certora

BAD PROOFS IN FORMAL VERIFICATION

Uri Kirstein Software Engineer and Developer Relations

BUGS AFTER FORMAL VERIFICATION

Critical Bug Payout Report

Jeff Wu
07 Jan 2022

Notional received a critical bug report from a whitehat hacker last night. The Notional team disabled the affected code in under an hour after it was reported. No user funds have been lost, and none are at risk. No user-facing functionality is affected. Users can continue to use Notional to lend and borrow at fixed rates safely. We have created a fix for this issue, and will deploy the change once our auditors have reviewed and confirmed it. Notional Finance Incorporated will pay the full $1 million bounty listed through Immunefi + a bonus of 100,000 NOTE.

Security remains our highest priority and we will continue to offer the top prize of $1 million via Immunefi for critical issues found. We submitted our code for audit by three independent audit firms. The affected code was present in all three of these audits. We also submitted our code for formal verification. The affected code was subject to a formal verification check explicitly designed to detect this particular vulnerability, but due to human error the check was not properly constructed and did not function as intended. Certora will issue a report detailing why that check did not work as intended in the coming days.

certora

LECTURE ROADMAP

What are proofs
in Formal Verification

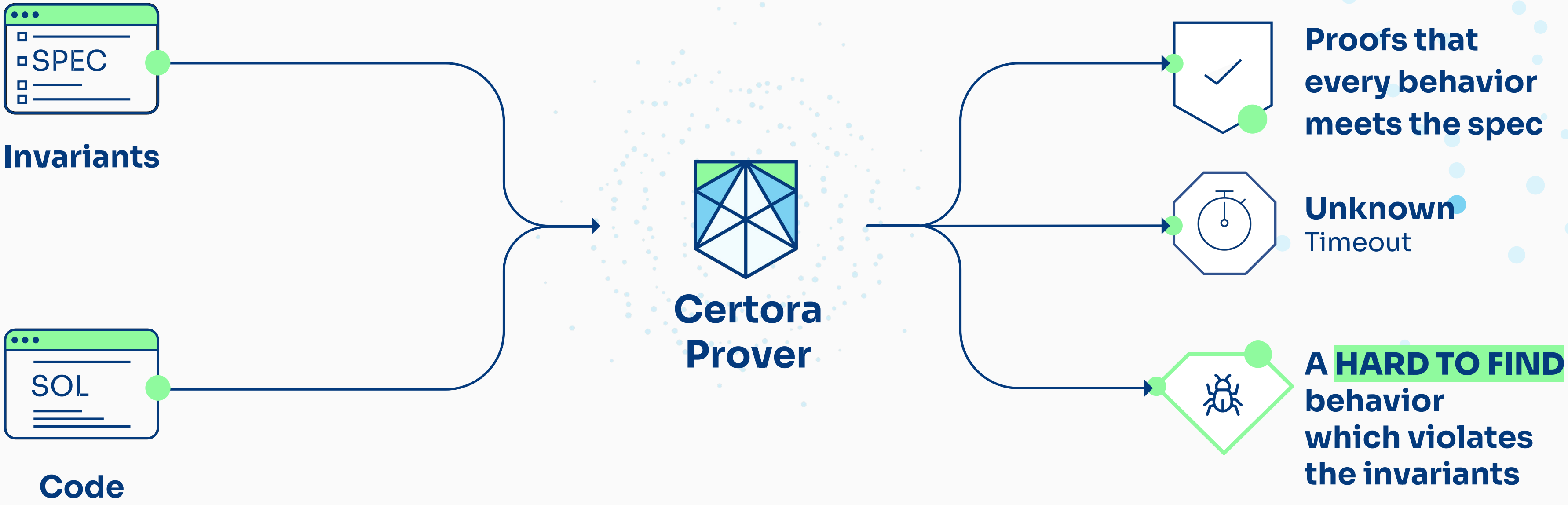
Types of
bad proofs

How to tell if
a proof is bad

Real life
example

certora

THE FORMAL VERIFICATION PROCESS



certora

SIMPLE EXAMPLE

SOLIDITY 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;
}
```

INVARIANT

$$\text{totalSupply} = \sum_{a:\,\text{address}} \text{balances}[a]$$

BUG

