

Solc-Verify: A Modular Verifier for Solidity Smart Contracts

Ákos Hajdu^{1,2}, Dejan Jovanović¹

¹SRI International

²Budapest University of Technology and Economics

SRI International[®]

Introduction

SRI International[®]

Blockchain

- Records **transactions**

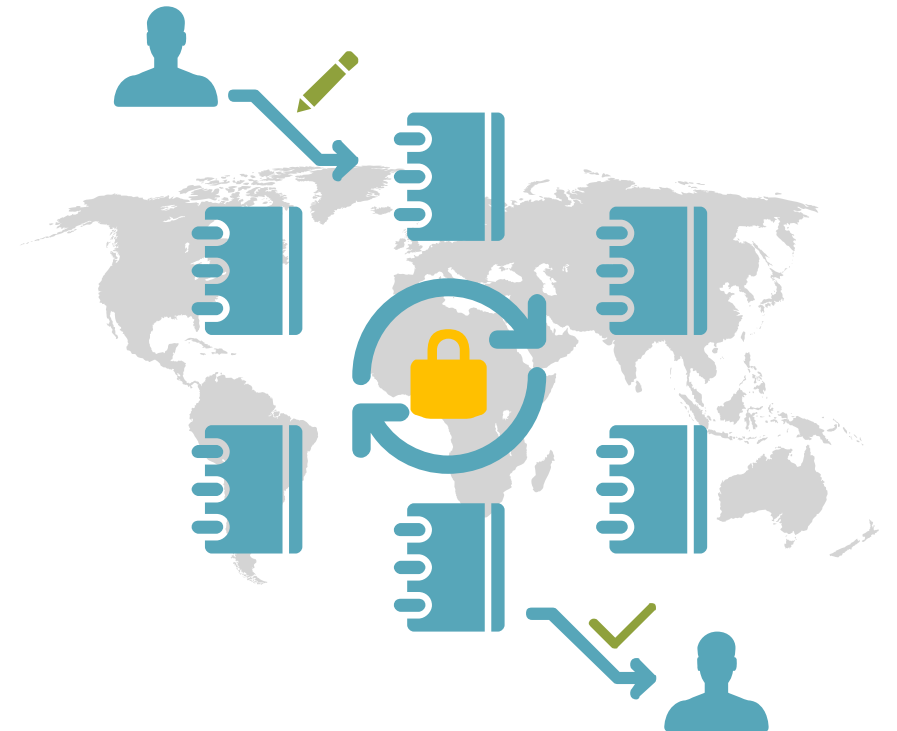
- Blocks linked by cryptographic hash
- Permanent and trusted

