TU WIEN INFORMATICS

Scaling Formal Verification to Realistic Code with Applications to DeFi

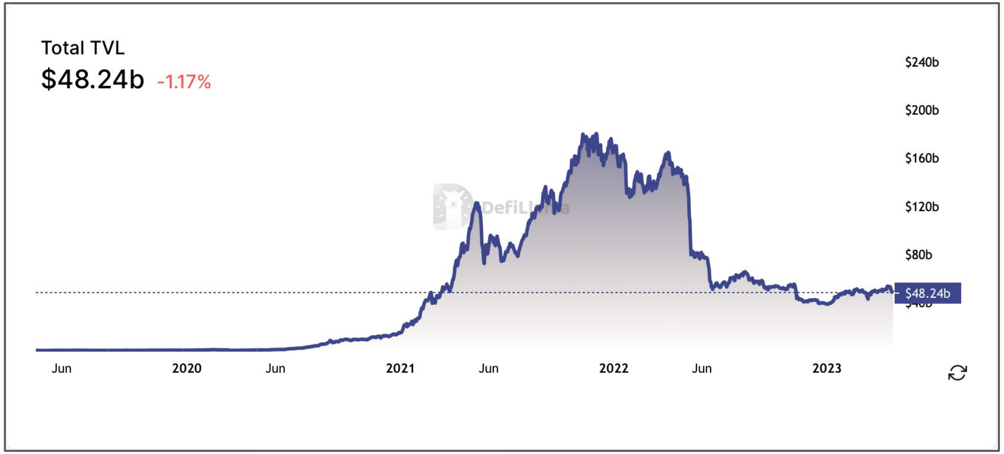
Mooly Sagiv

certora

TEL AVIV UNIVERSITY

DeFi in one slide

- Economic process completely defined by code
- Fairly complex code
- Examples
  - Landing
  - Exchange
  - Options
  - Auctions
- 50 Billion dollars in the bear market



certora

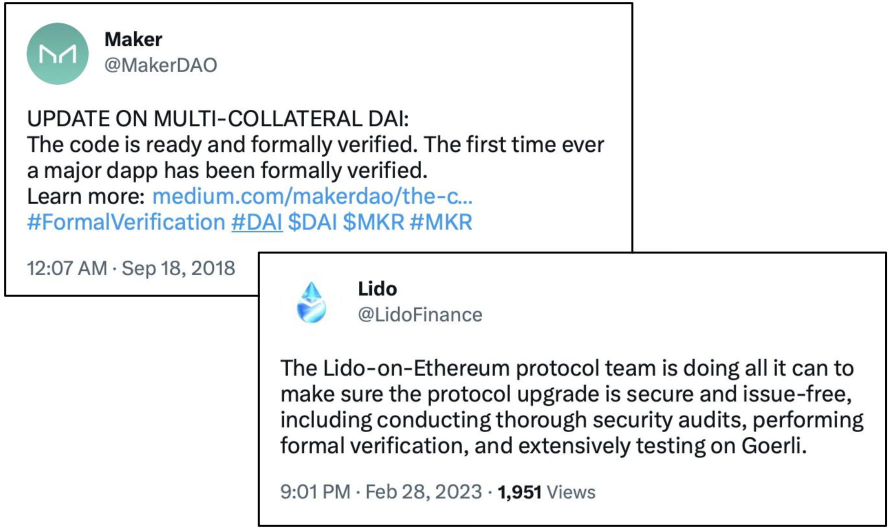
Interesting DeFi Bugs 2022/3

- **Euler Finance $200M** – DonateToReserves() function didn’t check for account debt health, allowing for bad debt to accrue and for the collateral to be liquidated at a large discount to the attacker
- **Yearn Finance V1 $10M** – Misconfiguration of one of the underlying asset addresses in the yUSDT pool allowed an attacker to drain the whole vault
- **Safemoon $9M** – Upgraded contract didn’t use access control for the burn() function. The attacker burned tokens from the Safemoon pool on a DEX, inflated the price and sold tokens into the pool
- **Platypus $8.5M** – EmergencyWithdraw() didn’t check for debt, so the attacker could take max loan for his collateral, and then simply emergency withdraw the collateral
- **Hundred $7.4M** – “First depositor” bug where the attacker could manipulate the exchange rate and borrow way more than allowed

certora

Why Formally Verify DeFi?

- ❗ Code is law
- 💰 Billions of dollars at stake
- Σ Code is typically small/modular
- 🐛 But bugs are hard to find
  Happens in rare scenarios
- 🔄 New code is produced frequently



Maker
@MakerDAO

UPDATE ON MULTI-COLLATERAL DAI:
The code is ready and formally verified. The first time ever a major dapp has been formally verified.
Learn more: medium.com/makerdao/the-c...
#FormalVerification #DAI $DAI $MKR #MKR

12:07 AM · Sep 18, 2018

Lido
@LidoFinance

The Lido-on-Ethereum protocol team is doing all it can to make sure the protocol upgrade is secure and issue-free, including conducting thorough security audits, performing formal verification, and extensively testing on Goerli.

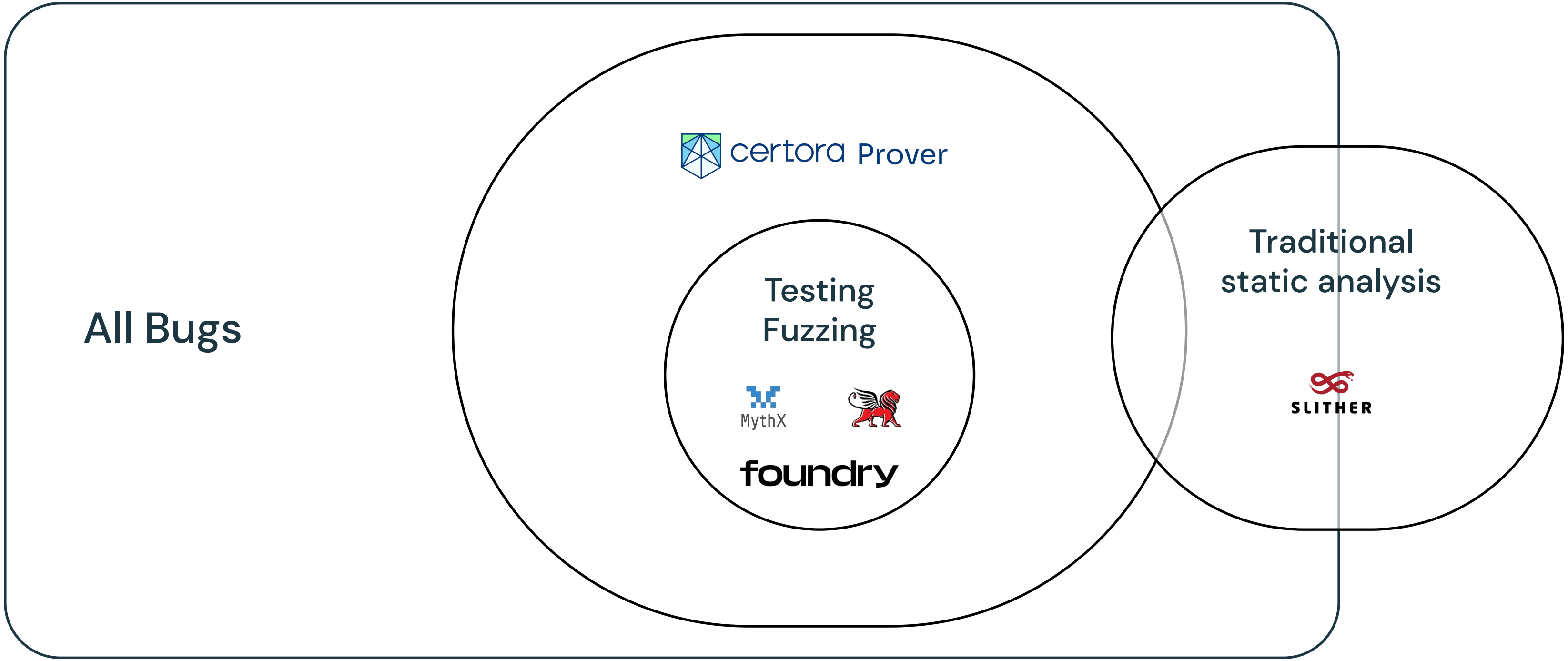
9:01 PM · Feb 28, 2023 · **1,951** Views

Code Security Tools

- Testing & Fuzzing
  - Easy to use
  - Hard to find (logical) corner cases
  - which depend on many events
- Static Analysis
  - False positives & negatives
- Automatic Formal Verification
  - Effective proof and bug finding
    - Start at arbitrary state
  - Computationally expensive
- Proof Assistants Spec ⇒ Code
  - Laborious efforts

certora

Competitive Landscape: Code Security

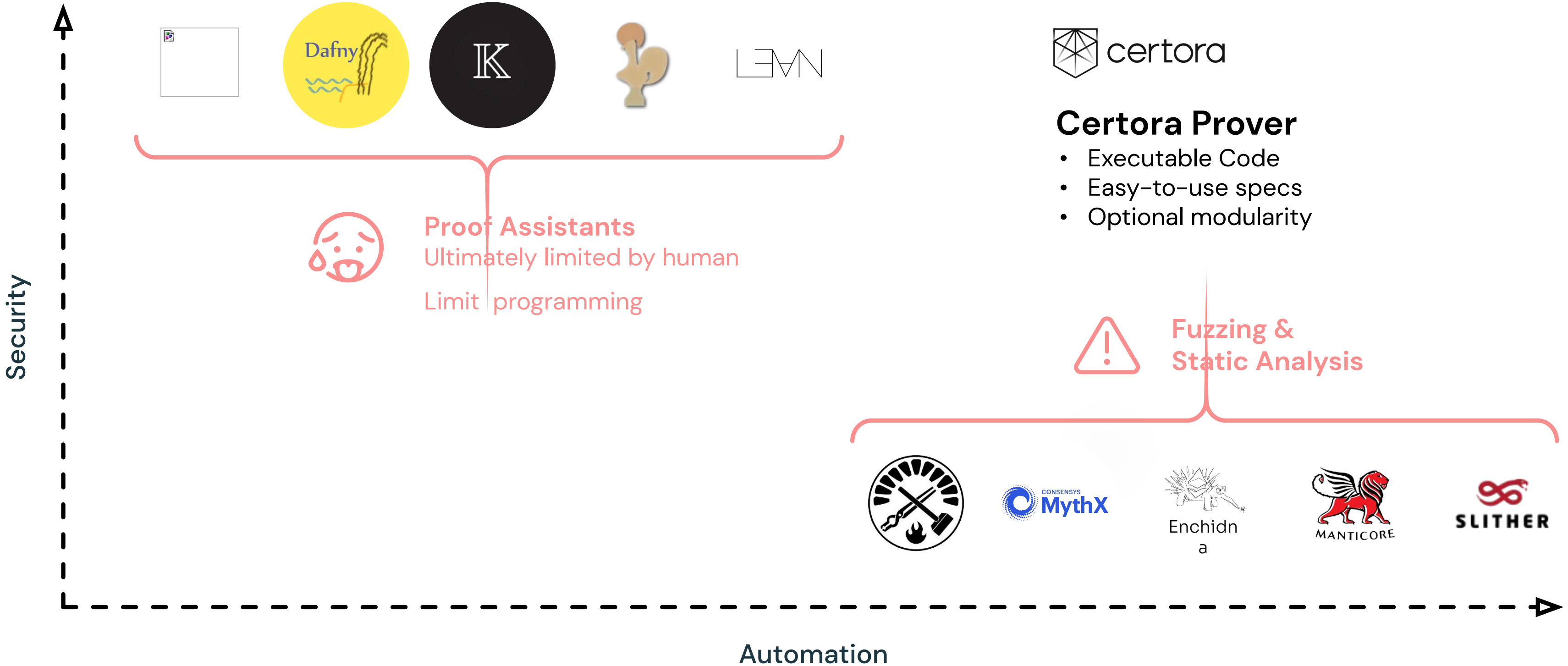


Slither In Action

```
uri@MacBook-Pro-7 Bank % slither . --solc solc4.25
Compilation warnings/errors on ./Bank.sol:
./Bank.sol:1:1: Warning: Source file does not specify required compiler version!Consider
contract Bank {
^ (Relevant source part starts here and spans across multiple lines).
./Bank.sol:26:2: Warning: Function state mutability can be restricted to view
        function getfunds(address account) public returns (uint256) {
 ^ (Relevant source part starts here and spans across multiple lines).
./Bank.sol:31:2: Warning: Function state mutability can be restricted to view
        function ercBalance() public returns (uint256) {
 ^ (Relevant source part starts here and spans across multiple lines).
./Bank.sol:34:2: Warning: Function state mutability can be restricted to pure
        function init_state() public {}
        ^-----------------------------^



Bank (Bank.sol#1-37) has incorrect ERC20 function interface:Bank.transfer(address,uint256
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-erc20-

solc-0.4.25 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versio

Function Bank.init_state() (Bank.sol#34) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-s

deposit(uint256) should be declared external:
        - Bank.deposit(uint256) (Bank.sol#4-6)
transfer(address,uint256) should be declared external:
        - Bank.transfer(address,uint256) (Bank.sol#8-12)
withdraw() should be declared external:
        - Bank.withdraw() (Bank.sol#14-18)
withdraw(uint256) should be declared external:
        - Bank.withdraw(uint256) (Bank.sol#20-24)
getfunds(address) should be declared external:
        - Bank.getfunds(address) (Bank.sol#26-28)
ercBalance() should be declared external:
        - Bank.ercBalance() (Bank.sol#31-33)
```

certora

State Of The Art In Code Quality Tools



Certora Prover

- Executable Code
- Easy-to-use specs
- Optional modularity

Proof Assistants
Ultimately limited by human

Limit programming

Fuzzing & Static Analysis

The Certora's Code Security Solution

Technology for checking bytecode for logical errors

- The Certora Prover:
  Automatic formal verification for bytecode programs
- Gambit: Simplifies CVL writing
- Foresight: Monitoring CVL properties
  - Utilize Certora Prover
  - Scale to all scenarios

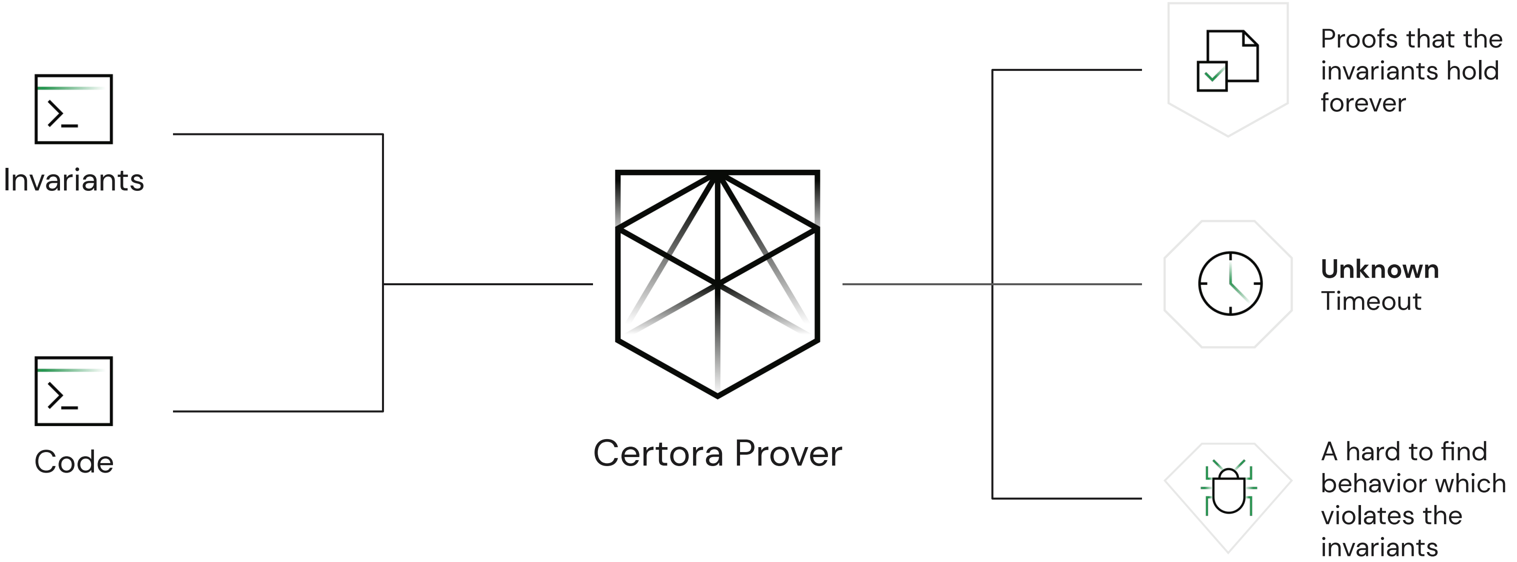
CVL a declarative language for expressing what (not how) the contract should do

- Application specific

certora

The Certora Prover:

