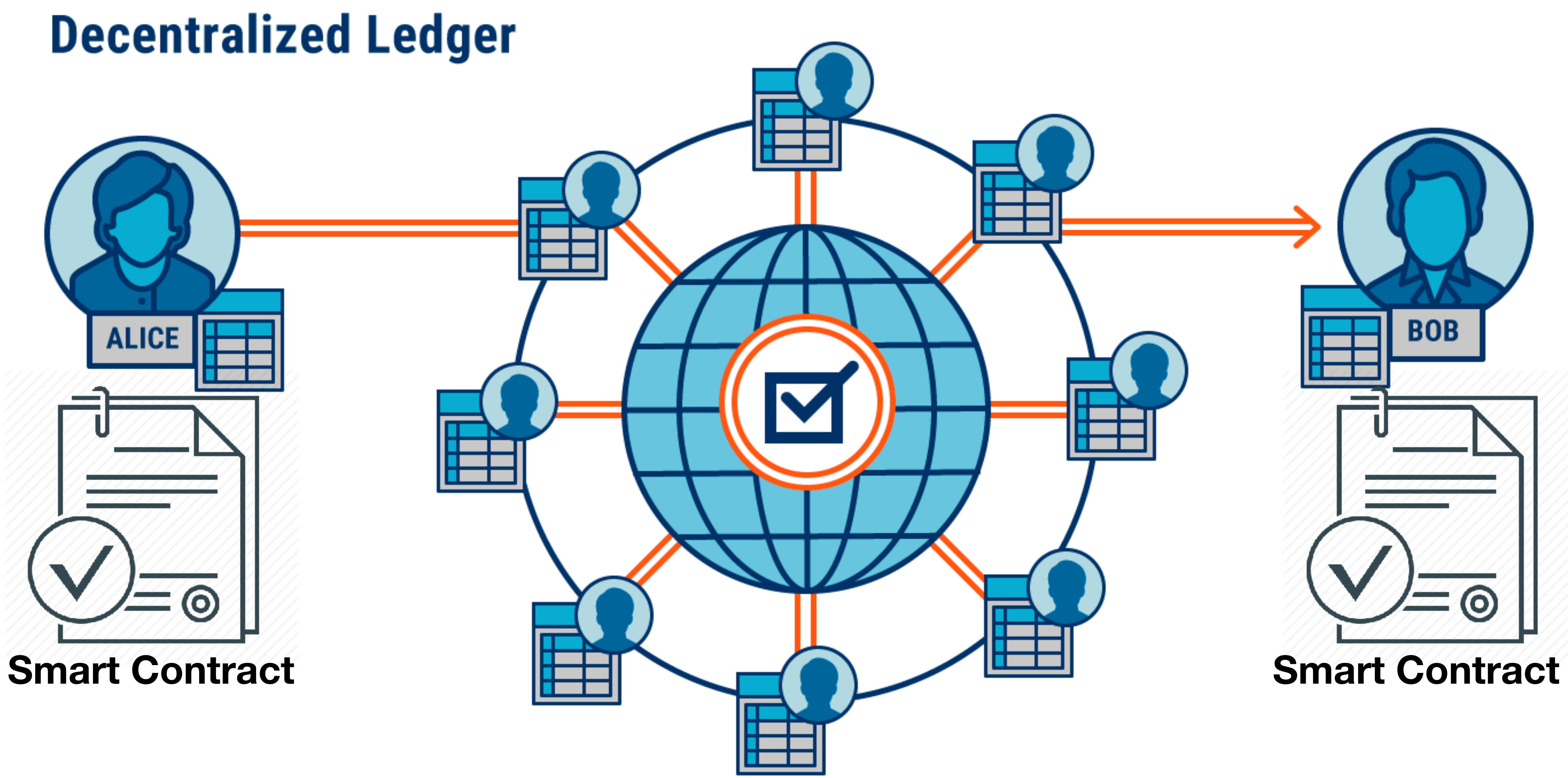
Lecture 13: Case II: Attack Synthesis for Smart Contracts