- **Decentralized** ledgers

- No trusted central party
- Consensus protocol

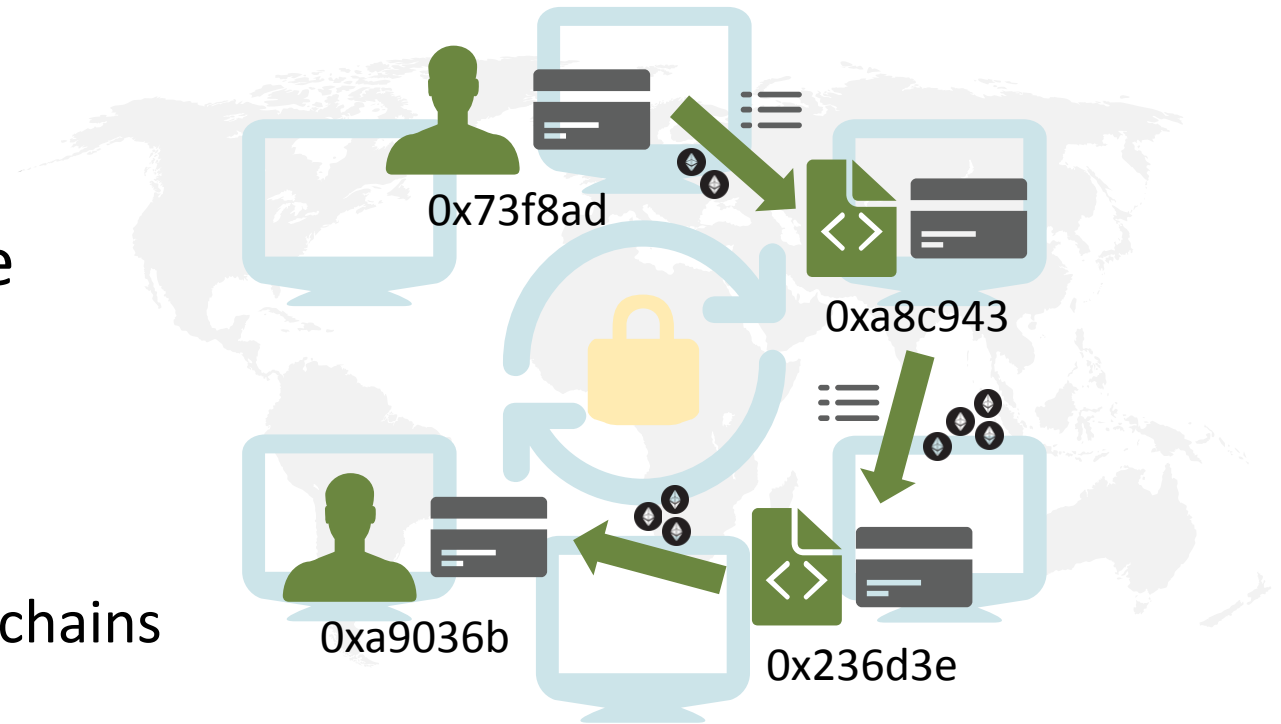
- Example: Bitcoin

- Users have balances
- Transactions transfer coins



Distributed Computing Platforms

- Ledger stores **data and code**
 - Smart contracts
 - Addresses, balances
- **Transactions execute** contract code
 - Operate on data, interactions
 - Consensus: identical execution
- Use cases
 - Tokens, multi-sig wallets, IoT, supply chains
- Example: Ethereum



Programming Ethereum: Solidity

State variable

Function

Function

```
contract SimpleBank {  
    mapping(address=>uint) user_balances;  
  
    function deposit() payable public {  
        user_balances[msg.sender] += msg.value;  
    }  
  
    function withdraw() public {  
        uint amount = user_balances[msg.sender];  
        if (amount > 0 && this.balance >= amount) {  
            (bool ok,) => msg.sender.call.value(amount)("");  
            if (!ok) revert();  
            user_balances[msg.sender] = 0;  
        }  
    }  
}
```



More Bugs

A Hacking of More Than \$50 Million Dashes Hopes in the World of Virtual Currency

By Nathaniel Popper

June 17, 2016

A hacker on Friday siphoned more than \$50 million from an [experimental virtual currency](#)

ETHEREUM, TECHNOLOGY

BatchOverflow Exploit Cripples Ethereum Tokens, Major Deposits



Sam Town



April 25, 2018



3 min read



5827 Views

Shut down of 0x Exchange v2.0 contract and migration to patched version



Will Warren in 0x Blog

Follow

Jul 13 · 2 min read

Today (7/12) at approximately 4:30 PM PT, we were made aware of a potential exploit in the 0x v2.0 [Exchange](#) contract by a third-party security researcher [samczsun](#). This vulnerability would allow an attacker to fill certain orders with invalid signatures. **This vulnerability does not effect the ZRX token contract; your digital assets are safe.**

GOOD JOB | By Jordan Pearson | Nov 7 2017, 11:24am

Someone 'Accidentally' Locked Away \$150M Worth of Other People's Ethereum Funds

able.

Hacked. Again

allet was hacked again:

[security-alert.html](#)

y funds can be moved out of the [ANY Parity] multi-

/companies/ICOs are using Parity-generated multisig wallets. is frozen and (probably) lost forever.

