Solc-Verify: A Modular Verifier for Solidity Smart Contracts

Ákos Hajdu, Dejan Jovanović

SRI International®

Introduction

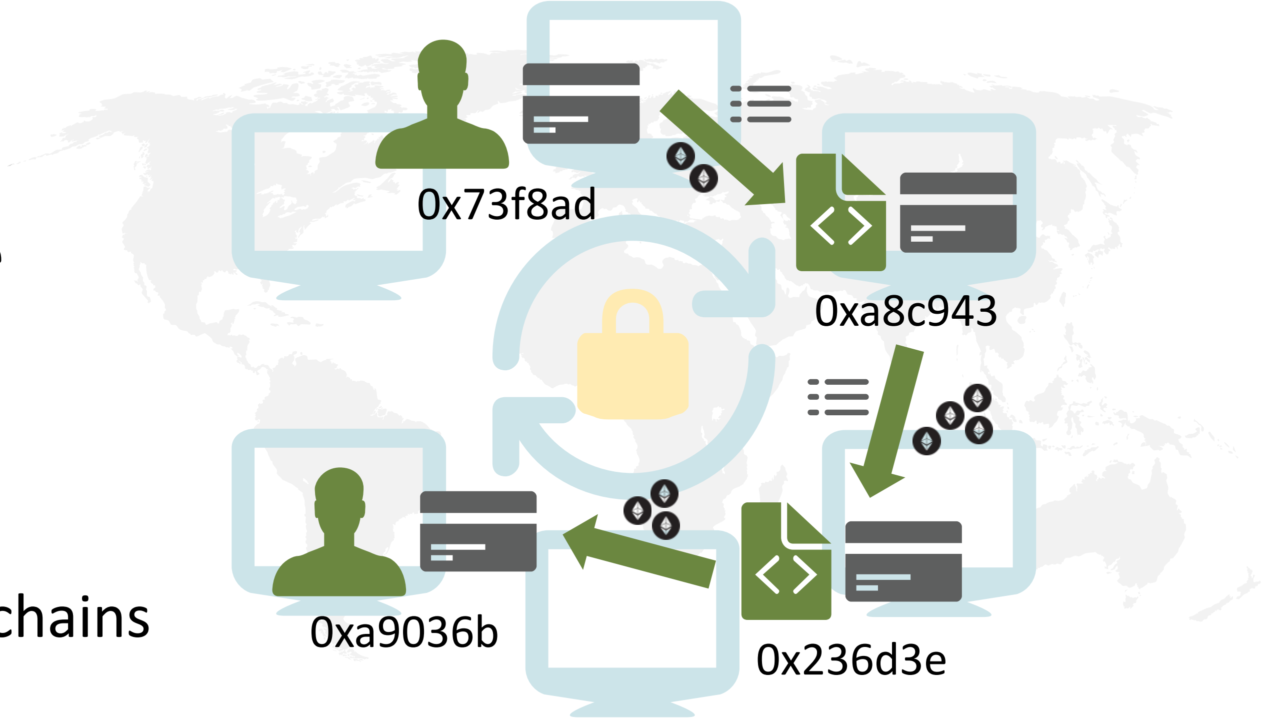
SRI International®

Blockchain

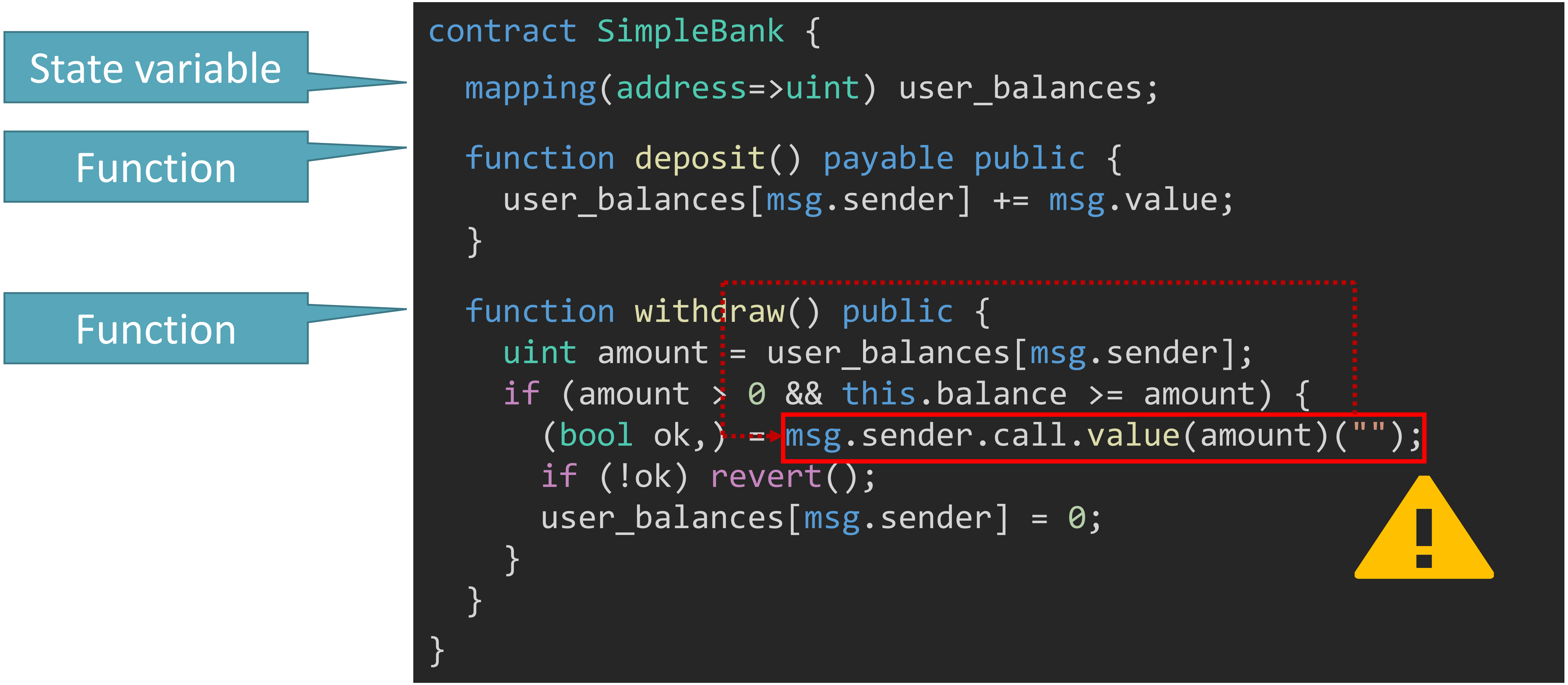
- Records transactions
  - Blocks linked by cryptographic hash
  - Permanent and trusted
- Decentralized ledgers
  - No trusted central party
  - Consensus protocol
- Example: Bitcoin
  - Users have balances
  - Transactions transfer coins