Yu Feng
Spring 2021

Summary of previous lecture

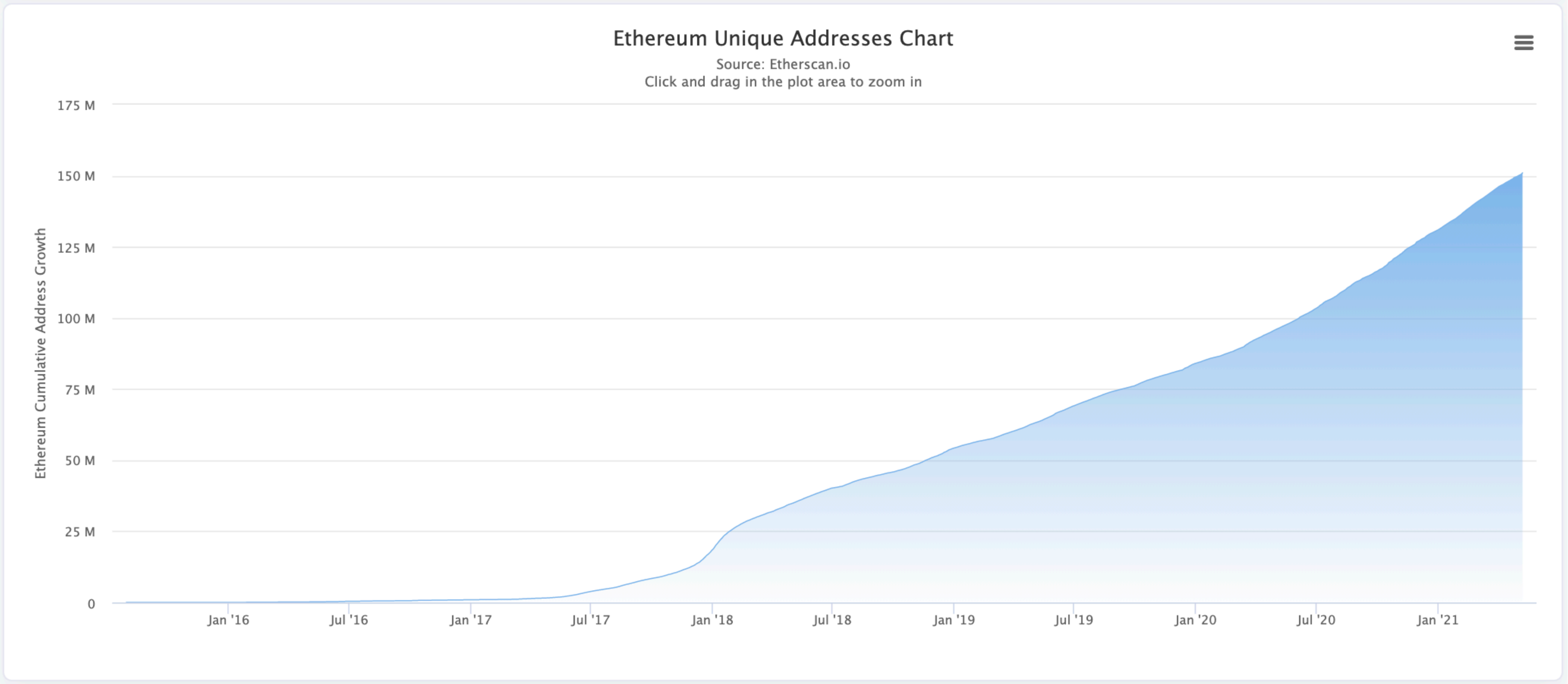
- R4 is out today
- 1st case study: Visualization Synthesis
- Today's topic: 2nd case study: Attack Synthesis

What is smart contract



https://www.cbinsights.com/research/what-is-blockchain-technology/

Motivation



https://etherscan.io/chart/address

Motivation

Smart Contract Bug Nearly Freezes Transfers in $800 Million Worth of Icon Tokens

ETHEREUM

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

Sam Town · Apr 25, 2018 · 3 min read

The DAO Attacked: Code Issue Leads to $60 Million Ether Theft

Smart contracts are immutable!

