SAILFISH: Vetting Smart Contract State-Inconsistency Bugs in Seconds

IEEE Symposium on Security & Privacy, May 2022.

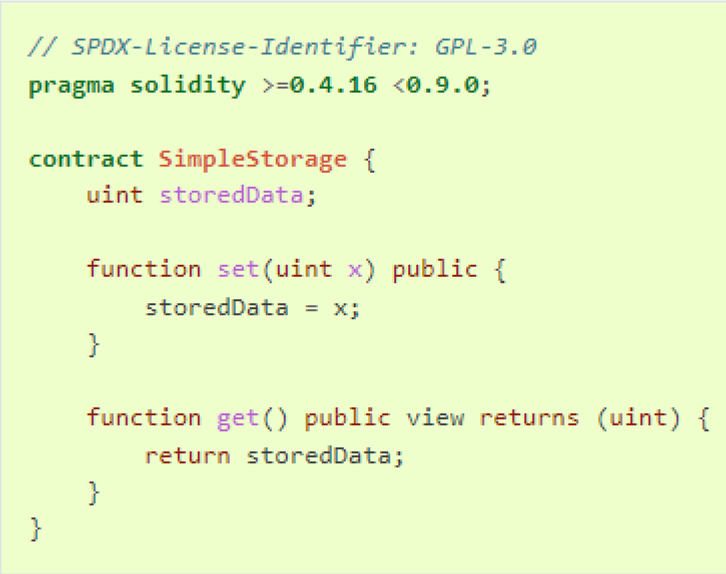
Priyanka Bose, Dipanjan Das, Yanju Chen, Y u Feng, Christopher Kruegel, and Giovanni Vigna

University of California, Santa Barbara

Background

- **Ethereum**: Ethereum is the community-run technology powering the cryptocurrency ether (ETH) and thousands of decentralized applications

- **Smart contract** : programs running on top of the Ethereum blockchain
  - **event**: a public/external method from outside the contract
  - **schedule**: a valid sequence of events
  - **contract state**: Δ = (V, B) ,V = {V1, V2, V3, …, Vn } is
    the set of all the storage variables of a contract,
    and B is its balance

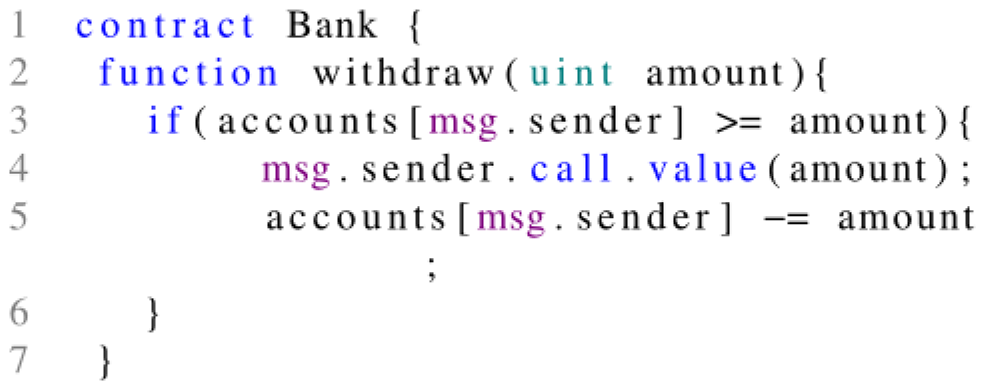


```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.4.16 <0.9.0;

contract SimpleStorage {
    uint storedData;

    function set(uint x) public {
        storedData = x;
    }

    function get() public view returns (uint) {
        return storedData;
    }
}
```

Background

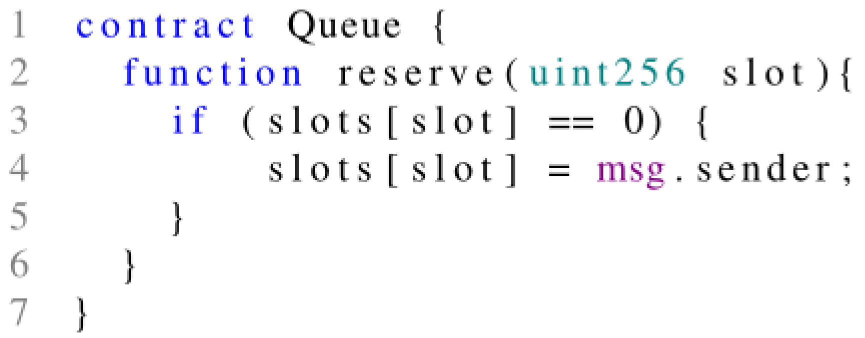
- **State inconsistency(SI)** :If those two schedules individually operate on the same initial state Δ, but yield different final states

Reentracny



```
contract Bank {
 function withdraw(uint amount){
   if(accounts[msg.sender] >= amount){
        msg.sender.call.value(amount);
        accounts[msg.sender] -= amount
              ;
   }
 }
```

Transaction Order Dependence

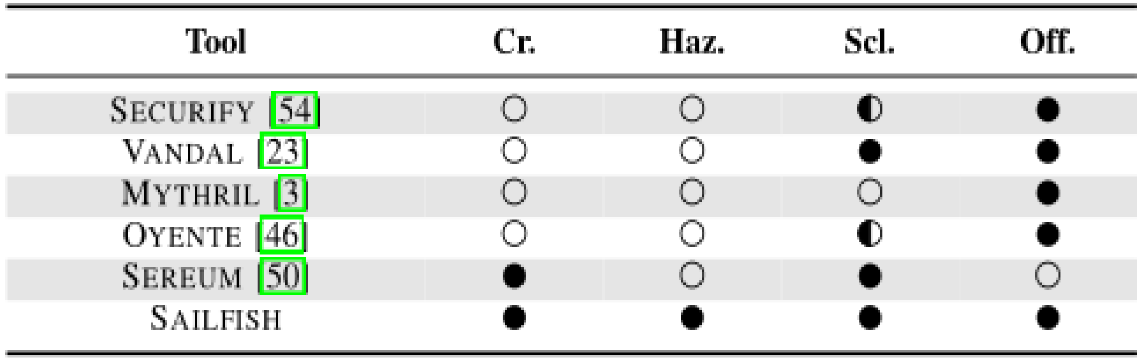


```
contract Queue {
  function reserve(uint256 slot){
    if (slots[slot] == 0) {
        slots[slot] = msg.sender;
    }
  }
}
```

