Formally Verifying Ethereum Smart Contracts by Overwhelming Horn Solvers

Leo Alt

Ethereum Foundation

leonardoalt
leo@ethereum.org
leonardoalt

```
contract BinaryMachine {
    uint x = 0;

    function setOne() public {
        x = 1;
    }

    function setZero() public {
        x = 0;
    }

    function invariant() public view {
        assert(x < 1);
    }
}
```

https://docs.soliditylang.org/en/v0.8.10/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

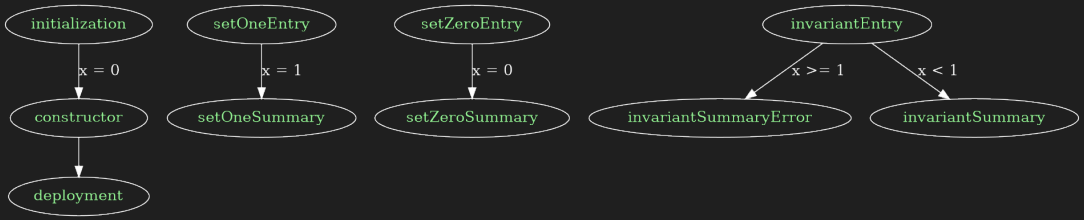
Nikolaj Bjørner[1(✉)], Arie Gurfinkel[2],
Ken McMillan[1], and Andrey Rybalchenko[3]

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

Horn solvers

- Actually use SMT solvers.
- Solve reachability on a transition system.
- Existential Positive Least Fixed-Point Logic (E+LFP) formulas match Hoare logic.
- Satisfiability in E+LFP = partial correctness of programs.
- Horn satisfiability = E+LFP satisfiability.

Control Flow Graph



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

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

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

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

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

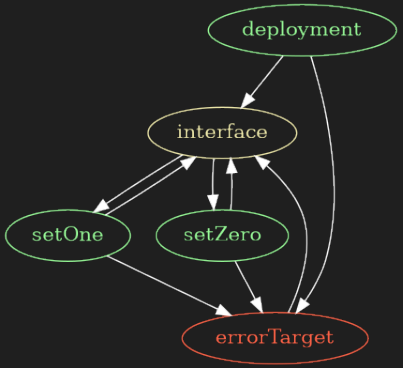
$$\mathbf{setZeroEntry}(\mathbf{error}, \mathbf{x}, \mathbf{x}') \implies \mathbf{setZeroSummary}(\mathbf{error}, \mathbf{x}, 0)$$

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

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

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

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

$$\mathbf{deployment}(\mathbf{error}, \mathbf{x}) \land \mathbf{error} = 0 \implies \mathbf{interface}(\mathbf{x})$$

$$\mathbf{deployment}(\mathbf{error}, \mathbf{x}) \land \mathbf{error} > 0 \implies \mathbf{errorTarget}(\mathbf{error})$$

$$\mathbf{interface}(\mathbf{x}) \land \mathbf{setOneSummary}(\mathbf{error}, \mathbf{x}, \mathbf{x}') \land \mathbf{error} = 0 \implies \mathbf{interface}(\mathbf{x}')$$

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

$$\mathbf{interface}(\mathbf{x}) \land \mathbf{setZeroSummary}(\mathbf{error}, \mathbf{x}, \mathbf{x}') \land \mathbf{error} = 0 \implies \mathbf{interface}(\mathbf{x}')$$

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

$$\mathbf{interface}(\mathbf{x}) \land \mathbf{invariantSummary}(\mathbf{error}, \mathbf{x}, \mathbf{x}') \land \mathbf{error} = 0 \implies \mathbf{interface}(\mathbf{x}')$$