Problem: attack synthesis

```
contract PausableToken {
bool flag = false;

function makeFlag(bool fg) {
 flag = fg;
}

function batchTransfer(address[] _receivers,
    uint256 _value) {
    uint cnt = _receivers.length;
    uint256 amount = uint256(cnt) * _value;
    require(flag);
    require(balances[msg.sender] >= amount);

    balances[msg.sender] =
      balances[msg.sender].sub(amount);
    for (uint i = 0; i < cnt; i++) {
      address recv = _receivers[i];
      balances[recv] =
        balances[recv].add(_value);
      Transfer(msg.sender, recv, _value);
    }
    return true;
  }
}
```

Victim

Given a vulnerable program, synthesizing an attack program to exploit the vulnerability

An attacker could access:

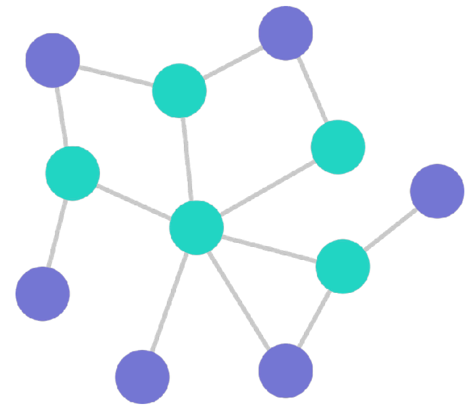
1) Bytecode
2) Public API
3) Vulnerability patterns

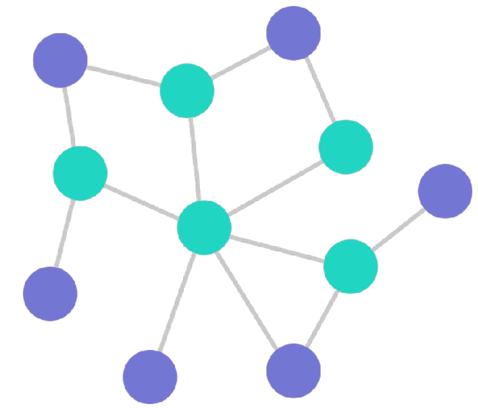
```
contract Attacker {
  ...
  function exploit() {
    VulContract v;
    v.makeFlag(true);
    v.batchTransfer([0x123, 0x456], 2^256 - 1);
  }
}
```

Attacker

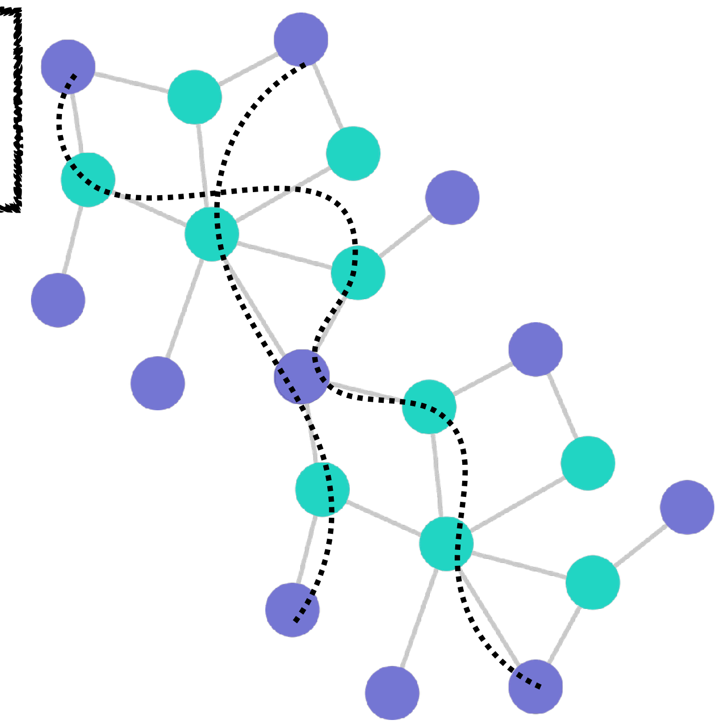
A vanilla solution

```
contract Ct {
  fun bar(int x) {…}
  fun goo(bool y) {…}
  fun foo(byte z) {…}
  …
}
```





