



Formally Verifying Ethereum Smart Contracts by Overwhelming Horn Solvers

Leo Alt

Ethereum Foundation

🔗 leonardoalt

✉ leo@ethereum.org

🐦 leonardoalt

<https://docs.soliditylang.org/en/v0.8.9/smtchecker.html>

Used to be: SMT-based assertion checker

Now wants to be: Horn-based model checker

Embedded in the Solidity compiler

Horn Clause Solvers for Program Verification

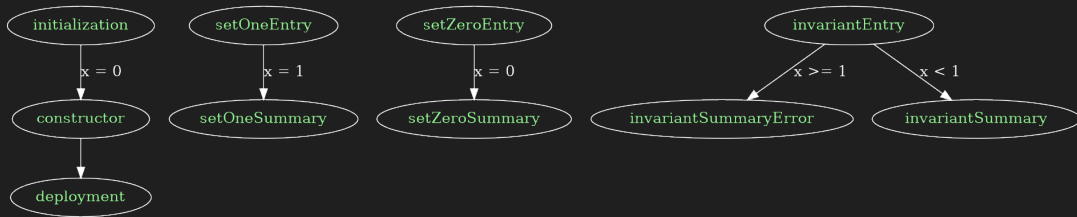
Nikolaj Bjørner¹ (✉), Arie Gurfinkel²,
Ken McMillan¹, and Andrey Rybalchenko³

¹ Microsoft Research, Redmond, USA
`nbjorner@microsoft.com`

² Software Engineering Institute, Pittsburgh, USA

³ Microsoft Research, Cambridge, UK

Control Flow Graph



$\text{error} = 0 \wedge \mathbf{x} = 0 \implies \text{constructor}(\text{error}, \mathbf{x})$

$\text{constructor}(\text{error}, \mathbf{x}) \implies \text{deployment}(\text{error}, \mathbf{x})$

$\text{error} = 0 \wedge \mathbf{x} = \mathbf{x}' \implies \text{setOneEntry}(\text{error}, \mathbf{x}, \mathbf{x}')$

$\text{setOneEntry}(\text{error}, \mathbf{x}, \mathbf{x}') \implies \text{setOneSummary}(\text{error}, \mathbf{x}, 1)$

$\text{error} = 0 \wedge \mathbf{x} = \mathbf{x}' \implies \text{setZeroEntry}(\text{error}, \mathbf{x}, \mathbf{x}')$

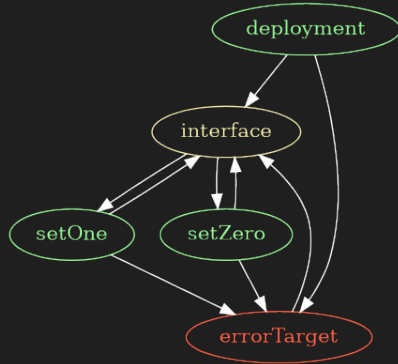
$\text{setZeroEntry}(\text{error}, \mathbf{x}, \mathbf{x}') \implies \text{setZeroSummary}(\text{error}, \mathbf{x}, 0)$

$\text{error} = 0 \wedge \mathbf{x} = \mathbf{x}' \implies \text{invariantEntry}(\text{error}, \mathbf{x}, \mathbf{x}')$

$\text{invariantEntry}(\text{error}, \mathbf{x}, \mathbf{x}') \wedge \mathbf{x} \geq 1 \wedge \text{errorCode} = 1 \implies \text{invariantSummary}(\text{errorCode}, \mathbf{x}, \mathbf{x}')$

$\text{invariantEntry}(\text{error}, \mathbf{x}, \mathbf{x}') \wedge \mathbf{x} < 1 \implies \text{invariantSummary}(\text{error}, \mathbf{x}, 1)$

Smart Contract Lifetime



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`

$\text{deployment}(\text{error}, x) \wedge \text{error} = 0 \implies \text{interface}(x)$