Problem & Addressing problem

- **cross-function**
- **over-approximate**
- **scalability**
- **offline bug detection**

- Hazardous access
  (a) operate on the same state variable
  (b) are reachable from public methods
  (c) at-least one is a write

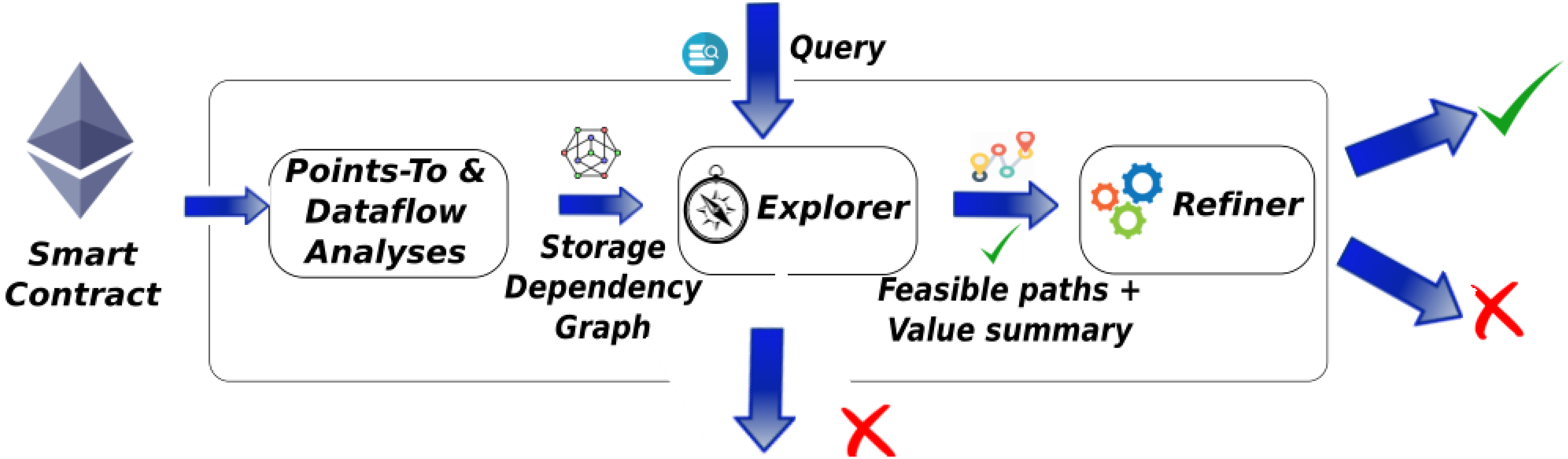


Tool	Cr.	Haz.	Scl.	Off.
SECURIFY [54]	○	○	◐	●
VANDAL [23]	○	○	●	●
MYTHRIL [3]	○	○	○	●
OYENTE [46]	○	○	◐	●
SEREUM [50]	●	○	●	○
SAILFISH	●	●	●	●

● Full ◐ Partial ○ No support. **Cr.**: Cross-function, **Haz.**: Hazardous access, **Scl.**: Scalability, **Off.**: Offline detection

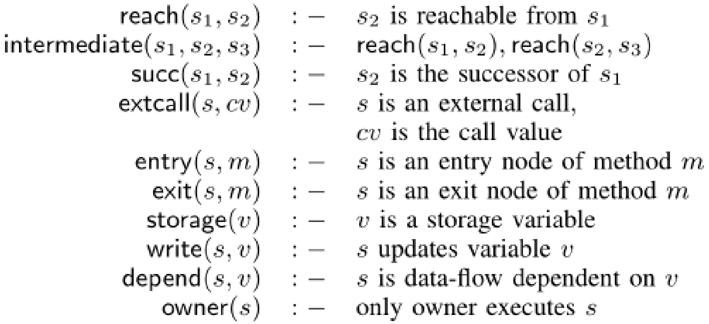
Overview

- **SAILFISH**, a **scalable** system for **automatically finding state-inconsistency bugs** in smart contracts

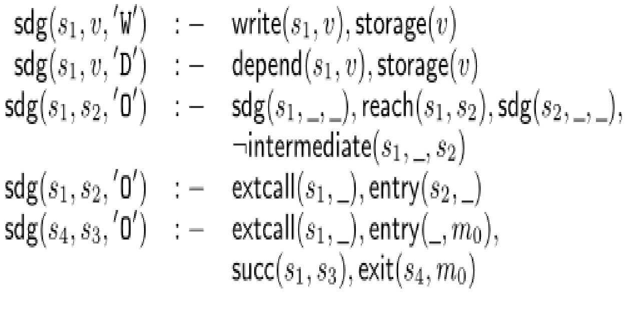


EXPLORER: LIGHTWEIGHT EXPLORATION OVER SDG

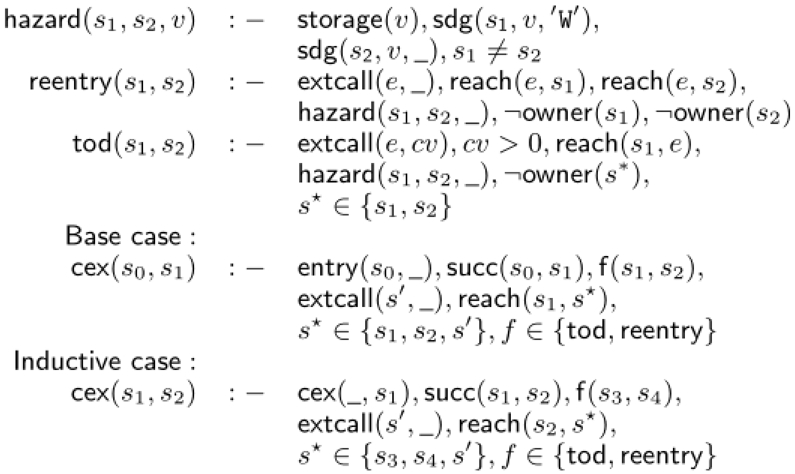
- Builds a storage dependency graph (SDG)
- over-approximates the read-write accesses
- graph queries over the SDG