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

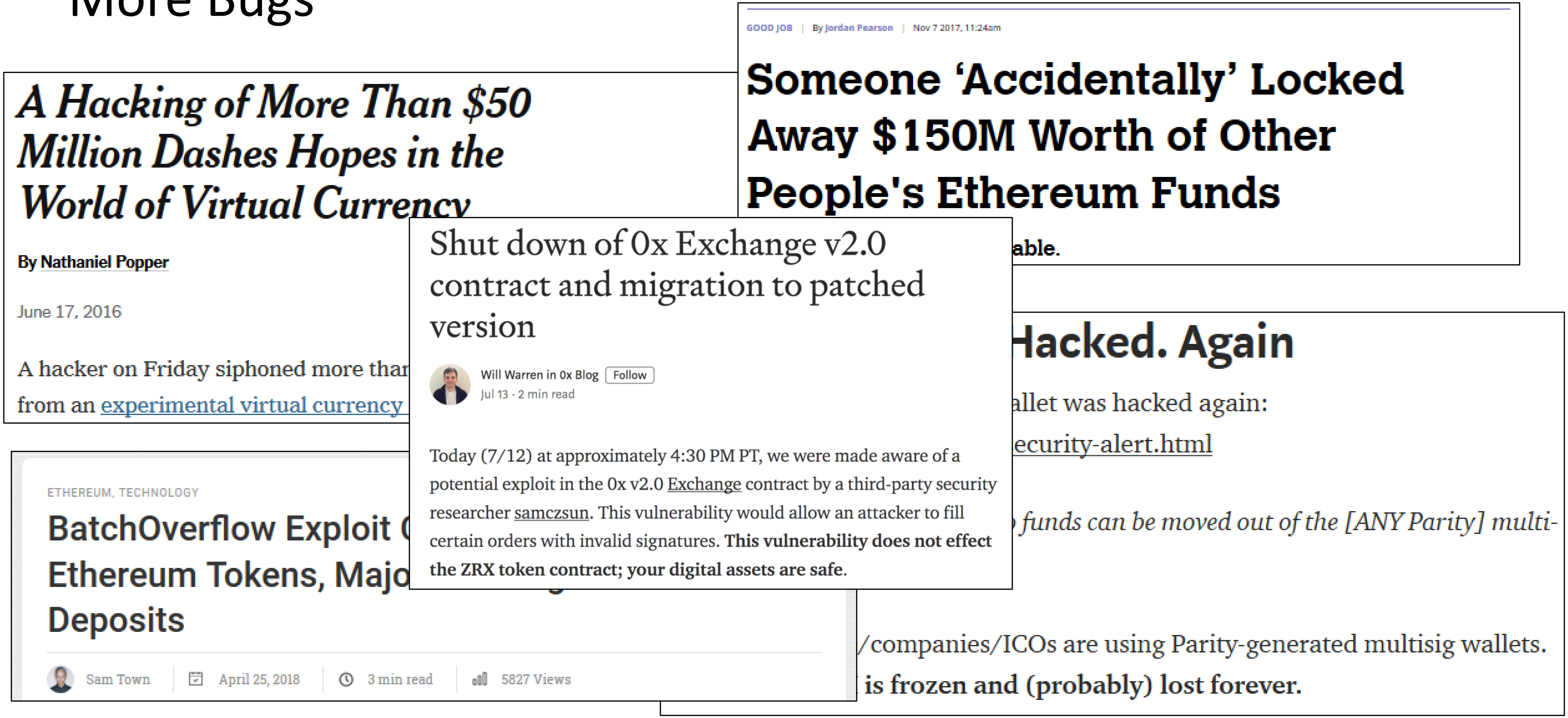


```
contract SimpleBank {
  mapping(address=>uint) user_balances;

  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 thar

from an experimental virtual currency

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

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

able.

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.**

Hacked. Again

allet was hacked again:

ecurity-alert.html

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

/companies/ICOs are using Parity-generated multisig wallets.