```
from="Alice"
to="Alice"
amount = 18
old.balances(Alice) = 20
new.balances(Alice) = 38
```

SIMPLE EXAMPLE

SOLIDITY CODE

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

INVARIANT

$$\text{totalSupply} = \sum_{a:\,\text{address}} \text{balances}[a]$$

PROOF

$$\sum_{a:\,\text{address}} \text{old.balances}[a] = \sum_{a:\,\text{address}} \text{new.balances}[a]$$

ADVANTAGES OF FORMAL VERIFICATION

Exhaustive

- Finds easy to miss bugs

Concrete counter examples

- Found bugs are verifiable

Proofs of correctness

- Hard to verify
- May be misleading!

certora

ANATOMY OF A PROPERTY

Certora Verification Language (CVL)

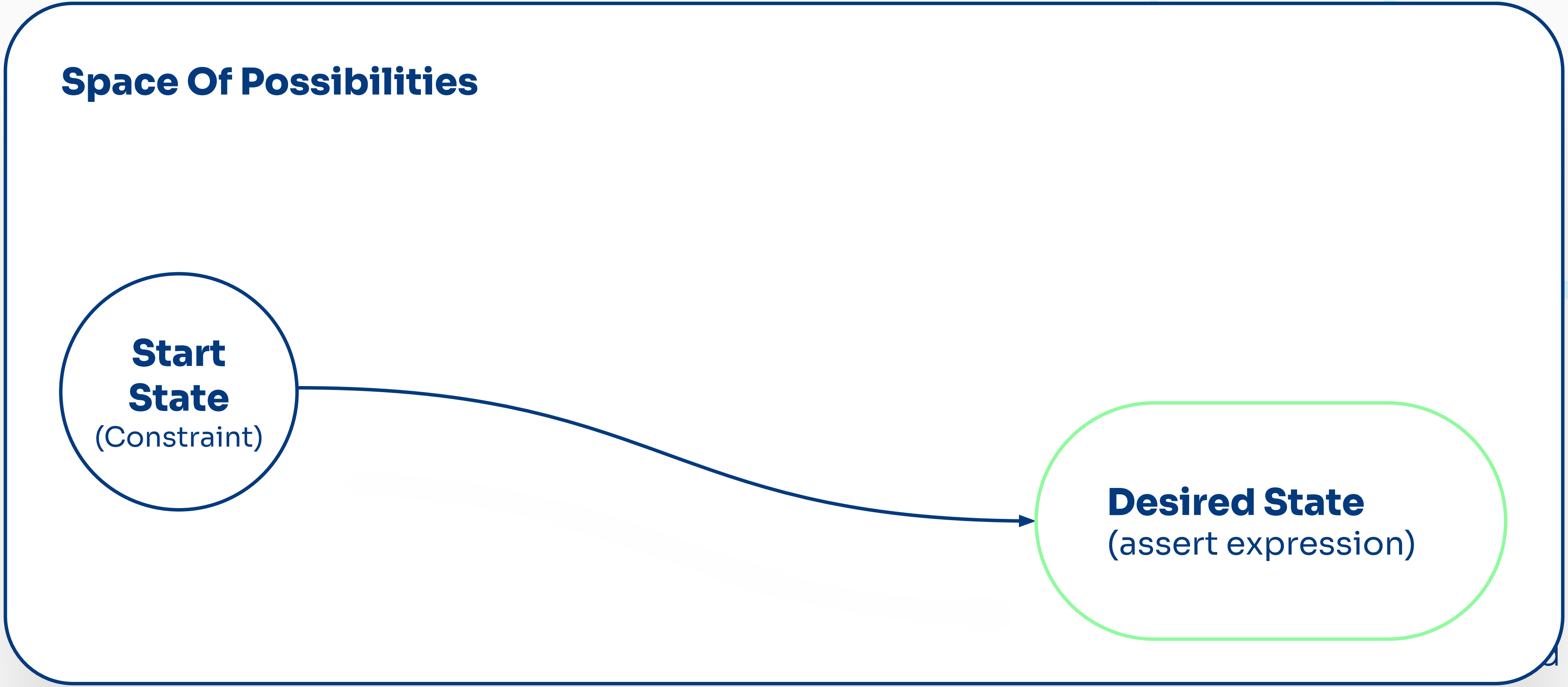
```
rule checkTransfer(address bob, uint256 amount) {
env e; /* calling context (msg.sender, block.timestamp, ... ) */
uint256 balanceBefore = balanceOf(bob);