For each candidate, perform symbolic execution to trigger the vulnerability

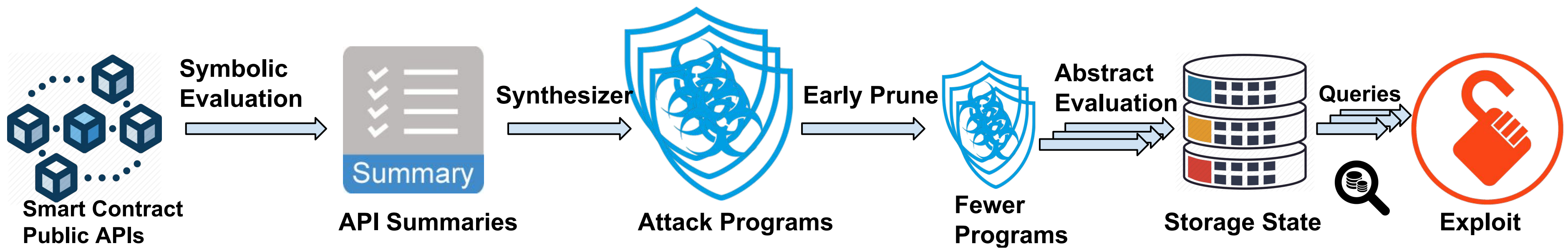


```
bar(…);
goo(…);
foo(…);
bar(…); goo(…);
…
bar(…); goo(…); foo(…);
```

Candidates

path explosion!

Our solution



Solver-aid Summary-based Symbolic

Parallel Synthesis

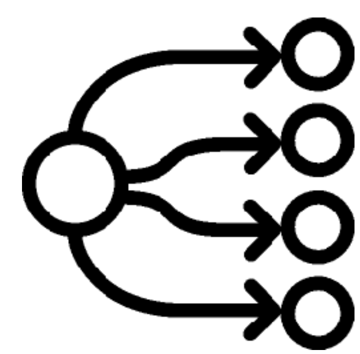
interpret$(P, x) \rightarrow$ Rosette $\rightarrow \phi(x)$

$$\phi_1 \mapsto \phi_1'$$

choose*$(m_1, \ldots, m_n)$; **choose***$(m_1, \ldots, m_n)$;

$$(\phi_1 \wedge \phi_2) \ldots \vee \ldots (\phi_2 \wedge \phi_n)$$

$$\phi_2 \mapsto \phi_2'$$



interpret$(m_2; m_n)$

interpret$(m_1; m_n)$

interpret$(m_1; m_2)$

Vulnerabilities as queries

```
contract PausableToken {
bool flag = false;

function makeFlag(bool fg) {
 flag = fg;
}

function batchTransfer(address[] _receivers,
    uint256 _value) {
    uint cnt = _receivers.length;
    uint256 amount = uint256(cnt) * _value;
    require(flag);
    require(balances[msg.sender] >= amount);

    balances[msg.sender] =
      balances[msg.sender].sub(amount);
    for (uint i = 0; i < cnt; i++) {
      address recv = _receivers[i];
      balances[recv] =
        balances[recv].add(_value);
      Transfer(msg.sender, recv, _value);
    }
    return true;
  }
}
```

First-order formulas over program states

$\exists\ arg_0, arg_1, r_1, r_2, r_3, call$
(&& (= r_3 ($\otimes$ r_1 r_2))
    (> $[\![r_2]\!]$ $[\![r_3]\!]$)
    (interfere? r_2 call.value)
    (interfere? arg_0 call.addr)
    (interfere? arg_1 call.value))
where $\otimes \in \{+, \times\}$

(a) Query for the BatchOverFlow

Symbolic evaluation

```
(define (smartscopy 𝒱 Υ K)
 (define (stmt) (apply choose* Υ))
 ;;Generate a symbolic attack program of size K.
 (define program (map (λ (x) (stmt)) (range K)))
 (define (progstate)
  ;;Program state has registers, memory, storage,
  ;;gas, and other global information.
  (progstate (for/vector ([i config]) 'reg)
             (init-memory)
             (init-storage)
             'gas  ;;gas consumption
             ...))
 (define i-pstate (send machine get-state ...))
 (define o-pstate (interpret program i-state))
 (define binding (solve (assert (𝒱 o-pstate))))
 (evaluate program binding))
```