Motivation

- **New paradigm** for developers
 - Semantic misalignments
- **Open world**
 - Publishing a contract == bug bounty
- **Permanent**
 - No reverting / patching
- **Consequences**
 - Real assets / money

Verification needed

- **Existing approaches**
 - Vulnerability patterns: MythX, Slither, ...
 - Theorem provers: KEVM, Scilla, ...
 - Finite automata: FSolidM, ...
 - Translation to SMT: Zeus, VeriSol, ...
- **Limitations**
 - Expressiveness
 - User-friendliness
 - False alarms, missed bugs
 - Manual actions

Our Goal

- Provide a **practical tool**
- Check **high-level**, user-specified **properties**
- Strike a balance between



Expressiveness

Wide range of specifications
to be expressed



User friendliness

Formal methods expertise
not required



Soundness, precision

No false alarms, no missed bugs



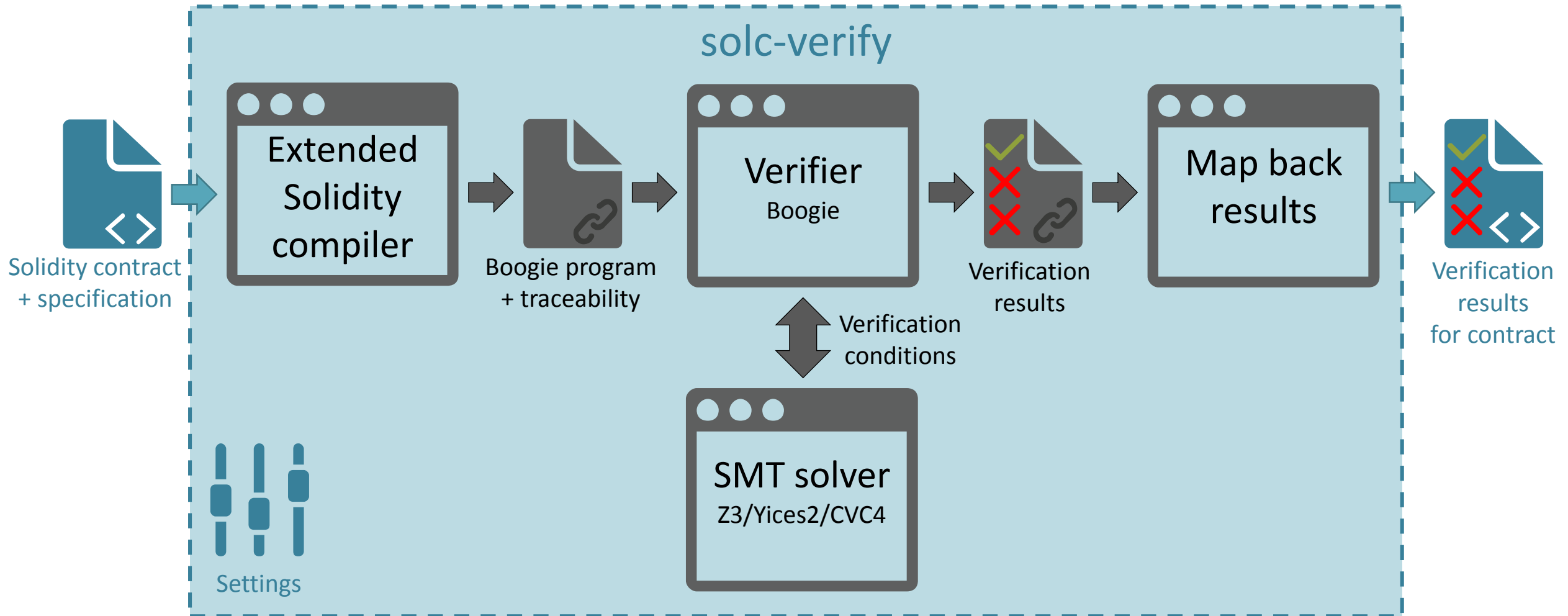
Automation

No user interaction required

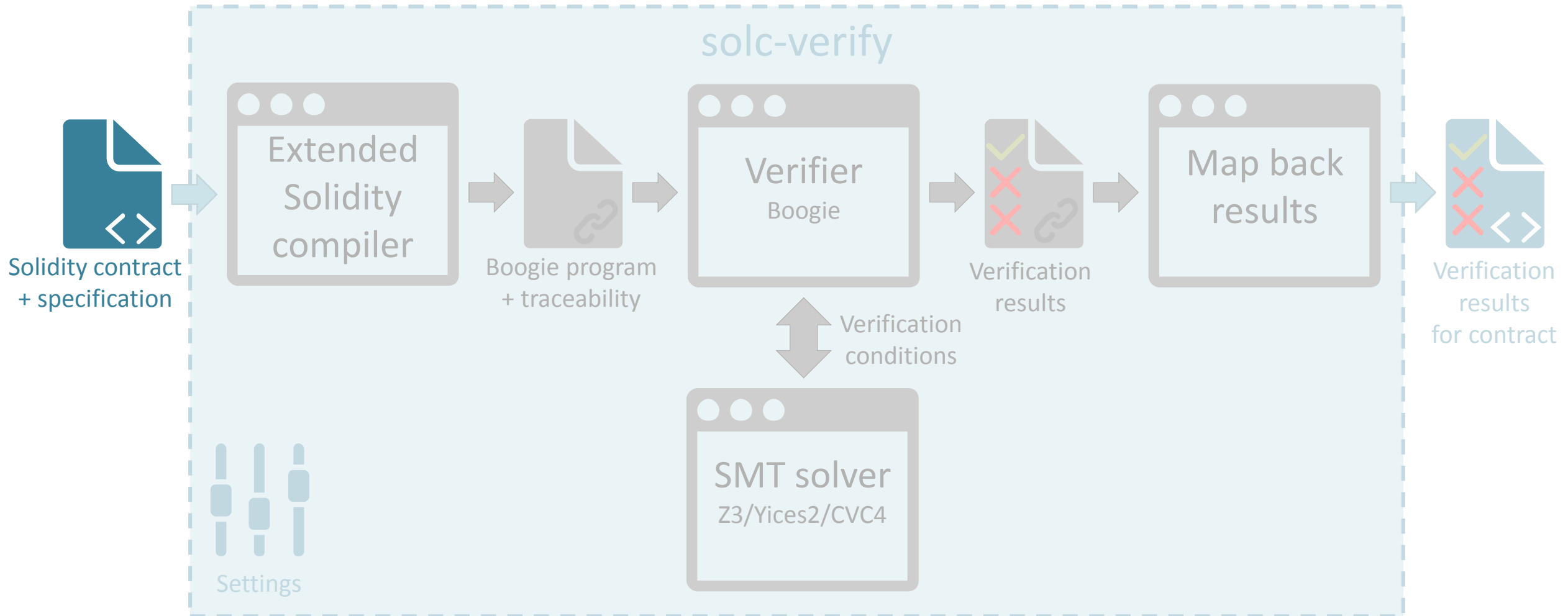
Solc-Verify

SRI International[®]