transfer(e, bob, amount);

assert balanceOf(bob) == balanceBefore + amount;
}
```

Precondition

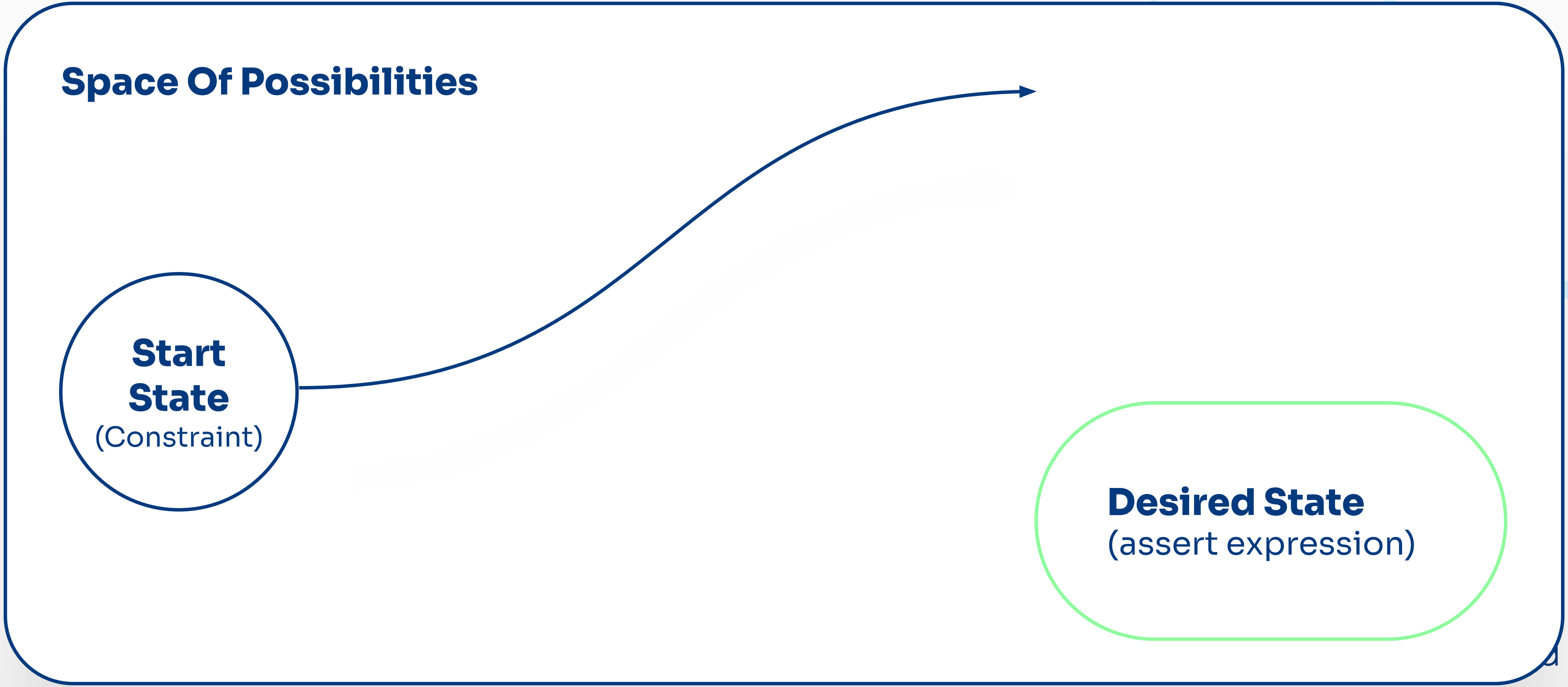
Operation

Postcondition

PROPERTY – VISUALIZATION OF WANTED BEHAVIOR



PROPERTY – A VIOLATED RULE



FALSE STATEMENTS

LOGIC DEFINES A FALSE STATEMENT AS THE EXISTENCE OF COUNTER EXAMPLE TO A CLAIM

LECTURE ROADMAP

What are proofs
in Formal Verification

Types of
bad proofs

How to tell if
a proof is bad

Real life
example

certora

DICTIONARY DEFINITION

Merriam-Webster dictionary

Vacuous:

- Empty
- Meaningless
- Lacking of significance
- Lacking contents which could or should be present

REAL LIFE EXAMPLE

Uri Kirstein - 29 years old, don’t have any children.

Statement **- If I let my children drink Colombian coffee, they will sleep better**

Logic

Given that I have no children, any statement about them is indisputable.

TRUE
(Vacuous)

REAL LIFE EXAMPLE

Uri Kirstein - 29 years old, don’t have any children.

Statement – **If I let my children drink Colombian coffee, they will sleep better**

TRUE
(Vacuous)

Statement

If I let my children drink Colombian coffee, they will not sleep at night

TRUE
(Vacuous)

PROPERTY – VISUALIZATION OF WANTED BEHAVIOR

VACUOUS RULE – CODE EXAMPLE

OpenZeppe