is frozen and (probably) lost forever.

ETHEREUM, TECHNOLOGY

BatchOverflow Exploit

Ethereum Tokens, Majo

Deposits

Sam Town | April 25, 2018 | 3 min read | 5827 Views

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: VeriSol, VerX, …
- 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

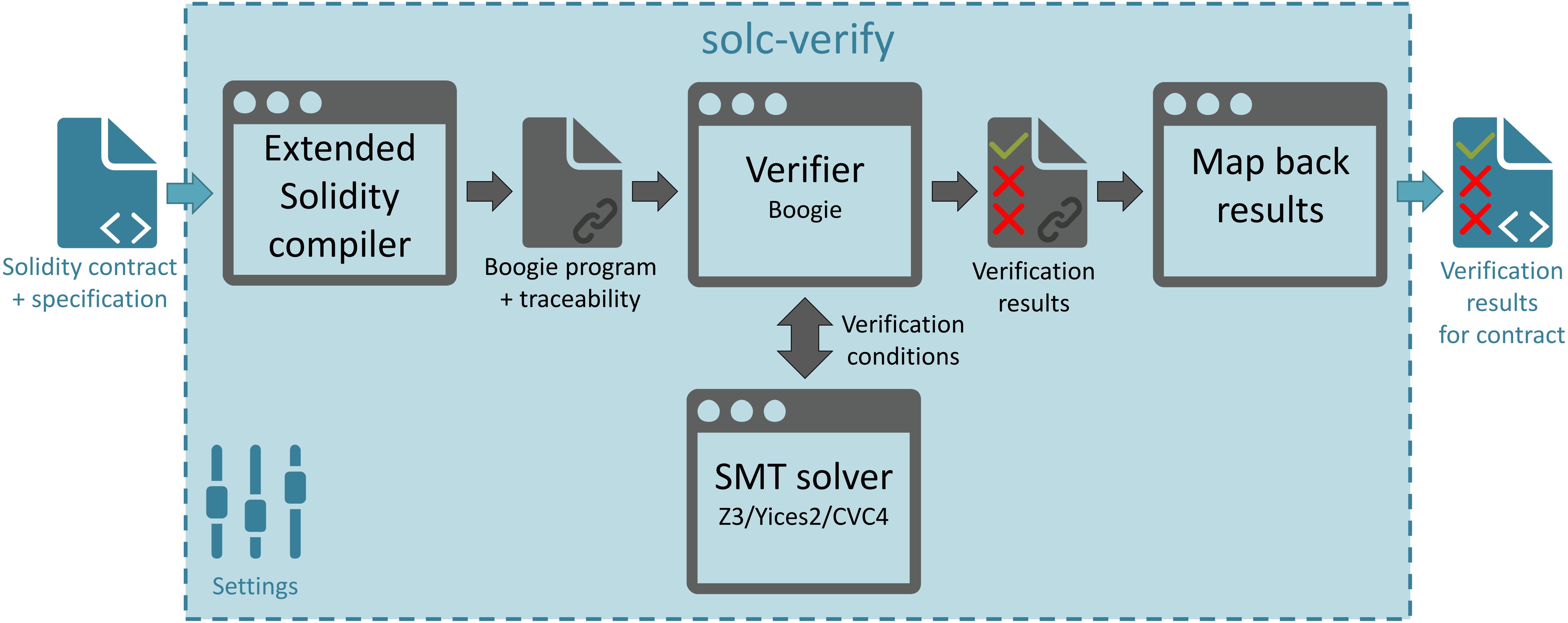
Soundness, precision
No false alarms, no missed bugs

Automation
No user interaction required

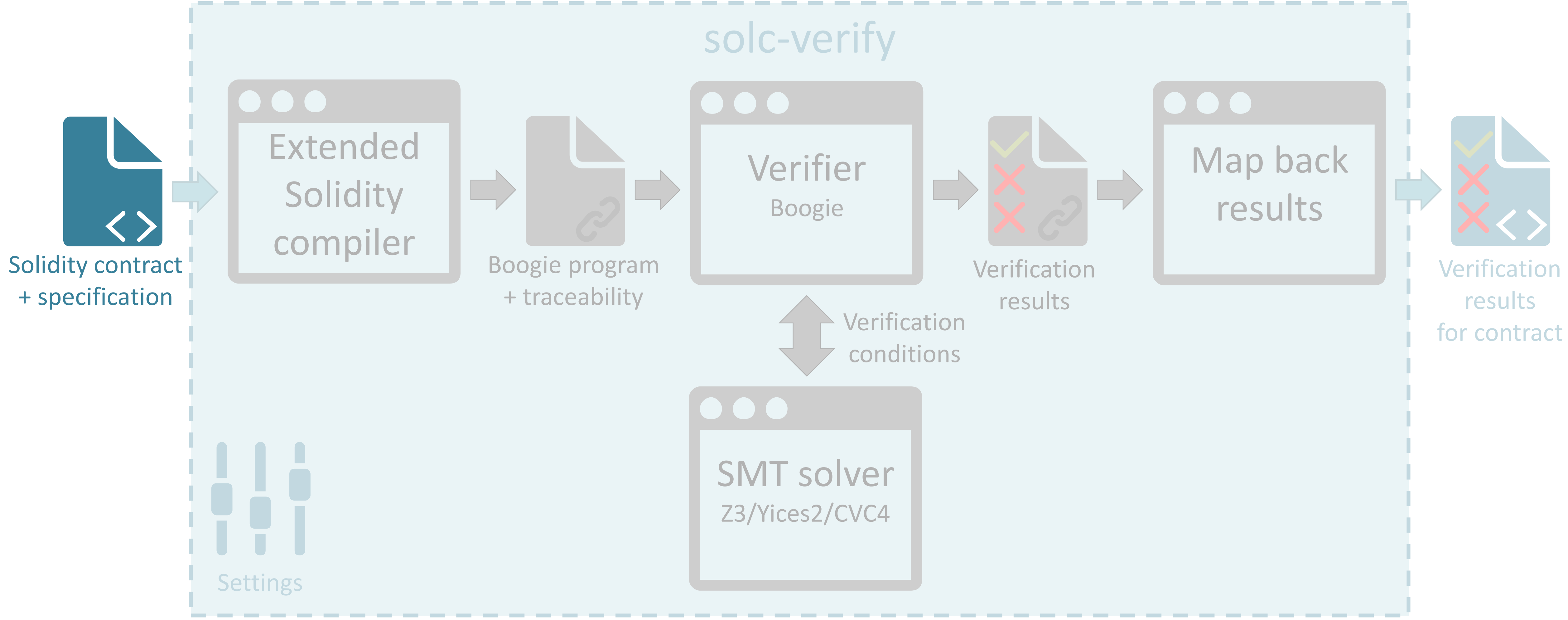
Solc-Verify

SRI International®

Overview

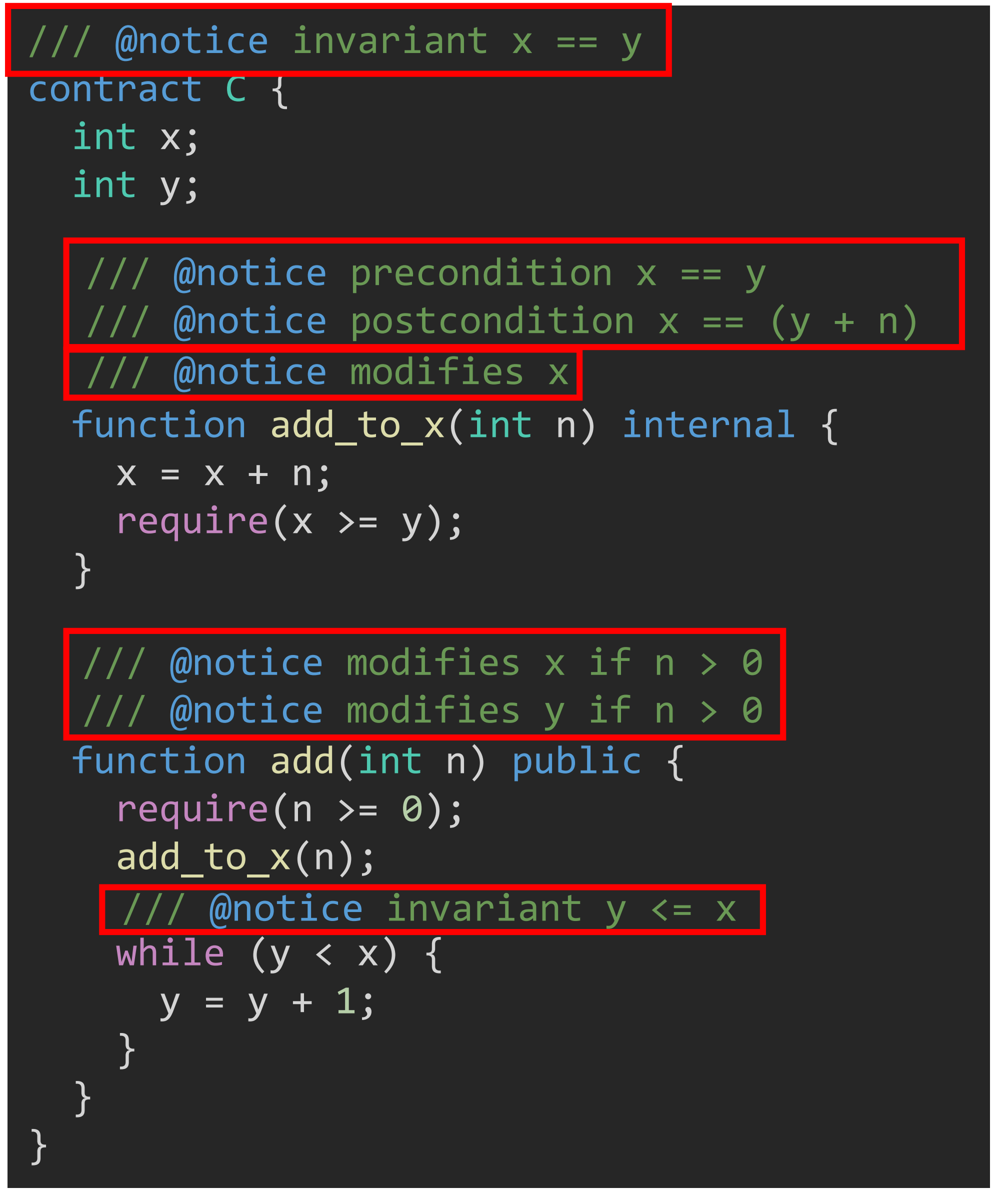

solc-verify
Solidity contract
