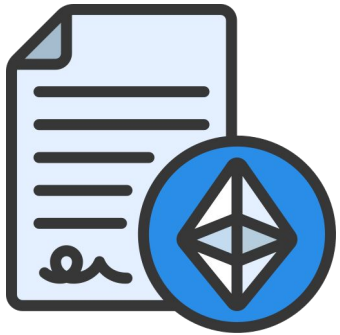
LLMs as verification oracles for Solidity

Massimo Bartoletti

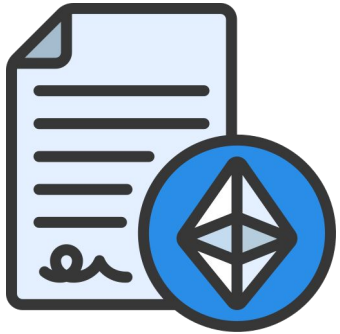
Enrico Lipparini

Livio Pompianu

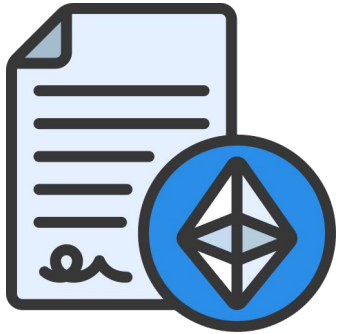
University of Cagliari, Italy



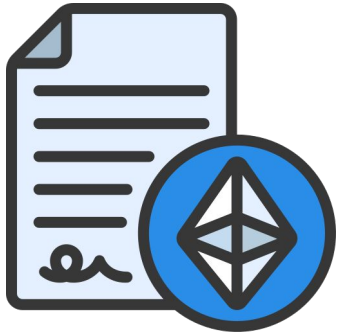
> $100 billion



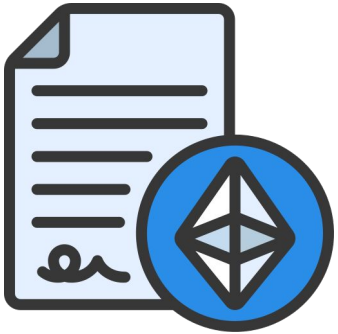
> $100 billion

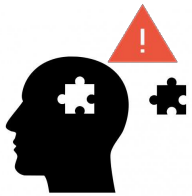


> $100 billion

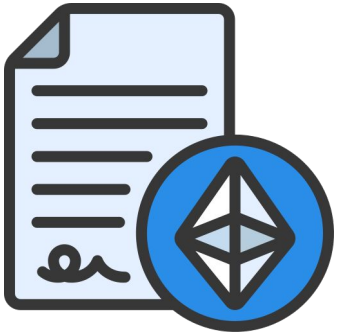


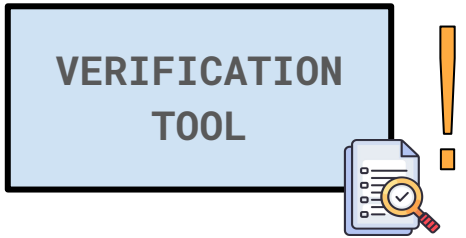
> $100 billion

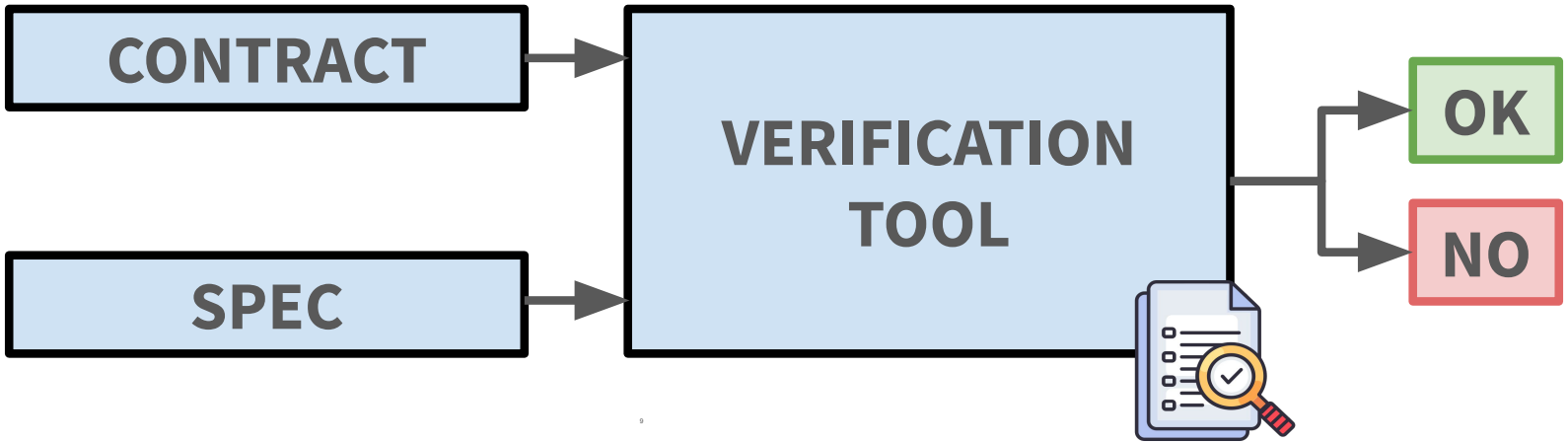




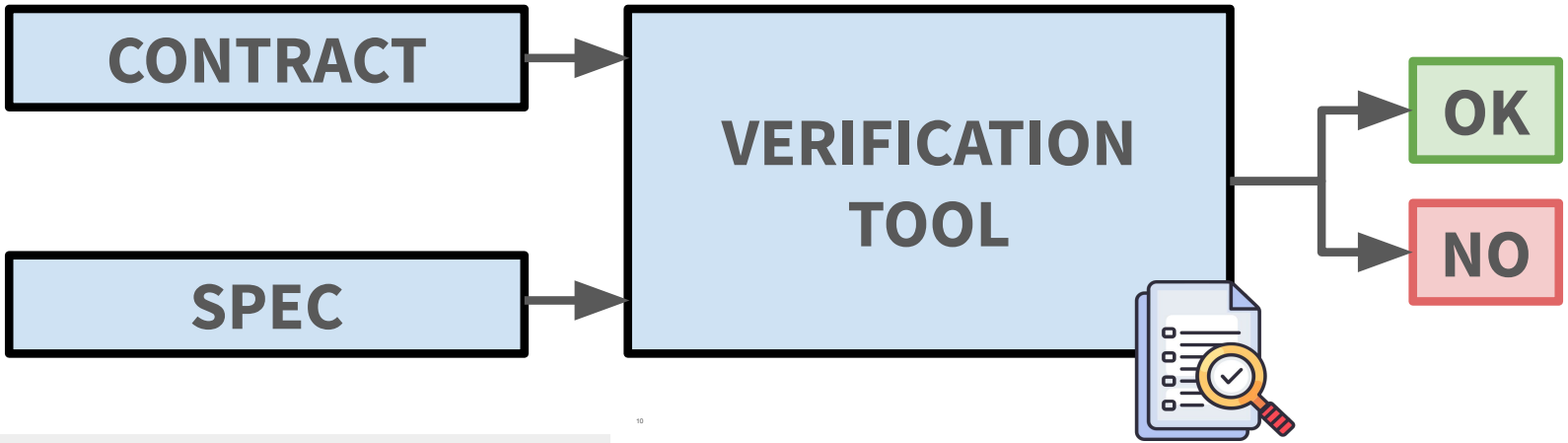