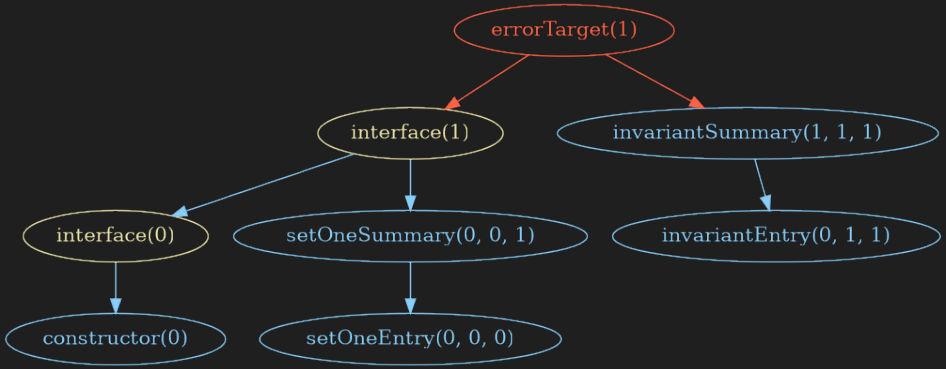
$$\mathbf{interface}(\mathbf{x}) \land \mathbf{invariantSummary}(\mathbf{error}, \mathbf{x}, \mathbf{x}') \land \mathbf{error} > 0 \implies \mathbf{errorTarget}(\mathbf{error})$$

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);
   |            ^^^^^^^^^^^^^
```

Counterexample graph



Correct code

```
contract BinaryMachine {
    uint x = 0;

    function setOne() public {
        x = 1;
    }

    function setZero() public {
        x = 0;
    }

    function invariant() public view {
        assert(x <= 2);
    }
}
```

Invariant

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

More inductive invariants

$$\mathbf{interface}(\mathbf{e}, \mathbf{x}) = \mathbf{x} < 2$$

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

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

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

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

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

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

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

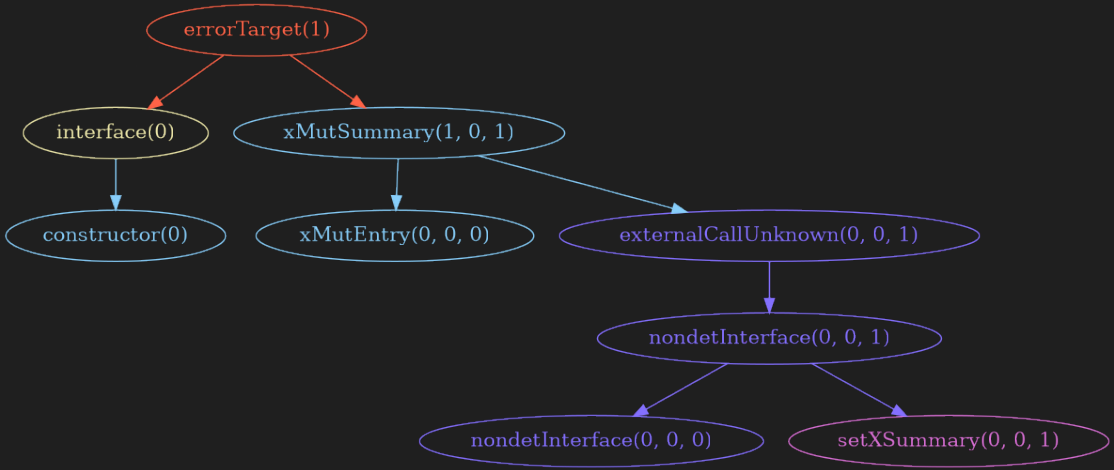
$$\mathbf{error} = 0 \wedge \mathbf{nondetInterface}(\mathbf{error}, \mathbf{x}, \mathbf{x}') \wedge \mathbf{xMut}(\mathbf{error}', \mathbf{x}', \mathbf{x}'') \implies \mathbf{nondetInterface}(\mathbf{error}', \mathbf{x}, \mathbf{x}'')$$

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




IT IS
MAGIC!
makeameme.org