Automatic formal verification



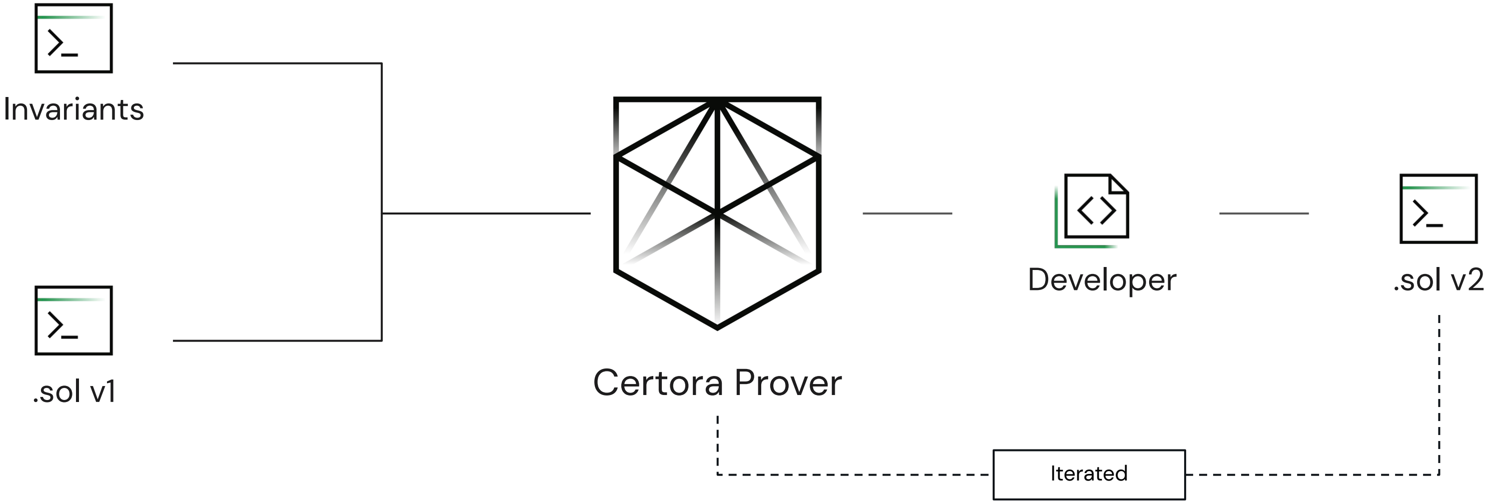
Proofs that the invariants hold forever

Unknown
Timeout

A hard to find behavior which violates the invariants

CI Integration

Verification driven development



Invariants

.sol v1

Certora Prover

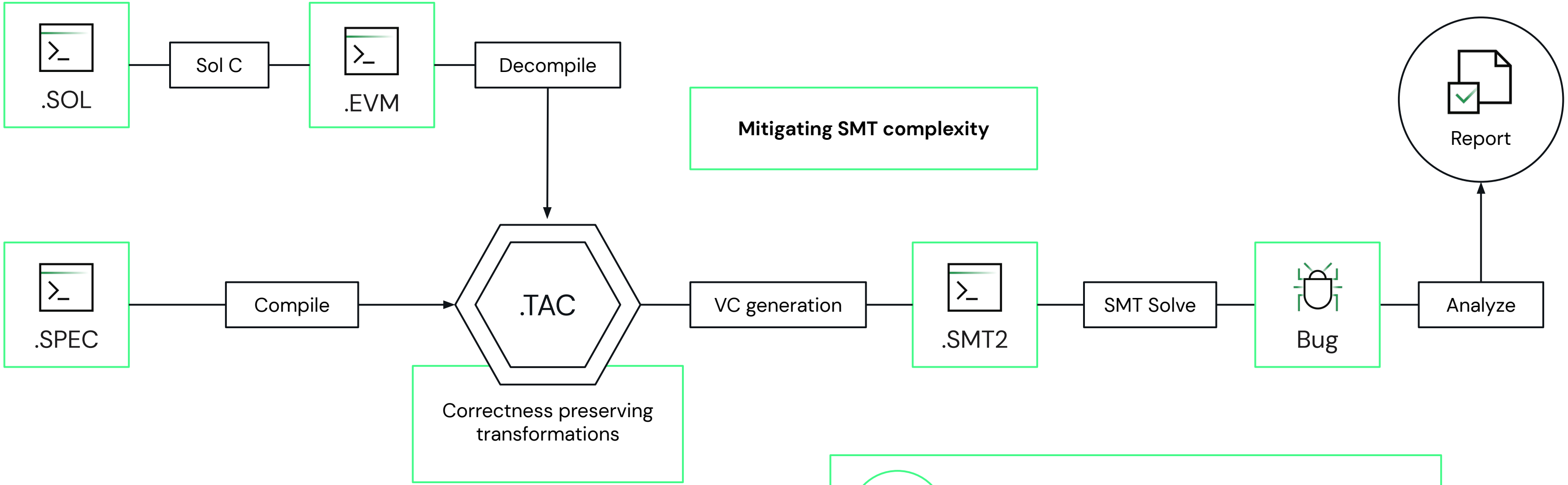
Developer

.sol v2

Iterated

certora

Certora Prover Architecture


.SOL
Sol C
.EVM
Decompile
Mitigating SMT complexity
Report
.SPEC
Compile
.TAC
VC generation
.SMT2
SMT Solve
Bug
Analyze
Correctness preserving transformations

1. **Trust but Verify**
2. **Abstract Interpretation:**
   Recover types
   Memory analysis

Solidity
@solidity_lang

"We thank all contributors who made this release possible. Special thanks goes to @johnadtoman of @CertoraInc for reporting the inline assembly memory side effects bug!"

certora

Simple Example

Money transfer

CODE

```
transfer (address from, address to, uint256
amount) {
  require (balances[from] ⩾ amount);
  balancesFrom = balances[from] - amount;
  balancesTo = balances[to] + amount;
  balances[from] = balancesFrom;
  balances[to] = balancesTo;
}
```

Certora
Prover

TEST

```
From = “Alice”
To = “Alice”
Amount = 18
old.balances(Alice) = 20
new.balances(Alice) = 38
```

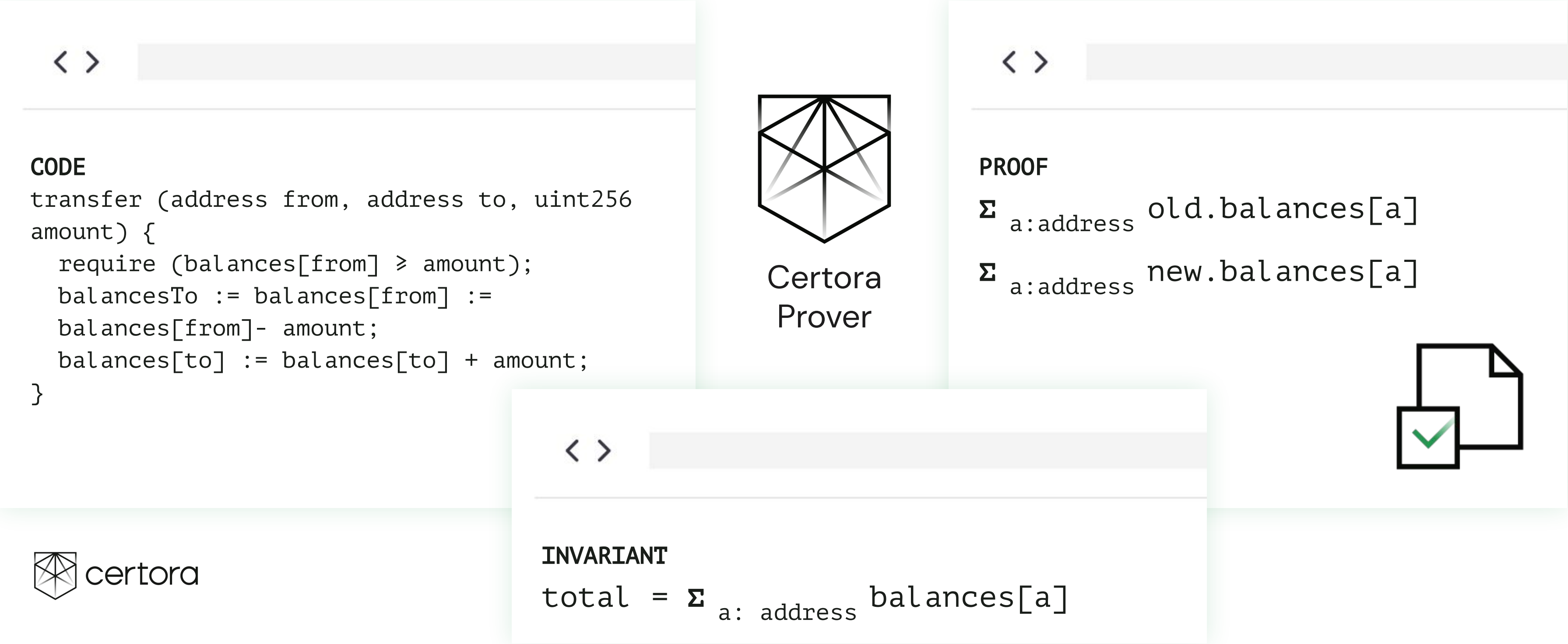
INVARIANT

$$total = \Sigma_{a:\ address}\ balances[a]$$

certora

Simple Example

Money transfer



CODE

```
transfer (address from, address to, uint256
amount) {
  require (balances[from] ⩾ amount);
  balancesTo := balances[from] :=
  balances[from]- amount;
  balances[to] := balances[to] + amount;
}
```

Certora
Prover

PROOF

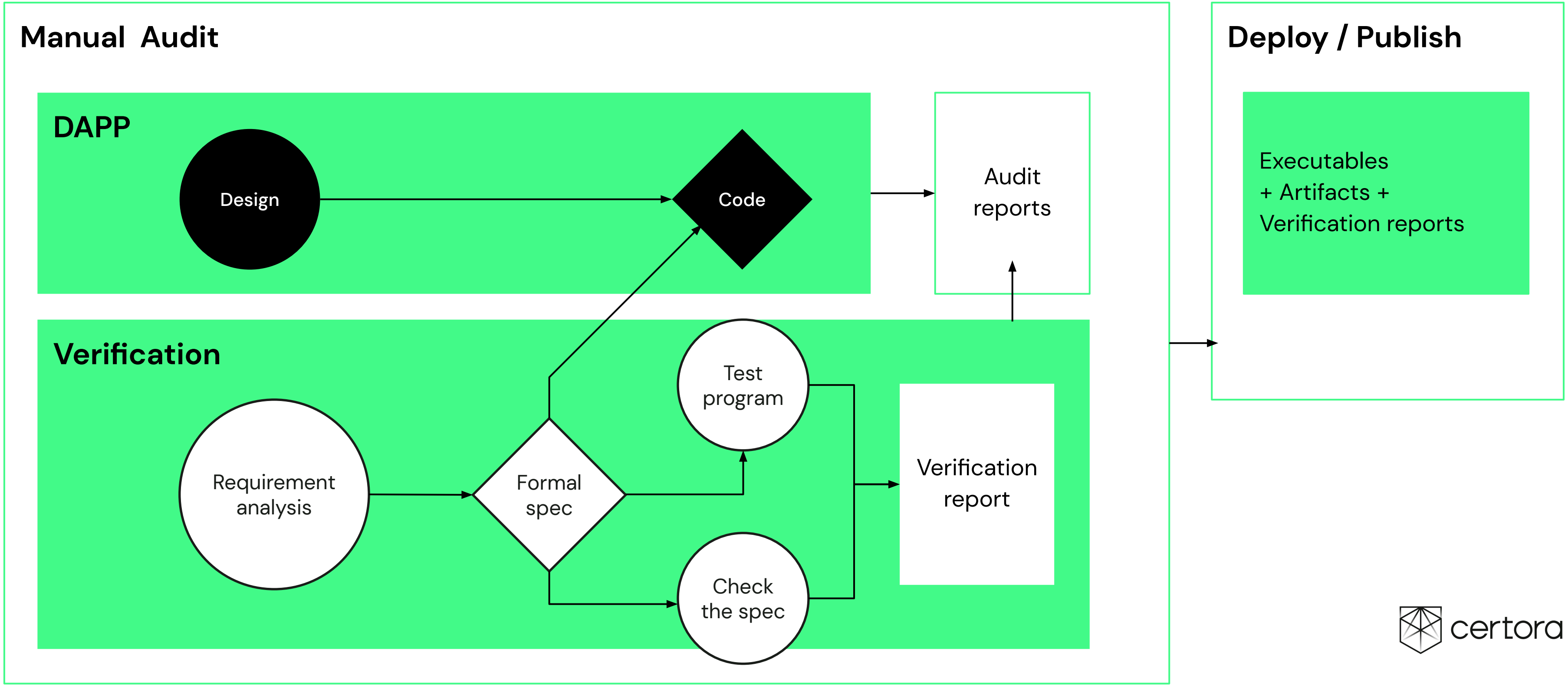
$$\Sigma_{a:address}\ \text{old.balances}[a]$$

$$\Sigma_{a:address}\ \text{new.balances}[a]$$

INVARIANT

$$\text{total} = \Sigma_{a:\ address}\ \text{balances}[a]$$

Where Does FV Fit In The Development Cycle



Manual Audit

DAPP

Design

Code

Audit reports

Verification

Requirement analysis

Formal spec

Test program

Check the spec

Verification report

Deploy / Publish

Executables
+ Artifacts +
Verification reports

Automatic Formal Verification

Finds Catastrophic Failures In Complex Code

- **Solvency**
  If everybody runs to the bank
  Bank still fulfills all commitments
- Users' money cannot be locked or lost

Bugs prevented by the Certora-Prover missed in manual audits by top

SushiSwap	$807M	AAVE	$6.5B	Compound	$2.7B	Balancer	$1.18B
Strategy	2	V3	1	Comet	5	V2	2
Trident	5	V2	2	V2	5		
KashiPair	3						
DutchAuction	1						

Automatic Formal Verification

1. Good Formal Specifications are hard to find
    - Complete specifications is impossible
    - Specs. which are good for the solver
    - Generic can get some errors
2. Bridging the gap between high level specifications and low-level code
3. Understanding the verifier results
    - Diagnosability
    - Correctness of spec
4. Code Complexity
    - Modular != Realistic
    - Nonlinear
    - Intercontract
    - Memory & Storage
    - Indirect calls
    - Dynamic loading
    - Unbounded entities
    - Assembly code
5. Mitigating SMT complexity
    - Splitting
    - Over-approximation and Axiomatisation
    - Learning lemmas
    - Slicing

certora

Good Specifications are hard to find

Solvency: Everybody can get the money in some way

- For every owner o:
    - For every asset a deposited by o:
        - For every reachable state s in which a was not withdrawn by o yet:
            - There exists a scenario starting from s in o can withdraw a

Weaker properties are checked

1. The system has sufficient assets to cover all holding
    - sum deposits >= sum possible withdrawals OR
    - current holdings >= sum obligations
2. Under certain conditions the owner can withdraw

A Four-Year Old Fundamental Bug Detected By Developer Using Formal Verification

- DAI is a stable coin launched by MakerDAO in 2019
- 6.5B$ market cap backed by **~10B$** collateral

May 2022

Kurt Barry
@Kurt_M_Barry

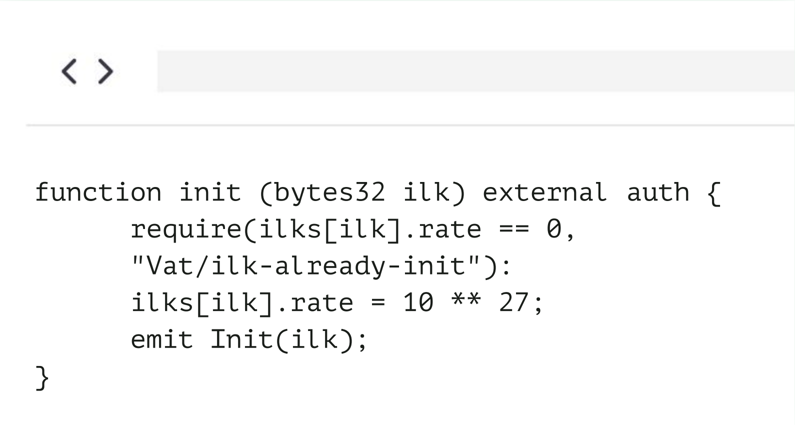
"Want to read about how we discovered that the foundational invariant of the Maker protocol was not, in fact, an invariant using @CertoraInc tech? Well, today you are in luck:

hackmd.io
When Invariants Aren't: DAI's Certora Surprise - H...
When Invariants Aren't: DAI's Certora Surprise
Authors: [Kurt Barry](https://twitter.com/Kurt_M_B