> $100 billion





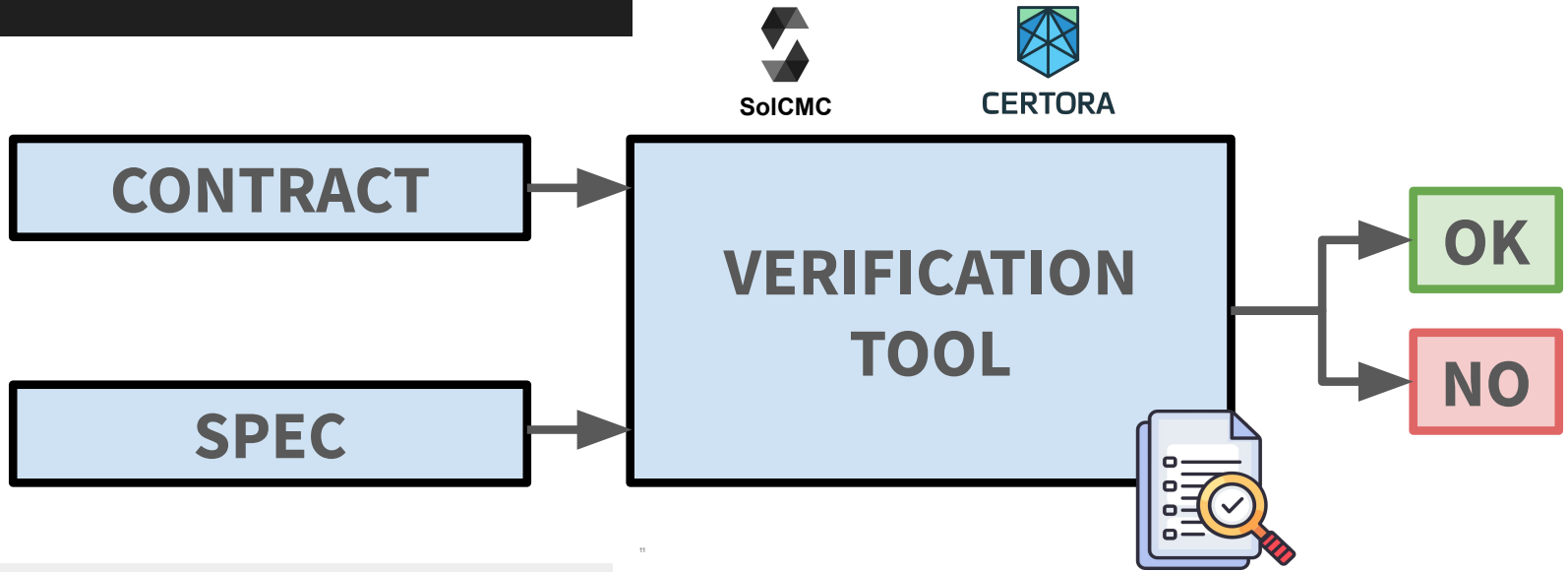

CONTRACT
SPEC
VERIFICATION TOOL
OK
NO

```
contract Bank {
    mapping (address user => uint credit) credits;

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



"after a non-reverting deposit(), the credits of msg.sender are increased by msg.value"

```
contract Bank {
    mapping (address user => uint credit) credits;

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



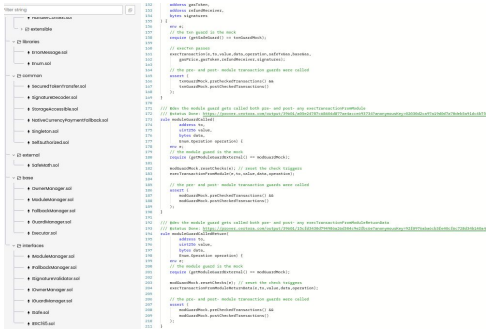
"after a non-reverting deposit(), the credits of msg.sender are increased by msg.value"