Overview



Overview



Specification

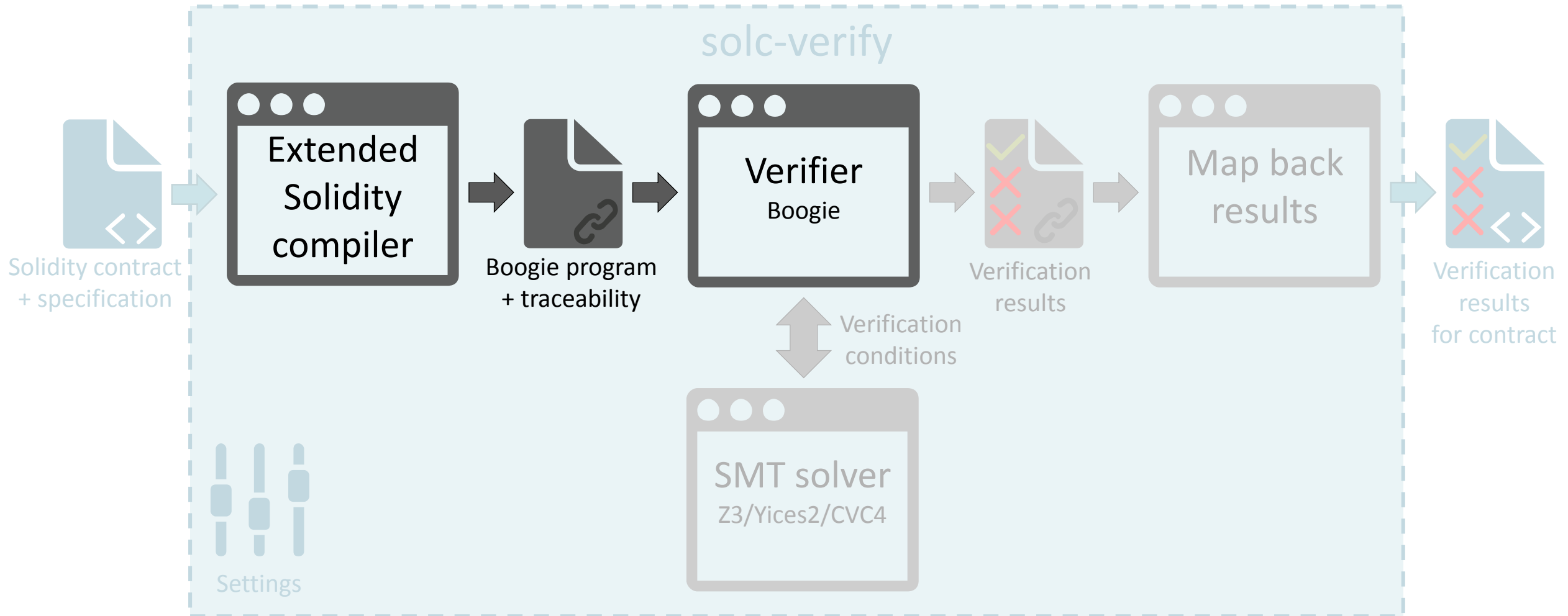
- Solidity provides
 - require, assert
- Our **annotation language**
 - Features
 - Pre/postconditions
 - Contract level invariants
 - Loop invariants
 - **Solidity expressions** (side effect free)
 - Scope of the annotated element
 - Quantifier free
 - Sum over collections (see later)
 - Might extend as needed

```
/// @notice invariant x == y
contract C {
    int x;
    int y;

    /// @notice precondition x == y
    /// @notice postcondition x == (y + n)
    function add_to_x(int n) internal {
        x = x + n;
        require(x >= y);
    }

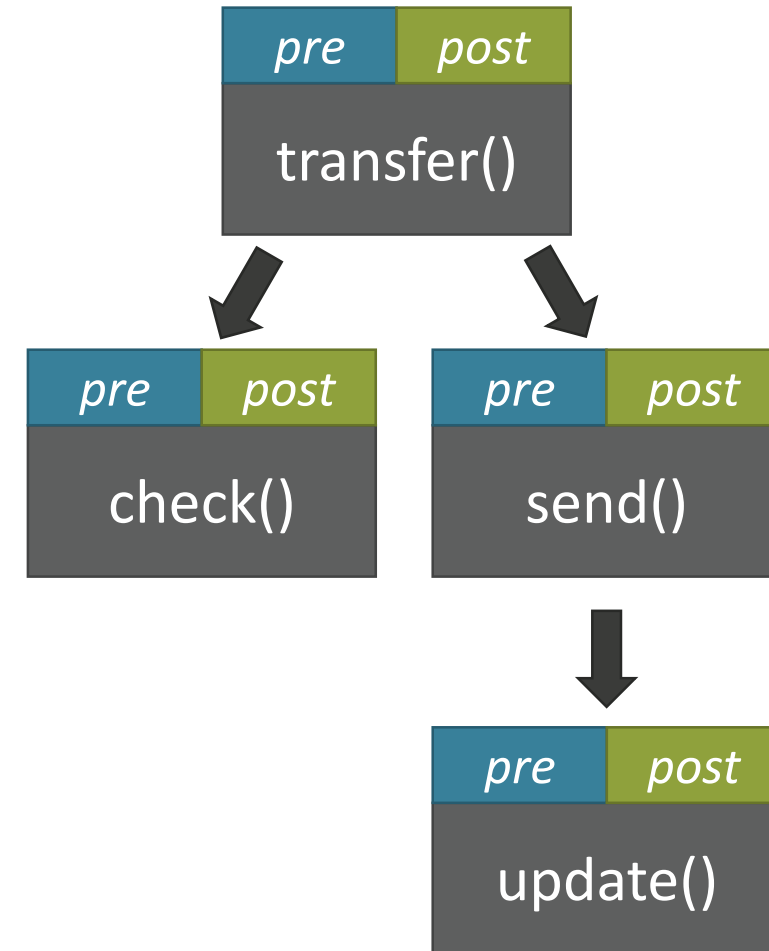
    function add(int n) public {
        require(n >= 0);
        add_to_x(n);
        /// @notice invariant y <= x
        while (y < x) {
            y = y + 1;
        }
    }
}
```