```
function bal
uint256 id)
override ret
    require
    "ERC115
    valid ow
return _bala
```

Certora Verification Language (CVL)

```
// If the user has a token, then the token should exist
rule held_tokens_should_exist {
  address user;
  uint256 token:
  require balanceOf(0, token) == 0;

  // This assumption was proven in a separate rule
  require balanceOf (user, token) <= totalSupplyOf (token);
  assert balanceOf (user, token) > 0 => token_exists (token);
```

VACUOUS RULE – CAN PROVE ANYTHING

Certora Verification Language (CVL)

```
// If the user has a token, then the token should exist
rule held_tokens_should_exist {
  address user;
  uint256 token:
  require balanceOf(0, token) == 0;

  assert 0 > 1;
```

VACUOUS RULES ARE A COMMON PROBLEM

"our experience has shown that typically
20% of specifications pass vacuously
during the first formal-verification runs
of a new hardware design, and that
vacuous passes always point to a real problem
in either the design or its specification or environment"

I. Beer, S. Ben-David, C. Eisner, and Y. Rodeh. Efficient detection of vacuity in ACTL formulas. Formal Methods in System Design, 18(2):141–162, 2001.

REACHABILITY CHECK

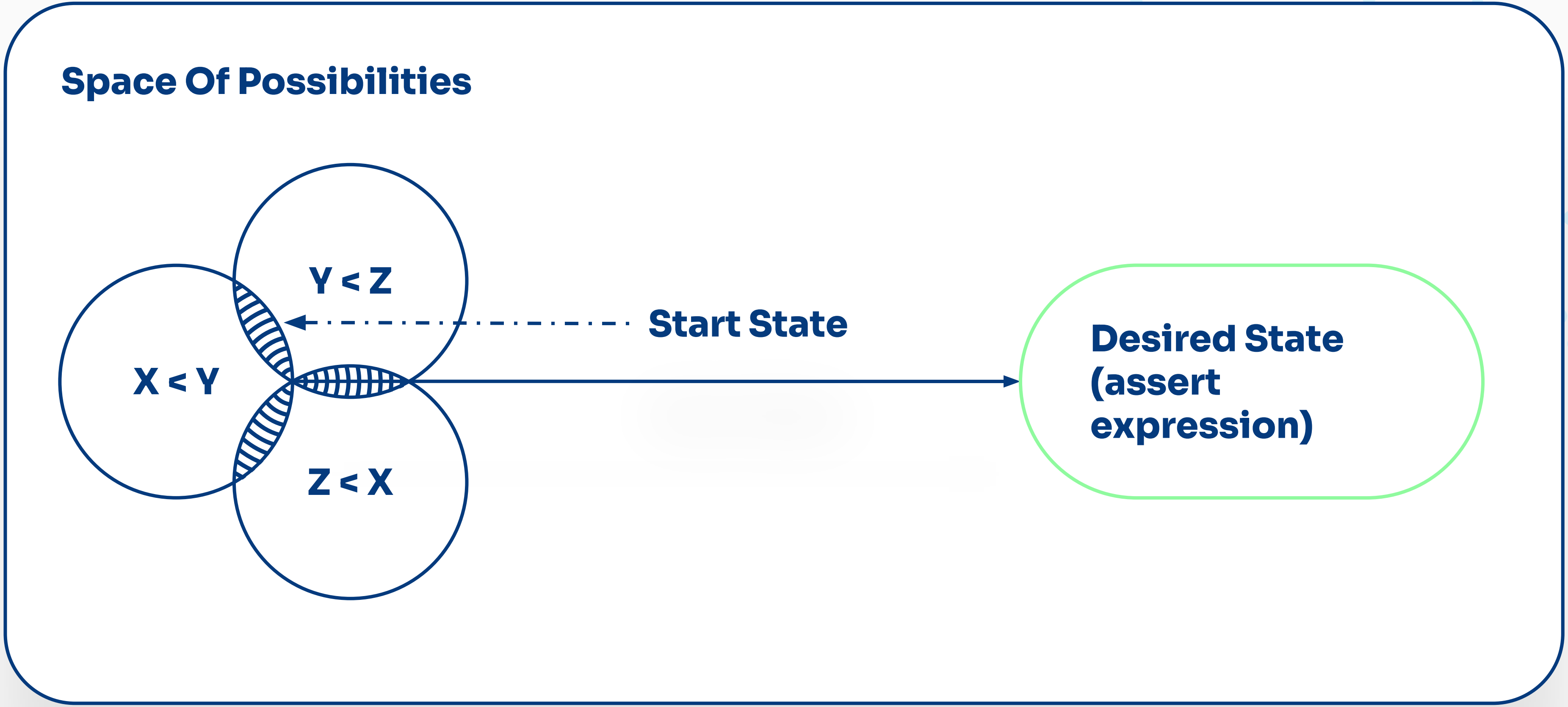
CVL

Discovering unreachability by adding assert false at the end of the rule

```
// If the rule passes, then it is vacuous
rule held_tokens should exist vacuity check {
address user;
uint256 token;
require balanceOf (0, token) == 0;