$\text{deployment}(\text{error}, x) \wedge \text{error} > 0 \implies \text{errorTarget}(\text{error})$

$\text{interface}(x) \wedge \text{setOneSummary}(\text{error}, x, x') \wedge \text{error} = 0 \implies \text{interface}(x')$

$\text{interface}(x) \wedge \text{setOneSummary}(\text{error}, x, x') \wedge \text{error} > 0 \implies \text{errorTarget}(\text{error})$

$\text{interface}(x) \wedge \text{setZeroSummary}(\text{error}, x, x') \wedge \text{error} = 0 \implies \text{interface}(x')$

$\text{interface}(x) \wedge \text{setZeroSummary}(\text{error}, x, x') \wedge \text{error} > 0 \implies \text{errorTarget}(\text{error})$

$\text{interface}(x) \wedge \text{invariantSummary}(\text{error}, x, x') \wedge \text{error} = 0 \implies \text{interface}(x')$

$\text{interface}(x) \wedge \text{invariantSummary}(\text{error}, x, x') \wedge \text{error} > 0 \implies \text{errorTarget}(\text{error})$

Counterexample

Warning: CHC: Assertion violation happens here.

Counterexample:

x = 1

Transaction trace:

BinaryMachine.constructor()

State: x = 0

BinaryMachine.setOne()

State: x = 1

BinaryMachine.invariant()

--> binary_machine.sol:13:3:

|

13

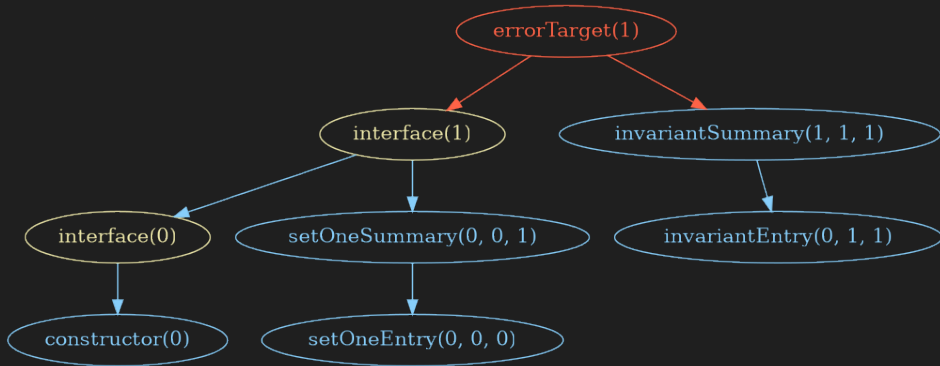
|

assert(x < 1);

|

AAAAAAAAAAAAAAAA

Counterexample graph



Invariant

```
Info: Contract invariant(s) for binary_machine.sol:BinaryMachine:  
!(x >= 2)
```

More inductive invariants

interface(e, x) = $x < 2$

invariantSummary(e, x, x') = $x' = 0 \vee x' = 1$

constructorSummary(e, x) = $e = 0 \wedge x' = 0$

setOneSummary(e, x, x') = $e = 0 \wedge x' = 1$

setZeroSummary(e, x, x') = $e = 0 \wedge x' = 0$

External calls & Reentrancy

```
interface Unknown {  
    function callMe() external;  
}  
  
contract ExtCall {  
    uint x;  
    function setX(uint _x) public { x = _x; }  
    function xMut(Unknown _u) public {  
        uint xPrev = x;  
        _u.callMe();  
        assert(xPrev == x);  
    }  
}
```

New CHC magic rules that encode reentrancy and help with synthesis of externally called unknown functions

$$\mathbf{error} = \mathbf{0} \implies \mathbf{nondetInterface}(\mathbf{error}, \mathbf{x}, \mathbf{x})$$

$$\mathbf{error} = \mathbf{0} \wedge \mathbf{nondetInterface}(\mathbf{error}, \mathbf{x}, \mathbf{x}') \wedge \mathbf{setX}(\mathbf{error}', \mathbf{x}', \mathbf{x}'') \implies \mathbf{nondetInterface}(\mathbf{error}', \mathbf{x}, \mathbf{x}'')$$