Overview



Verification

- **Functional correctness** w.r.t completed transactions
 - **Expected failure**: explicit guards (require, revert)
 - **Unexpected failure**: assertion, overflow
 - **Specification violation**: pre/postconditions, invariants
 - Reentrancy: check invariant at external call
- **Modular verification**
 - $pre \wedge body \rightarrow post$
 - Replace calls with their specification
 - Discharge verification conditions to SMT solver



Translation

- State variables → 1D global heap
- Functions → procedures
- Extra semantics of the blockchain
 - E.g., balances, payments
- Similar to program verification, but much more in the details
 - Blockchain semantics
 - Message passing
 - Transactional behavior

```
contract SimpleBank {  
    mapping(address=>uint) user_balances;  
  
    function deposit() payable public {  
        user_balances[msg.sender] += msg.value;  
    }  
}
```

```
var _balance: [address]int;  
var user_balances: [address][address]int;  
  
procedure deposit(_this: address,  
                 _msg_sender: address,  
                 _msg_value: int) {  
    _balance[_this] += _msg_value;  
    user_balances[_this][_msg_sender] += _msg_value;  
}
```

Arithmetic – Model of Computation

- Solidity

8-256 bit, overflow

```
uint8 x = 255;  
uint8 y = 1;  
x + y == 0;
```



- solc-verify

Integers (SMT)

```
int x = 255;  
int y = 1;  
x + y == 256;
```

Not precise

Bitvector (SMT)

```
bv8 x = 255bv8;  
bv8 y = 1bv8;  
x + y == 0bv8;
```

Not scalable

256 bits default
(see example later)

Modular

```
int x = 255;  
int y = 1;  
(x + y) % 256 == 0;
```

Precise & scalable

- Checking for overflows

- Range check of every operation

- False alarms

- Compute precise & unbounded, compare at end of block

- No alarm if developer checks

```
function f(uint x, uint y) {  
    uint z = x + y;  
    require (z >= x);  
}
```



```
int z = (x + y) % 255;  
int z0 = x + y;  
assume (z >= x);  
assert (z == z0);
```