```
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

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

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

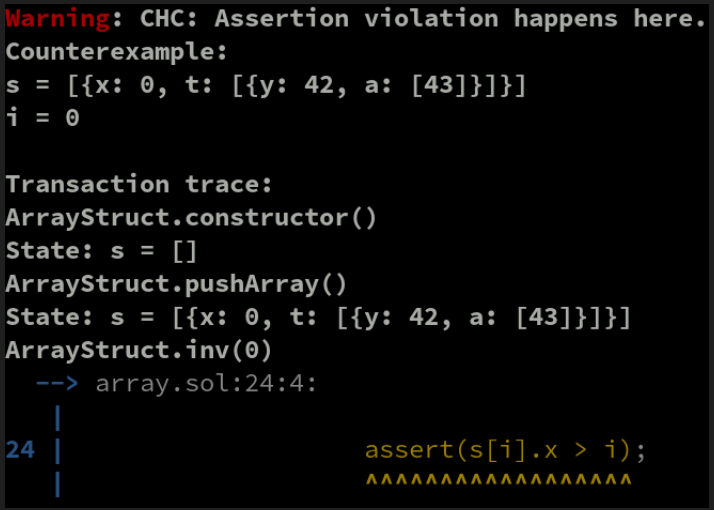
```
contract ArrayStruct {
    struct T {
        uint y;
        uint[] a;
    }
    struct S {
        uint x;
        T[] t;
    }

    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);
    }
}
```

```
(declare-datatypes
        ((|uint_array_tuple| 0))
        (((|uint_array_tuple| (|uint_array_tuple_accessor_array| (Array Int Int)) (|uint_array_tuple_accessor_length| Int))))
)
(declare-datatypes
        ((|struct ArrayStruct.T| 0))
        (((|struct ArrayStruct.T| (|struct ArrayStruct.T_accessor_y| Int) (|struct ArrayStruct.T_accessor_a| |uint_array_tuple|))))
)
(declare-datatypes
        ((|struct ArrayStruct.T_array_tuple| 0))
        (((|struct ArrayStruct.T_array_tuple|
                (|struct ArrayStruct.T_array_tuple_accessor_array| (Array Int |struct ArrayStruct.T|))
                (|struct ArrayStruct.T_array_tuple_accessor_length| Int)
        )))
)
(declare-datatypes
        ((|struct ArrayStruct.S| 0))
        (((|struct ArrayStruct.S| (|struct ArrayStruct.S_accessor_x| Int) (|struct ArrayStruct.S_accessor_t| |struct ArrayStruct.T_array_tuple|))))
)
(declare-datatypes
        ((|struct ArrayStruct.S_array_tuple| 0))
        (((|struct ArrayStruct.S_array_tuple|
                (|struct ArrayStruct.S_array_tuple_accessor_array| (Array Int |struct ArrayStruct.S|))
                (|struct ArrayStruct.S_array_tuple_accessor_length| Int)
        )))
)
```