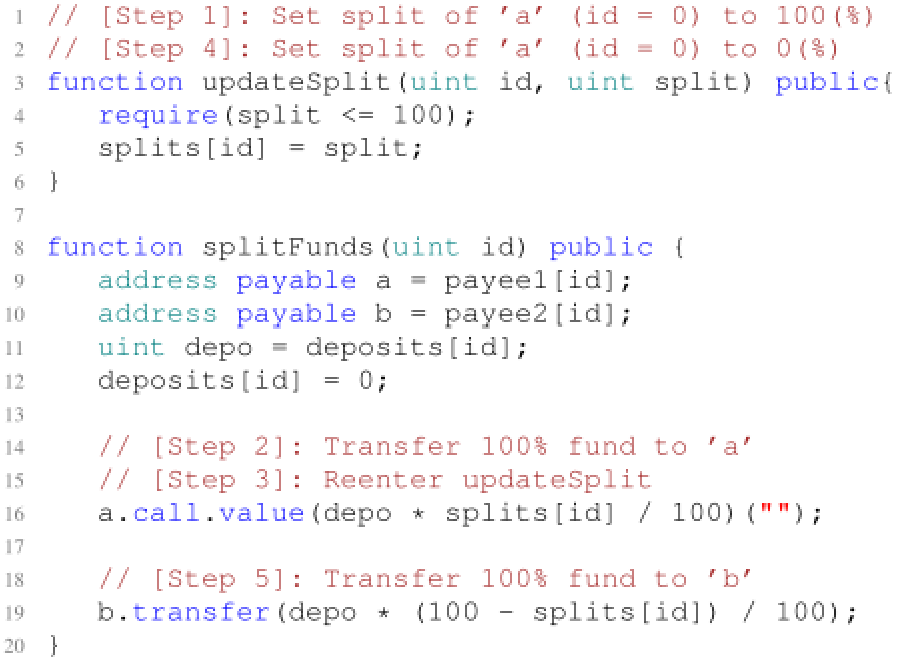
$$
\begin{array}{rcl}
\mathsf{reach}(s_1, s_2) & :- & s_2 \text{ is reachable from } s_1 \\
\mathsf{intermediate}(s_1, s_2, s_3) & :- & \mathsf{reach}(s_1, s_2), \mathsf{reach}(s_2, s_3) \\
\mathsf{succ}(s_1, s_2) & :- & s_2 \text{ is the successor of } s_1 \\
\mathsf{extcall}(s, cv) & :- & s \text{ is an external call,} \\
 & & cv \text{ is the call value} \\
\mathsf{entry}(s, m) & :- & s \text{ is an entry node of method } m \\
\mathsf{exit}(s, m) & :- & s \text{ is an exit node of method } m \\
\mathsf{storage}(v) & :- & v \text{ is a storage variable} \\
\mathsf{write}(s, v) & :- & s \text{ updates variable } v \\
\mathsf{depend}(s, v) & :- & s \text{ is data-flow dependent on } v \\
\mathsf{owner}(s) & :- & \text{only owner executes } s
\end{array}
$$



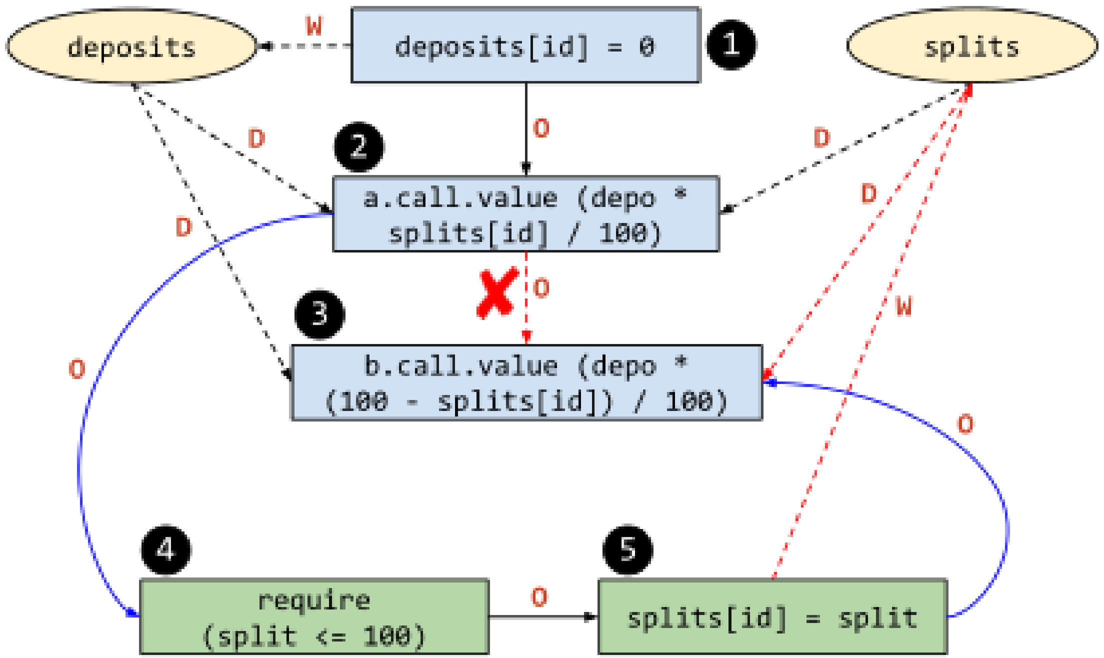
$$
\begin{array}{rcl}
\mathsf{sdg}(s_1, v, \texttt{'W'}) & :- & \mathsf{write}(s_1, v), \mathsf{storage}(v) \\
\mathsf{sdg}(s_1, v, \texttt{'D'}) & :- & \mathsf{depend}(s_1, v), \mathsf{storage}(v) \\
\mathsf{sdg}(s_1, s_2, \texttt{'O'}) & :- & \mathsf{sdg}(s_1, \_, \_), \mathsf{reach}(s_1, s_2), \mathsf{sdg}(s_2, \_, \_), \\
 & & \neg\mathsf{intermediate}(s_1, \_, s_2) \\
\mathsf{sdg}(s_1, s_2, \texttt{'O'}) & :- & \mathsf{extcall}(s_1, \_), \mathsf{entry}(s_2, \_) \\
\mathsf{sdg}(s_4, s_3, \texttt{'O'}) & :- & \mathsf{extcall}(s_1, \_), \mathsf{entry}(\_, m_0), \\
 & & \mathsf{succ}(s_1, s_3), \mathsf{exit}(s_4, m_0)
\end{array}
$$