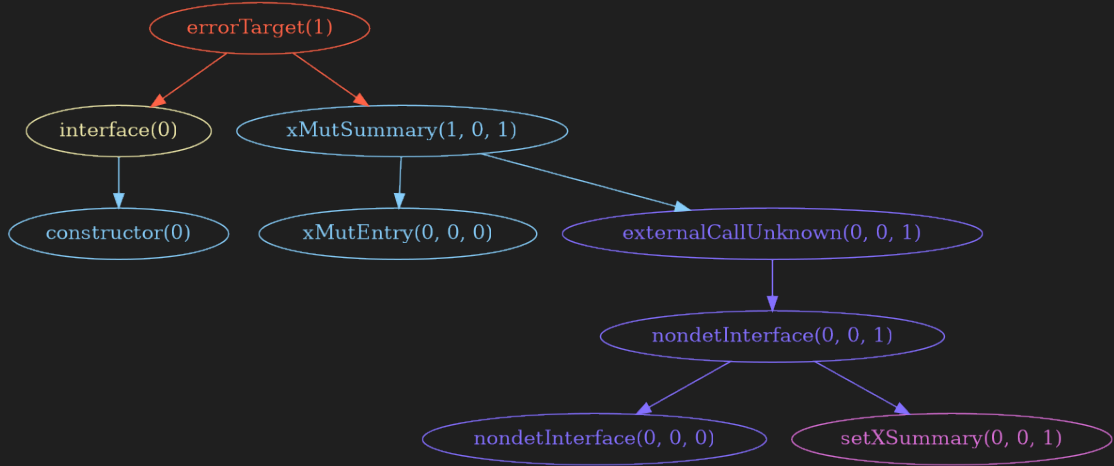
$$\mathbf{error} = \mathbf{0} \wedge \mathbf{nondetInterface}(\mathbf{error}, \mathbf{x}, \mathbf{x}') \wedge \mathbf{xMut}(\mathbf{error}', \mathbf{x}', \mathbf{x}'') \implies \mathbf{nondetInterface}(\mathbf{error}', \mathbf{x}, \mathbf{x}'')$$

Counterexample

```
Warning: CHC: Assertion violation happens here.
Counterexample:
x = 1
_u = 0
xPrev = 0

Transaction trace:
ExtCall.constructor()
State: x = 0
ExtCall.xMut(0)
    _u.callMe() -- untrusted external call, synthesized as:
        ExtCall.setX(1) -- reentrant call
--> ext_call.sol:11:3:
    |
11 |         assert(xPrev == x);
    |         ^^^^^^^^^^^^^^^^^^
```

Counterexample graph synthesizing external calls





```

interface Unknown {
    function callMe() external;
}

contract ExtCall {
    uint x;
    bool lock;

    modifier mutex {
        require(!lock);
        lock = true;
        _;
        lock = false;
    }

    function setX(uint _x) public mutex { x = _x; }
    function xMut(Unknown _u) public mutex {
        uint xPrev = x;
        _u.callMe();
        assert(xPrev == x);
    }
}

```

External call property

```
Info: Reentrancy property(ies) for ext_call.sol:ExtCall:  
((!lock || ((x' + ((- 1) * x)) = 0)) && (<errorCode> <= 0) && (lock' || !lock))  
<errorCode> = 0 -> no errors  
<errorCode> = 1 -> Assertion failed at assert(xPrev == x)
```

External call property

$$\text{externalCallUnknown}(e, x, \text{lock}, x', \text{lock}') = \text{lock} \implies x = x'$$