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);
^^^^^^^^^^^^^^^^^

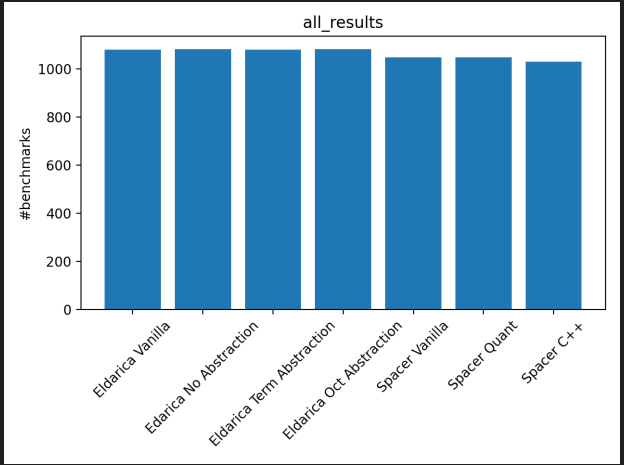
Uninterpreted Functions

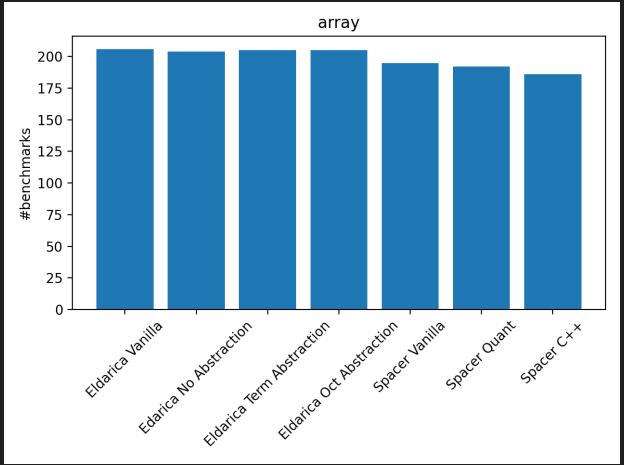
(Array <input types as tuple> <return types as tuple>)

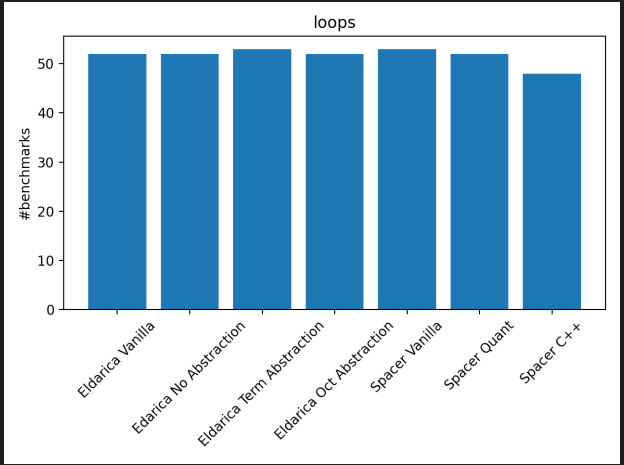
- ABI encode/decode functions
- Crypto functions
- Blockchain state/balances
- Global storage

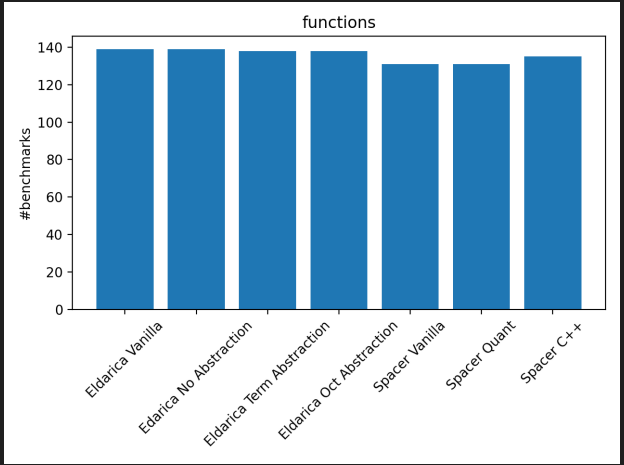
Solvers we run on our benchmarks

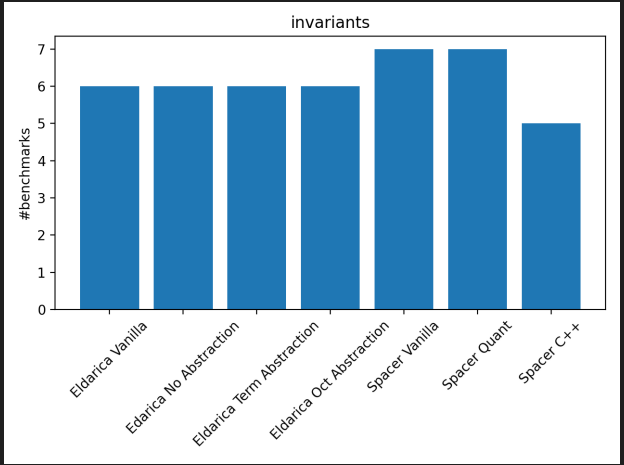
- z3/Spacer
- Eldarica
- Golem – Work in progress