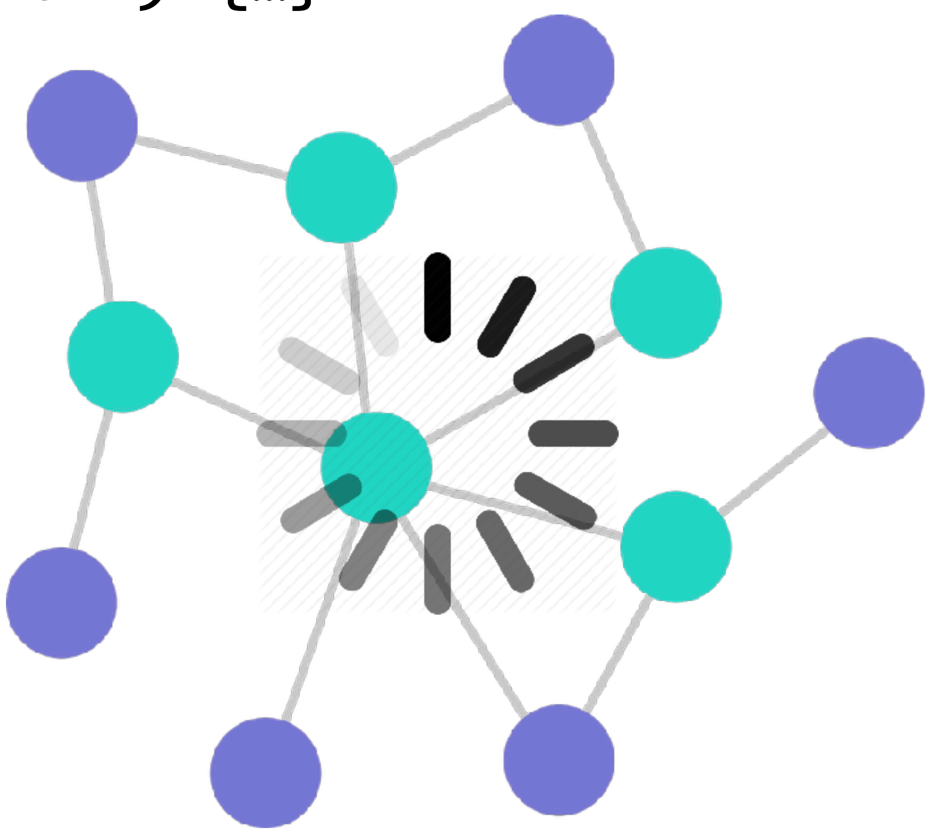
Fig. 10. SmartScopy implementation in Rosette.

Torlak, Emina, and Rastislav Bodik. "A lightweight symbolic virtual machine for solver-aided host languages." PLDI'14.

Redundant computation

The same method is evaluated over and over again!