// This assumption was proven in a separate rule
require balanceOf (user, token) <= totalSupplyOf (token) ;
assert balanceOf (user, token) > 0 => token exists (token);
assert false;
```

We expect the rule to reach the assert false at the end and fail

DISJOINT PRECONDITIONS – UNREACHABILITY VISUALIZATION



VACUOUS ASSERTIONS – TAUTOLOGY DEFINITION

Wikipedia &
Oxford Dictionary

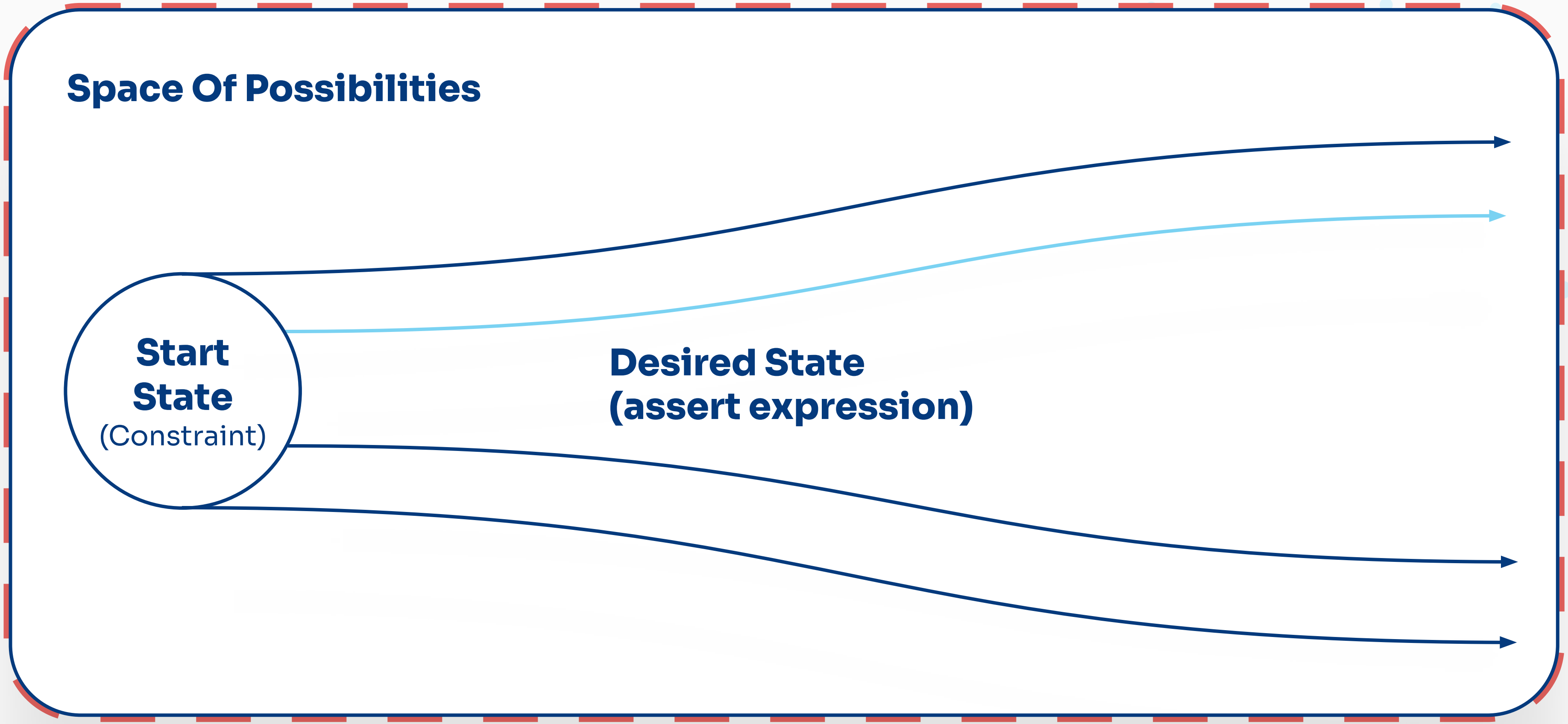
Vacuous assertions:

- The saying of the same thing twice in different words
- A propositional statement that is always true
- A formula or assertion that is true in every possible interpretation

TAUTOLOGY EXAMPLE

```
rule something_is_always_transferred {
    address recipient;
    uint256 balance_before_transfer = balanceOf (recipient) ;
    require balanceOf (recipient) == 0;

    uint256 amount;
    require amount > 0;

    transfer(recipient, amount);

    uint256 balance after transfer = balance0f(recipient);;
    assert balanceOf(recipient) <= balance after transfer;
}
```

TAUTOLOGY VISUALIZATION



FINDING TAUTOLOGIES

Remove all preconditions and the operations, then check if the rule still passes