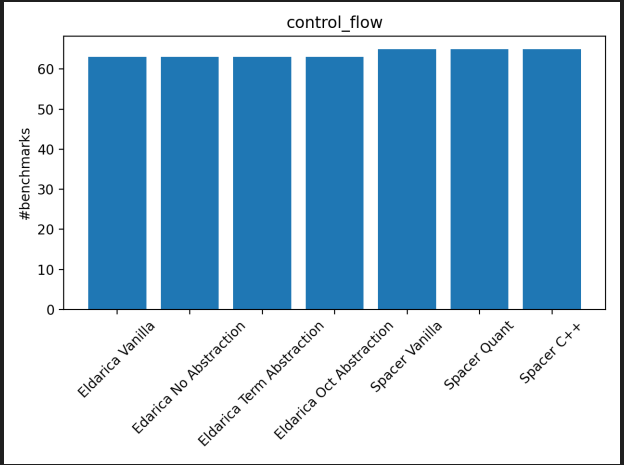

all_results
#benchmarks
0
200
400
600
800
1000
Eldarica Vanilla
Edarica No Abstraction
Eldarica Term Abstraction
Eldarica Oct Abstraction
Spacer Vanilla
Spacer Quant
Spacer C++

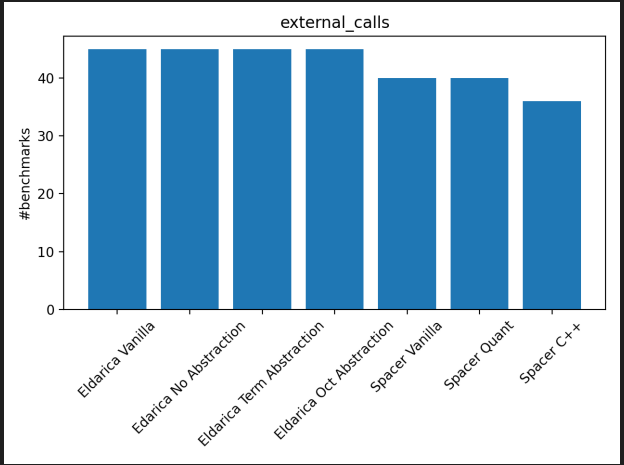

array
#benchmarks
0
25
50
75
100
125
150
175
200
Eldarica Vanilla
Edarica No Abstraction
Eldarica Term Abstraction
Eldarica Oct Abstraction
Spacer Vanilla
Spacer Quant
Spacer C++

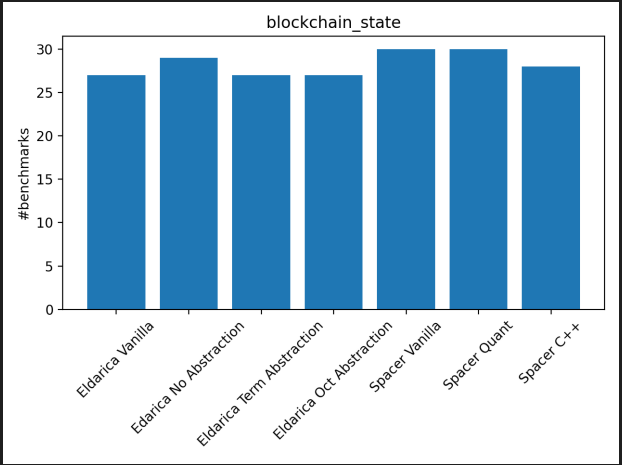

loops
#benchmarks
0
10
20
30
40
50
Eldarica Vanilla
Edarica No Abstraction
Eldarica Term Abstraction
Eldarica Oct Abstraction
Spacer Vanilla
Spacer Quant
Spacer C++