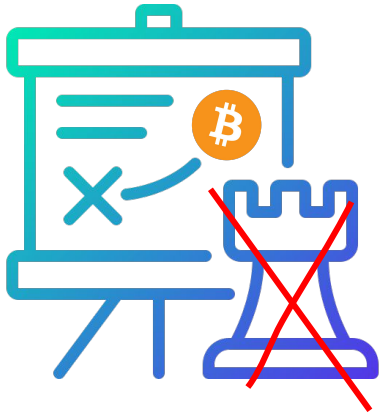
Formal Verification

issues

Formal Verification

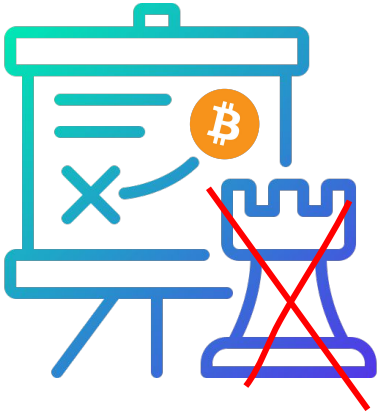
issues

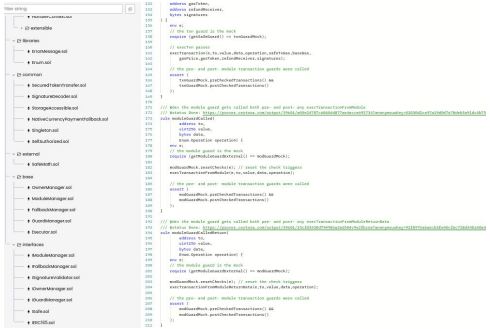


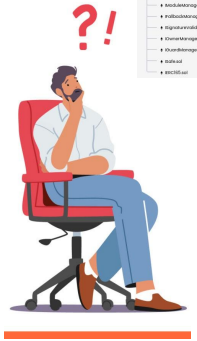


Formal Verification

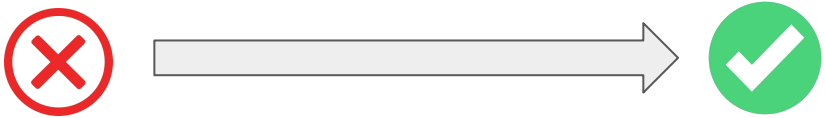
issues





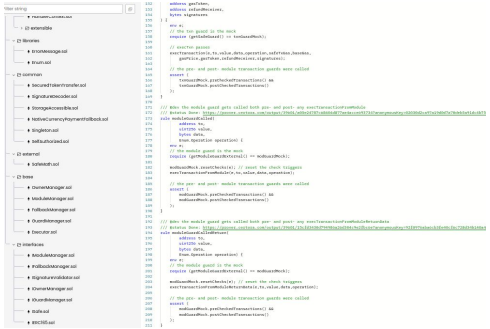


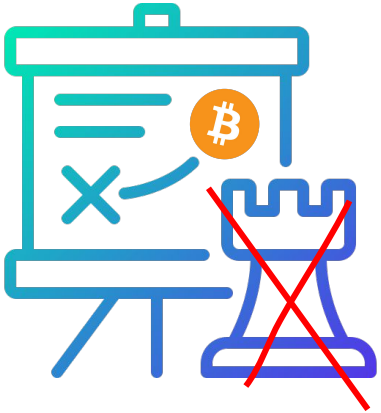
Semantics abstraction

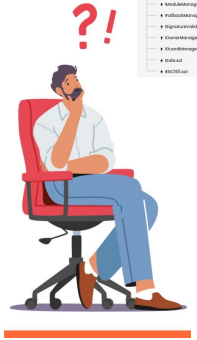


Formal Verification

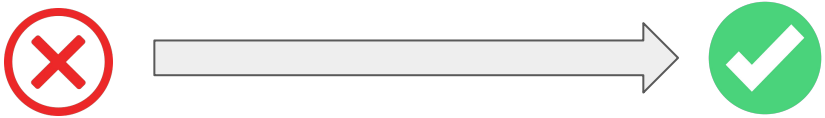
issues







Semantics abstraction

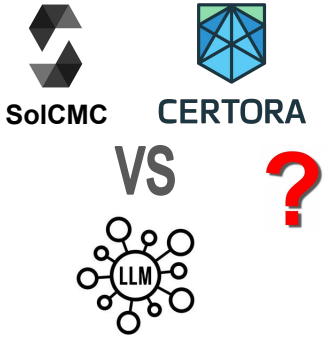


VERIFICATION TOOL

RQ1