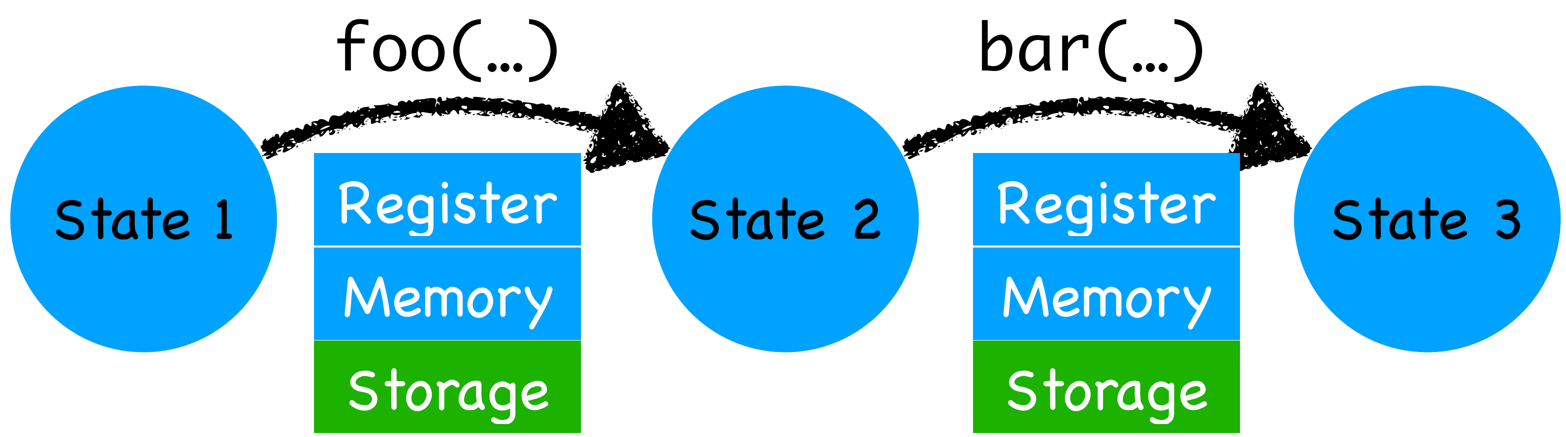
```
foo(int x) {…}
```



Control flow graph

```
foo(…);
foo(…); goo(…);
goo(…); foo(…);
…
bar(…); goo(…); foo(…);
```

Summary generation



Only updates on storage are preserved across different states!

Generate summaries that soundly
model side effects on the storage

Summary generation

```
(define (get-summary s φ)
  (match s
   [call(x, y, z) call(⟦x⟧, ⟦y⟧, ⟦z⟧)@φ]
   [sstore(x, y)  sstore(x, ⟦y⟧)@φ]
   [_             ⟦_⟧)]))
```

(a) Procedure for Summary Generation

```
(define (interpret-summary s@φ Γ)
  (define s_Γ@φ_Γ (substitute s@φ Γ))
  (match s_Γ
   [call(x_Γ, y_Γ, z_Γ) (when φ_Γ call(x_Γ, y_Γ, z_Γ))]
   [sstore(x_Γ, y_Γ) (when φ_Γ sstore(x_Γ, y_Γ))]
   [_  no-op]))
```

(b) Procedure for Summary Interpretation

Summary generation

```
contract EubChainIco is PausableToken {
...
function vestedTransfer(address _to,
                        uint256 _amount){
  ...
  require(_amount > 0);
  vesting.amount = _amount.sub(1);
  transfer(msg.sender,_to,vesting.amount);

  uint256 v1 = _amount - 15;
  uint256 wei = v1;
  uint t1 = vesting.startTime;

  emit VestTransfer(msg.sender,
                    _to, wei, t1, _);
  ...
}
```

```
assert(_amount > 0);
r1 := _amount - 1;
sstore(vesting.amount, _amount - 1);
call(msg.sender, _to, _amount - 1);

r2 := amount - 15;
r3 := amount - 15;
r4 := sload(vesting.startTime);
no-op;
```

Symbolic evaluation

```
sstore(vesting.amount, _amount - 1)
    [_amount>0];
call(msg.sender, _to, _amount - 1)
    [_amount>0];
```

Summary extraction

Parallel Synthesis

For complex contracts, the constraints generated by the attack programs are still hard to solve

$$\textbf{choose*}(m_1, \ldots, m_n); \textbf{choose*}(m_1, \ldots, m_n);$$

$$(\phi_1 \wedge \phi_2 \wedge \ldots) \ldots \vee \ldots (\phi_2 \wedge \phi_n \wedge \ldots)$$

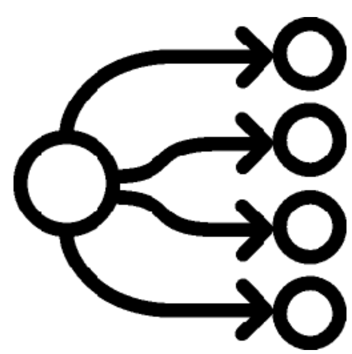
The gigantic formula is pushed as a path condition from top to all statements!

Parallel Synthesis

Early concretization via parallel symbolic evaluation

$$\textbf{choose*}(m_1, \ldots, m_n); \textbf{choose*}(m_1, \ldots, m_n);$$

$m_2; m_n$ $m_1; m_2$ $m_1; m_n$ ...



interpret$(m_2; m_n)$

interpret$(m_1; m_n)$

interpret$(m_1; m_2)$

...