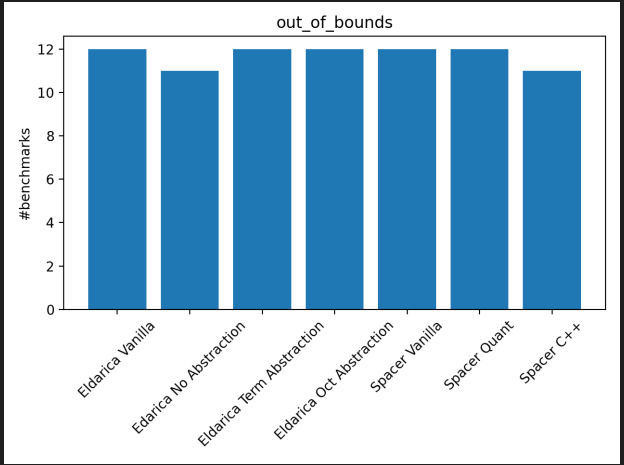

functions
#benchmarks
0
20
40
60
80
100
120
140
Eldarica Vanilla
Edarica No Abstraction
Eldarica Term Abstraction
Eldarica Oct Abstraction
Spacer Vanilla
Spacer Quant
Spacer C++

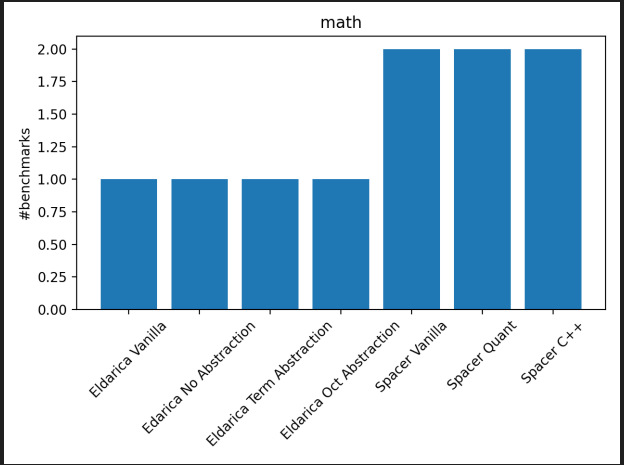

invariants
#benchmarks
0
1
2
3
4
5
6
7
Eldarica Vanilla
Edarica No Abstraction
Eldarica Term Abstraction
Eldarica Oct Abstraction
Spacer Vanilla
Spacer Quant
Spacer C++

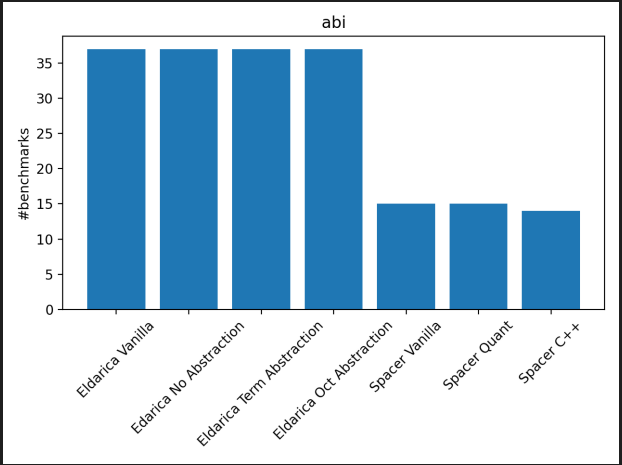

control_flow
#benchmarks
0
10
20
30
40
50
60
Eldarica Vanilla
Edarica No Abstraction
Eldarica Term Abstraction
Eldarica Oct Abstraction
Spacer Vanilla
Spacer Quant
Spacer C++