+ specification
Extended Solidity compiler
Boogie program
+ traceability
Verifier
Boogie
Verification conditions
SMT solver
Z3/Yices2/CVC4
Verification results
Map back results
Verification results for contract
Settings

Overview


solc-verify
Solidity contract
+ specification
Extended Solidity compiler
Boogie program
+ traceability
Verifier
Boogie
Verification conditions
SMT solver
Z3/Yices2/CVC4
Verification results
Map back results
Verification results for contract
Settings

Specification

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

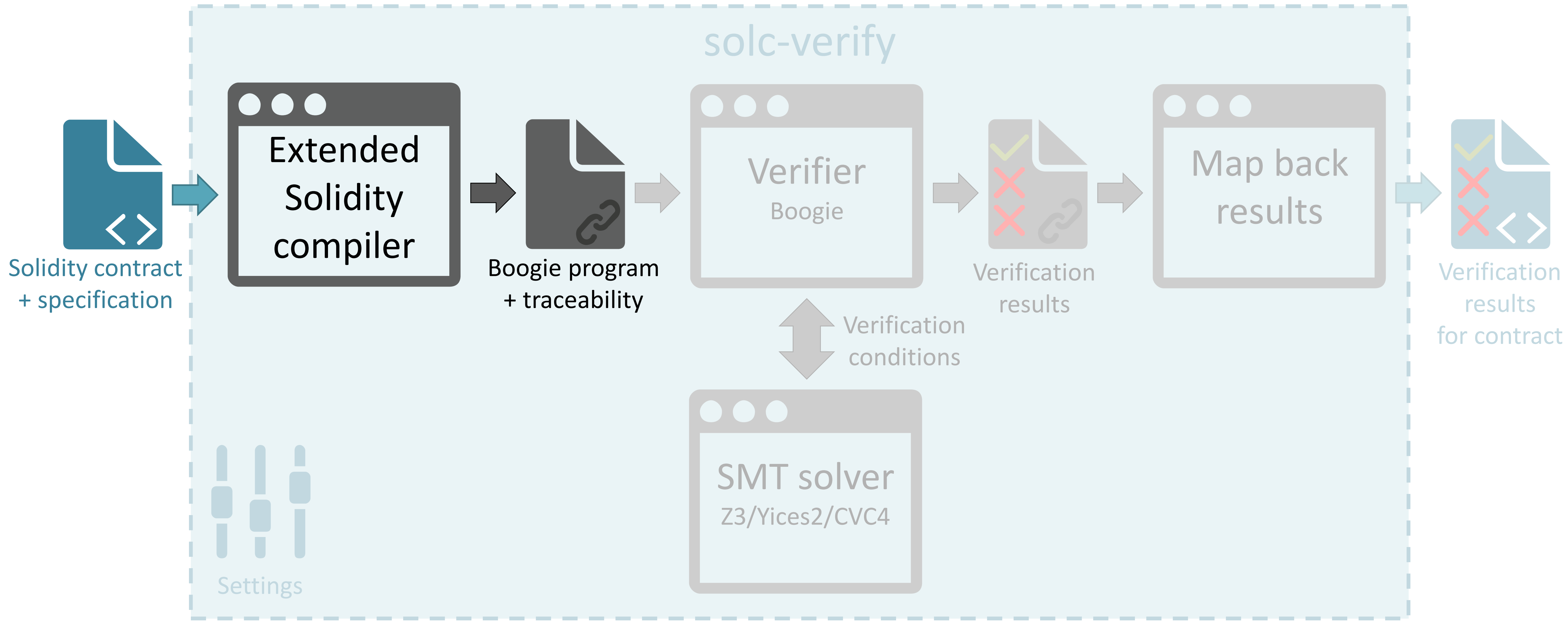