Evaluation

- How does Solar perform compared to the state-of-the-arts
- How effective is our summary-based symbolic evaluation

Compare against teEther

Data set: 25K smart contracts from etherscan

Vulnerability	#TP	Solar (198)		teEther (179)	
		#FP	#FN	#FP	#FN
Attack Control	181	17	0	19	21

	Solar	teEther
Running time	8s	31s

Krupp et. al. Usenix Security'18

Compare against ContractFuzzer

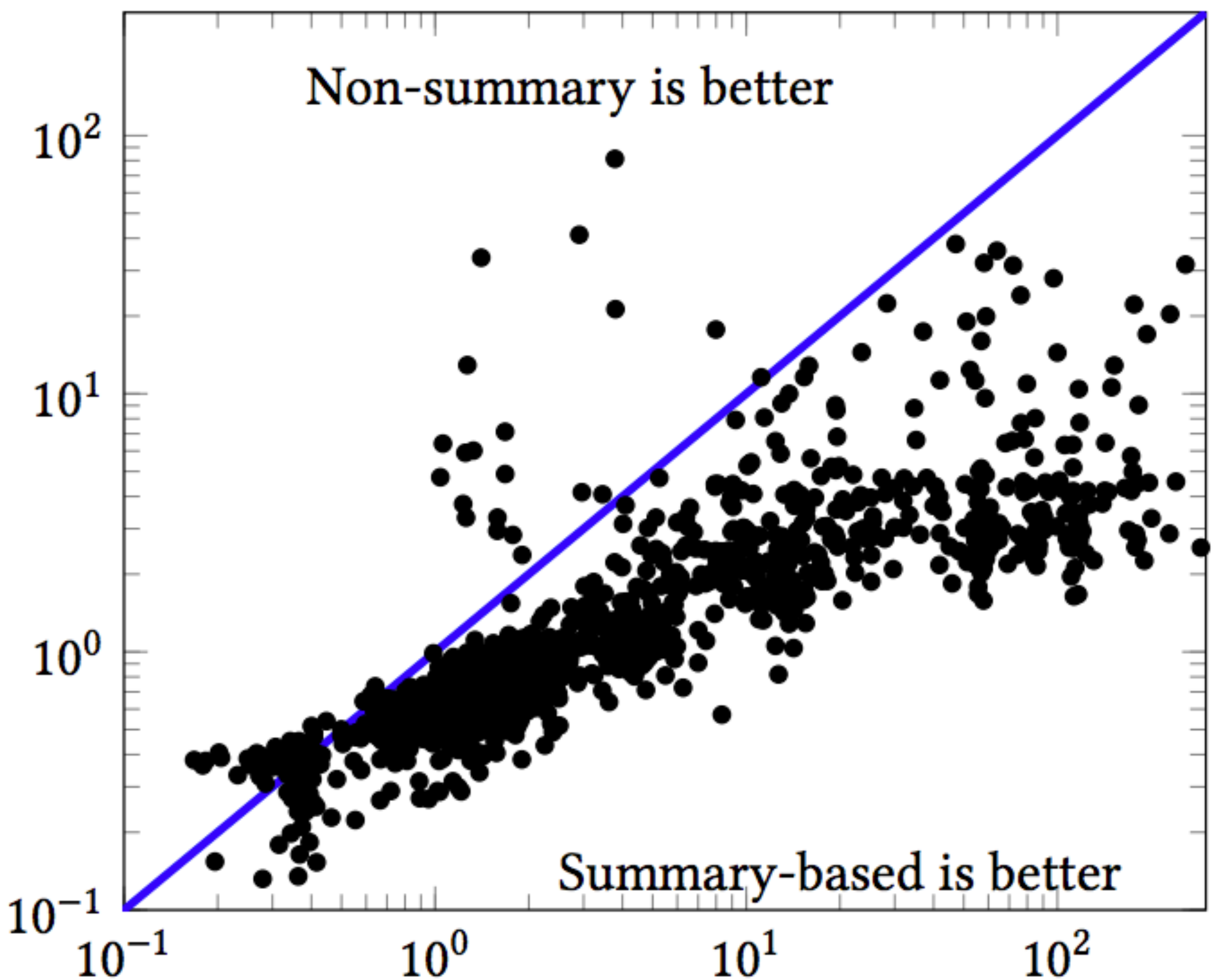
Data set: 100 smart contracts from ContractFuzzer

Vulnerability	Solar			ContractFuzzer		
	No.	FP	FN	No.	FP	FN
Timestamp	16	0	1	13	4	7
Gasless send	17	0	0	14	3	6
Bad random	9	0	0	5	1	5