Nonlinear Horn clauses

- ◆ Queries
- ◆ Inheritance
- ◆ Deployment procedure
- ◆ Function calls
- ◆ Reentrancy / Unsafe external calls

Optimistic encoding

- ◆ Loops
- ◆ Nonlinear Integer Arithmetic
- ◆ BitVectors for bitwise operations
- ◆ SMT (tuples and arrays) for Solidity (arrays and mappings)
- ◆ Tuples for nonrecursive structs
- ◆ Heavy theory combination


```

contract ArrayStruct {
    struct T {
        uint y;
        uint[] a;
    }
    struct S {
        uint x;
        T[] t;
    }

    S[] s;

    function pushArray() public {
        uint sLen = s.length;
        s.push();
        s[sLen].x = sLen;
        s[sLen].t.push();
        s[sLen].t[0].y = 42;
        s[sLen].t[0].a.push(43);
    }

    function inv(uint i) public view {
        if (i < s.length)
            assert(s[i].x > i);
    }
}

```


Warning: CHC: Assertion violation happens here.

Counterexample:

```
s = [{x: 0, t: [{y: 42, a: [43]}]}]
```

```
i = 0
```

Transaction trace:

```
ArrayStruct.constructor()
```

```
State: s = []
```

```
ArrayStruct.pushArray()
```

```
State: s = [{x: 0, t: [{y: 42, a: [43]}]}]
```

```
ArrayStruct.inv(0)
```

```
--> array.sol:24:4:
```

```
24 | | assert(s[i].x > i);  
   | | ^^^^^^^^^^^^^^^^^
```

Uninterpreted Functions

(Array <input types as tuple> <return types as tuple>)