$$
\begin{array}{rcl}
\mathsf{hazard}(s_1, s_2, v) & :- & \mathsf{storage}(v), \mathsf{sdg}(s_1, v, \texttt{'W'}), \\
 & & \mathsf{sdg}(s_2, v, \_), s_1 \neq s_2 \\
\mathsf{reentry}(s_1, s_2) & :- & \mathsf{extcall}(e, \_), \mathsf{reach}(e, s_1), \mathsf{reach}(e, s_2), \\
 & & \mathsf{hazard}(s_1, s_2, \_), \neg\mathsf{owner}(s_1), \neg\mathsf{owner}(s_2) \\
\mathsf{tod}(s_1, s_2) & :- & \mathsf{extcall}(e, cv), cv > 0, \mathsf{reach}(s_1, e), \\
 & & \mathsf{hazard}(s_1, s_2, \_), \neg\mathsf{owner}(s^*), \\
 & & s^\star \in \{s_1, s_2\} \\
\text{Base case :} & & \\
\mathsf{cex}(s_0, s_1) & :- & \mathsf{entry}(s_0, \_), \mathsf{succ}(s_0, s_1), \mathsf{f}(s_1, s_2), \\
 & & \mathsf{extcall}(s', \_), \mathsf{reach}(s_1, s^\star), \\
 & & s^\star \in \{s_1, s_2, s'\}, f \in \{\mathsf{tod}, \mathsf{reentry}\} \\
\text{Inductive case :} & & \\
\mathsf{cex}(s_1, s_2) & :- & \mathsf{cex}(\_, s_1), \mathsf{succ}(s_1, s_2), \mathsf{f}(s_3, s_4), \\
 & & \mathsf{extcall}(s', \_), \mathsf{reach}(s_2, s^\star), \\
 & & s^\star \in \{s_3, s_4, s'\}, f \in \{\mathsf{tod}, \mathsf{reentry}\}
\end{array}
$$



```
// [Step 1]: Set split of ’a’ (id = 0) to 100(%)
// [Step 4]: Set split of ’a’ (id = 0) to 0(%)
function updateSplit(uint id, uint split) public{
    require(split <= 100);
    splits[id] = split;
}

function splitFunds(uint id) public {
    address payable a = payee1[id];
    address payable b = payee2[id];
    uint depo = deposits[id];
    deposits[id] = 0;

    // [Step 2]: Transfer 100% fund to ’a’
    // [Step 3]: Reenter updateSplit
    a.call.value(depo * splits[id] / 100)("");

    // [Step 5]: Transfer 100% fund to ’b’
    b.transfer(depo * (100 - splits[id]) / 100);
}
```



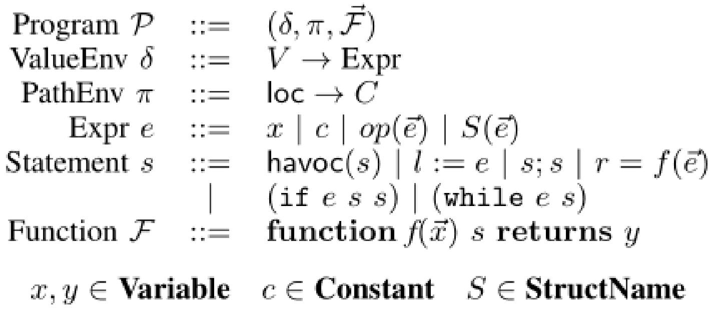
- hazardous access： pairs <3,5>
- counter-example： root→ 2 → 4 → 5→ 3