```
/// @notice invariant x == y
contract C {
  int x;
  int y;

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

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

Overview


solc-verify
Solidity contract
+ specification
Extended
Solidity
compiler
Boogie program
+ traceability
Verifier
Boogie
Verification
results
Map back
results
Verification
results
for contract
Verification
conditions
SMT solver
Z3/Yices2/CVC4
Settings

Translation

- State variables → 1D global heap
- Functions → procedures
- Extra semantics of the blockchain
  - E.g., balances, payments

```
contract SimpleBank {
  mapping(address=>uint) user_balances;

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

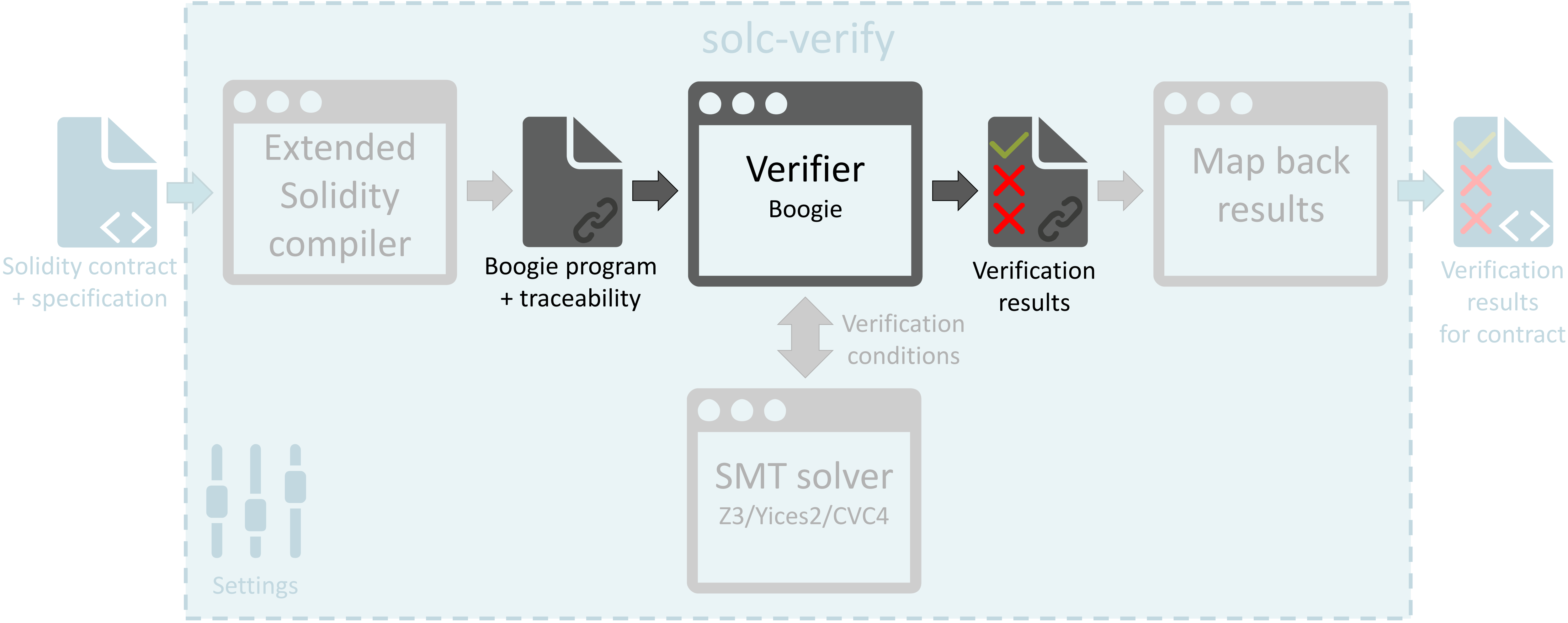
- Similar to program verification, but much more in the details
  - Blockchain semantics
  - Message passing
  - Transactional behavior
  - Memory models