A Four-Year Old Fundamental Bug

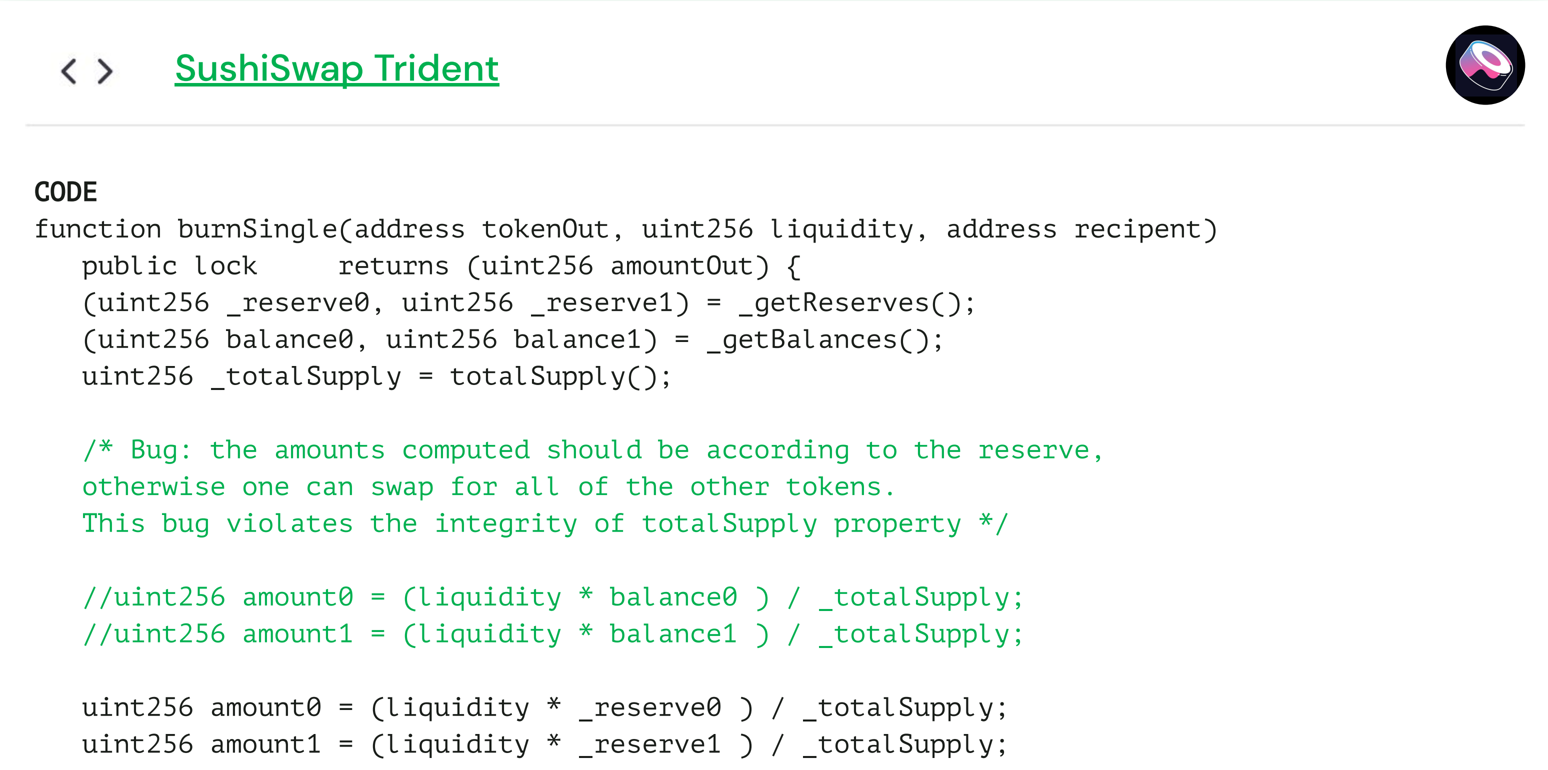
- **Bug**
  init function can change the rate of anilk with an Art greater than zero, breaking the FEoD

$$\cancel{\text{Vat.debt} = \text{Vat.vice} + \sum_i \text{Vat.ilks}[i].\text{Art} \cdot \text{Vat.ilks}[i].\text{rate}}$$



```
function init (bytes32 ilk) external auth {
      require(ilks[ilk].rate == 0,
      "Vat/ilk-already-init"):
      ilks[ilk].rate = 10 ** 27;
      emit Init(ilk);
}
```

The Critical Burn Bug Prevented



SushiSwap Trident

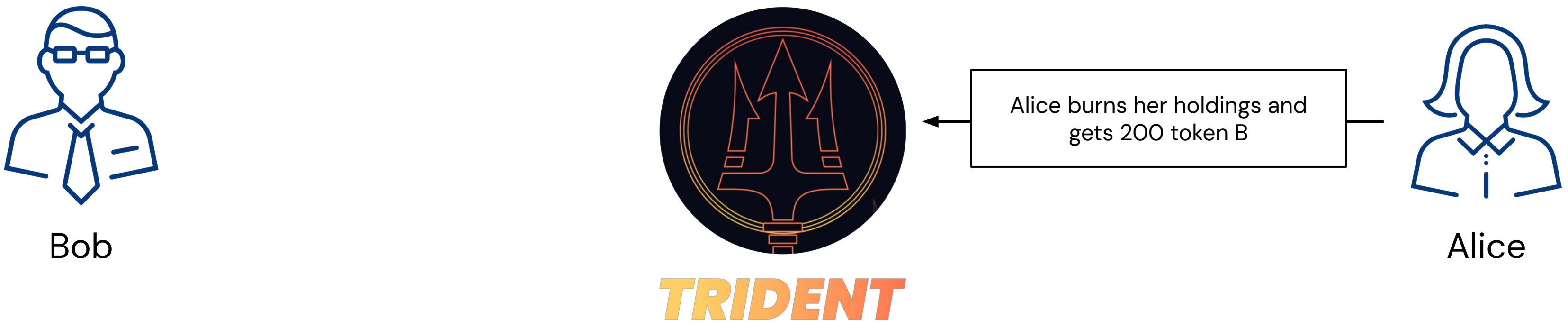
CODE

```
function burnSingle(address tokenOut, uint256 liquidity, address recipent)
   public lock     returns (uint256 amountOut) {
   (uint256 _reserve0, uint256 _reserve1) = _getReserves();
   (uint256 balance0, uint256 balance1) = _getBalances();
   uint256 _totalSupply = totalSupply();

   /* Bug: the amounts computed should be according to the reserve,
   otherwise one can swap for all of the other tokens.
   This bug violates the integrity of totalSupply property */

   //uint256 amount0 = (liquidity * balance0 ) / _totalSupply;
   //uint256 amount1 = (liquidity * balance1 ) / _totalSupply;

   uint256 amount0 = (liquidity * _reserve0 ) / _totalSupply;
   uint256 amount1 = (liquidity * _reserve1 ) / _totalSupply;
```

Certora Prover:

Searches For Catastrophic Failures In Complex Code

~~Token A > 0 ⇔ Token B > 0~~



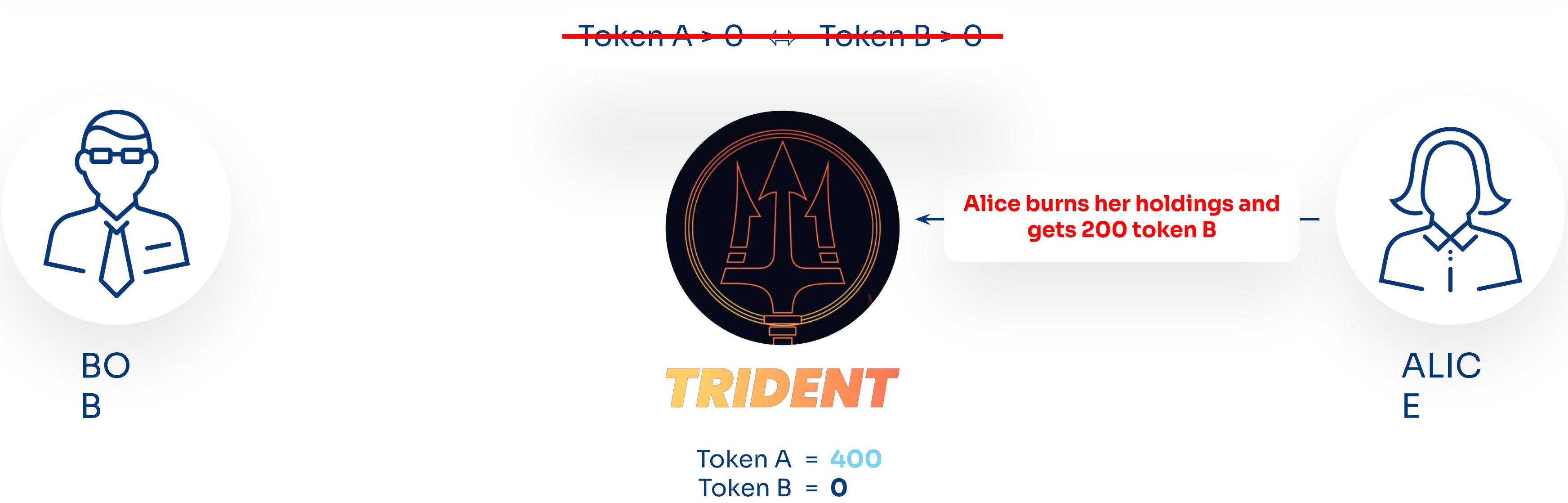
Alice burns her holdings and gets 200 token B

Token A = 400
Token B = 0

certora

CERTORA PROVER: SEARCHES FOR CATASTROPHIC FAILURES IN COMPLEX CODE

Sushi Trident | 2000 lines of Solidity code | 24,547 lines of EVM code | ∞ Behaviors



~~Token A > 0 ⇔ Token B > 0~~

Alice burns her holdings and gets 200 token B

BOB

TRIDENT

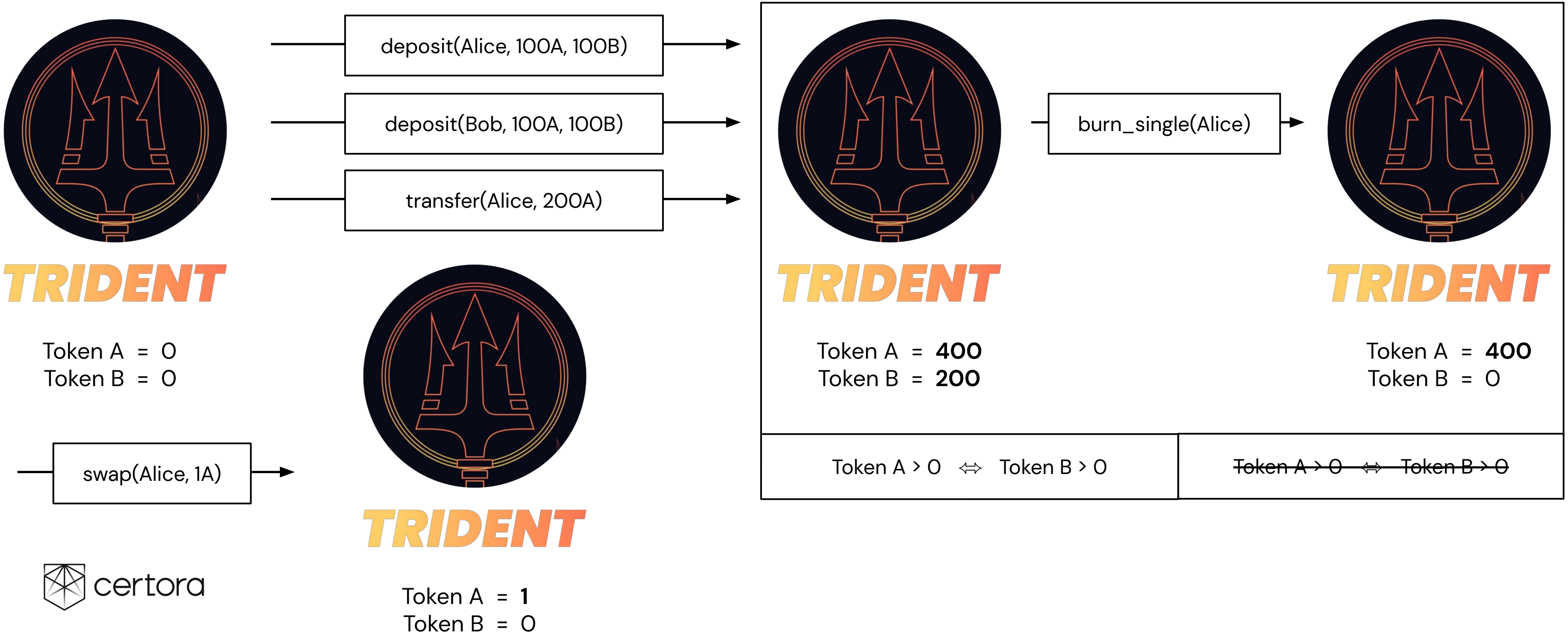
ALICE

Token A = 400
Token B = **0**

Resolved on November 4th, 2021

Certora Prover:

Searches For Catastrophic Failures In Complex Code



Token A = 0
Token B = 0

deposit(Alice, 100A, 100B)
deposit(Bob, 100A, 100B)
transfer(Alice, 200A)

Token A = **400**
Token B = **200**

burn_single(Alice)

Token A = **400**
Token B = 0

Token A > 0 ⇔ Token B > 0

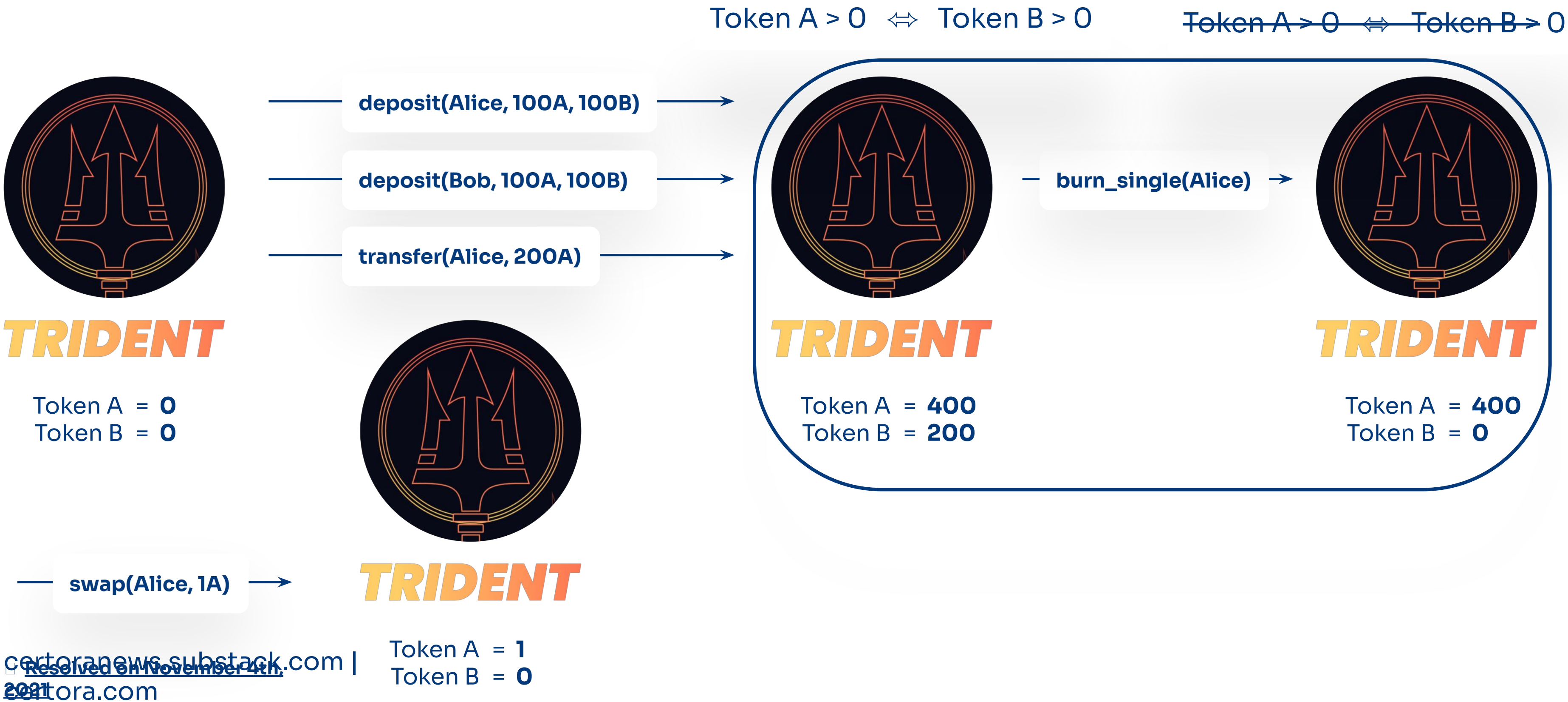
~~Token A > 0 ⇔ Token B > 0~~

swap(Alice, 1A)

Token A = **1**
Token B = 0

certora

CERTORA PROVER: SEARCHES FOR CATASTROPHIC FAILURES IN COMPLEX CODE



Token A > 0 ⇔ Token B > 0

~~Token A > 0 ⇔ Token B >~~ 0

deposit(Alice, 100A, 100B)

deposit(Bob, 100A, 100B)

transfer(Alice, 200A)

burn_single(Alice)

swap(Alice, 1A)