Arithmetic – Sum of Collections

- Invariant over **sum of collections**
 - Common in wallets, tokens (ERC20)
- Not expressible in Solidity/FOL
- Our abstraction
 - **Shadow variable** for each collection
 - Update shadow with collection

```
/// @notice invariant sum(user_balances) <= this.balance
contract SimpleBank {
    mapping(address=>uint) user_balances;

    function deposit() payable public {
        // ...
    }

    function withdraw() public {
        // ...
    }
}
```

Examples and Demo

SRI International[®]

Annotated Contracts – Reentrancy Detection (DAO)


- Report every external call?
 - False alarms
- Contract invariant
 - Does not hold at external call
- Fixed version: deduct balance first
 - No false alarm, invariant holds

DEMO

```
/// @notice invariant sum(user_balances) <= this.balance
contract SimpleBank {
    mapping(address=>uint) user_balances;

    function deposit() payable public {
        user_balances[msg.sender] += msg.value;
    }

    function withdraw() public {
        uint amount = user_balances[msg.sender];
        if (amount > 0 && this.balance >= amount) {
            (bool ok,) = msg.sender.call.value(amount)("");
            if (!ok) revert();
            user_balances[msg.sender] = 0;
        }
    }
}
```



Annotated Contracts – Overflow Detection (BEC token)

- Integers: **cannot detect**
- Range check after every operation: **false alarms**
- Bitvectors: **scale up to 16 bits (Z3)**
- Modular arithmetic, delayed checks: **overflow reported, no other false alarms**
 - Fixed version: **no alarms**
- Annotations: **high-level property proved**

DEMO

```
/// @notice invariant sum(balances) == totalSupply
contract BecToken {
    using SafeMath for uint256;
    uint256 totalSupply;
    mapping(address => uint256) balances;
    function batchTransfer(address[] _receivers, uint256 _value) {
        uint cnt = _receivers.length;
        uint256 amount = uint256(cnt) * _value;
        require(cnt > 0 && cnt <= 20);
        require(_value > 0 && balances[msg.sender] >= amount);
        balances[msg.sender] = balances[msg.sender].sub(amount);
        /// @notice invariant totalSupply ==
            sum(balances) + (cnt - i) * _value
        /// @notice invariant i <= cnt
        for (uint i = 0; i < cnt; i++)
            balances[_receivers[i]] = balances[_receivers[i]].add(_value);
    }
}
```



Unannotated Contracts

- 37 531 contracts
- 7 836 accepted by compiler 0.4.25
- Roughly 50% can be processed
 - Small differences between encodings
 - Missing features: structs, enums, special members, returning arrays, ...
- No annotations
 - Require, assert, overflows
- Inconsistent usage of assert and require



```
uint z = x + y;  
assert (z >= x);
```

```
assert (now >= saleEnd);
```

```
assert (msg.sender == owner);
```

```
bool ok = msg.sender.call("...");  
assert (ok);
```


Unannotated Contracts – Example

- VestChain

- If guard against overflow → **require**
- If implicit assumption on fixed-cap → **explicit invariant**


```
uint256 public totalSupply;
mapping (address => uint256) holders;

function transfer(address _to, uint256 _val) {
    require(holders[msg.sender] >= _val);
    require(msg.sender != _to);
    assert(_val <= holders[msg.sender]);
    holders[msg.sender] -= _val;
    holders[_to] += _val;
    assert(holders[_to] >= _val);
}
```



Unannotated Contracts – Example

- FoodStore
 - Overflow

```
function buyFood(uint32 _bundles) {  
    uint cost = _bundles * price;   
    require(msg.value >= cost);  
    uint fundsExcess = msg.value - cost;  
    if (fundsExcess > 1 finney) {  
        msg.sender.transfer(fundsExcess);  
    }  
}
```

Conclusions

SRI International[®]