```
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;
}
```

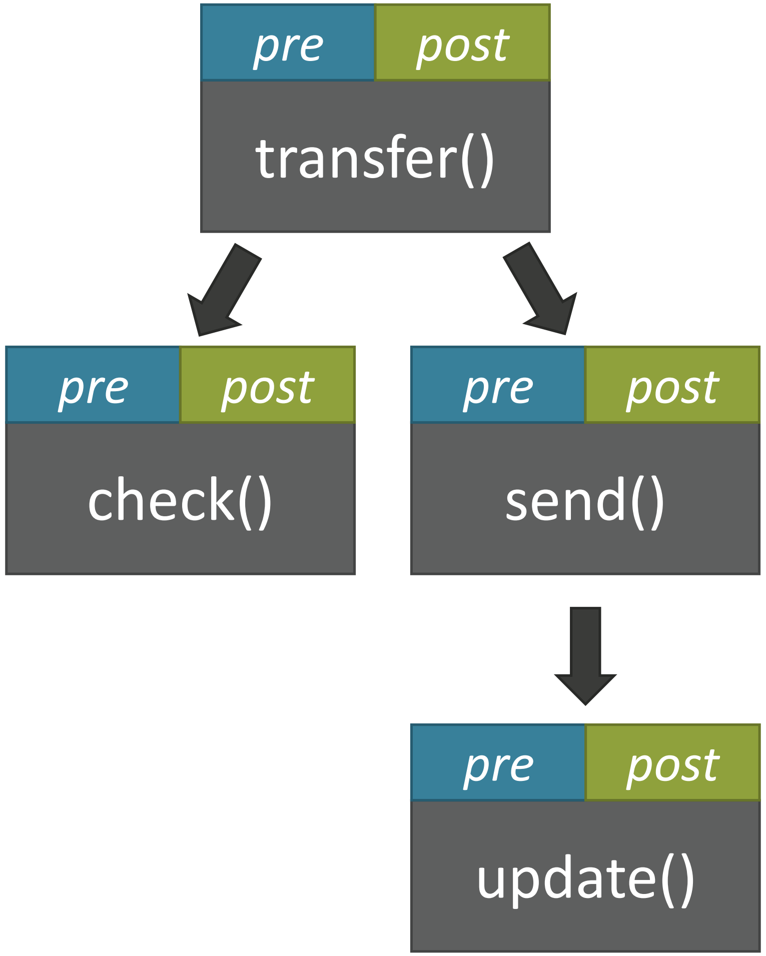
Overview


solc-verify
Solidity contract + specification
Extended Solidity compiler
Boogie program + traceability
Verifier
Boogie
Verification results
Map back results
Verification results for contract
Verification conditions
SMT solver
Z3/Yices2/CVC4
Settings

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* ∧ *body* → *post*
  - Replace calls with their specification
  - Discharge verification conditions to SMT solver

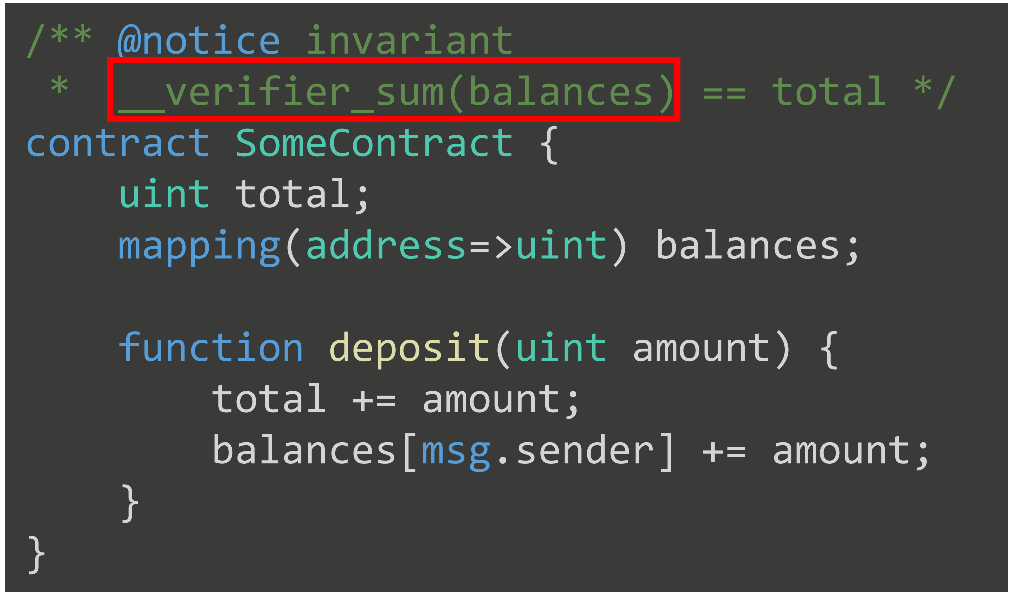


Challenges

SRI International®

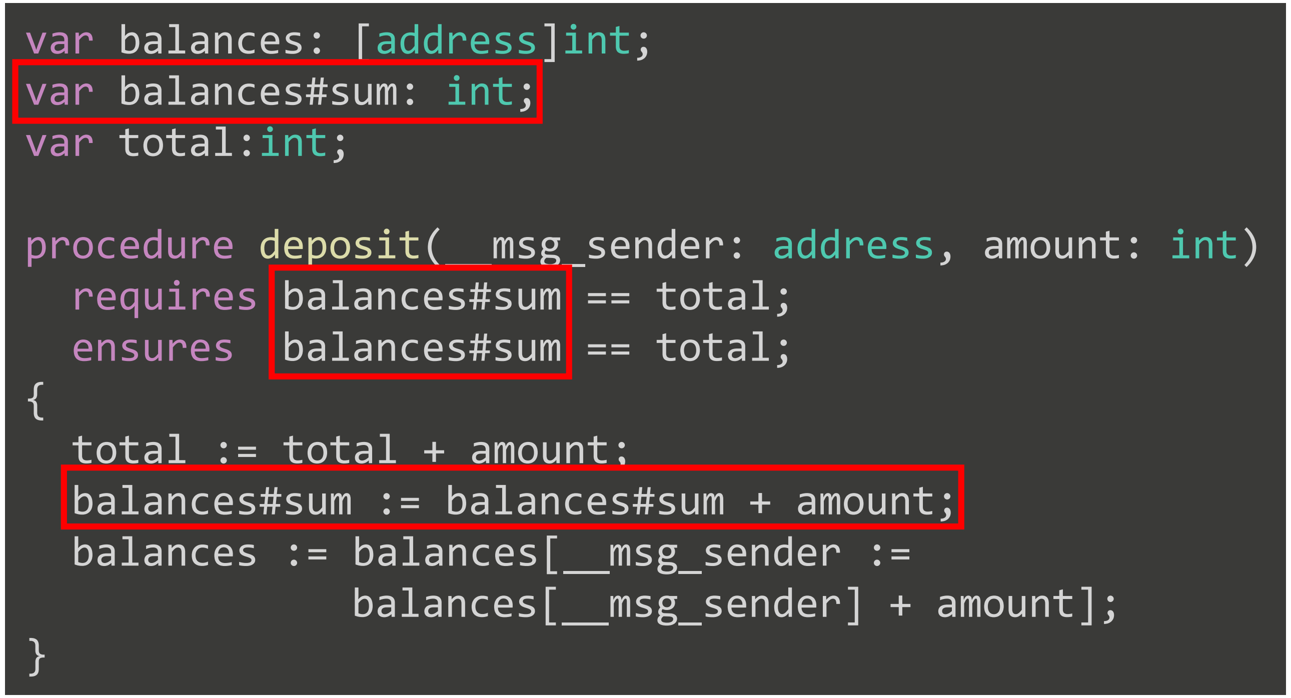
Sum Function

- Sum of mappings not possible in Solidity (nor in FOL)
- Use shadow variable