Token A = **0**
Token B = **0**

Token A = **400**
Token B = **200**

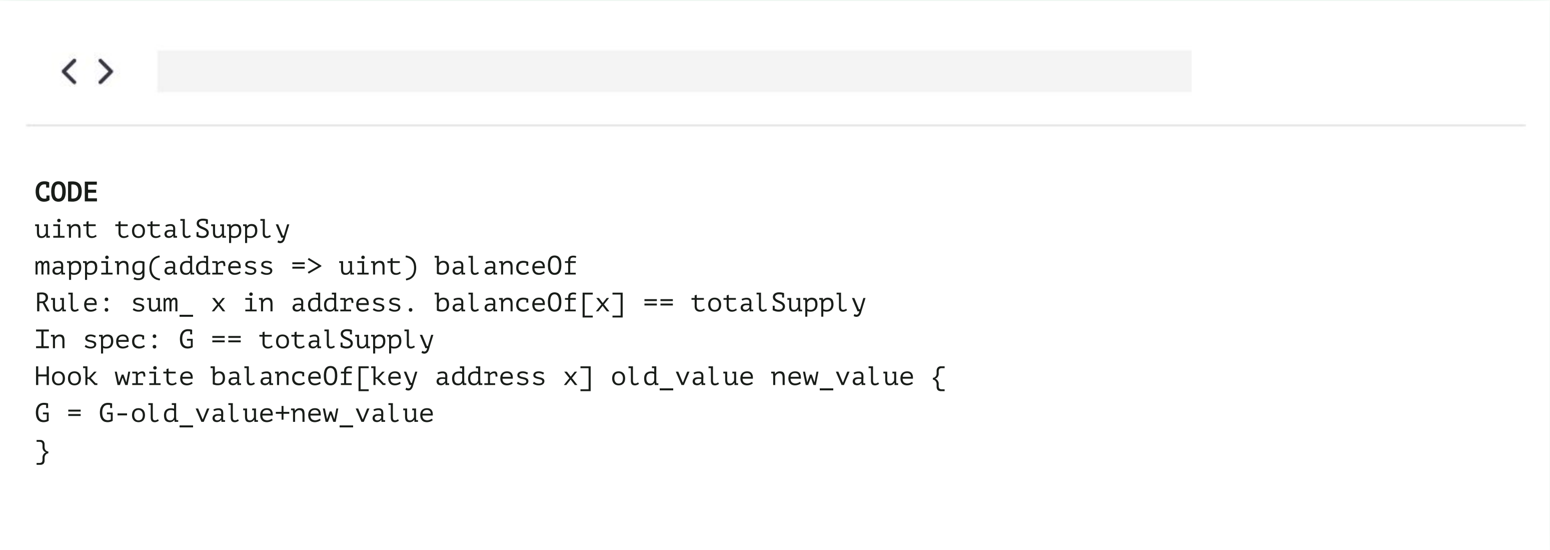
Token A = **400**
Token B = **0**

Token A = **1**
Token B = **0**

Resolved on November 4th, 2021

Bridging the gap between high level spec and low level bytecode

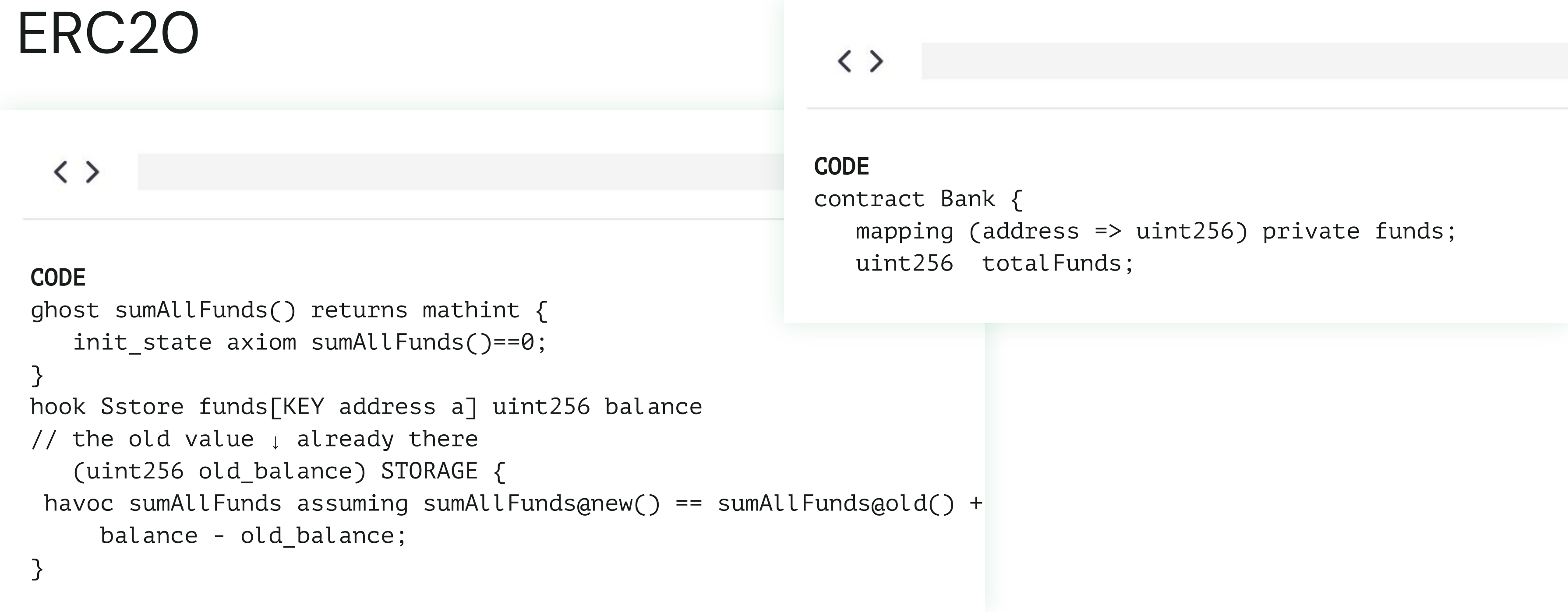
ERC20



CODE

```
uint totalSupply
mapping(address => uint) balanceOf
Rule: sum_ x in address. balanceOf[x] == totalSupply
In spec: G == totalSupply
Hook write balanceOf[key address x] old_value new_value {
G = G-old_value+new_value
}
```

ERC20

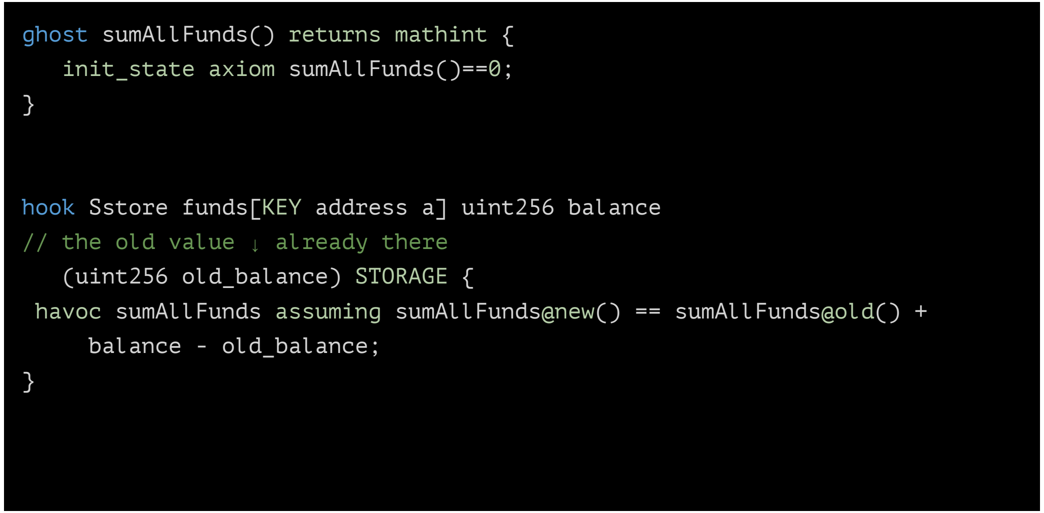


CODE

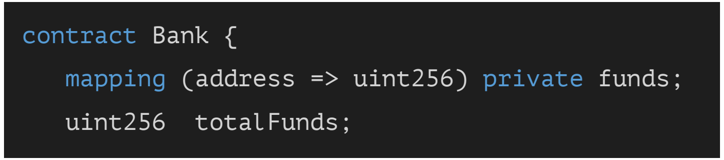
```
ghost sumAllFunds() returns mathint {
   init_state axiom sumAllFunds()==0;
}
hook Sstore funds[KEY address a] uint256 balance
// the old value ↓ already there
   (uint256 old_balance) STORAGE {
 havoc sumAllFunds assuming sumAllFunds@new() == sumAllFunds@old() +
     balance - old_balance;
}
```

CODE

```
contract Bank {
   mapping (address => uint256) private funds;
   uint256  totalFunds;
```



```
ghost sumAllFunds() returns mathint {
   init_state axiom sumAllFunds()==0;
}


hook Sstore funds[KEY address a] uint256 balance
// the old value ↓ already there
   (uint256 old_balance) STORAGE {
 havoc sumAllFunds assuming sumAllFunds@new() == sumAllFunds@old() +
     balance - old_balance;
}
```



```
contract Bank {
   mapping (address => uint256) private funds;
   uint256  totalFunds;
```

Digesting the Prover’s results

certora

What happens when the tool finds a counterexample to induction?

Overview of counterexample

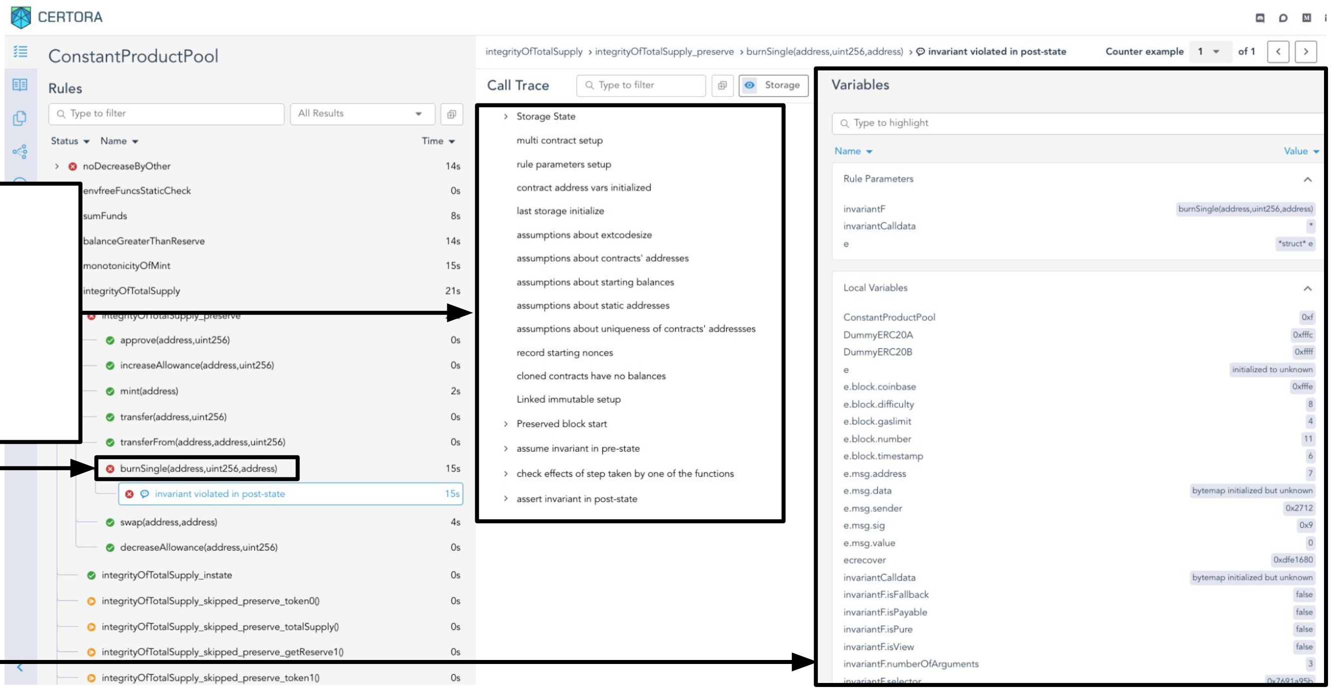
```
invariant integrityOfTotalSupply()
    (totalSupply() == 0 <=> getReserve0() == 0) &&
    (totalSupply() == 0 <=> getReserve1() == 0)
```

Detailed trace:

- Pre-state
- Assumption
- Induction step
- Postcondition check

Violated induction step

List of values



ConstantProductPool

Rules

Name	Time
noDecreaseByOther	14s
envfreeFuncsStaticCheck	0s
sumFunds	8s
balanceGreaterThanReserve	14s
monotonicityOfMint	15s
integrityOfTotalSupply	21s
integrityOfTotalSupply_preserve	
approve(address,uint256)	0s
increaseAllowance(address,uint256)	0s
mint(address)	2s
transfer(address,uint256)	0s
transferFrom(address,address,uint256)	0s
burnSingle(address,uint256,address)	15s
invariant violated in post-state	15s
swap(address,address)	4s
decreaseAllowance(address,uint256)	0s
integrityOfTotalSupply_instate	0s
integrityOfTotalSupply_skipped_preserve_token0()	0s
integrityOfTotalSupply_skipped_preserve_totalSupply()	0s
integrityOfTotalSupply_skipped_preserve_getReserve0()	0s
integrityOfTotalSupply_skipped_preserve_token1()	0s

integrityOfTotalSupply › integrityOfTotalSupply_preserve › burnSingle(address,uint256,address) › invariant violated in post-state

Call Trace

- Storage State
- multi contract setup
- rule parameters setup
- contract address vars initialized
- last storage initialize
- assumptions about extcodesize
- assumptions about contracts' addresses
- assumptions about starting balances
- assumptions about static addresses
- assumptions about uniqueness of contracts' addressses
- record starting nonces
- cloned contracts have no balances
- Linked immutable setup
- Preserved block start
- assume invariant in pre-state
- check effects of step taken by one of the functions
- assert invariant in post-state

Variables

Rule Parameters

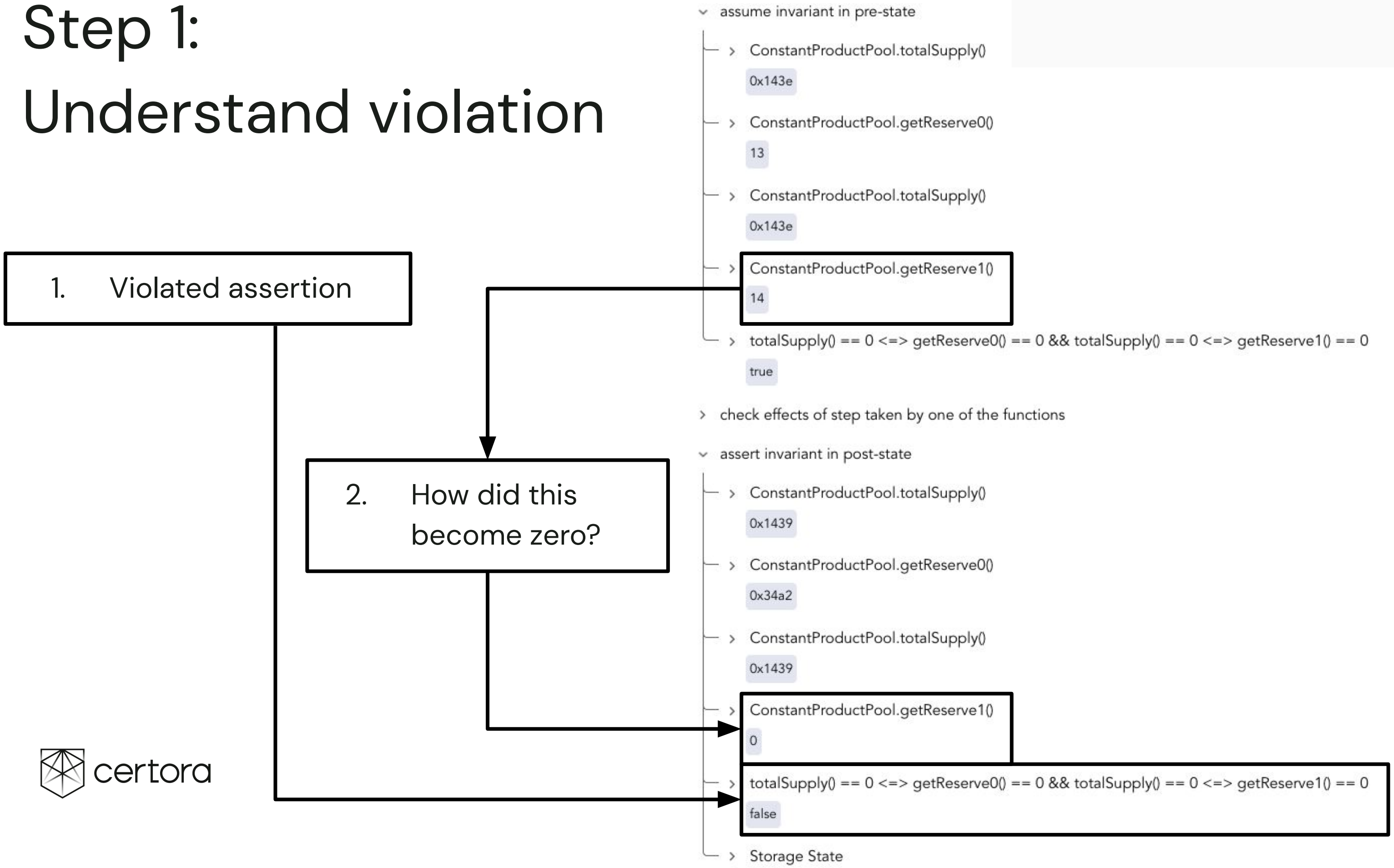
Name	Value
invariantF	burnSingle(address,uint256,address)
invariantCalldata	*
e	*struct* e