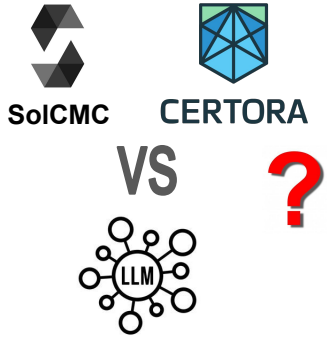
RQ1

RQ2



RQ1

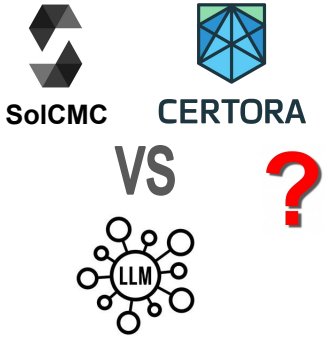
RQ2



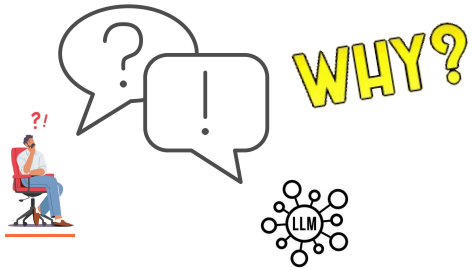
RQ3

RQ1

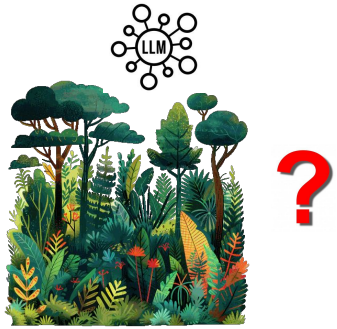
RQ2

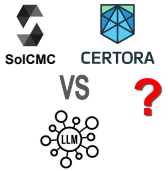


RQ3



RQ4





RQ1 RQ2 RQ3 RQ4

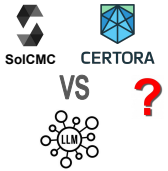
Controlled dataset

bitbart / **contracts-verification-benchmark**

- 5 use cases (→LP)
- 63 mutations
- 159 properties in natural language
- 2034 verification tasks
- specs for SolCMC / Certora

Certora Reports

- Safe (smart account wallet)
- InfiniFi (yield-bearing deposits)
- 75 properties (written in CVL + NL)