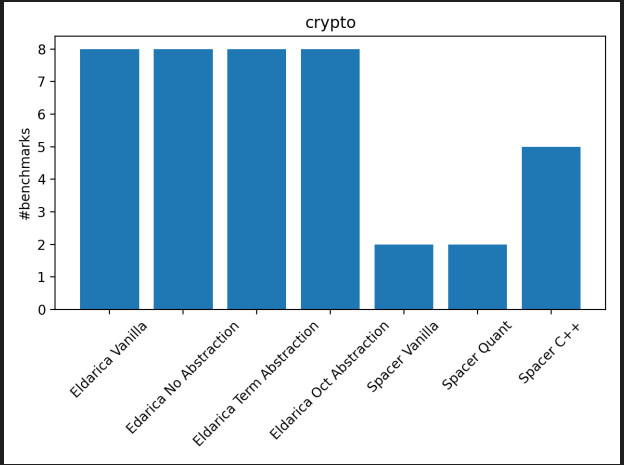

external_calls
#benchmarks
0
10
20
30
40
Eldarica Vanilla
Edarica No Abstraction
Eldarica Term Abstraction
Eldarica Oct Abstraction
Spacer Vanilla
Spacer Quant
Spacer C++


blockchain_state
#benchmarks
0
5
10
15
20
25
30
Eldarica Vanilla
Edarica No Abstraction
Eldarica Term Abstraction
Eldarica Oct Abstraction
Spacer Vanilla
Spacer Quant
Spacer C++


out_of_bounds
#benchmarks
0
2
4
6
8
10
12
Eldarica Vanilla
Edarica No Abstraction
Eldarica Term Abstraction
Eldarica Oct Abstraction
Spacer Vanilla
Spacer Quant
Spacer C++


math
#benchmarks
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Eldarica Vanilla
Edarica No Abstraction
Eldarica Term Abstraction
Eldarica Oct Abstraction
Spacer Vanilla
Spacer Quant
Spacer C++


abi
#benchmarks
0
5
10
15
20
25
30
35
Eldarica Vanilla
Edarica No Abstraction
Eldarica Term Abstraction
Eldarica Oct Abstraction
Spacer Vanilla
Spacer Quant
Spacer C++


crypto
#benchmarks
0
1
2
3
4
5
6
7
8
Eldarica Vanilla
Edarica No Abstraction
Eldarica Term Abstraction
Eldarica Oct Abstraction
Spacer Vanilla
Spacer Quant
Spacer C++

The solvers work well when solving complex instances in a smaller scale

How about larger problems?