🔹 ABI encode/decode functions

🔹 Crypto functions

🔹 Blockchain state/balances

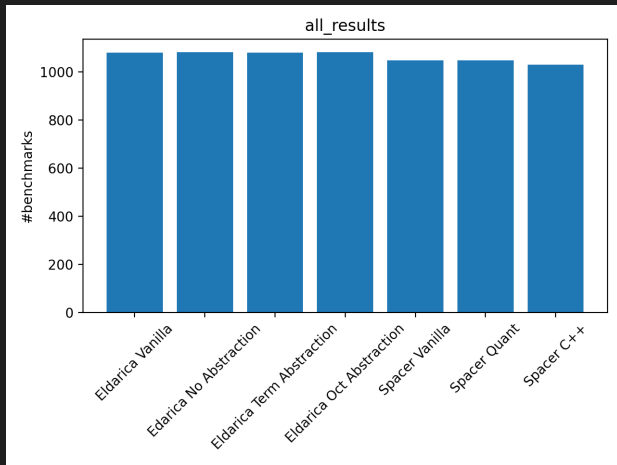
🔹 Global storage

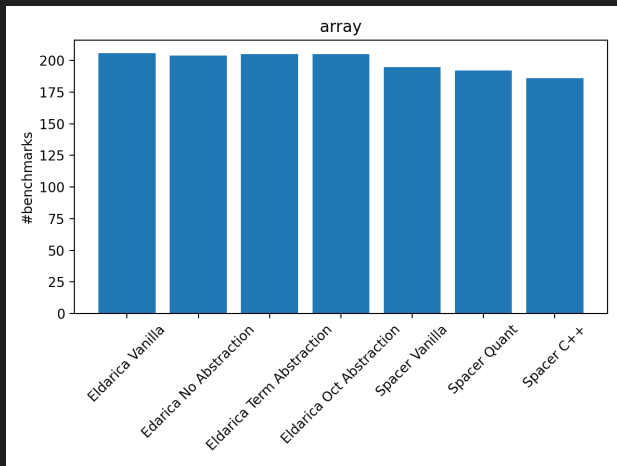
Solvers we run on our benchmarks

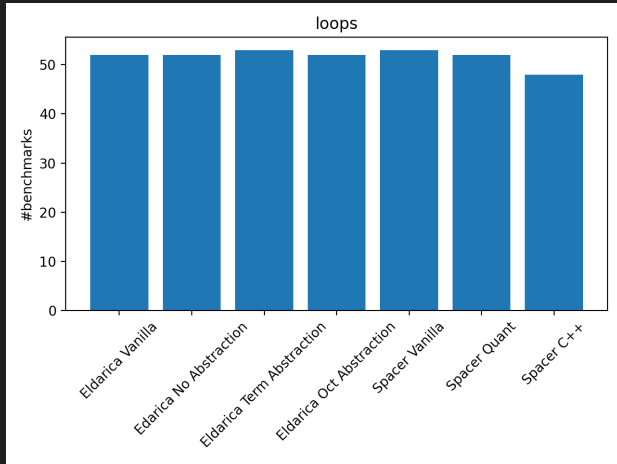
🔹 z3/Spacer

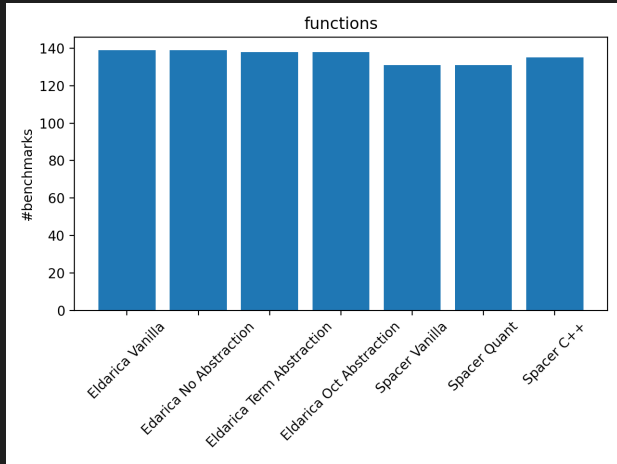
🔹 Eldarica

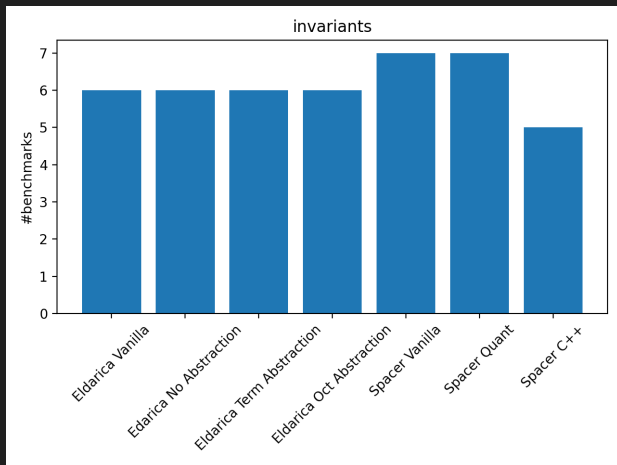
🔹 Golem – Work in progress

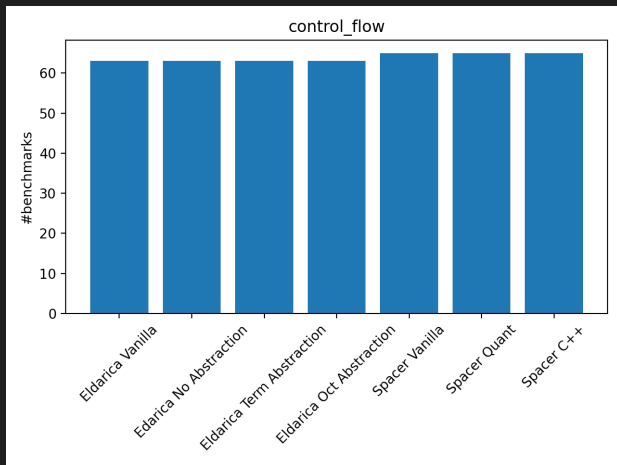


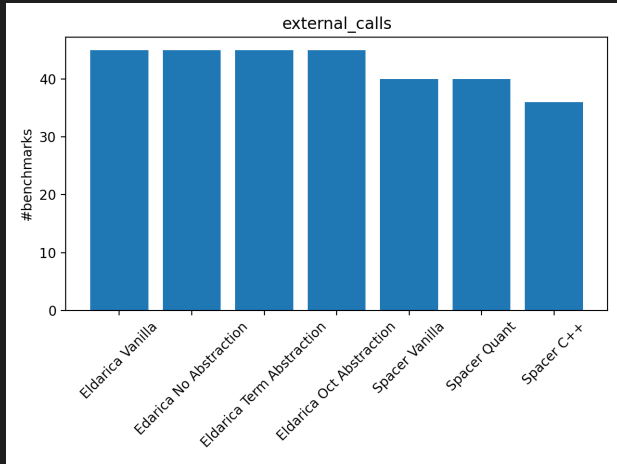


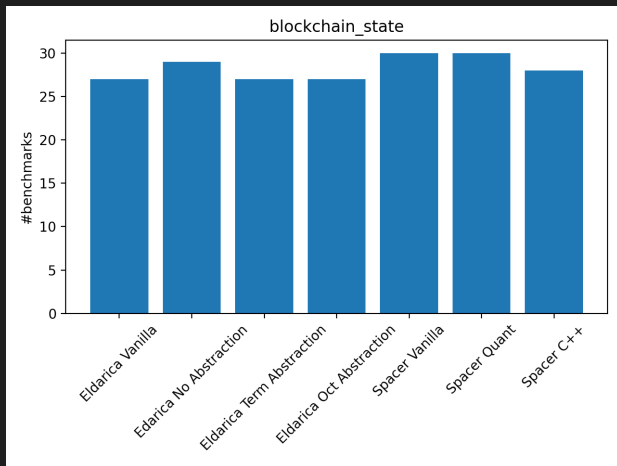


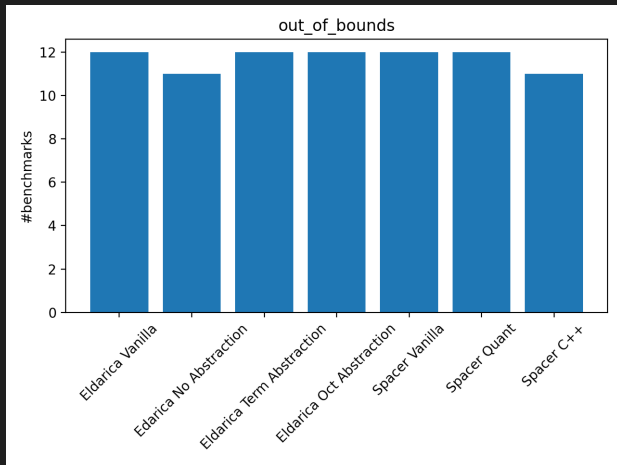


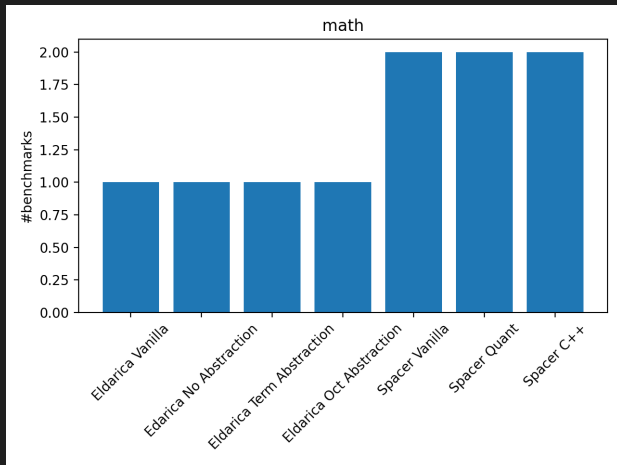


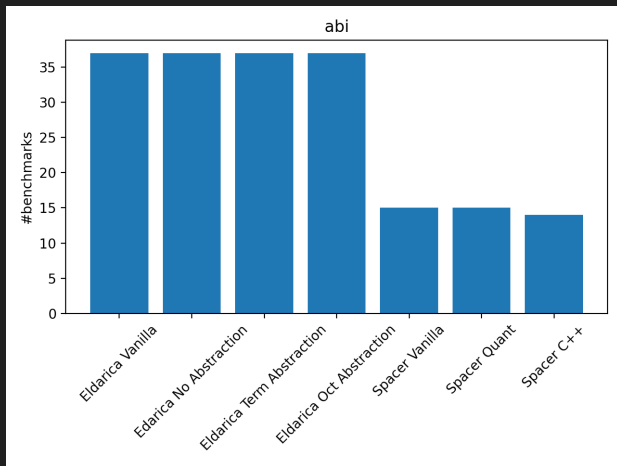


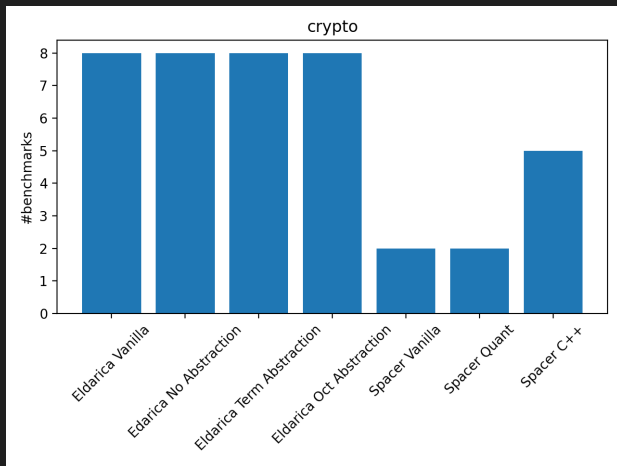