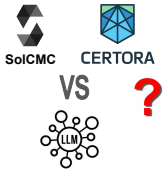


RQ1

RQ2

Dataset	Tool	F1 score
specs expressible in Certora		
$DS_{\|CVL}$	Certora	77%
$DS_{\|CVL}$	GPT-5	92%
$DS_{\|CVL}$	GPT-4	68%
specs expressible in SolCMC		
$DS_{\|SolCMC}$	SolCMC	79%
$DS_{\|SolCMC}$	GPT-5	90%
$DS_{\|SolCMC}$	GPT-4	66%

RQ1



RQ2

specs expressible in Certora

specs expressible in SolCMC

Dataset	Tool	F1 score
$DS_{\mid CVL}$	Certora	77%
$DS_{\mid CVL}$	GPT-5	92%
$DS_{\mid CVL}$	GPT-4	68%
$DS_{\mid SolCMC}$	SolCMC	79%
$DS_{\mid SolCMC}$	GPT-5	90%
$DS_{\mid SolCMC}$	GPT-4	66%

specs *not* expressible in Certora

$DS_{\mid \neg CVL}$	GPT-5	91%

specs *not* expressible in SolCMC

$DS_{\mid \neg SolCMC}$	GPT-5	93%

RQ3

PoC generated by GPT-5

- Deploy with payees [Owner O, Alice A], shares [1, 2] (totalShares = 3).
- Send 3 wei to the contract.
- Call release(O): pays 1 wei to O.
- Call release(A): pays 2 wei to A. Now totalReleased = 3, balance = 0, released[O]=1, released[A]=2, totalReceived = 3.
- Send 2 wei to the contract (balance = 2, totalReleased = 3, totalReceived = 5).
- Call release(A): payment = floor(5*2/3) - 2 = 1; pays 1 wei; totalReleased = 4; balance = 1; released[A] = 3; totalReceived remains 5.
- Call release(O): payment = floor(5*1/3) - 1 = 0; else-branch executes and transfers 1 wei to owner; totalReleased stays 4; balance becomes 0; totalReceived becomes 4.
- Now for A: RHS = floor(4*2/3) = 2, but released[A] = 3 and releasable(A) = floor(4*2/3) - 3 = -1 (underflow → revert).

Hence the property “released[A] + releasable(A) == (totalReceived * shares[A]) / totalShares” is violated (releasable(A) cannot be computed).

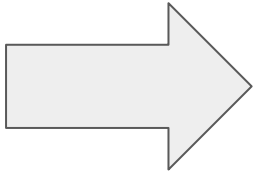
RQ3

PoC generated by GPT-5

- Deploy with payees [Owner O, Alice A], shares [1, 2] (totalShares = 3).
- Send 3 wei to the contract.
- Call release(O): pays 1 wei to O.
- Call release(A): pays 2 wei to A. Now totalReleased = 3, balance = 0, released[O]=1, released[A]=2, totalReceived = 3.
- Send 2 wei to the contract (balance = 2, totalReleased = 3, totalReceived = 5).
- Call release(A): payment = floor(5*2/3) - 2 = 1; pays 1 wei; totalReleased = 4; balance = 1; released[A] = 3; totalReceived remains 5.
- Call release(O): payment = floor(5*1/3) - 1 = 0; else-branch executes and transfers 1 wei to owner; totalReleased stays 4; balance becomes 0; totalReceived becomes 4.
- Now for A: RHS = floor(4*2/3) = 2, but released[A] = 3 and releasable(A) = floor(4*2/3) - 3 = -1 (underflow → revert).

Hence the property “released[A] + releasable(A) == (totalReceived * shares[A]) / totalShares” is violated (releasable(A) cannot be computed).

Manual validation in Forge



```
contract PaymentSplitterTest is Test {

    PaymentSplitter public ps;
      bart, last month • test/fair-split-eq_v9.t.sol
    function setUp() public {
    }

    function test_fair_split_eq() public {

        address o = address(0x1);
        address b = address(0x2);
        address c = address(0x3);

        address[] memory payees = new address[](2);
        payees[0] = address(o);
        payees[1] = address(b);

        uint256[] memory shares = new uint256[](2);
        shares[0] = 1;
        shares[1] = 1;

        vm.prank(o);
        ps = new PaymentSplitter(payees, shares);

        // C sends 100 wei to the contract
        vm.deal(address(c), 2);
        vm.prank(c);
        (bool success,) = address(ps).call{value:2}("");
        assert(success);

        ps.release(payable(b));
        ps.release(payable(c));

        // rhs = (totalReceived * shares[b]) / totalShares
        uint totalReceived = address(ps).balance + ps.getTotalReleased();
        uint rhs = (totalReceived * ps.getShares(b)) / ps.getTotalShares();
        assertEq(rhs, 0);

        // lhs = released[b] + releasable(b)
        vm.expectRevert();
        uint releasable_b = ps.releasable(b);

        // violation: lhs (reverted) != rhs
    }
}
```

