SolCMC: Solidity Compiler's Model Checker

Leo Alt[1], Martin Blicha[2], Antti Hyvärinen[2], Natasha Sharygina[2]

Ethereum Foundation[1] – Formal Verification Team
University of Lugano[2], Switzerland – Software Verification Lab

leonardoalt
leo@ethereum.org
leonardoalt

- Solidity: main language for Ethereum smart contracts
- 4.7 million compiler downloads in July
- ~200 billion USD held by smart contracts

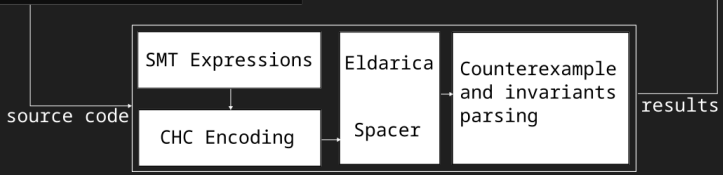
```
contract Token {
    mapping (address => uint) public balances;

    constructor(uint amt) {
        balances[msg.sender] = amt;
    }

    function transfer(address to, uint amt) external {
        balances[msg.sender] -= amt;
        balances[to] += amt;
    }
}
```

```sol
contract Token {
    mapping (address => uint) public balances;
    uint totalSupply;

    constructor(uint amt) {
        balances[msg.sender] = amt;
        totalSupply = amt;
    }

    function burn(uint amt) external {
        balances[msg.sender] -= amt;
        totalSupply -= amt;
    }

    function transfer(address to, uint amt) external {
        uint supply = totalSupply;

        balances[msg.sender] -= amt;
        balances[to] += amt;

        Receiver(to).receiveToken(msg.sender, amt);

        assert(totalSupply == supply);
    }
}
```

```
leo@horn ~/devel/cav_reentrancy master ± forge build
[⠒] Compiling...
[⠢] Compiling 9 files with 0.8.15
[⠆] Solc 0.8.15 finished in 1.52s
Compiler run successful (with warnings)
warning[6328]: Warning: CHC: Assertion violation happens here.
Counterexample:
totalSupply = 0
to = 0x07ba
amt = 0
supply = 39

Transaction trace:
Token.constructor(39){ msg.sender: 0x1adf }
State: totalSupply = 39
Token.transfer(0x07ba, 0){ msg.sender: 0x07b9 }
    Receiver(to).receiveToken(msg.sender, amt) -- untrusted external call, synthesized as:
        Token.burn(39){ msg.sender: 0x1adf } -- reentrant call
  --> src/Token.sol:30:9:
   |
30 |         assert(totalSupply == supply);
   |         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

```
contract Token {
    mapping (address => uint) public balances;
    uint totalSupply;

    constructor(uint amt) {
        balances[msg.sender] = amt;
        totalSupply = amt;
    }

    function burn(uint amt) external {
        balances[msg.sender] -= amt;
        totalSupply -= amt;
    }

    function transfer(address to, uint amt) external {
        uint supply = totalSupply;

        balances[msg.sender] -= amt;
        balances[to] += amt;

        Receiver(to).receiveToken(msg.sender, amt);

        assert(totalSupply == supply);
    }
}
```

```
leo@horn ~/devel/fet/reentrancy/ master ± forge build
[⠒] Compiling...
[⠒] Compiling 9 files with 0.8.15
[⠢] Solc 0.8.15 finished in 1.52s
Compiler run successful (with warnings)
warning[6328]: Warning: CHC: Assertion violation happens here.
Counterexample:
totalSupply = 0
to = 0x07ba
amt = 0
supply = 39

Transaction trace:
Token.constructor(39)( msg.sender: 0x1adf )
State: totalSupply = 39
Token.transfer(0x07ba, 0)( msg.sender: 0x07b9 )
    Receiver(to).receiveToken(msg.sender, amt) -- untrusted external call, synthesized as:
        Token.burn(39)( msg.sender: 0x1adf ) -- reentrant call
  --> src/Token.sol:30:9:
   |
30 |         assert(totalSupply == supply);
   |         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```



More features

- Contract invariants and reentrancy properties
- Several verification targets
- Up-to-date with the latest Solidity versions
- External calls to unknown code, unbounded txs, loops, function abstraction
- Binaries for Linux, OSX, Windows, WebAssembly (runs on browsers too!)