Local Variables

Name	Value
ConstantProductPool	0xf
DummyERC20A	0xfffc
DummyERC20B	0xffff
e	initialized to unknown
e.block.coinbase	0xfffe
e.block.difficulty	8
e.block.gaslimit	4
e.block.number	11
e.block.timestamp	6
e.msg.address	7
e.msg.data	bytemap initialized but unknown
e.msg.sender	0x2712
e.msg.sig	0x9
e.msg.value	0
ecrecover	0xdfe1680
invariantCalldata	bytemap initialized but unknown
invariantF.isFallback	false
invariantF.isPayable	false
invariantF.isPure	false
invariantF.isView	false
invariantF.numberOfArguments	3
invariantF.selector	0x7691a95b

```
invariant integrityOfTotalSupply()
    (totalSupply() == 0 <=> getReserve0() == 0) &&
    (totalSupply() == 0 <=> getReserve1() == 0)
```

Step 1:
Understand violation



```
˅ assume invariant in pre-state
  ├ › ConstantProductPool.totalSupply()
  │     0x143e
  ├ › ConstantProductPool.getReserve0()
  │     13
  ├ › ConstantProductPool.totalSupply()
  │     0x143e
  ├ › ConstantProductPool.getReserve1()
  │     14
  └ › totalSupply() == 0 <=> getReserve0() == 0 && totalSupply() == 0 <=> getReserve1() == 0
        true
› check effects of step taken by one of the functions
˅ assert invariant in post-state
  ├ › ConstantProductPool.totalSupply()
  │     0x1439
  ├ › ConstantProductPool.getReserve0()
  │     0x34a2
  ├ › ConstantProductPool.totalSupply()
  │     0x1439
  ├ › ConstantProductPool.getReserve1()
  │     0
  ├ › totalSupply() == 0 <=> getReserve0() == 0 && totalSupply() == 0 <=> getReserve1() == 0
  │     false
  └ › Storage State
```

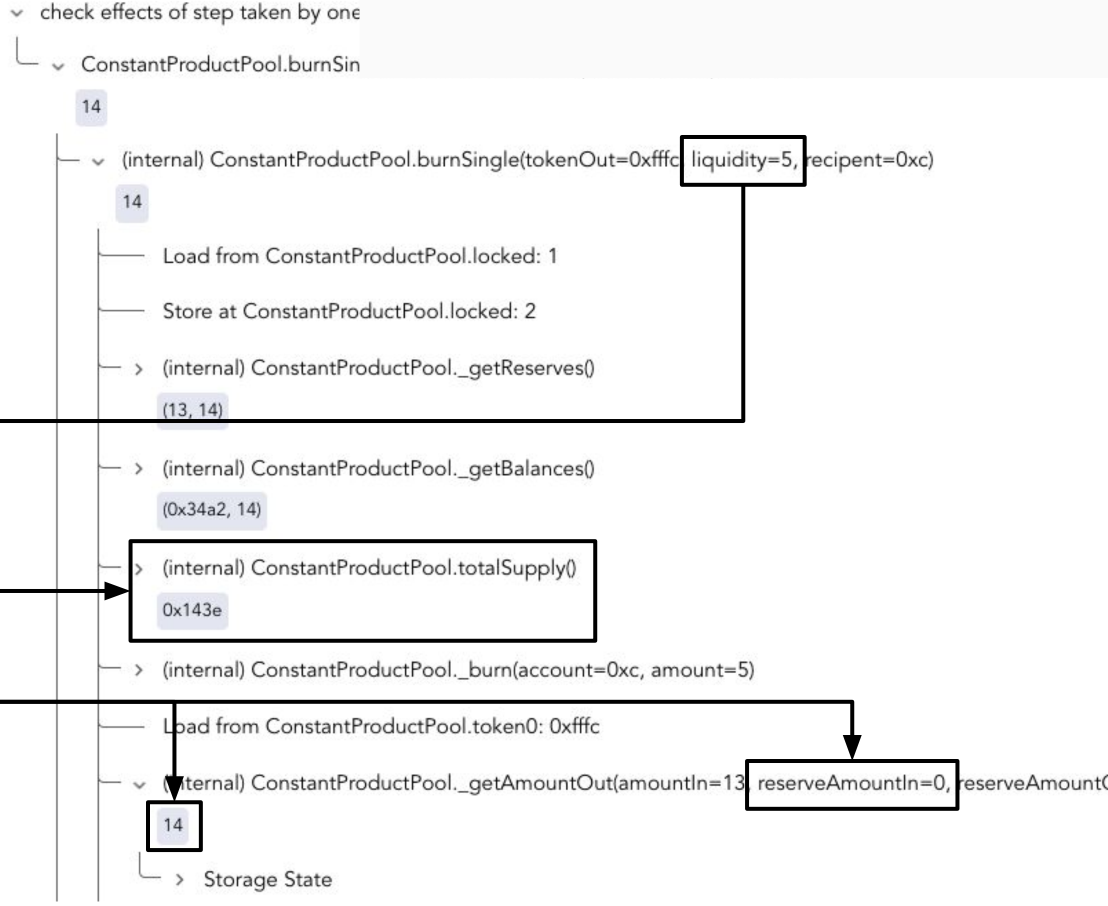
1. Violated assertion

2. How did this become zero?

certora

```
invariant integrityOfTotalSupply()
    (totalSupply() == 0 <=> getReserve0() == 0) &&
    (totalSupply() == 0 <=> getReserve1() == 0)
```

Step 2: Understand the change



```
check effects of step taken by one
  ConstantProductPool.burnSin
    14
    (internal) ConstantProductPool.burnSingle(tokenOut=0xfffc, liquidity=5, recipent=0xc)
      14
      Load from ConstantProductPool.locked: 1
      Store at ConstantProductPool.locked: 2
      (internal) ConstantProductPool._getReserves()
        (13, 14)
      (internal) ConstantProductPool._getBalances()
        (0x34a2, 14)
      (internal) ConstantProductPool.totalSupply()
        0x143e
      (internal) ConstantProductPool._burn(account=0xc, amount=5)
      Load from ConstantProductPool.token0: 0xfffc
      (internal) ConstantProductPool._getAmountOut(amountIn=13, reserveAmountIn=0, reserveAmount
        14
        Storage State
```

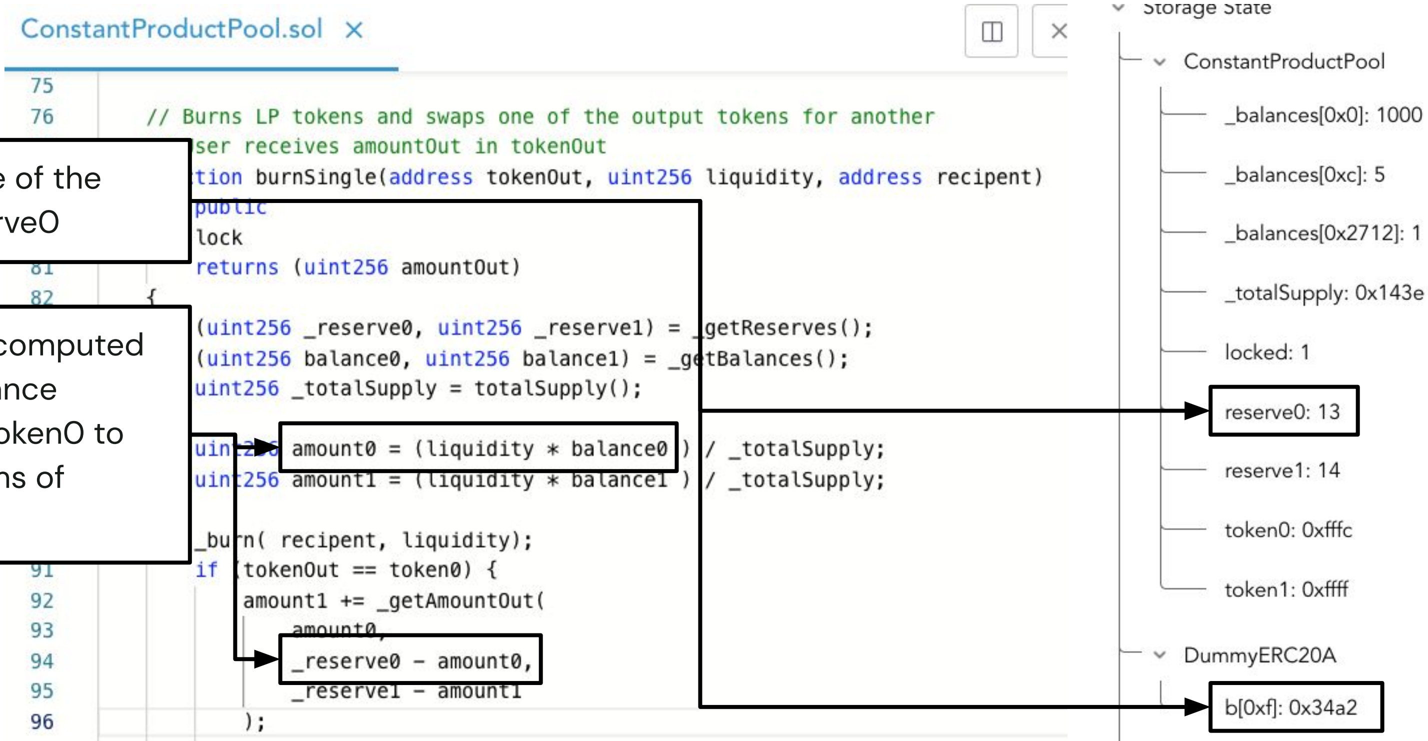
1. Burning 5 out of 0x143e tokens for only token1
2. How can one get all the reserve?

certora

Step 3: Understand pre-state and code

1. Token0 balance of the system >> reserve0

2. Amount out is computed in terms of balance
But swapping token0 to token1 is in terms of reserve



ConstantProductPool.sol

```
    // Burns LP tokens and swaps one of the output tokens for another
       ser receives amountOut in tokenOut
       tion burnSingle(address tokenOut, uint256 liquidity, address recipent)
       public
       lock
       returns (uint256 amountOut)
    {
       (uint256 _reserve0, uint256 _reserve1) = _getReserves();
       (uint256 balance0, uint256 balance1) = _getBalances();
       uint256 _totalSupply = totalSupply();

       uint256 amount0 = (liquidity * balance0 ) / _totalSupply;
       uint256 amount1 = (liquidity * balance1 ) / _totalSupply;

       _burn( recipent, liquidity);
       if (tokenOut == token0) {
           amount1 += _getAmountOut(
               amount0,
               _reserve0 - amount0,
               _reserve1 - amount1
           );
```

Storage State
- ConstantProductPool
  - _balances[0x0]: 1000
  - _balances[0xc]: 5
  - _balances[0x2712]: 1
  - _totalSupply: 0x143e
  - locked: 1
  - reserve0: 13
  - reserve1: 14
  - token0: 0xfffc
  - token1: 0xffff
- DummyERC20A
  - b[0xf]: 0x34a2

certora

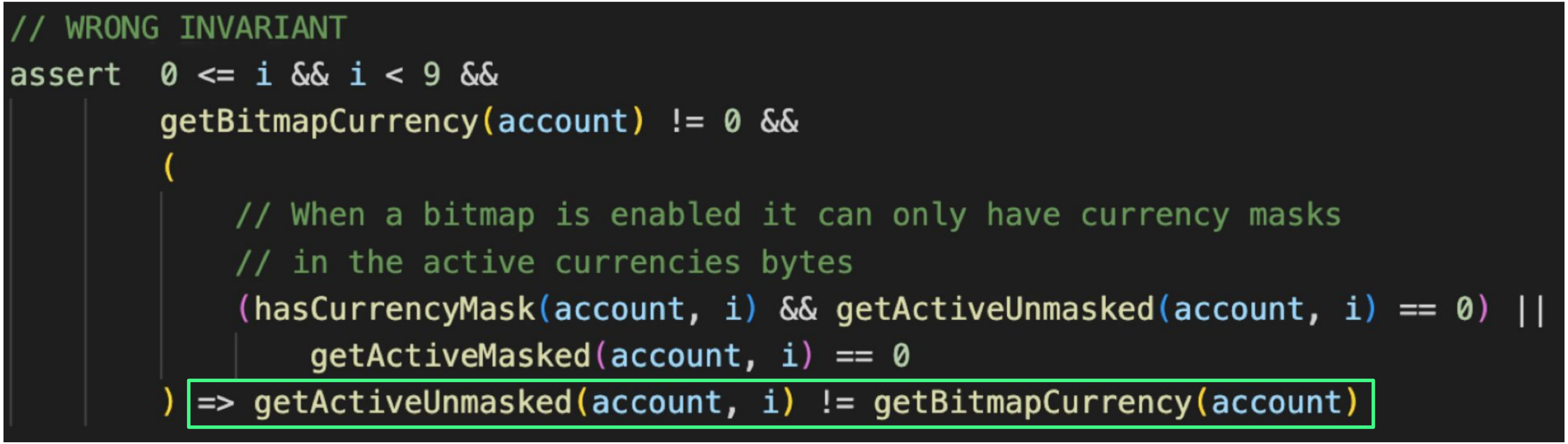
What happens when the tool successfully verifies the code?

Tautological rule example – Notional

Notional
@NotionalFinance

The white hat who reported the bug will receive $1 million USD and a 100,000 NOTE bonus for their efforts and our bounty program will continue.

