CHAINSECURITY

ICE center@ETH

Certifying functional correctness of Ethereum smart contracts

Dr. Petar Tsankov

Co-founder and Chief scientist, ChainSecurity

Senior researcher, ICE center, ETH Zurich

@ptsankov

ICE center@ETH

Inter-disciplinary research center at the #1 CS department in Europe

Blockchain security

Safety of AI

Security and privacy

CHAINSECURITY

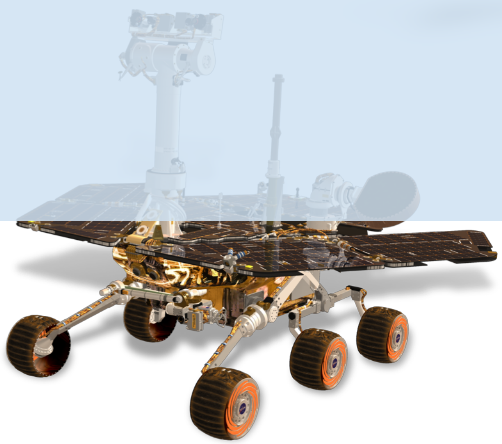
Next-generation blockchain security using automated reasoning

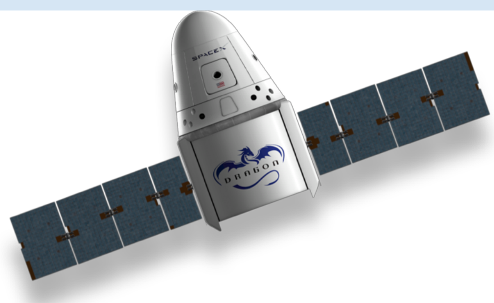
https://chainsecurity.com
@chain_security

ethereum foundation

WEB3

What do these have in common?

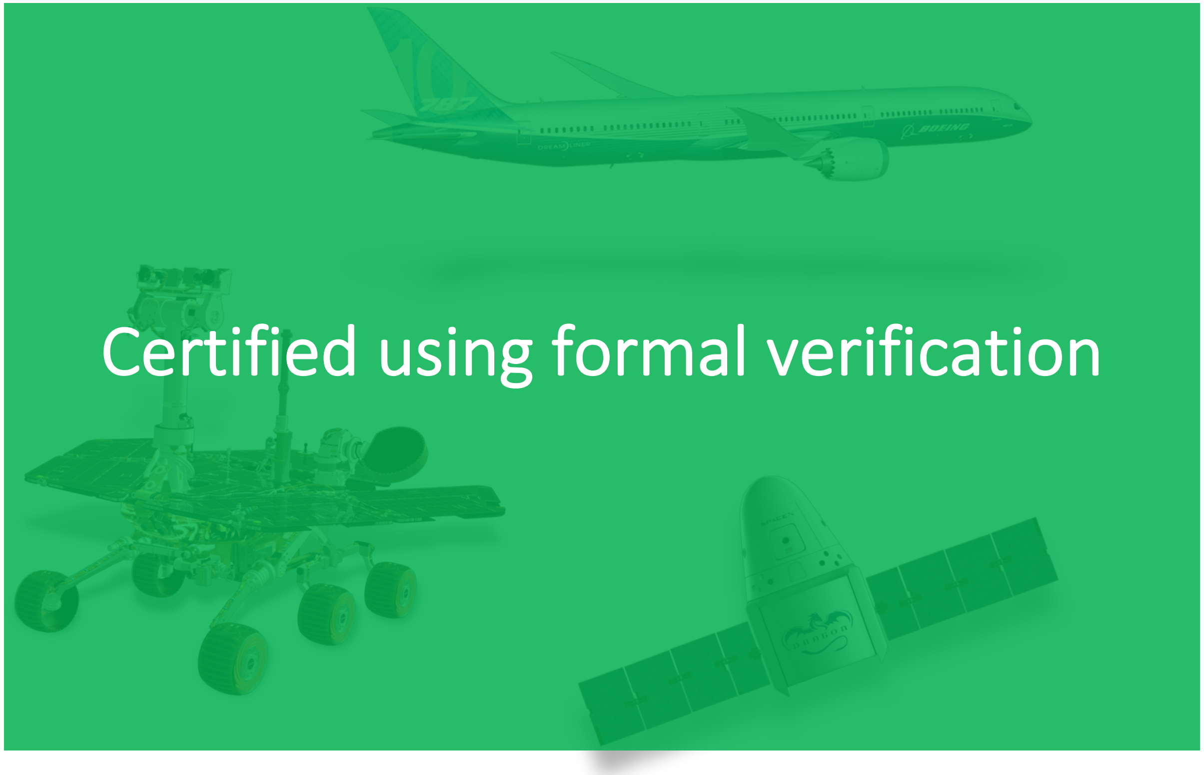


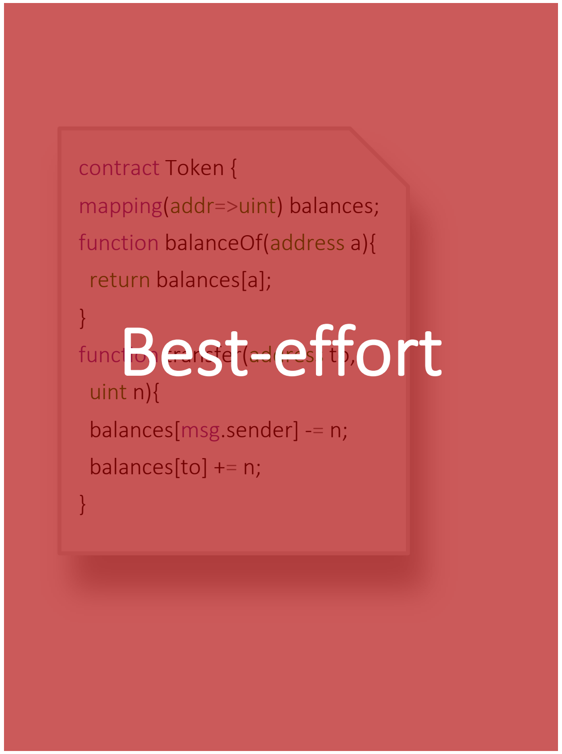


```
contract Token {
mapping(addr=>uint) balances;
function balanceOf(address a){
  return balances[a];
}
function transfer(address to,
  uint n){
  balances[msg.sender] -= n;
  balances[to] += n;
}
```