Partial support / Under dev	No support
Inline assembly	Memory/storage aliasing
Ext calls to trusted code	Low level opcodes
Report loop invariants	Most of inline assembly
Selfdestruct	Function pointers

Horn Clause Solvers for Program Verification

Nikolaj Bjørner[1(✉)], Arie Gurfinkel[2],
Ken McMillan[1], and Andrey Rybalchenko[3]

[1] Microsoft Research, Redmond, USA
nbjorner@microsoft.com
[2] Software Engineering Institute, Pittsburgh, USA
[3] Microsoft Research, Cambridge, UK

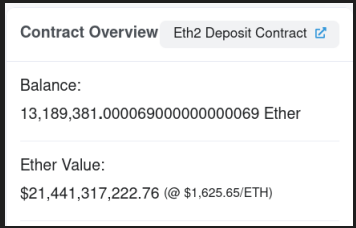
Accurate Smart Contract Verification Through Direct Modelling

Matteo Marescotti[1(✉)], Rodrigo Otoni[1(✉)], Leonardo Alt[2(✉)],
Patrick Eugster[1(✉)], Antti E. J. Hyvärinen[1(✉)], and Natasha Sharygina[1(✉)]

[1] Università della Svizzera italiana, Lugano, Switzerland
{matteo.marescotti,rodrigo.benedito.otoni,patrick.thomas.eugster,
antti.hyvaerinen,natasha.sharygina}@usi.ch
[2] Ethereum Foundation, Zug, Switzerland
leo@ethereum.org

Deposit Contract

- Functional correctness of the deposit function
- Nonlinear Horn clauses, Arrays, ADTs, LIA, BV
- Eldarica proves the assertion in 22.4s



ERC777 – OpenZeppelin Implementation

- Contract invariant: total supply does not change after a transfer
- 1200 Solidity LOC and several complex language features
- Eldarica proves in ~3 minutes that the property does not hold
- Synthesized malicious reentrant call and 21 function calls

```
import "./contracts/token/ERC777/ERC777.sol";

contract ERC777Property is ERC777 {
    constructor(
        address[] memory defaultOperators_,
        uint amt_
    ) ERC777("ERC777", "E7", defaultOperators_) {
        _mint(msg.sender, amt_, "", "");
    }

    function transfer(address recipient, uint256 amount)
        public override returns (bool) {

        uint prevSupply = totalSupply();
        bool result = ERC777.transfer(recipient, amount);
        uint postSupply = totalSupply();

        assert(prevSupply == postSupply);

        return result;

    }
}
```

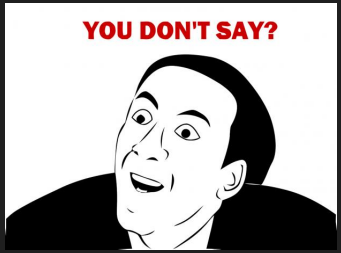
ERC777 – OpenZeppelin Implementation

- Protect external functions from reentrancy
- Mutex modifier
- Eldarica proves in 26.2 seconds that the property holds

```
import "./contracts/token/ERC777/ERC777_mutex.sol";

contract ERC777Property is ERC777 {
    constructor(
        address[] memory defaultOperators_,
        uint amt_
    ) ERC777("ERC777", "E7", defaultOperators_) {
        ERC777._mint(msg.sender, amt_, "", "");
    }

    function transfer(address recipient, uint256 amount)
        mutex public override returns (bool) {

        uint prevSupply = totalSupply();
        bool result = ERC777.transfer(recipient, amount);
        uint postSupply = totalSupply();

        assert(prevSupply == postSupply);

        return result;

    }
}
```

Conclusions

maintaining a model checker inside a production compiler is a lot of work



it is possible to verify unmodified real world contracts and Horn solvers are doing great!

SolCMC is also pushing Horn solvers

- **LRA-TS-par**allel, **T**ransition **S**ystems (**LRA-TS-par**)
- **A**lgebraic **D**ata-**T**ypes, **Nonlin**ear clauses (**ADT-Nonlin**)
- **LIA**, **Arrays** and **non-rec**ursive ADT, **Nonlin**ear clauses (**LIA-Nonlin-Arrays-nonrecADT**)

New in 2022

New Benchmarks

- **Solidity CHC Benchmarks** (Thanks to *Leonardo Alt*)
  Nonlinear clauses; LIA + Arrays + non-recursive ADTs
  Source: Solidity SMTChecker, Eth2 Deposit Contract, and OpenZeppelin's ERC777 implementation

https://github.com/leonardoalt/chc_benchmarks_solidity

Chat to us!

- University of Lugano has open positions (PhD/Post-doc)
- Ethereum Foundation FV related projects:
  - SolCMC – Solidity Model Checking
    https://docs.soliditylang.org/en/latest/smtchecker.html
  - hevm – EVM Symbolic Execution
    https://fv.ethereum.org/2020/07/28/symbolic-hevm-release/
  - Act – Smart Contract Specification + Coq/SMT/EVM
    https://fv.ethereum.org/2021/08/31/act-0.1
  - Zero Knowledge Circuits (SMT Workshop)
    https://github.com/lvella/polynomial-solver
  - SolSMT solver (SMT-Comp – SMT Workshop)

Thank you!