Eth 2 Deposit Contract

- ◆ Fundamental for Eth2
- ◆ Deposit on Eth1 to be a validator on Eth2
- ◆ Currently holds around 33.5 billion USD

End-to-End Formal Verification of Ethereum 2.0 Deposit Smart Contract

Daejun Park¹(✉) , Yi Zhang^{1,2}, and Grigore Rosu^{1,2}

¹ Runtime Verification, Inc., Urbana, IL, USA
daejun.park@runtimeverification.com

² University of Illinois at Urbana-Champaign, Urbana, IL, USA
{yzhng173,grosu}@illinois.edu

- ◆ Main property: sparse Merkle Tree is equivalent to the standard one
- ◆ Functional correctness of the deposit function

Same 7 configurations of Eldarica and Spacer

Timeout: 1 hour

- ❖ Eldarica Vanilla: OOM after 14.8 minutes
- ❖ Eldarica No Abstraction: OOM after 12.6 minutes
- ❖ Eldarica Term Abstraction: OOM after 18.5 minutes
- ❖ Eldarica Oct Abstraction: OOM after 18.6 minutes
- ❖ Spacer Vanilla: Answered Unknown after 1h
- ❖ Spacer Quant: Answered Unknown after 1h
- ❖ Spacer C++: Answered Unknown after 1h

Same 7 configurations of Eldarica and Spacer

Timeout: 1 hour

- ❖ Eldarica Vanilla: OOM after 34.3 minutes
- ❖ Eldarica No Abstraction: SAT after 38s!
- ❖ Eldarica Term Abstraction: OOM after 34.2 minutes
- ❖ Eldarica Oct Abstraction: OOM after 34.2 minutes
- ❖ Spacer Vanilla: Answered Unknown after 1h
- ❖ Spacer Quant: Answered Unknown after 33.8 minutes
- ❖ Spacer C++: Answered Unknown after 1h

How about even larger problems?