Eth 2 Deposit Contract

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

```
function deposit(bytes calldata pubkey,
    bytes calldata withdrawal_credentials,
    bytes calldata signature,
    bytes32 deposit_data_root
) override external payable {
    // Extended ABI length checks since dynamic types are used.
    require(pubkey.length == 48, "DepositContract: invalid pubkey length");
    require(withdrawal_credentials.length == 32, "DepositContract: invalid withdrawal_credentials length");
    require(signature.length == 96, "DepositContract: invalid signature length");
    // Check deposit amount
    require(msg.value >= 1 ether, "DepositContract: deposit value too low");
    require(msg.value % 1 gwei == 0, "DepositContract: deposit value not multiple of gwei");
    uint deposit_amount = msg.value / 1 gwei;
    require(deposit_amount <= type(uint64).max, "DepositContract: deposit value too high");

    // Emit `DepositEvent` log
    bytes memory amount = to_little_endian_64(uint64(deposit_amount));
    emit DepositEvent(
        pubkey,
        withdrawal_credentials,
        amount,
        signature,
        to_little_endian_64(uint64(deposit_count))
    );
    // Compute deposit data root (`DepositData` hash tree root)
    bytes32 pubkey_root = sha256(abi.encodePacked(pubkey, bytes16(0)));
    bytes32 signature_root = sha256(abi.encodePacked(
        sha256(abi.encodePacked(signature[:64])),
        sha256(abi.encodePacked(signature[64:], bytes32(0)))
    ));
    bytes32 node = sha256(abi.encodePacked(
        sha256(abi.encodePacked(pubkey_root, withdrawal_credentials)),
        sha256(abi.encodePacked(amount, bytes24(0), signature_root))
    ));
    // Verify computed and expected deposit data roots match
    require(node == deposit_data_root, "DepositContract: reconstructed DepositData does not match supplied deposit_data_root");
    // Avoid overflowing the Merkle tree (and prevent edge case in computing `branch`)
    require(deposit_count < MAX_DEPOSIT_COUNT, "DepositContract: merkle tree full");
    // Add deposit data root to Merkle tree (update a single `branch` node)
    deposit_count += 1;
    uint size = deposit_count;
    for (uint height = 0; height < DEPOSIT_CONTRACT_TREE_DEPTH; height++) {
        if ((size & 1) == 1) {
            branch[height] = node;
            return;
        }
        node = sha256(abi.encodePacked(branch[height], node));
        size /= 2;
    }
    // As the loop should always end prematurely with the `return` statement,
    // this code should be unreachable. We assert `false` just to be safe.
    assert(false);
}
```

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

Daejun Park[1(✉)], 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

Replace

```
if ((size & 1) == 1)
```

by

```
if ((size % 2) == 1)
```

Logically equivalent, but bytecode and gas are different

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 18MB, 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

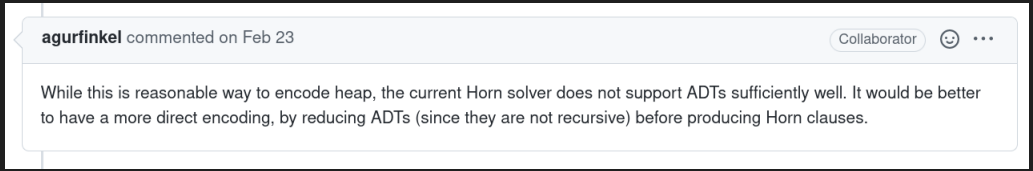
Timeout: 1 hour

- 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

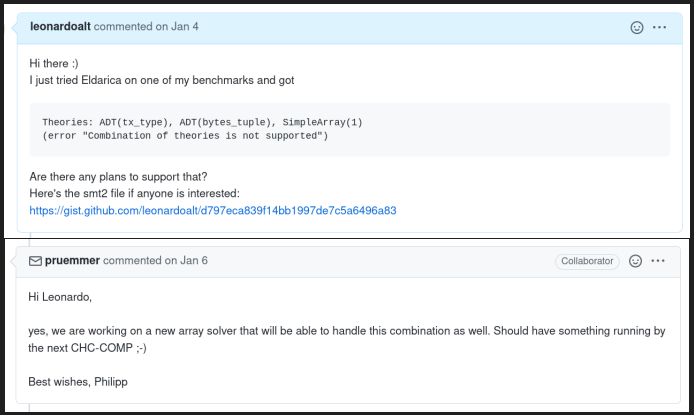
Why did Eldarica perform better than Spacer on the larger instances?

My guess: ADTs, we use them *a lot*

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



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

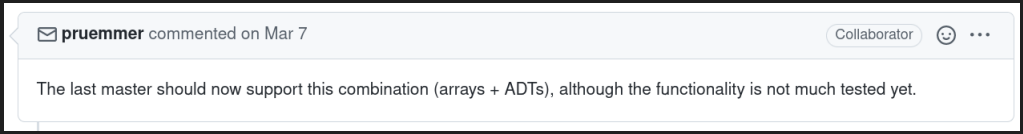
pruemmer commented on Jan 6 — Collaborator

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



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!