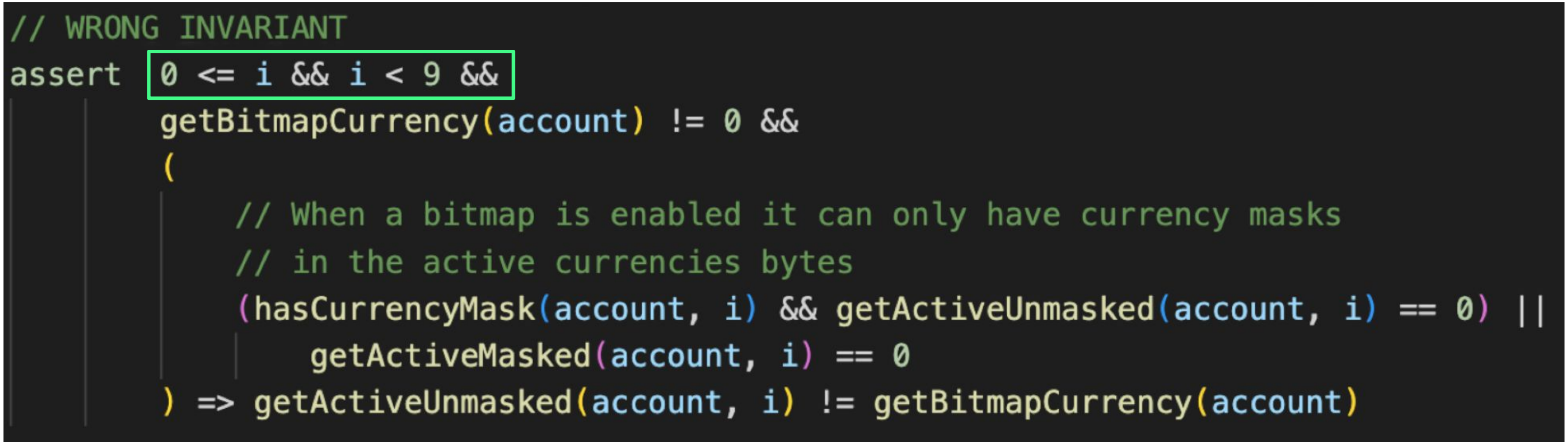
Tautological rule example – Notional



```
// WRONG INVARIANT
assert  0 <= i && i < 9 &&
        getBitmapCurrency(account) != 0 &&
        (
            // When a bitmap is enabled it can only have currency masks
            // in the active currencies bytes
            (hasCurrencyMask(account, i) && getActiveUnmasked(account, i) == 0) ||
                getActiveMasked(account, i) == 0
        ) => getActiveUnmasked(account, i) != getBitmapCurrency(account)
```

An asset cannot be both **bitmap** and **active**

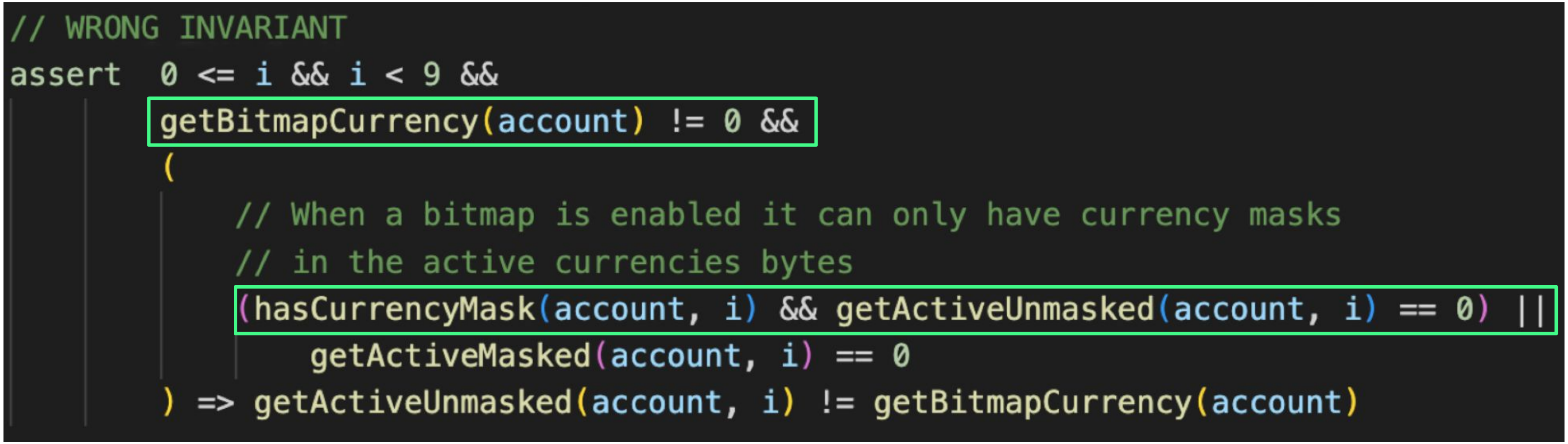
Tautological rule example – Notional



```
// WRONG INVARIANT
assert  0 <= i && i < 9 &&
        getBitmapCurrency(account) != 0 &&
        (
            // When a bitmap is enabled it can only have currency masks
            // in the active currencies bytes
            (hasCurrencyMask(account, i) && getActiveUnmasked(account, i) == 0) ||
                getActiveMasked(account, i) == 0
        ) => getActiveUnmasked(account, i) != getBitmapCurrency(account)
```

An asset cannot be both **bitmap** and **active**

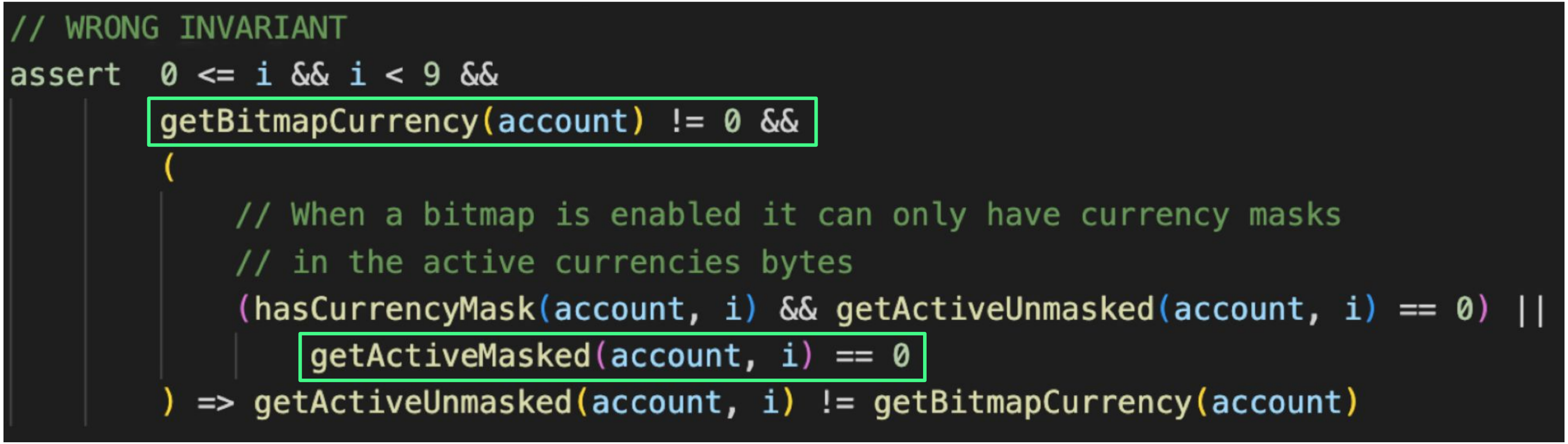
Tautological rule example – Notional



```
// WRONG INVARIANT
assert  0 <= i && i < 9 &&
        getBitmapCurrency(account) != 0 &&
        (
            // When a bitmap is enabled it can only have currency masks
            // in the active currencies bytes
            (hasCurrencyMask(account, i) && getActiveUnmasked(account, i) == 0) ||
                getActiveMasked(account, i) == 0
        ) => getActiveUnmasked(account, i) != getBitmapCurrency(account)
```

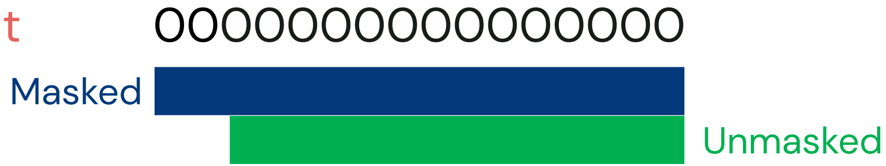
If the bitmap currency is not zero, and the active currency is zero, then the bitmap and active currencies are different

Tautological rule example – Notional

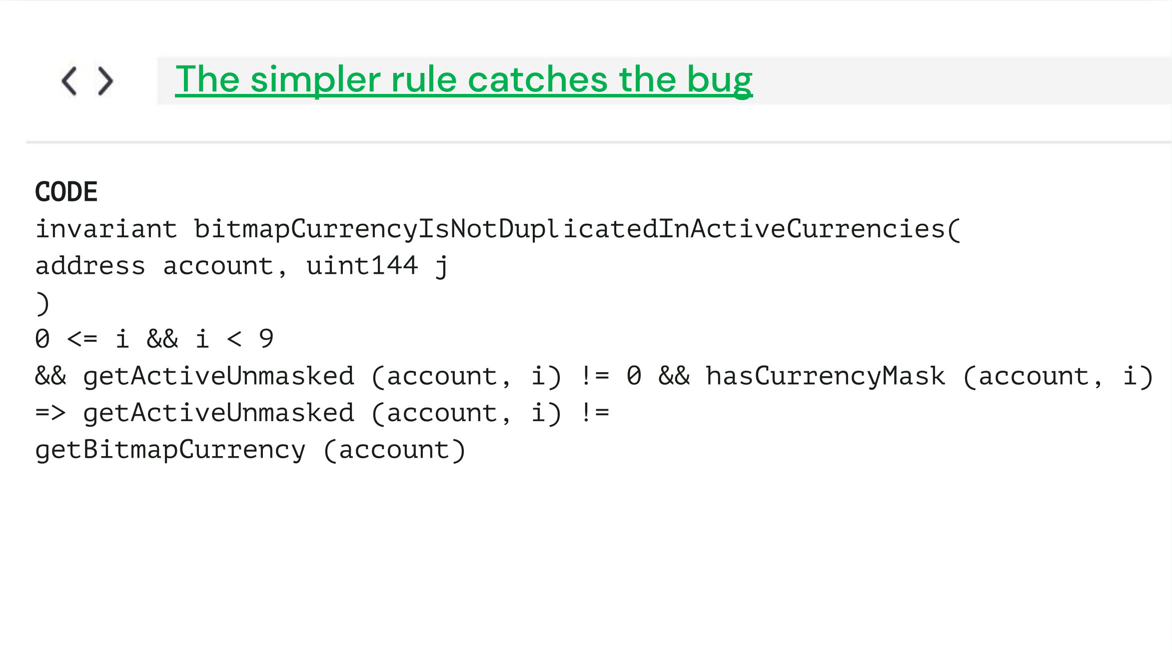


```
// WRONG INVARIANT
assert  0 <= i && i < 9 &&
        getBitmapCurrency(account) != 0 &&
        (
            // When a bitmap is enabled it can only have currency masks
            // in the active currencies bytes
            (hasCurrencyMask(account, i) && getActiveUnmasked(account, i) == 0) ||
                getActiveMasked(account, i) == 0
        ) => getActiveUnmasked(account, i) != getBitmapCurrency(account)
```

Same tautological statement



Less is more



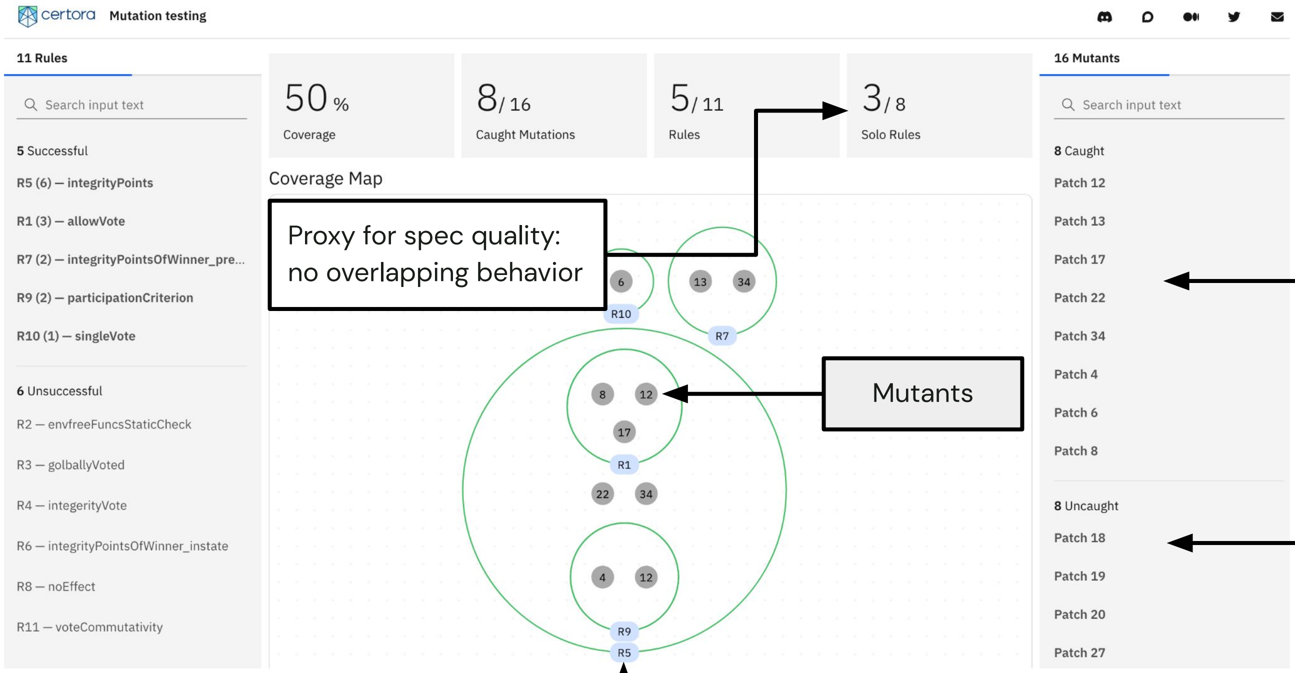
The simpler rule catches the bug

CODE

```
invariant bitmapCurrencyIsNotDuplicatedInActiveCurrencies(
address account, uint144 j
)
0 <= i && i < 9
&& getActiveUnmasked (account, i) != 0 && hasCurrencyMask (account, i)
=> getActiveUnmasked (account, i) !=
getBitmapCurrency (account)
```

certora

Mutating Testing for Improving Spec(Chandra Nandi)



certora **Mutation testing**

11 Rules

Search input text

5 Successful

R5 (6) — integrityPoints

R1 (3) — allowVote

R7 (2) — integrityPointsOfWinner_pre...

R9 (2) — participationCriterion

R10 (1) — singleVote

6 Unsuccessful

R2 — envfreeFuncsStaticCheck

R3 — golballyVoted

R4 — integerityVote

R6 — integrityPointsOfWinner_instate

R8 — noEffect

R11 — voteCommutativity

50 %	8/ 16	5/ 11	3/ 8
Coverage	Caught Mutations	Rules	Solo Rules

Coverage Map

16 Mutants

Search input text

8 Caught

Patch 12

Patch 13

Patch 17

Patch 22

Patch 34

Patch 4

Patch 6

Patch 8

8 Uncaught

Patch 18

Patch 19

Patch 20

Patch 27

Proxy for spec quality: no overlapping behavior

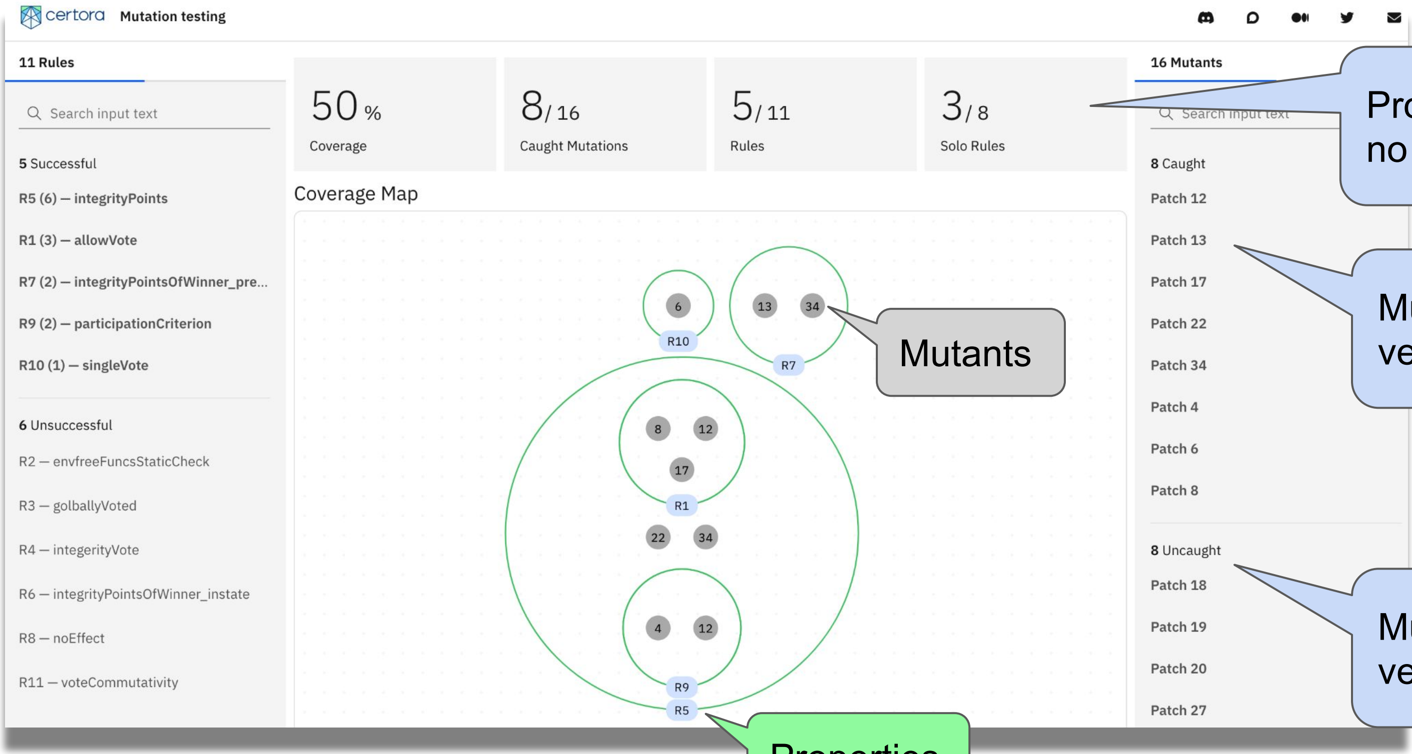
Mutants

Mutants that fail verification

Mutants that pass verification

Properties

Gambit: Mutating Testing for Improving Spec



certora **Mutation testing**

11 Rules

Search input text

5 Successful

R5 (6) — integrityPoints

R1 (3) — allowVote

R7 (2) — integrityPointsOfWinner_pre...