RQ3

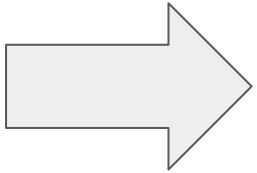
Correctness	92%
Completeness	89%
Coherence	95%

PoC generated by GPT-5

- Deploy with payees [Owner O, Alice A], shares [1, 2] (totalShares = 3).
- Send 3 wei to the contract.
- Call release(O): pays 1 wei to O.
- Call release(A): pays 2 wei to A. Now totalReleased = 3, balance = 0, released[O]=1, released[A]=2, totalReceived = 3.
- Send 2 wei to the contract (balance = 2, totalReleased = 3, totalReceived = 5).
- Call release(A): payment = floor(5*2/3) - 2 = 1; pays 1 wei; totalReleased = 4; balance = 1; released[A] = 3; totalReceived remains 5.
- Call release(O): payment = floor(5*1/3) - 1 = 0; else-branch executes and transfers 1 wei to owner; totalReleased stays 4; balance becomes 0; totalReceived becomes 4.
- Now for A: RHS = floor(4*2/3) = 2, but released[A] = 3 and releasable(A) = floor(4*2/3) - 3 = -1 (underflow → revert).

Hence the property “released[A] + releasable(A) == (totalReceived * shares[A]) / totalShares” is violated (releasable(A) cannot be computed).

Manual validation in Forge



```
contract PaymentSplitterTest is Test {

    PaymentSplitter public ps;
      bart, last month • test/fair-split-eq_v9.t.sol
    function setUp() public {
    }

    function test_fair_split_eq() public {

        address o = address(0x1);
        address b = address(0x2);
        address c = address(0x3);

        address[] memory payees = new address[](2);
        payees[0] = address(o);
        payees[1] = address(b);

        uint256[] memory shares = new uint256[](2);
        shares[0] = 1;
        shares[1] = 1;

        vm.prank(o);
        ps = new PaymentSplitter(payees, shares);

        // C sends 100 wei to the contract
        vm.deal(address(c), 2);
        vm.prank(c);
        (bool success,) = address(ps).call{value:2}("");
        assert(success);

        ps.release(payable(b));
        ps.release(payable(c));

        // rhs = (totalReceived * shares[b]) / totalShares
        uint totalReceived = address(ps).balance + ps.getTotalReleased();
        uint rhs = (totalReceived * ps.getShares(b)) / ps.getTotalShares();
        assertEq(rhs, 0);

        // lhs = released[b] + releasable(b)
        vm.expectRevert();
        uint releasable_b = ps.releasable(b);

        // violation: lhs (reverted) != rhs
    }
}
```

```
contract Bank {
    mapping (address user => uint credit) credits;

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

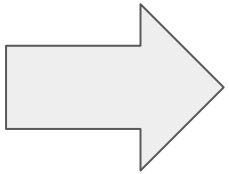
*"after a non-reverting **deposit()**,*
the credits of msg.sender
are increased by msg.value"

- Deploy with payees [Owner O, Alice A], shares [1, 2] (totalShares = 3).
- Send 3 wei to the contract.
- Call release(O): pays 1 wei to O.
- Call release(A): pays 2 wei to A. Now totalReleased = 3, balance = 0, released[O]=1, released[A]=2, totalReceived = 3.
- Send 2 wei to the contract (balance = 2, totalReleased = 3, totalReceived = 5).
- Call release(A): payment = floor(5*2/3) - 2 = 1; pays 1 wei; totalReleased = 4; balance = 1; released[A] = 3; totalReceived remains 5.
- Call release(O): payment = floor(5*1/3) - 1 = 0; else-branch executes and transfers 1 wei to owner; totalReleased stays 4; balance becomes 0; totalReceived becomes 4.
- Now for A: RHS = floor(4*2/3) = 2, but released[A] = 3 and releasable(A) = floor(4*2/3) - 3 = -1 (underflow → revert).

Hence the property “released[A] + releasable(A) == (totalReceived * shares[A]) / totalShares” is violated (releasable(A) cannot be computed).

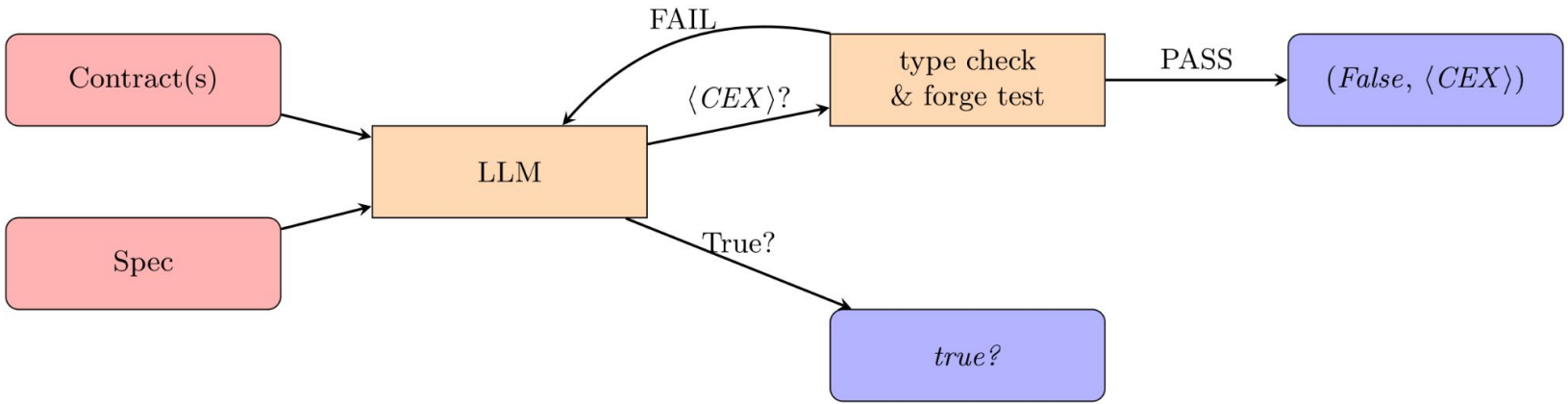
[M. Bartoletti & E. Lipparini, VERIFAI 2026]