REFINER:SYMBOLIC EVALUATION WITH VALUE SUMMARY

- **value-summary analysis**
- **symbolic verifier**

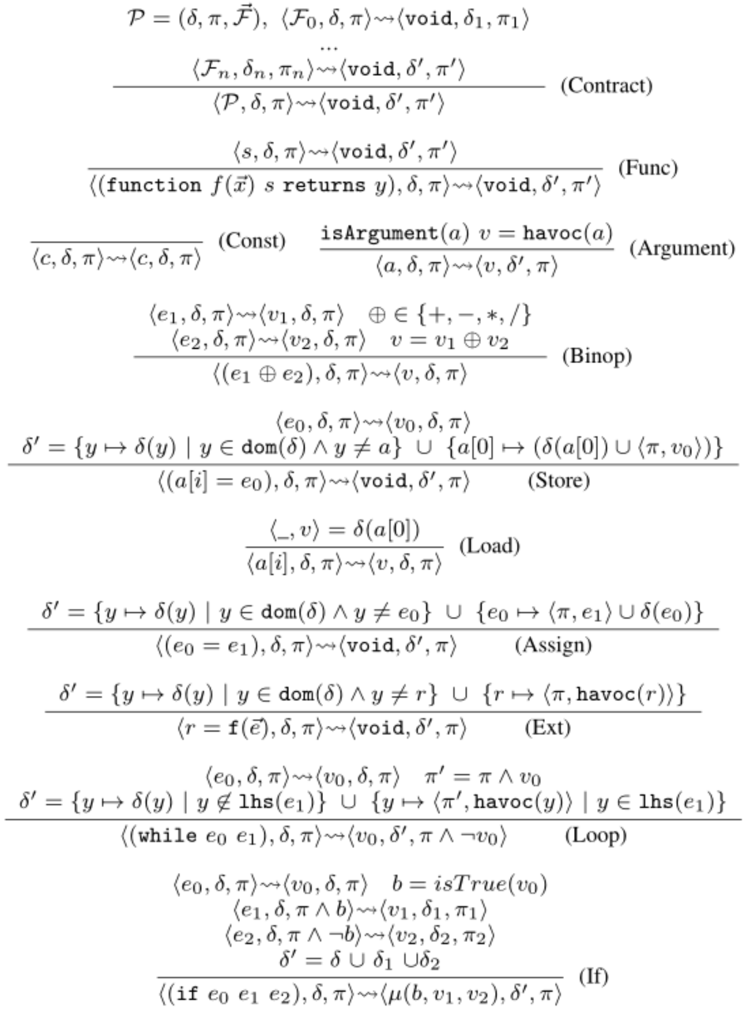
VSA

- Precision
- Scalability
- Application



$$
\begin{array}{rcl}
\text{Program } \mathcal{P} & ::= & (\delta, \pi, \vec{\mathcal{F}}) \\
\text{ValueEnv } \delta & ::= & V \rightarrow \text{Expr} \\
\text{PathEnv } \pi & ::= & \mathsf{loc} \rightarrow C \\
\text{Expr } e & ::= & x \mid c \mid op(\vec{e}) \mid S(\vec{e}) \\
\text{Statement } s & ::= & \mathsf{havoc}(s) \mid l := e \mid s; s \mid r = f(\vec{e}) \\
 & \mid & (\texttt{if } e\ s\ s) \mid (\texttt{while } e\ s) \\
\text{Function } \mathcal{F} & ::= & \textbf{function } f(\vec{x})\ s \textbf{ returns } y
\end{array}
$$

$x, y \in$ **Variable** $\quad c \in$ **Constant** $\quad S \in$ **StructName**



$$\frac{\mathcal{P} = (\delta, \pi, \vec{\mathcal{F}}),\ \langle \mathcal{F}_0, \delta, \pi \rangle \rightsquigarrow \langle \texttt{void}, \delta_1, \pi_1 \rangle \quad \ldots \quad \langle \mathcal{F}_n, \delta_n, \pi_n \rangle \rightsquigarrow \langle \texttt{void}, \delta', \pi' \rangle}{\langle \mathcal{P}, \delta, \pi \rangle \rightsquigarrow \langle \texttt{void}, \delta', \pi' \rangle}\ \text{(Contract)}$$

$$\frac{\langle s, \delta, \pi \rangle \rightsquigarrow \langle \texttt{void}, \delta', \pi' \rangle}{\langle (\texttt{function } f(\vec{x})\ s \texttt{ returns } y), \delta, \pi \rangle \rightsquigarrow \langle \texttt{void}, \delta', \pi' \rangle}\ \text{(Func)}$$

$$\frac{}{\langle c, \delta, \pi \rangle \rightsquigarrow \langle c, \delta, \pi \rangle}\ \text{(Const)} \qquad \frac{\texttt{isArgument}(a)\ v = \texttt{havoc}(a)}{\langle a, \delta, \pi \rangle \rightsquigarrow \langle v, \delta', \pi \rangle}\ \text{(Argument)}$$

$$\frac{\langle e_1, \delta, \pi \rangle \rightsquigarrow \langle v_1, \delta, \pi \rangle \quad \oplus \in \{+, -, *, /\} \qquad \langle e_2, \delta, \pi \rangle \rightsquigarrow \langle v_2, \delta, \pi \rangle \quad v = v_1 \oplus v_2}{\langle (e_1 \oplus e_2), \delta, \pi \rangle \rightsquigarrow \langle v, \delta, \pi \rangle}\ \text{(Binop)}$$

$$\frac{\langle e_0, \delta, \pi \rangle \rightsquigarrow \langle v_0, \delta, \pi \rangle \qquad \delta' = \{y \mapsto \delta(y) \mid y \in \texttt{dom}(\delta) \wedge y \neq a\}\ \cup\ \{a[0] \mapsto (\delta(a[0]) \cup \langle \pi, v_0 \rangle)\}}{\langle (a[i] = e_0), \delta, \pi \rangle \rightsquigarrow \langle \texttt{void}, \delta', \pi \rangle}\ \text{(Store)}$$