```
/** @notice invariant
 *  __verifier_sum(balances) == total */
contract SomeContract {
    uint total;
    mapping(address=>uint) balances;

    function deposit(uint amount) {
        total += amount;
        balances[msg.sender] += amount;
    }
}
```



```
var balances: [address]int;
var balances#sum: int;
var total:int;

procedure deposit(__msg_sender: address, amount: int)
  requires balances#sum == total;
  ensures  balances#sum == total;
{
  total := total + amount;
  balances#sum := balances#sum + amount;
  balances := balances[__msg_sender :=
              balances[__msg_sender] + amount];
}
```

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

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);
```

✓

Complex Data Types and Memory Models

```
struct S {
  int x;
  T[] ts;
}
```

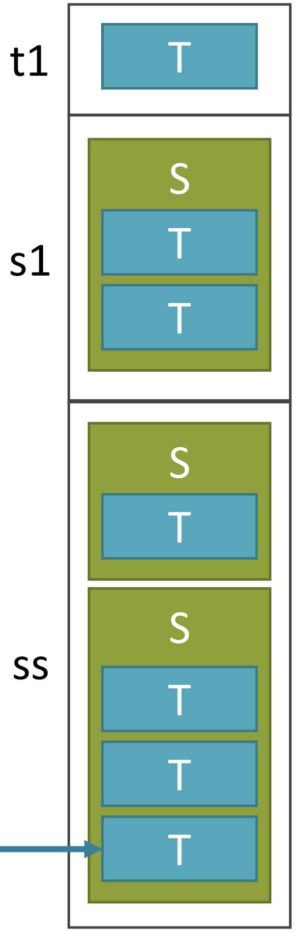
```
struct T {
  int z;
}
```

- Storage: non-aliasing

```
contract C {
   T   t1;
   S   s1;
   S[] ss;
}
```

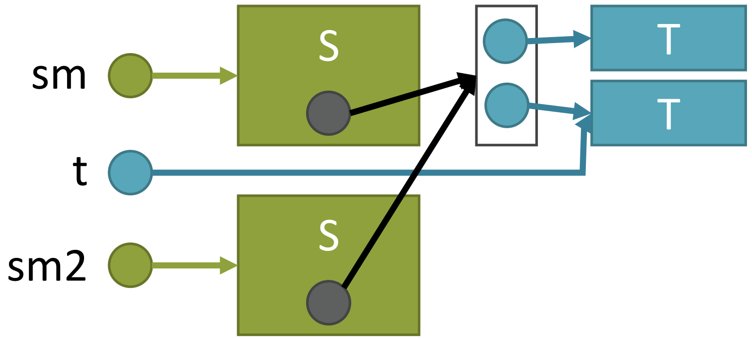
- Local storage pointers

```
function f() public {
  T storage tp = ss[1].ts[2];
  g(tp);
}
function g(T storage t) internal {
  t.z = 5;
}
```