ERC-777 OpenZeppelin Implementation

- ◆ 1.2k lines of Solidity code
- ◆ 8 files
- ◆ Assembly, inheritance, strings, arrays, nested mappings, loops, hash, external calls
- ◆ Property: transfer hooks do not change the total supply
- ◆ SMTLib2 instance of 121MB, with 1231 rules

```

function _callTokensReceived(
    address operator,
    address from,
    address to,
    uint256 amount,
    bytes memory userData,
    bytes memory operatorData,
    bool requireReceptionAck
) private {
    uint prevTotal = _totalSupply;

    address implementer = _ERC1820_REGISTRY.getInterfaceImplementer(to, _TOKENS_RECIPIENT_INTERFACE_HASH);
    if (implementer != address(0)) {
        IERC777Recipient(implementer).tokensReceived(operator, from, to, amount, userData, operatorData);
    } else if (requireReceptionAck) {
        require(!to.isContract(), "ERC777: token recipient contract has no implementer for ERC777TokensRecipient");
    }

    assert(_totalSupply == prevTotal);
}

```

6 configurations of Eldarica and Spacer

Timeout: 1 hour

❖ Eldarica Vanilla: UNSAT after 82s!

❖ Eldarica No Abstraction: UNSAT after 56.6s!

❖ Eldarica Term: UNSAT after 81.5s!

❖ Eldarica Oct: UNSAT after 56.1s!

❖ Spacer Vanilla: Answered Unknown after 5.2s

❖ Spacer Quant: Answered Unknown after 5.2s

ERC-777 OpenZeppelin Implementation

Property: transfer hooks do not change the total supply

- ✧ UNSAT: total supply can be modified via reentrancy

- ✧ Counterexample with 113 nodes and 116 edges

- ✧ Use a mutex to forbid reentrancy

6 configurations of Eldarica and Spacer

Timeout: 1 hour

◆ Eldarica Vanilla: SAT after 128.1s!

◆ Eldarica No Abstraction: SAT after 85.3s!

◆ Eldarica Term: SAT after 128.1s!

◆ Eldarica Oct: SAT after 96s!

◆ Spacer Vanilla: Answered Unknown after 5.2s

◆ Spacer Quant: Answered Unknown after 5.5s

Why did Eldarica perform better than Spacer
on the larger instances?

<https://github.com/Z3Prover/z3/issues/5049>

agurfinkel commented on Feb 23

Collaborator



While this is reasonable way to encode heap, the current Horn solver does not support ADTs sufficiently well. It would be better to have a more direct encoding, by reducing ADTs (since they are not recursive) before producing Horn clauses.


<https://github.com/uuverifiers/eldarica/issues/33>

leonardoalt commented on Jan 4

Hi there :)
I just tried Eldarica on one of my benchmarks and got

```
Theories: ADT(tx_type), ADT(bytes_tuple), SimpleArray(1)
(error "Combination of theories is not supported")
```

Are there any plans to support that?
Here's the smt2 file if anyone is interested:
<https://gist.github.com/leonardoalt/d797eca839f14bb1997de7c5a6496a83>

 **pruemmer** commented on Jan 6

Hi Leonardo,

yes, we are working on a new array solver that will be able to handle this combination as well. Should have something running by the next CHC-COMP ;-)

Best wishes, Philipp

<https://github.com/uuverifiers/eldarica/issues/33>

✉ **pruemmer** commented on Mar 7

Collaborator



The last master should now support this combination (arrays + ADTs), although the functionality is not much tested yet.



<https://gitter.im/chc-comp/Lobby>

<https://github.com/chc-comp>

Conclusions and Future Work

- ◆ Impressive work from both Spacer and Eldarica!
- ◆ Good news not only for smart contracts
- ◆ Direct modelling and optimistic encoding paid off
- ◆ Encoding has room for improvement
- ◆ Solver strategies

Thank you!