R9 (2) — participationCriterion

R10 (1) — singleVote

6 Unsuccessful

R2 — envfreeFuncsStaticCheck

R3 — golballyVoted

R4 — integerityVote

R6 — integrityPointsOfWinner_instate

R8 — noEffect

R11 — voteCommutativity

50 %	8/ 16	5/ 11	3/ 8
Coverage	Caught Mutations	Rules	Solo Rules

Coverage Map

16 Mutants

Search input text

8 Caught

Patch 12

Patch 13

Patch 17

Patch 22

Patch 34

Patch 4

Patch 6

Patch 8

8 Uncaught

Patch 18

Patch 19

Patch 20

Patch 27

Proxy for spec quality: no overlapping behavior

Mutants

Mutants that failed verification

Mutants that passed verification

Properties

UNSAT Core

- For Unsat formulas SMT solver generates a minimal Unsat subset of clauses
- Generalizes tautology and vacuity checking
- Utilized to check specs
- Low level code is a challenge

UNSAT Cores - Mapping CVL -> TAC

CVL

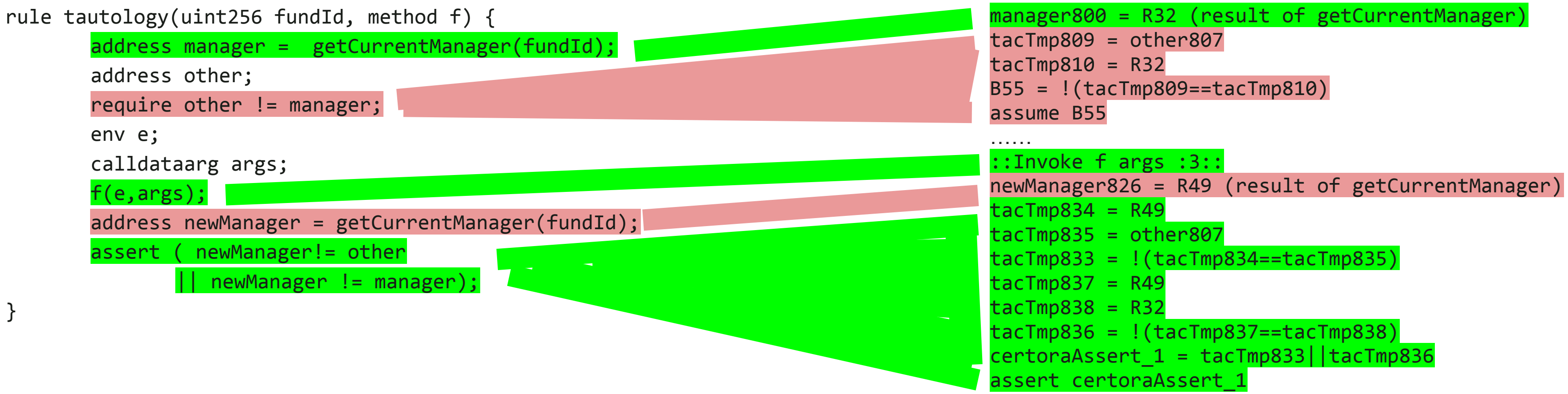
```
rule tautology(uint256 fundId, method f) {
       address manager =  getCurrentManager(fundId);
       address other;
       require other != manager;
       env e;
       calldataarg args;
       f(e,args);
       address newManager = getCurrentManager(fundId);
       assert ( newManager!= other
               || newManager != manager);
}
```

TAC

```
manager800 = R32 (result of getCurrentManager)
tacTmp809 = other807
tacTmp810 = R32
B55 = !(tacTmp809==tacTmp810)
assume B55
......
::Invoke f args :3::
newManager826 = R49 (result of getCurrentManager)
tacTmp834 = R49
tacTmp835 = other807
tacTmp833 = !(tacTmp834==tacTmp835)
tacTmp837 = R49
tacTmp838 = R32
tacTmp836 = !(tacTmp837==tacTmp838)
certoraAssert_1 = tacTmp833||tacTmp836
assert certoraAssert_1
```

certora

UNSAT Cores - Mapping CVL -> TAC



CVL

```
rule tautology(uint256 fundId, method f) {
        address manager =  getCurrentManager(fundId);
        address other;
        require other != manager;
        env e;
        calldataarg args;
        f(e,args);
        address newManager = getCurrentManager(fundId);
        assert ( newManager!= other
                || newManager != manager);
}
```

TAC

```
manager800 = R32 (result of getCurrentManager)
tacTmp809 = other807
tacTmp810 = R32
B55 = !(tacTmp809==tacTmp810)
assume B55
......
::Invoke f args :3::
newManager826 = R49 (result of getCurrentManager)
tacTmp834 = R49
tacTmp835 = other807
tacTmp833 = !(tacTmp834==tacTmp835)
tacTmp837 = R49
tacTmp838 = R32
tacTmp836 = !(tacTmp837==tacTmp838)
certoraAssert_1 = tacTmp833||tacTmp836
assert certoraAssert_1
```

certora

UNSAT Cores - Mapping TAC -> SMT

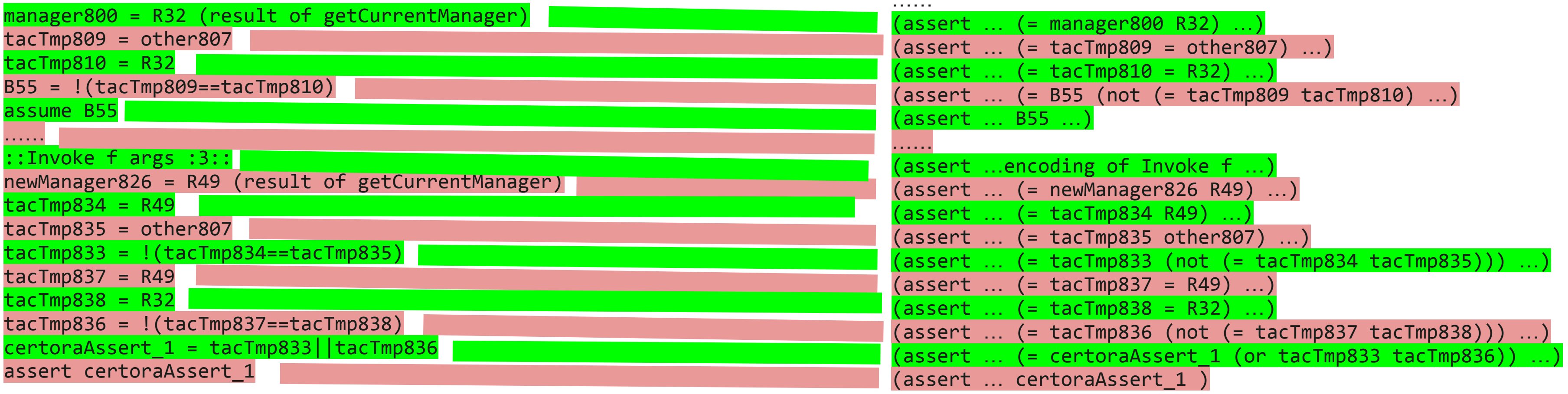
TAC

```
manager800 = R32 (result of getCurrentManager)
tacTmp809 = other807
tacTmp810 = R32
B55 = !(tacTmp809==tacTmp810)
assume B55
......
::Invoke f args :3::
newManager826 = R49 (result of getCurrentManager)
tacTmp834 = R49
tacTmp835 = other807
tacTmp833 = !(tacTmp834==tacTmp835)
tacTmp837 = R49
tacTmp838 = R32
tacTmp836 = !(tacTmp837==tacTmp838)
certoraAssert_1 = tacTmp833||tacTmp836
assert certoraAssert_1
```

SMT

```
(set-logic QF_UFDTLIA)
......
......
(assert ... (= manager800 R32) ...)
(assert ... (= tacTmp809 = other807) ...)
(assert ... (= tacTmp810 = R32) ...)
(assert ... (= B55 (not (= tacTmp809 tacTmp810) ... )
(assert ... B55 ...)
......
(assert ...encoding of Invoke f ...)
(assert ... (= newManager826 R49) ...)
(assert ... (= tacTmp834 R49) ...)
(assert ... (= tacTmp835 other807) ...)
(assert ... (= tacTmp833 (not (= tacTmp834 tacTmp835))) ...)
(assert ... (= tacTmp837 = R49) ...)
(assert ... (= tacTmp838 = R32) ...)
(assert ... (= tacTmp836 (not (= tacTmp837 tacTmp838))) ...)
(assert ... (= certoraAssert_1 (or tacTmp833 tacTmp836)) ...)
(assert ... certoraAssert_1 )
......
(check-sat)
(get-unsat-core)
```

UNSAT Cores - Mapping TAC -> SMT



TAC

```
manager800 = R32 (result of getCurrentManager)
tacTmp809 = other807
tacTmp810 = R32
B55 = !(tacTmp809==tacTmp810)
assume B55
......
::Invoke f args :3::
newManager826 = R49 (result of getCurrentManager)
tacTmp834 = R49
tacTmp835 = other807
tacTmp833 = !(tacTmp834==tacTmp835)
tacTmp837 = R49
tacTmp838 = R32
tacTmp836 = !(tacTmp837==tacTmp838)
certoraAssert_1 = tacTmp833||tacTmp836
assert certoraAssert_1
```

SMT

```
(set-logic QF_UFDTLIA)
......
......
(assert … (= manager800 R32) …)
(assert … (= tacTmp809 = other807) …)
(assert … (= tacTmp810 = R32) …)
(assert … (= B55 (not (= tacTmp809 tacTmp810) …)
(assert … B55 …)
......
(assert …encoding of Invoke f …)
(assert … (= newManager826 R49) …)
(assert … (= tacTmp834 R49) …)
(assert … (= tacTmp835 other807) …)
(assert … (= tacTmp833 (not (= tacTmp834 tacTmp835))) …)
(assert … (= tacTmp837 = R49) …)
(assert … (= tacTmp838 = R32) …)
(assert … (= tacTmp836 (not (= tacTmp837 tacTmp838))) …)
(assert … (= certoraAssert_1 (or tacTmp833 tacTmp836)) …)
(assert … certoraAssert_1 )
......
(check-sat)
(get-unsat-core)
```

certora

UNSAT Cores – SMT Solving

SMT

```
(set-logic QF_UFDTLIA)
......
......
(assert … (= manager800 R32) …)
(assert … (= tacTmp809 = other807) …)
(assert … (= tacTmp810 = R32) …)
(assert … (= B55 (not (= tacTmp809 tacTmp810) …)
(assert … B55 …)
......
(assert …encoding of Invoke f …)
(assert … (= newManager826 R49) …)
(assert … (= tacTmp834 R49) …)
(assert … (= tacTmp835 other807) …)
(assert … (= tacTmp833 (not (= tacTmp834 tacTmp835))) …)
(assert … (= tacTmp837 = R49) …)
(assert … (= tacTmp838 = R32) …)
(assert … (= tacTmp836 (not (= tacTmp837 tacTmp838))) …)
(assert … (= certoraAssert_1 (or tacTmp833 tacTmp836)) …)
(assert … certoraAssert_1 …)
......
(check-sat)
(get-unsat-core)
```

UNSAT Cores - SMT Solving

SMT

```
(set-logic QF_UFDTLIA)
......
......
(assert ... (= manager800 R32) ...)
(assert ... (= tacTmp809 = other807) ...)
(assert ... (= tacTmp810 = R32) ...)
(assert ... (= B55 (not (= tacTmp809 tacTmp810) ...)
(assert ... B55 ...)
......
(assert ...encoding of Invoke f ...)
(assert ... (= newManager826 R49) ...)
(assert ... (= tacTmp834 R49) ...)
(assert ... (= tacTmp835 other807) ...)
(assert ... (= tacTmp833 (not (= tacTmp834 tacTmp835))) ...)
(assert ... (= tacTmp837 = R49) ...)
(assert ... (= tacTmp838 = R32) ...)
(assert ... (= tacTmp836 (not (= tacTmp837 tacTmp838))) ...)
(assert ... (= certoraAssert_1 (or tacTmp833 tacTmp836)) ...)
(assert ... certoraAssert_1 ...)
......
(check-sat)
(get-unsat-core)
```

UNSAT

UNSAT Cores – SMT Solving

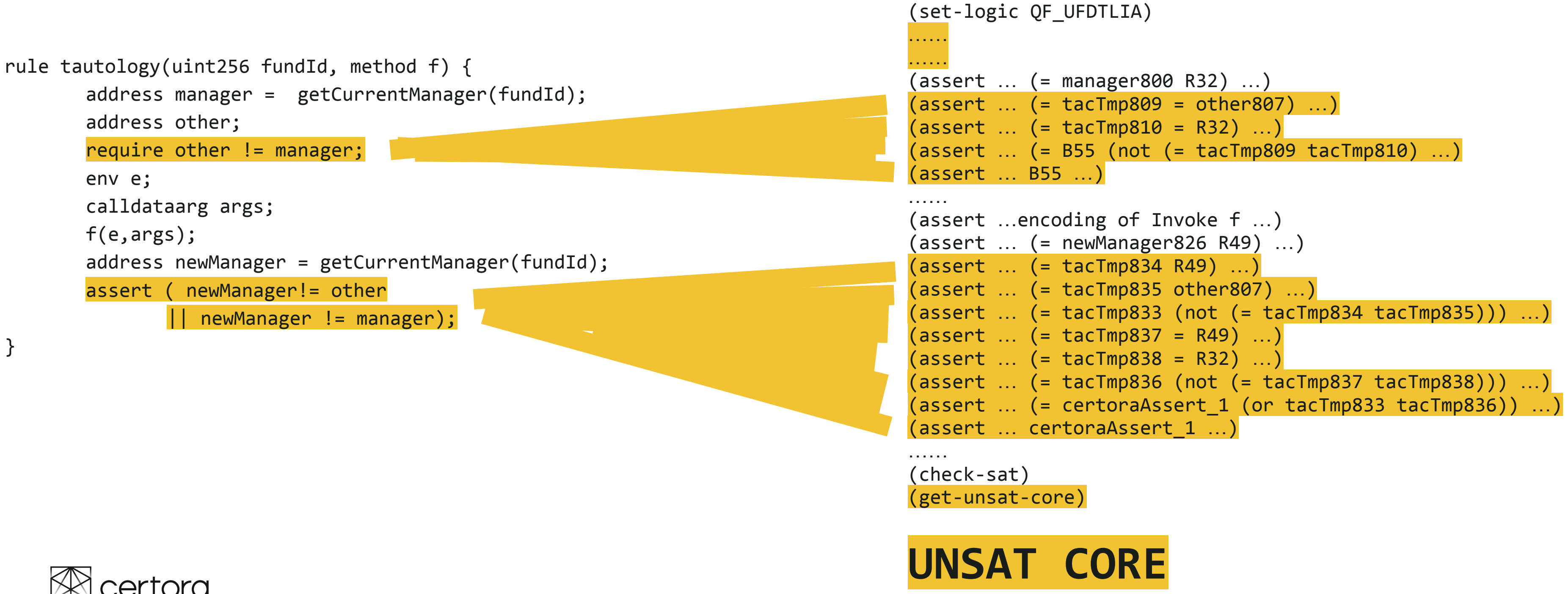
SMT

```
(set-logic QF_UFDTLIA)
......
......
(assert … (= manager800 R32) …)
(assert … (= tacTmp809 = other807) …)
(assert … (= tacTmp810 = R32) …)
(assert … (= B55 (not (= tacTmp809 tacTmp810) …)
(assert … B55 …)
......
(assert …encoding of Invoke f …)
(assert … (= newManager826 R49) …)
(assert … (= tacTmp834 R49) …)
(assert … (= tacTmp835 other807) …)
(assert … (= tacTmp833 (not (= tacTmp834 tacTmp835))) …)
(assert … (= tacTmp837 = R49) …)
(assert … (= tacTmp838 = R32) …)
(assert … (= tacTmp836 (not (= tacTmp837 tacTmp838))) …)
(assert … (= certoraAssert_1 (or tacTmp833 tacTmp836)) …)
(assert … certoraAssert_1 …)
......
(check-sat)
(get-unsat-core)
```

UNSAT CORE

certora

UNSAT Cores - BackMapping CVL <- TAC <- SMT



CVL

```
rule tautology(uint256 fundId, method f) {
        address manager =  getCurrentManager(fundId);
        address other;
        require other != manager;
        env e;
        calldataarg args;
        f(e,args);
        address newManager = getCurrentManager(fundId);
        assert ( newManager!= other
                || newManager != manager);
}
```

SMT

```
(set-logic QF_UFDTLIA)
......
......
(assert … (= manager800 R32) …)
(assert … (= tacTmp809 = other807) …)
(assert … (= tacTmp810 = R32) …)
(assert … (= B55 (not (= tacTmp809 tacTmp810) …)
(assert … B55 …)
......
(assert …encoding of Invoke f …)
(assert … (= newManager826 R49) …)
(assert … (= tacTmp834 R49) …)
(assert … (= tacTmp835 other807) …)
(assert … (= tacTmp833 (not (= tacTmp834 tacTmp835))) …)
(assert … (= tacTmp837 = R49) …)
(assert … (= tacTmp838 = R32) …)
(assert … (= tacTmp836 (not (= tacTmp837 tacTmp838))) …)
(assert … (= certoraAssert_1 (or tacTmp833 tacTmp836)) …)
(assert … certoraAssert_1 …)
......
(check-sat)
(get-unsat-core)
```