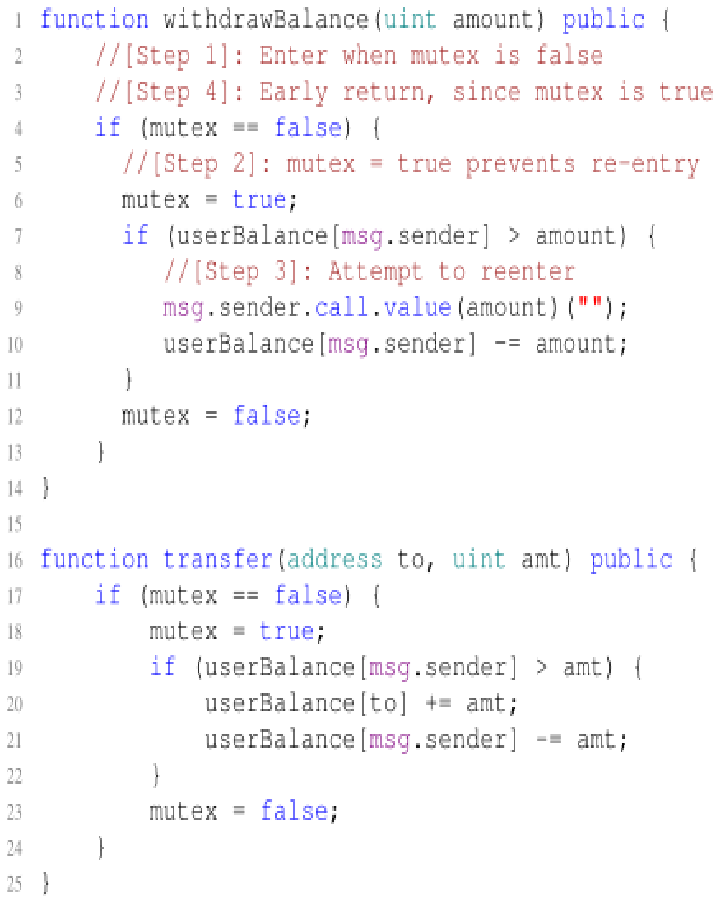
$$\frac{\langle \_, v \rangle = \delta(a[0])}{\langle a[i], \delta, \pi \rangle \rightsquigarrow \langle v, \delta, \pi \rangle}\ \text{(Load)}$$

$$\frac{\delta' = \{y \mapsto \delta(y) \mid y \in \texttt{dom}(\delta) \wedge y \neq e_0\}\ \cup\ \{e_0 \mapsto \langle \pi, e_1 \rangle \cup \delta(e_0)\}}{\langle (e_0 = e_1), \delta, \pi \rangle \rightsquigarrow \langle \texttt{void}, \delta', \pi \rangle}\ \text{(Assign)}$$

$$\frac{\delta' = \{y \mapsto \delta(y) \mid y \in \texttt{dom}(\delta) \wedge y \neq r\}\ \cup\ \{r \mapsto \langle \pi, \texttt{havoc}(r) \rangle\}}{\langle r = \texttt{f}(\vec{e}), \delta, \pi \rangle \rightsquigarrow \langle \texttt{void}, \delta', \pi \rangle}\ \text{(Ext)}$$

$$\frac{\langle e_0, \delta, \pi \rangle \rightsquigarrow \langle v_0, \delta, \pi \rangle \quad \pi' = \pi \wedge v_0 \qquad \delta' = \{y \mapsto \delta(y) \mid y \notin \texttt{lhs}(e_1)\}\ \cup\ \{y \mapsto \langle \pi', \texttt{havoc}(y) \rangle \mid y \in \texttt{lhs}(e_1)\}}{\langle (\texttt{while } e_0\ e_1), \delta, \pi \rangle \rightsquigarrow \langle v_0, \delta', \pi \wedge \neg v_0 \rangle}\ \text{(Loop)}$$

$$\frac{\langle e_0, \delta, \pi \rangle \rightsquigarrow \langle v_0, \delta, \pi \rangle \quad b = isTrue(v_0) \qquad \langle e_1, \delta, \pi \wedge b \rangle \rightsquigarrow \langle v_1, \delta_1, \pi_1 \rangle \qquad \langle e_2, \delta, \pi \wedge \neg b \rangle \rightsquigarrow \langle v_2, \delta_2, \pi_2 \rangle \qquad \delta' = \delta \cup \delta_1 \cup \delta_2}{\langle (\texttt{if } e_0\ e_1\ e_2), \delta, \pi \rangle \rightsquigarrow \langle \mu(b, v_1, v_2), \delta', \pi \rangle}\ \text{(If)}$$



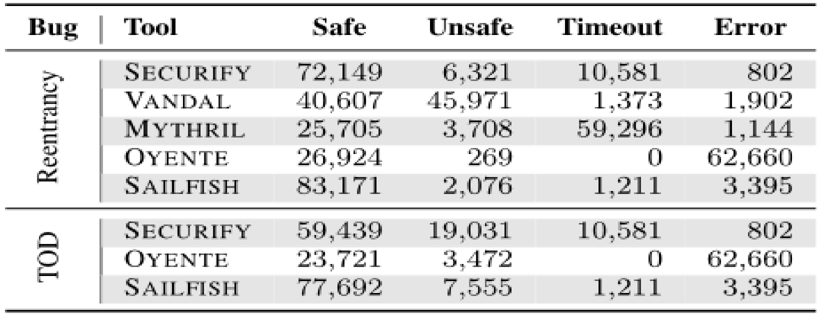
```
1  function withdrawBalance(uint amount) public {
2      //[Step 1]: Enter when mutex is false
3      //[Step 4]: Early return, since mutex is true
4      if (mutex == false) {
5        //[Step 2]: mutex = true prevents re-entry
6        mutex = true;
7        if (userBalance[msg.sender] > amount) {
8           //[Step 3]: Attempt to reenter
9           msg.sender.call.value(amount)("");
10          userBalance[msg.sender] -= amount;
11       }
12       mutex = false;
13     }
14 }
15
16 function transfer(address to, uint amt) public {
17     if (mutex == false) {
18         mutex = true;
19         if (userBalance[msg.sender] > amt) {
20             userBalance[to] += amt;
21             userBalance[msg.sender] -= amt;
22         }
23         mutex = false;
24     }
25 }
```

- Mutex:{<mutex=false,false},<mutex=false,true>}
- π: mutex == false ∧ userBalance[msg.sender] > amount
- δ = {mutex 7→ true, ...} (line 9)