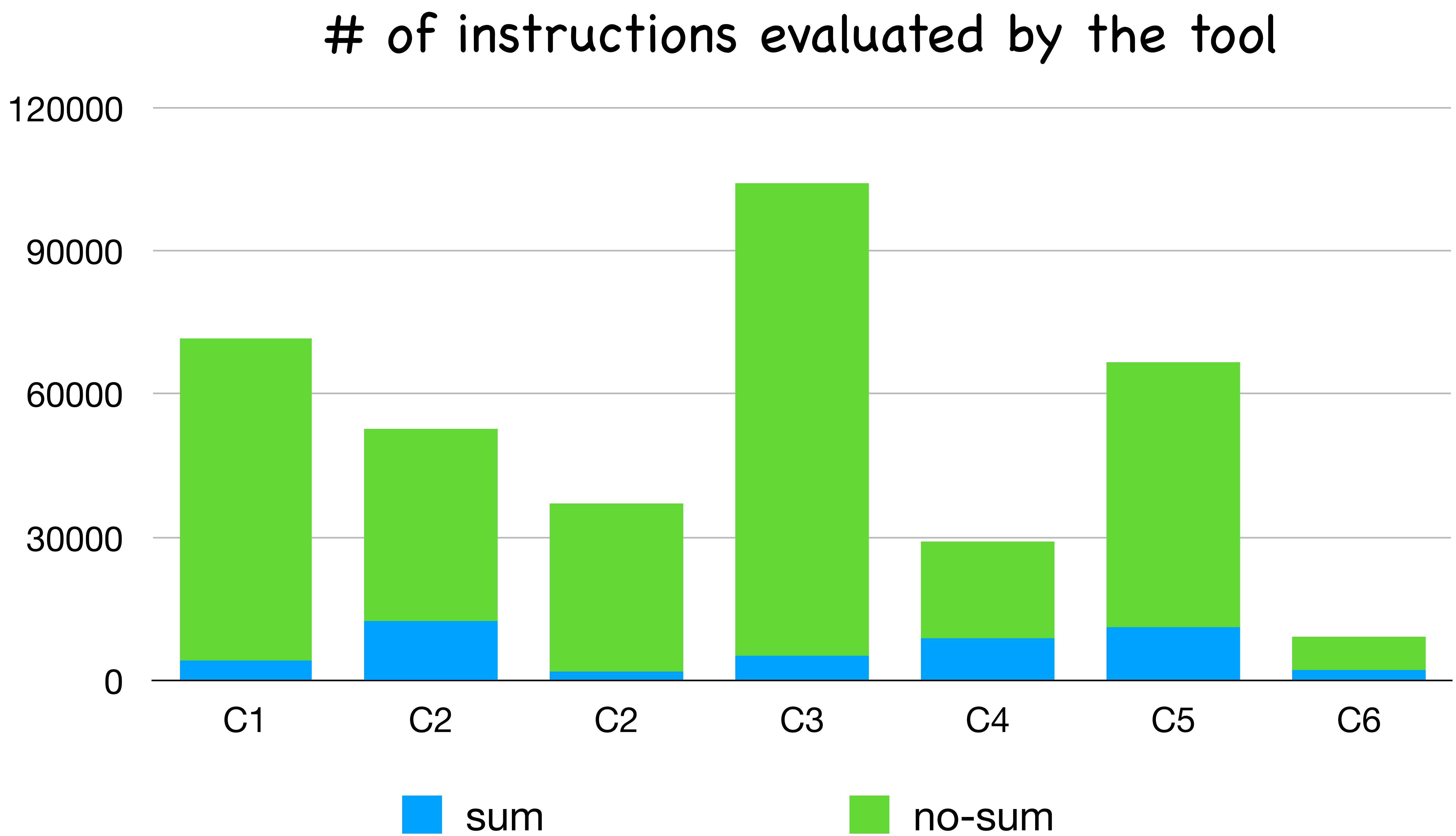
	Solar	ContractFuzzer
Running time	11s	>10min

Jiang et. al. Ase'18

Impact of summary analysis



Impact of summary analysis



TODOs by next lecture

- Start to work on your final report/project! (40%)