```
abstract contracts[] cs;

contract BankTest is Test {
  Bank immutable bank;

  constructor() {
    abstract address bank_deployer;
    vm.prank(bank_deployer);
    bank = new Bank();
  }

  function test_withdraw_assets_credit_others_violation()
    abstract transaction[] txs;
    abstract address user;
    uint256 user_creditsBefore = bank.credits(user);

    abstract uint256 amount;
    abstract address sender;
    vm.prank(sender);
    bank.withdraw(amount); // should not revert

    uint256 user_creditsAfter = bank.credits(user);

    assertNotEq(sender, user, "user equal to sender");
    assertNotEq(user_creditsBefore, user_creditsAfter, "cr
  }
}
```



```
contract Attacker {
 Bank public bank;
 User public user;

 constructor(Bank _bank) {
  bank = _bank;
 }

 function setUser(User _user) external {
  user = _user;
 }

 // When receiving ETH from Bank.withdraw, forward it to User,
 // which deposits into Bank, crediting the User.
 receive() external payable {
  user.doDeposit{value: msg.value}(bank);
 }
}

contract BankTest is Test {
 constructor() {
  address bank_deployer = address(0xBEEF);
  [...]  // deployment of bank
 }

 function test_withdraw_assets_credit_others_violation() public {
  User u = new User();
  Attacker a = new Attacker(bank);
  a.setUser(u);
  vm.deal(address(a), 1 ether);
  vm.prank(address(a));
  bank.deposit{value: 1 ether}();