Evaluation

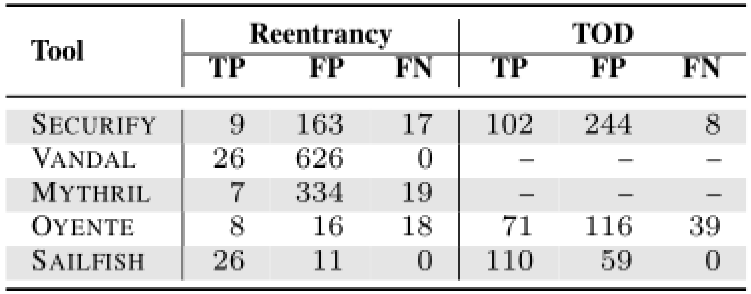
Dataset:89,853 smart contracts, small ([0, 500)), medium([500, 1000)) large ([1000, ∞))

Comparison against other tools



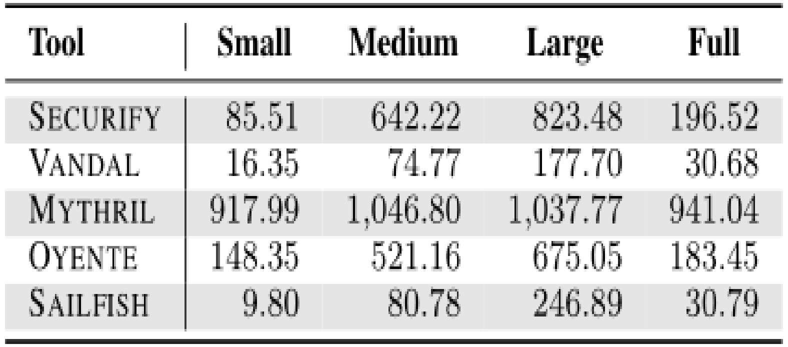
Bug	Tool	Safe	Unsafe	Timeout	Error
Reentrancy	SECURIFY	72,149	6,321	10,581	802
	VANDAL	40,607	45,971	1,373	1,902
	MYTHRIL	25,705	3,708	59,296	1,144
	OYENTE	26,924	269	0	62,660
	SAILFISH	83,171	2,076	1,211	3,395
TOD	SECURIFY	59,439	19,031	10,581	802
	OYENTE	23,721	3,472	0	62,660
	SAILFISH	77,692	7,555	1,211	3,395

Ground truth determination



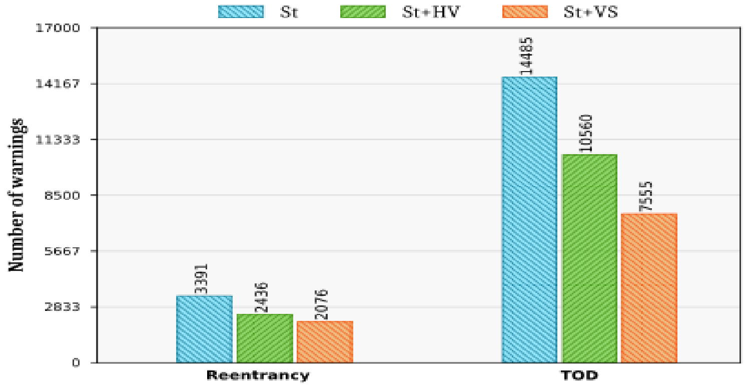
Tool	Reentrancy			TOD		
	TP	FP	FN	TP	FP	FN
SECURIFY	9	163	17	102	244	8
VANDAL	26	626	0	–	–	–
MYTHRIL	7	334	19	–	–	–
OYENTE	8	16	18	71	116	39
SAILFISH	26	11	0	110	59	0

Performance analysis



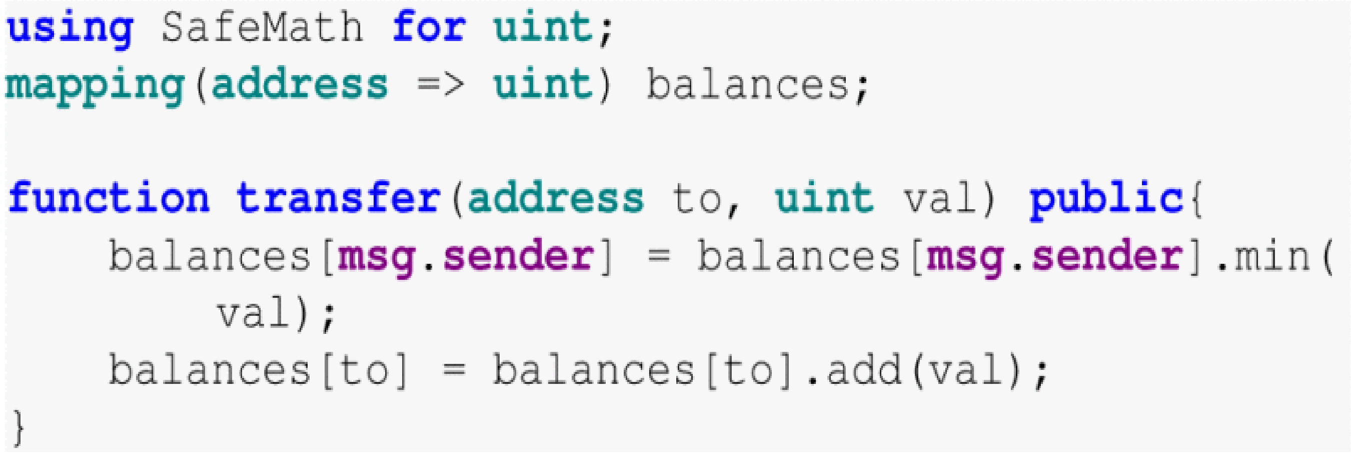
Tool	Small	Medium	Large	Full
SECURIFY	85.51	642.22	823.48	196.52
VANDAL	16.35	74.77	177.70	30.68
MYTHRIL	917.99	1,046.80	1,037.77	941.04
OYENTE	148.35	521.16	675.05	183.45
SAILFISH	9.80	80.78	246.89	30.79