```
rule something_is_always_transferred_vacuity_check {
    uint256 balance_after_transfer = balanceOf(recipient);
    assert balanceOf(recipient) <= balance_after_transfer;
}
```

LECTURE ROADMAP

What are proofs
in Formal Verification

Types of
bad proofs

How to tell if
a proof is bad

Real life
example

INVARIANTS

Wikipedia &
Oxford Dictionary

Invariant

- Always the same
- Never changing
- A logical assertion that is always held to be true
- A property which remains unchanged after operations or transformations of a certain type are applied

certora

PROOF BY INDUCTION

1. **The base case – after constructor**

2. **The step – any external/public function**
    a. **Assume the invariant**
    b. **Call the function**
    c. **Check if the invariant is still true**

TAUTOLOGICAL INVARIANT

A non-zero asset **cannot** be both **bitmap** and **active**

```
// BAD 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)
```

TAUTOLOGICAL INVARIANT

A non-zero asset **cannot** be both **bitmap** and **active**

```
// BAD 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)
```

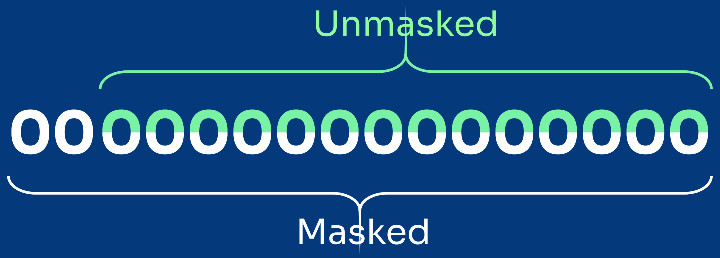
TAUTOLOGICAL INVARIANT

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

```
// BAD 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)
```

TAUTOLOGICAL INVARIANT

Same tautological statement




```
// BAD 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)
```

THE BUG

```
/// @notice Enables a bitmap currency for msg.sender, account cannot have any assets when this call
/// occurs. Will revert if the account already has a bitmap currency set.
/// @param currencyId the currency to enable the bitmap for.
/// @dev emit:AccountSettled emit:AccountContextUpdate
/// @dev auth:msg.sender
function enableBitmapCurrency(uint16 currencyId) external {
    require(msg.sender != address(this)); // dev: no internal call to enableBitmapCurrency
    require(currencyId <= maxCurrencyId); // dev: invalid currency id
    address account = msg.sender;
    (AccountContext memory accountContext, /* didSettle */) = _settleAccountIfRequired(account);
    accountContext.enableBitmapForAccount(currencyId, block.timestamp);
    accountContext.setAccountContext(account);
}
```

certora

THE BUG

1. Enable a bitmap currency on your account, eg. ETH.
2. Deposit a second currency into your account, eg. DAI.
3. Call *enableBitmapForAccount* a second time, switching your bitmap currency to DAI. Due to a logic error, the system believes that it would have to check DAI twice in free collateral, effectively doubling the DAI collateral believed to be present in the account.
4. Borrow in significant amounts without sufficient collateral; drain funds

FIXED INVARIANT

A non-zero asset **cannot** be both **bitmap** and **active**

```
// BAD INVARIANT
assert 0 <= i && i < 9 &&
     getActiveUnmasked(account, i) != 0 &&
      hasCurrencyMask(account, i)
       => getActiveUnmasked(account, i) != getBitmapCurrency(account)
```

AFTERMATH

1. The fixed invariant catches the bug in enableBitmapCurrency
2. The fixed invariant verifies the bug fix
3. The tautology automatic detection finds the problem in the bad invariant

TAKEAWAYS

Writing specifications is hard

Suspect, don't trust

- When the prover reports a bug, it is always useful
- When you get a proof, be suspicious

Check your spec!

Human reviews

Automatic checks

A right specification can prevent Billion $ bugs

certora

THANK YOU!

Uri Kirstein Software Engineer and Developer Relations