UNSAT CORE

UNSAT Cores - Vacuous Spec

CVL

```
rule tautology(uint256 fundId, method f) {
       address manager =  getCurrentManager(fundId);
       address other;
       require other != manager;
       env e;
       calldataarg args;
       f(e,args);
       address newManager = getCurrentManager(fundId);
       assert ( newManager!= other
              || newManager != manager);
}
```

certora

UNSAT Cores – Vacuous Spec

CVL

```
rule tautology(uint256 fundId, method f) {
        address manager =  getCurrentManager(fundId);
        address other;
        require other != manager;
        env e;
        calldataarg args;
        f(e,args);
        address newManager = getCurrentManager(fundId);
        assert ( newManager!= other
                || newManager != manager);
}
```

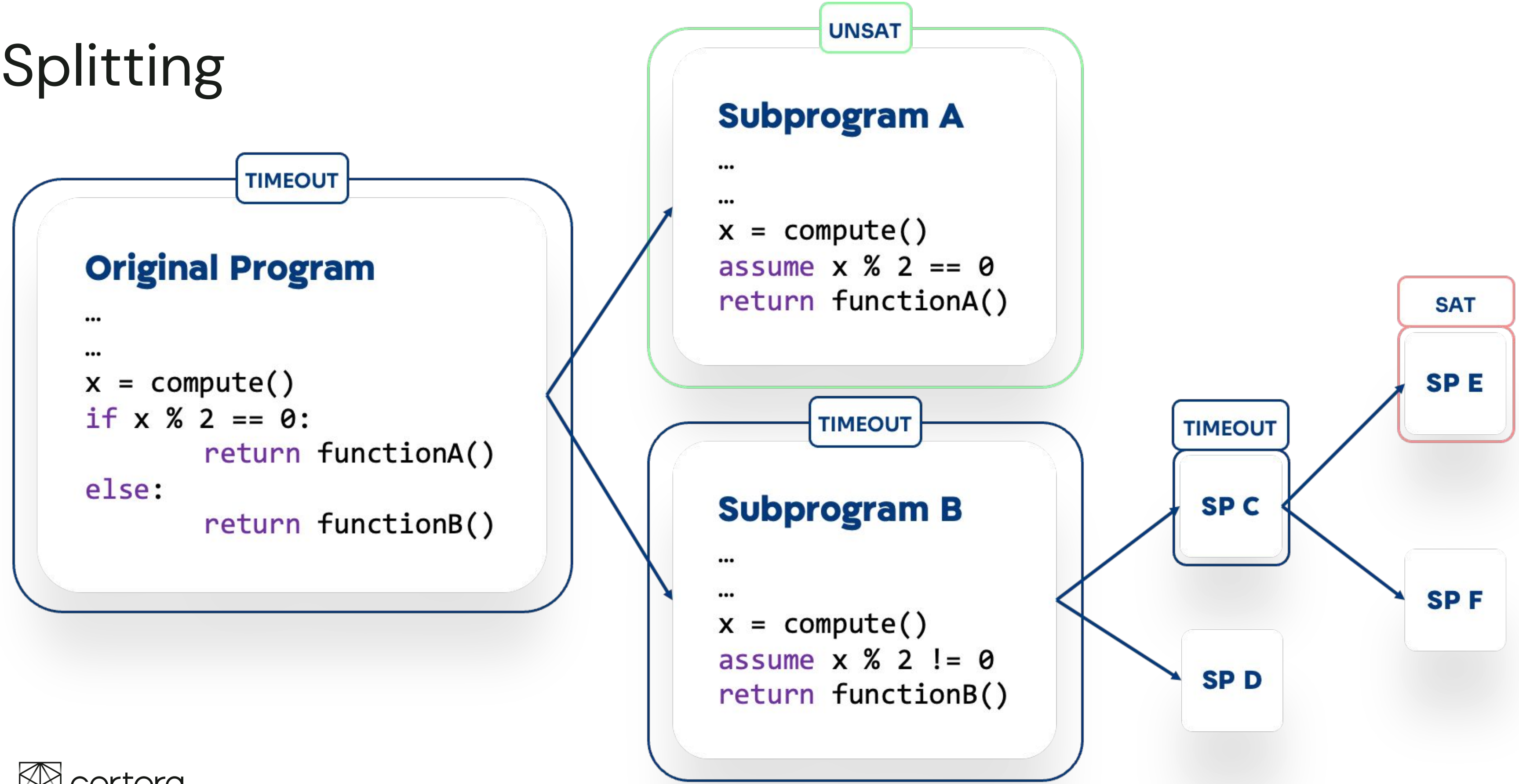
Irrelevant function calls and variable assignments!!!

certora

Mitigating SMT Complexity

certora

Splitting



TIMEOUT

Original Program

```
…
…
x = compute()
if x % 2 == 0:
        return functionA()
else:
        return functionB()
```

UNSAT

Subprogram A

```
…
…
x = compute()
assume x % 2 == 0
return functionA()
```

TIMEOUT

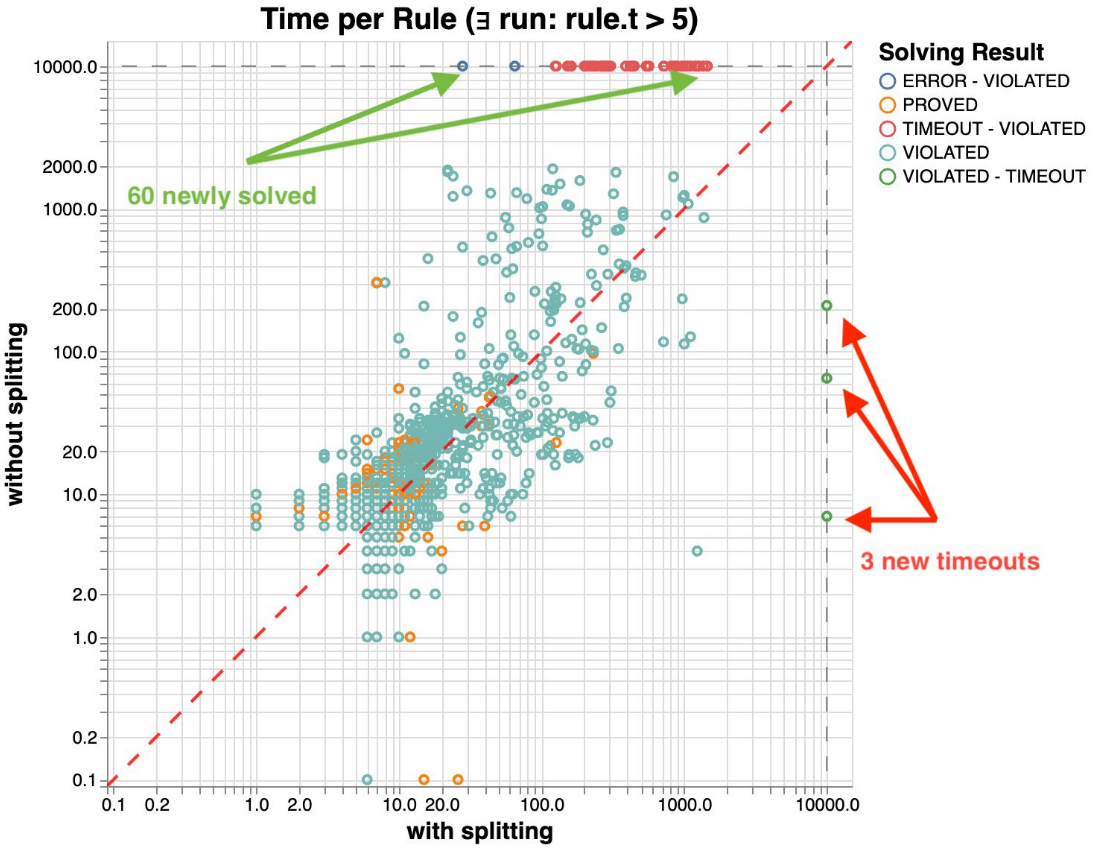
Subprogram B

```
…
…
x = compute()
assume x % 2 != 0
return functionB()
```

certora

Splitting

Experimental Evaluation



60 newly solved

3 new timeouts

certora

Custom Over Approximation and Axiomatisation

Basic Encodings

- Precise Nonlinear Integer Arithmetic (NIA)
- Over-approximating Linear Integer Arithmetic (LIA)
- We run a portfolio of LIA and NIA encodings with many solvers in parallel

Example LIA Multiplication Axiomatisation

a∗b modelled with an uninterpreted function a·b with axioms:

$a \cdot 0 = 0$ $\quad$ $a > 0, b > 0 \Longrightarrow a \cdot b > 0$ $\quad$ $a > 0, b > 0 \Longrightarrow a \cdot b \geq a, a \cdot b \geq b$

$a \cdot b = b \cdot a$ $\quad$ $a > 0, b < 0 \Longrightarrow a \cdot b < 0$ $\quad$ …and many other

Domain Specific Overflow Axiom Example

(a>0 && b>0) => ((a*b % 2^256)/a == b <=> a*b < 2^256)

Assumption from the program

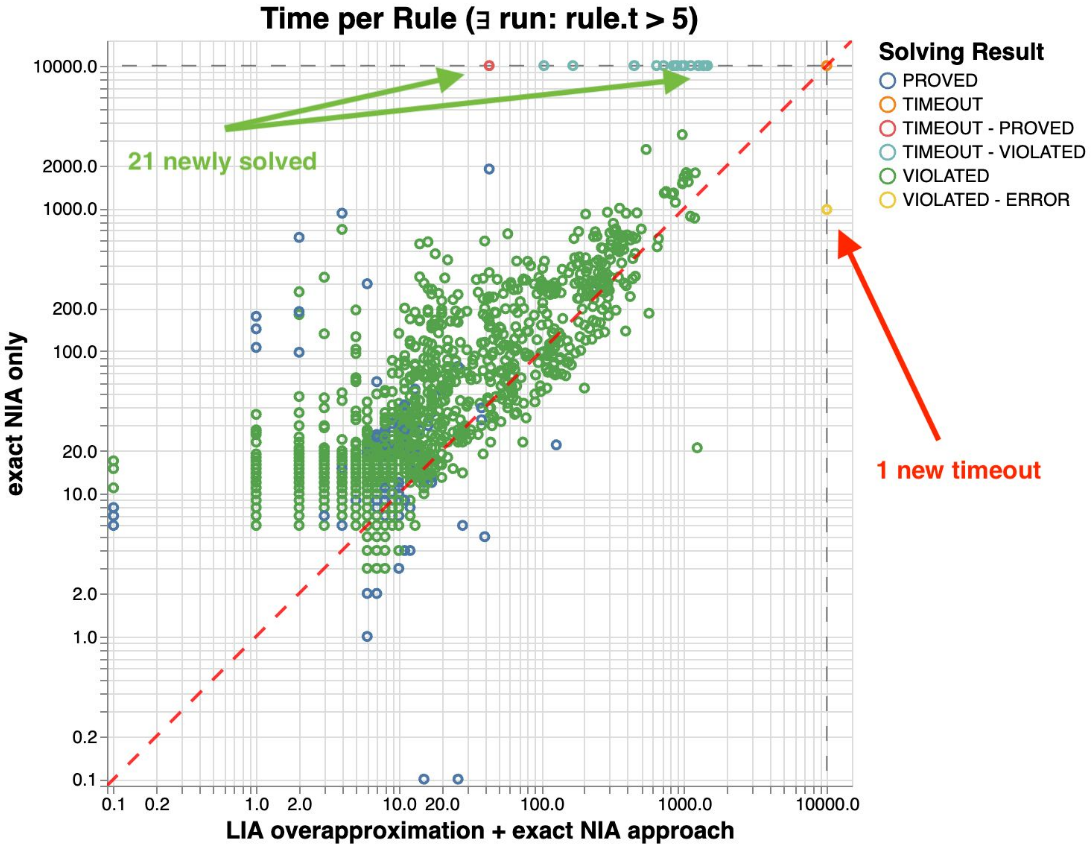
Equivality hint for the solver

To verify: no overflow

(2^256 −1 = max uint)

LIA + NIA Portfolio

Experimental Evaluation



certora

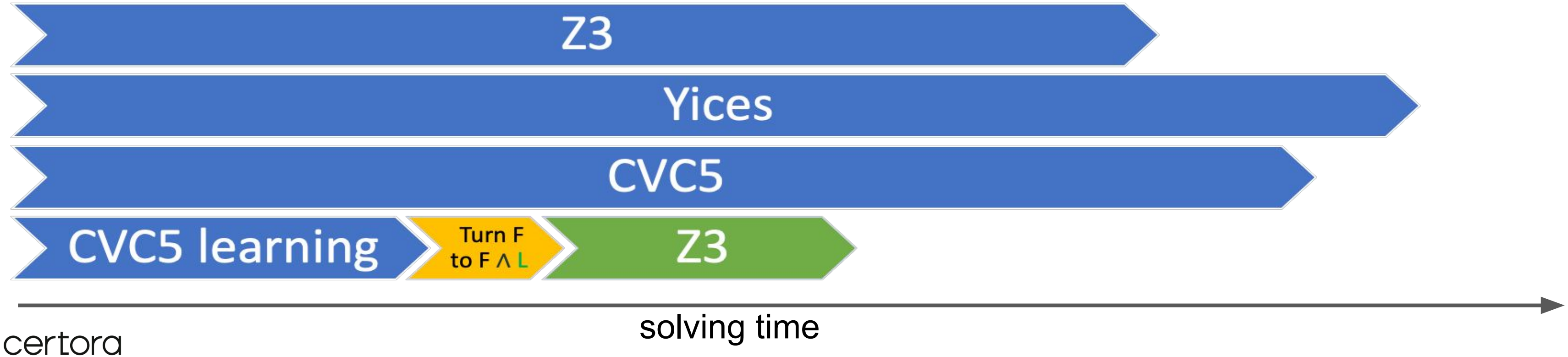
Learned Lemmas Sharing

Given a formula F, an SMT solver says:

- F is SAT, or
- F is UNSAT, or
- Timeout, but learned F => L for some L

Example

$$
\begin{aligned}
L \equiv\ & (x = 5)\ \land \\
& (y <= 10\ \lor\ y > 20\ )\ \land \\
& (y < 100)\ \land \\
& (z = x\ \lor\ z > 10)
\end{aligned}
$$



Grounding Quantifiers (Basic Idea)

Let “assume forall x. f(x) = 0” appear in the verification condition (VC)

Natural SMT Encoding

- via universal quantification

Our Encoding

- Collect all instances of f appearing in the VC, say f(1), f(x), f(y + 2)
- Replace assume forall x. f(x) = 0 in the VC with:
    - assume f(1) = 0
    - assume f(x) = 0
    - assume f(y + 2) = 0

certora

AI & SMT better together

- Memory in EVM is a monolithic array of bytes
- SMT must infer facts about non-aliasing
- Certora Prover performs a pre-processing of bytecode
    - Ensure non-interference between accesses to memory regions
    - Resolve external function calls which improves precision
    - Simplify the SMT formulae, lowers burden on solvers

No Analysis (imprecise)	No mem-splitting rewrites (slow)
`149/5348` tasks had spurious counterexamples	upto `42` min slowdown in SMT solving time

unresolved external function calls treated as “havocs”

e.g., due to complex / too many read-over-write axioms

Take Away

1. Formal verification is useful for DeFi
    a. Developers and security researchers can write specifications
    b. Code is law
    c. Ratio of lines per $$$
    d. Code is tricky
    e. Bugs have huge costs
    f. Security budget is high
2. But more is needed
    a. Specifications
    b. Static Analysis
    c. SMT
    d. Programming Language design
3. Certora takes a first step

certora

More Take Away(Jaroslav Bendik)

1. LIA is sometimes good over-approximation for NIA
    a. The program contains non-linear but the specs do not
2. Unsat-cores are key to spec quality checking
3. Array theory & bitvectors drastically complicate reasoning
4. Most interesting specs initially time-out
    a. Sometimes solved with user provided summaries
    b. Abstract interpretation is a key
5. Domain specific axioms can make SMT faster
    a. Overflow checking
6. Memory analysis is key
    a. Eliminate low level storage

Myths And Reality About Formal Verification

Myths	Reality
FV can only prove absence of bugs	Biggest value of FV is finding bugs
Hardest problem is computational	Hardest problem is specification
FV produces bullet-proof code	FV drastically improves code security
FV replaces auditing	FV improves auditing
FV comes last as a one-time deal	FV comes first and guarantees code upgrade safety

Conclusion

Bug finding is hard

- Auditing is good but not enough
- Fuzzing from initial state until....

Static analysis and SMT better together

- Static analysis reduces SMT complexity
- SMT eliminates false alarms by static analysis

Automatic Formal Verification

- Arbitrary state
- Finds tricky bugs
- Proves high level properties of low level code
- Arbitrary programming style