Conclusions

- Solc-Verify
 - **Modular verifier** for smart contracts
 - Specification **annotations**
 - Translation to Boogie/SMT
- Properties
 - Express **high-level properties** in **user-friendly** way
 - **Sound** and **automated** backend
- Current state
 - **Open source**, under development
 - Up-to date with latest compiler
 - Support for structs, access control specs, ...
- Future work
 - Cover missing Solidity features
 - Translation validation
 - Invariant inference

```
/// @notice invariant x == y
contract C {
    int x; int y;

    /// @notice precondition x == y
    /// @notice postcondition x == (y + n)
    function add_to_x(int n) internal {
        x = x + n;
        require(x >= y);
    }

    function add(int n) public {
        require(n >= 0);
        add_to_x(n);
        /// @notice invariant y <= x
        while (y < x) { y = y + 1; }
    }
}
```

github.com/SRI-CSL/solidity

SRI International[®]

Translation

```
1  contract A {  
2    int public x;  
3    function set(int _x) public { x = _x; }  
4  }  
5  contract B {  
6    A a;  
7    function setXofA(uint x) public { a.set(x); }  
8    function getXofA() public returns (uint) {  
9      return a.x();  
10   }  
11 }
```

```
1  var x: [address]int;  
2  procedure set(_this: address, _x: int) {  
3    x := x[_this :=_x];  
4  }  
5  var a: [address]address;  
6  procedure setXofA(_this: address, x: int) {  
7    call set(a[_this], x);  
8  }  
9  procedure getXofA(_this: address) returns (r: int) {  
10   r := x[a[_this]];  
11 }
```

Translation

```
1 contract Wallet {  
2     address owner;  
3  
4     modifier onlyOwner() {  
5         require(msg.sender == owner);  
6     }  
7  
8     function receive() payable public {  
9         // Actions could be performed here  
10    }  
11    function pay(address to, uint amount)  
12        public onlyOwner {  
13        to.transfer(amount);  
14    }
```

```
1 var _balance: [address]int;  
2  
3 var owner: [address]address;  
4  
5 procedure receive(_this: address, _msg_sender: address,  
6     _msg_value: int) {  
7     _balance := _balance[_this := _balance[_this] + _msg_value];  
8     // Actions could be performed here  
9 }  
10 procedure pay(_this: address, _msg_sender: address, _msg_value:  
11     int, to: address, amount: int) {  
12     assume(_msg_sender == owner[_this]);  
13     assume(_balance[_this] >= amount);  
14     _balance := _balance[_this := _balance[_this] - amount];  
15     _balance := _balance[to := _balance[to] + amount];  
16 }
```

Unannotated Contracts

- PreSale

- Assertion checks stronger condition, **can fail**
- Weaker condition → **false alarm** due to modular reasoning
 - Lift to **contract invariant**
 - Use **require** in the beginning and **assert** in the end

```
uint256 maxEther = 1000 ether;  
uint256 etherRaised = 0;  
  
function () external payable {  
    assert(etherRaised < maxEther);  
    require(msg.value != 0);  
    require(etherRaised + msg.value <= maxEther);  
  
    etherRaised += msg.value;  
}
```




Unannotated Contracts

- MainframeTokenDistribution
 - Overflow

```
uint public totalDistributed;

function distributeTokens(address tokenOwner, address[] recipients, uint[] values) onlyOwner {
    require(recipients.length == values.length);
    for(uint i = 0; i < recipients.length; i++) {
        if(values[i] > 0) {
            require(mainframeToken.transferFrom(tokenOwner, recipients[i], values[i]));
            totalDistributed += values[i];
        }
    }
}
```



Etherscan

Encoding	int	bv	mod	mod-overflow
Translated	4096	3919	3926	3926
CVC4	4090 (0.71s)	3837 (0.99s)	3921 (0.72s)	3911 (0.79s)
YICES2	3892 (1.15s)	3854 (0.86s)	3903 (0.75s)	3859 (0.87s)
z3	3897 (1.24s)	3831 (1.10s)	3892 (0.87s)	3894 (0.88s)