Ablation study

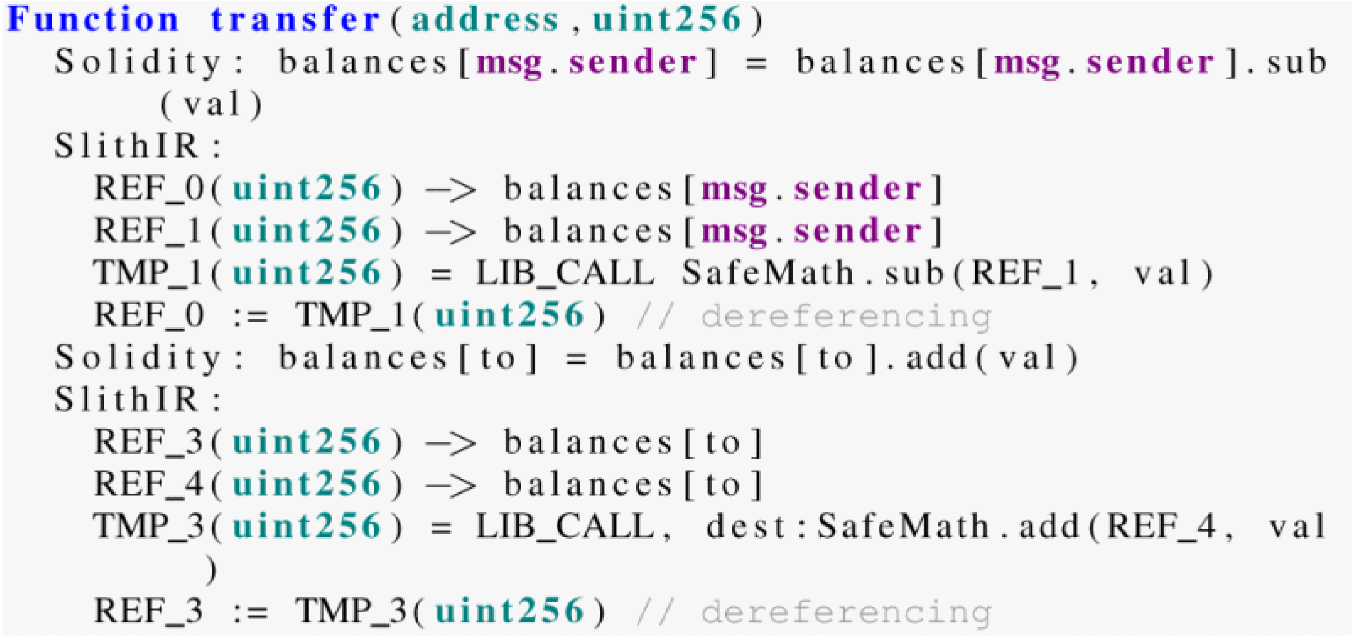


Limitation

- source-code dependency
- potential unsoundness

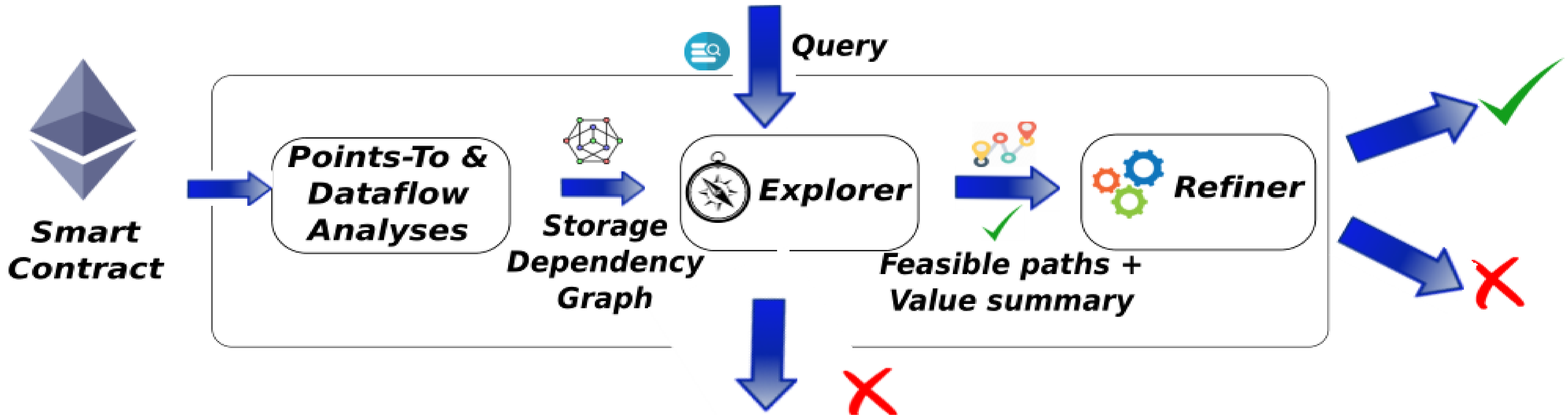


```
using SafeMath for uint;
mapping(address => uint) balances;

function transfer(address to, uint val) public{
    balances[msg.sender] = balances[msg.sender].min(
        val);
    balances[to] = balances[to].add(val);
}
```



```
Function transfer(address,uint256)
  Solidity: balances[msg.sender] = balances[msg.sender].sub
      (val)
  SlithIR:
    REF_0(uint256) -> balances[msg.sender]
    REF_1(uint256) -> balances[msg.sender]
    TMP_1(uint256) = LIB_CALL SafeMath.sub(REF_1, val)
    REF_0 := TMP_1(uint256) // dereferencing
  Solidity: balances[to] = balances[to].add(val)
  SlithIR:
    REF_3(uint256) -> balances[to]
    REF_4(uint256) -> balances[to]
    TMP_3(uint256) = LIB_CALL, dest:SafeMath.add(REF_4, val
        )
    REF_3 := TMP_3(uint256) // dereferencing
```

Conclusion



- model state-inconsistency detection as **hazardous access queries** over SDG
- a novel **value-summary analysis** that efficiently computes global constraints over storage variables