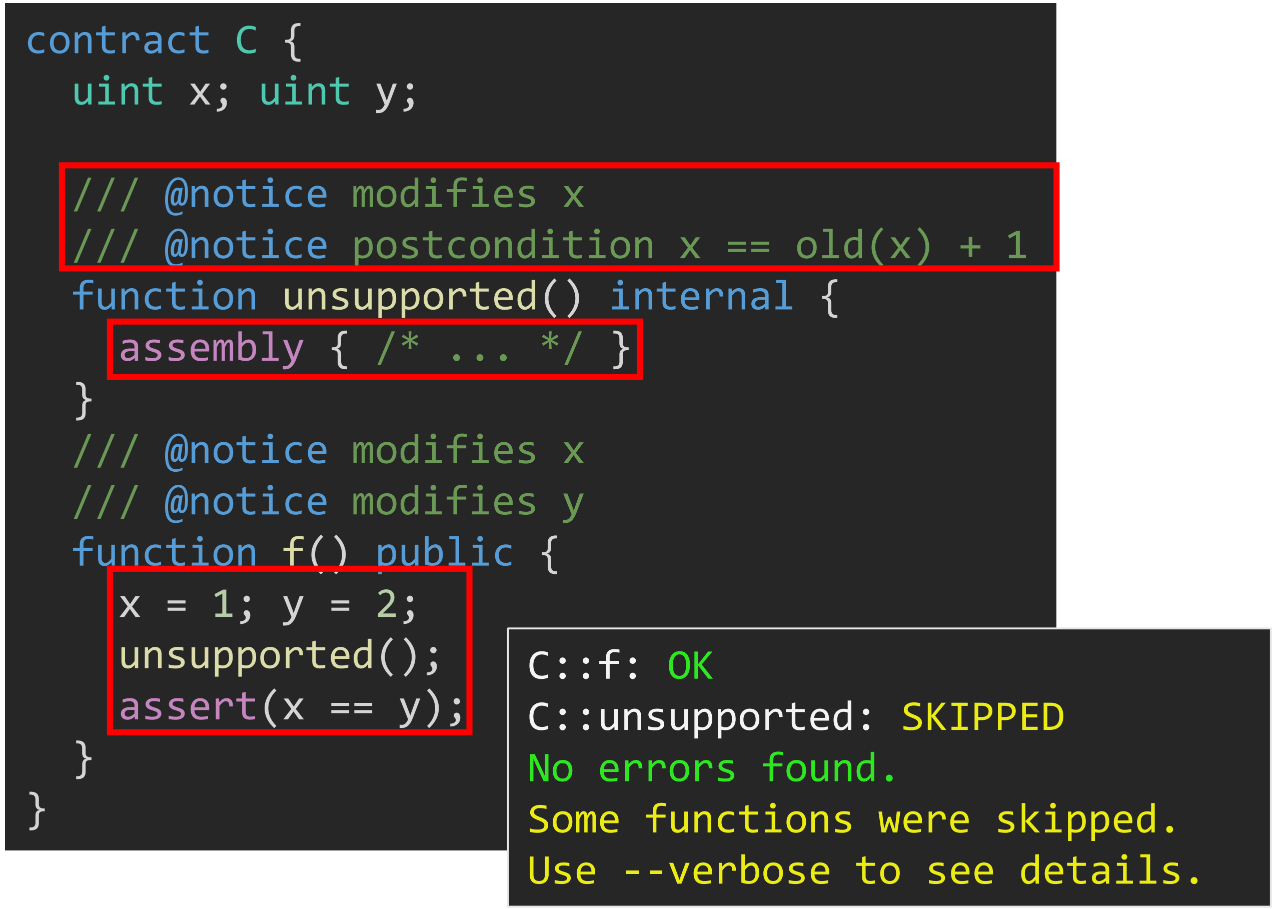
- Memory: all references

```
function f() public pure {
   S memory sm = S(1, new T[](2));
   T memory t = sm.ts[1];
   S memory sm2 = S(2, sm.ts);
}
```



Implementation

- Working inside compiler
  - Reuse modules
    - E.g., parsing specifications
  - Internal AST with extra information
- Testing
  - Test cases with “main” function
  - Verifier vs. test network
  - Language coverage on syntax tests
- Unsupported elements
  - Partial results



```
contract C {
  uint x; uint y;

  /// @notice modifies x
  /// @notice postcondition x == old(x) + 1
  function unsupported() internal {
    assembly { /* ... */ }
  }
  /// @notice modifies x
  /// @notice modifies y
  function f() public {
    x = 1; y = 2;
    unsupported();
    assert(x == y);
  }
}
```

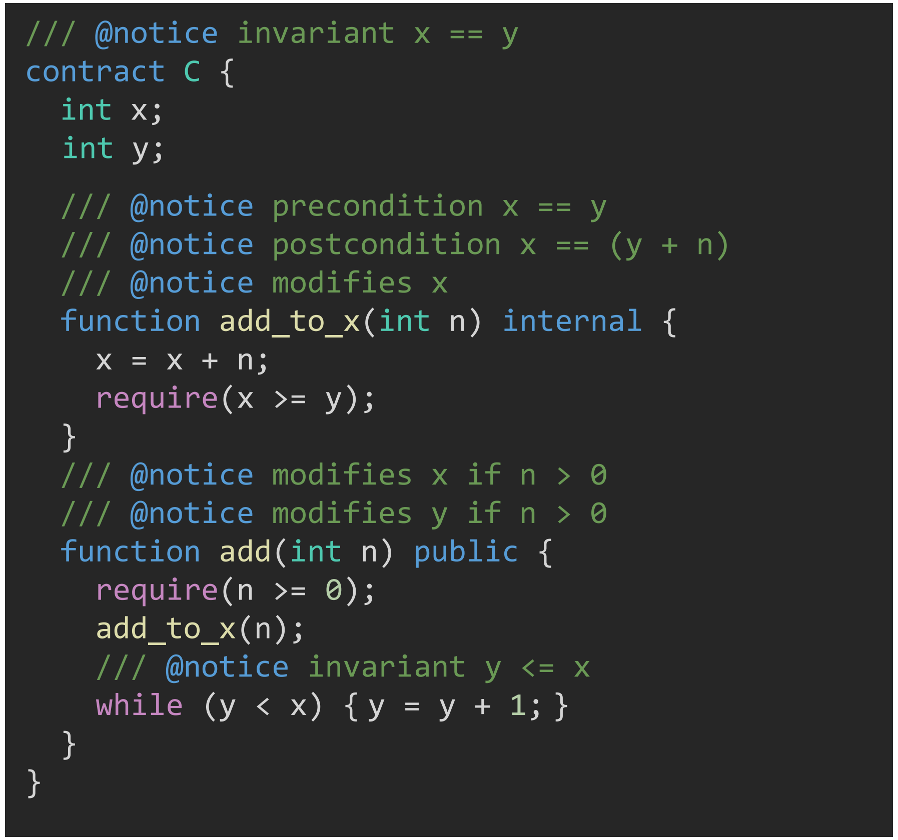
```
C::f: OK
C::unsupported: SKIPPED
No errors found.
Some functions were skipped.
Use --verbose to see details.
```

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
- Future work
  - Cover missing Solidity features
  - Translation validation
  - Invariant inference
  - Implicit specifications
  - Experiments (with/without specs)



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

  /// @notice precondition x == y
  /// @notice postcondition x == (y + n)
  /// @notice modifies x
  function add_to_x(int n) internal {
    x = x + n;
    require(x >= y);
  }
  /// @notice modifies x if n > 0
  /// @notice modifies y if n > 0
  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

hajduakos.github.io

csl.sri.com/users/dejan/

SRI International®