  address user = address(u);
  [...] // def. of user_creditsBefore
  uint256 amount = 1 ether;
  address sender = address(a);
  [...] // call to bank.withdraw and asserts
 }
```

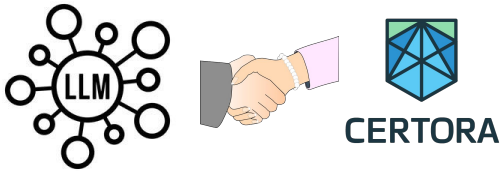
[M. Bartoletti & E. Lipparini, VERIFAI 2026]



results / # iter.	Total	(TRUE, *true?*)	(FALSE, *true?*)	(FALSE, *False*)
Total	109	57	4	48
0	90	46	4	40
1	15	10	0	5
2	4	1	0	3
3	0	0	0	0

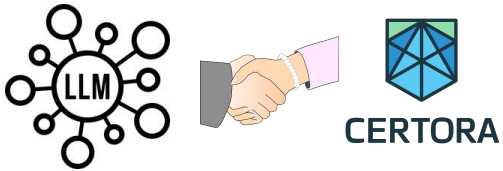
In the wild (RQ4)

In the wild (RQ4)



In the wild (RQ4)

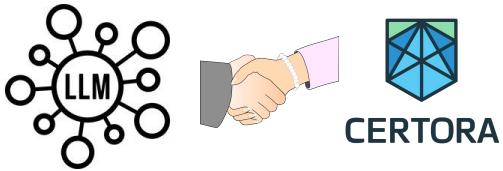
Spotting inconsistencies between NL property - formal spec



In the wild (RQ4)

Spotting inconsistencies between NL property - formal spec

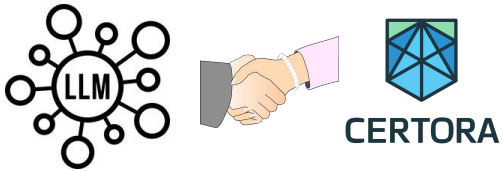
Completing trace violation



In the wild (RQ4)

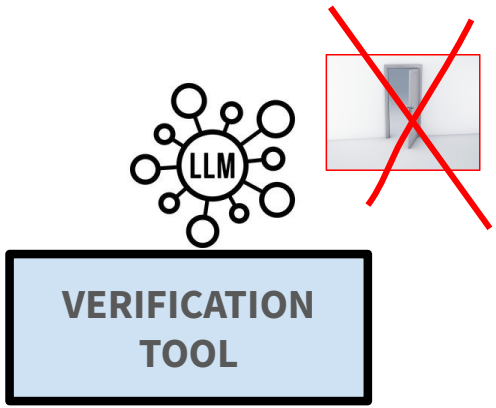
Spotting inconsistencies between NL property - formal spec

Completing trace violation

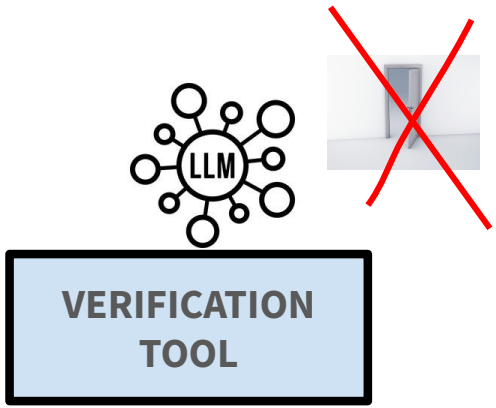


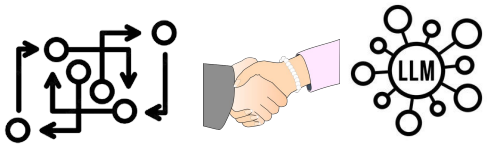
Flattened learning curve

Takeaway message

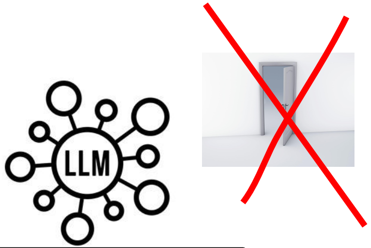


Takeaway message

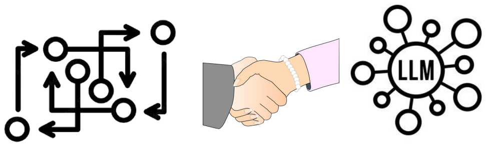


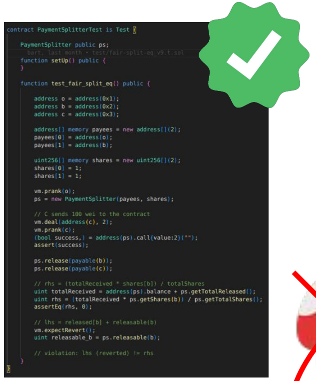


Takeaway message



VERIFICATION TOOL





```
contract PaymentSplitterTest is Test {

    PaymentSplitter public ps;
      bart, last month • test/fair-split-eq_v9.t.sol
    function setUp() public {
    }

    function test_fair_split_eq() public {

        address o = address(0x1);
        address b = address(0x2);
        address c = address(0x3);

        address[] memory payees = new address[](2);
        payees[0] = address(o);
        payees[1] = address(b);

        uint256[] memory shares = new uint256[](2);
        shares[0] = 1;
        shares[1] = 1;

        vm.prank(o);
        ps = new PaymentSplitter(payees, shares);

        // C sends 100 wei to the contract
        vm.deal(address(c), 2);
        vm.prank(c);
        (bool success,) = address(ps).call{value:2}("");
        assert(success);

        ps.release(payable(b));
        ps.release(payable(c));

        // rhs = (totalReceived * shares[b]) / totalShares
        uint totalReceived = address(ps).balance + ps.getTotalReleased();
        uint rhs = (totalReceived * ps.getShares(b)) / ps.getTotalShares();
        assertEq(rhs, 0);

        // lhs = released[b] + releasable(b)
        vm.expectRevert();
        uint releasable_b = ps.releasable(b);

        // violation: lhs (reverted) != rhs
    }
}
```

~~hallucinations~~