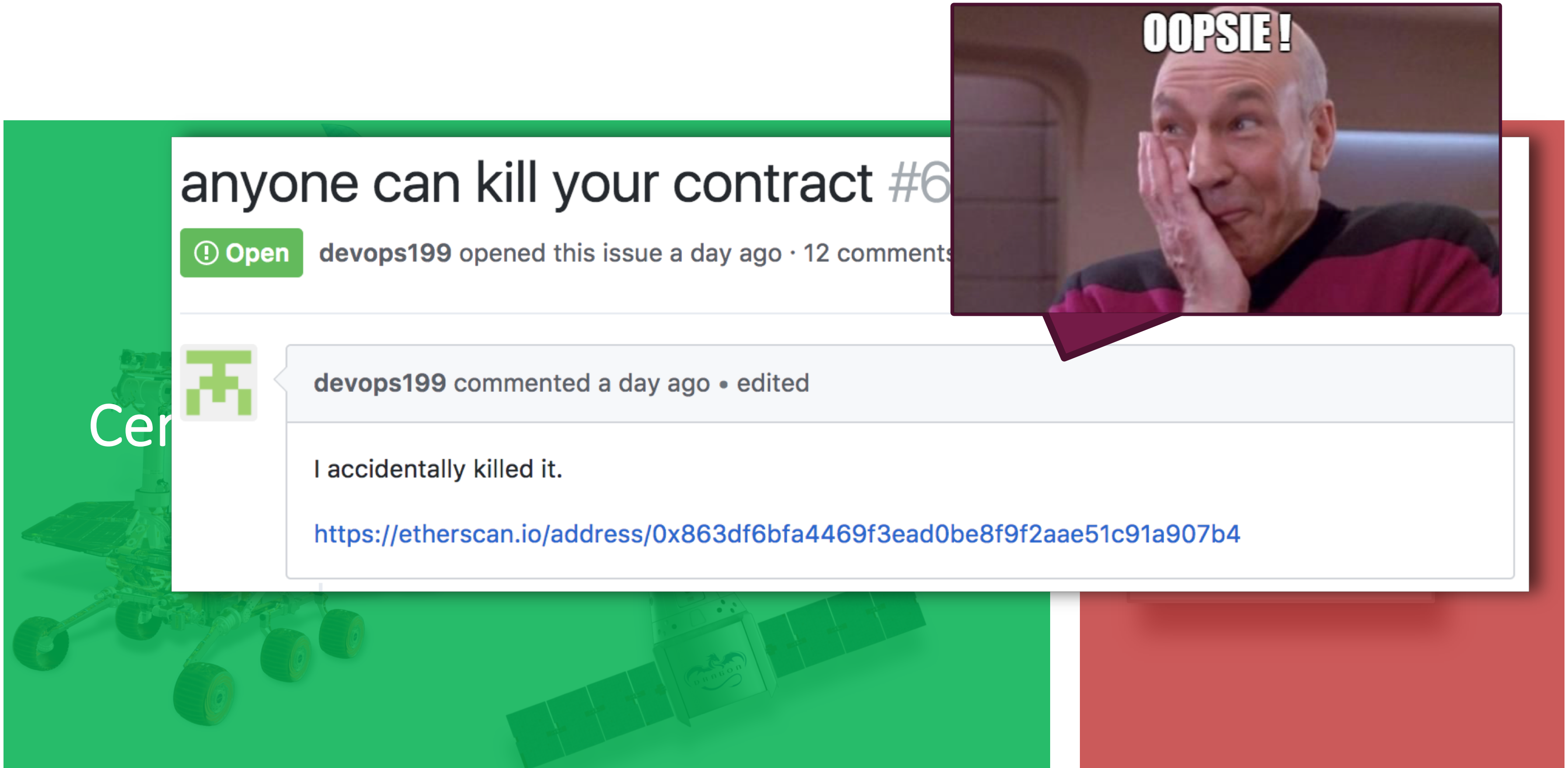
Must not fail!

What sets ***them apart***?





What sets *them apart*?



Our mission

Bring formal security guarantees to contracts

- Mathematically model **all** behaviors of smart contracts
- **Prove** that no bugs can occur
- Scale via **automation** and state-of-the-art research

Our mission

Bring formal security guarantees to contracts

- Scale via **automation** and state-of-the-art research

VerX

Formal verifier for certifying ***custom functional specifications*** of Ethereum contracts

Why is it hard to *certify* the custom behavior of smart contracts?

Note:

Find generic vulnerabilities

$\neq$ Certify **custom behavior**

Functional correctness

Crowdsale

```
uint raised;
uint goal;
uint closeTime;

function invest() { .. }
function close(){ .. }
```

Escrow

```
mapping(address => uint) deposits;

function deposit() { .. }
function withdraw() { .. }
function claimRefund() { .. }
```

Requirements

- Sum of all deposits equals the escrow's ether balance
- Investors cannot claim refunds after the goal is reached

Step 1: *Formalize* requirements

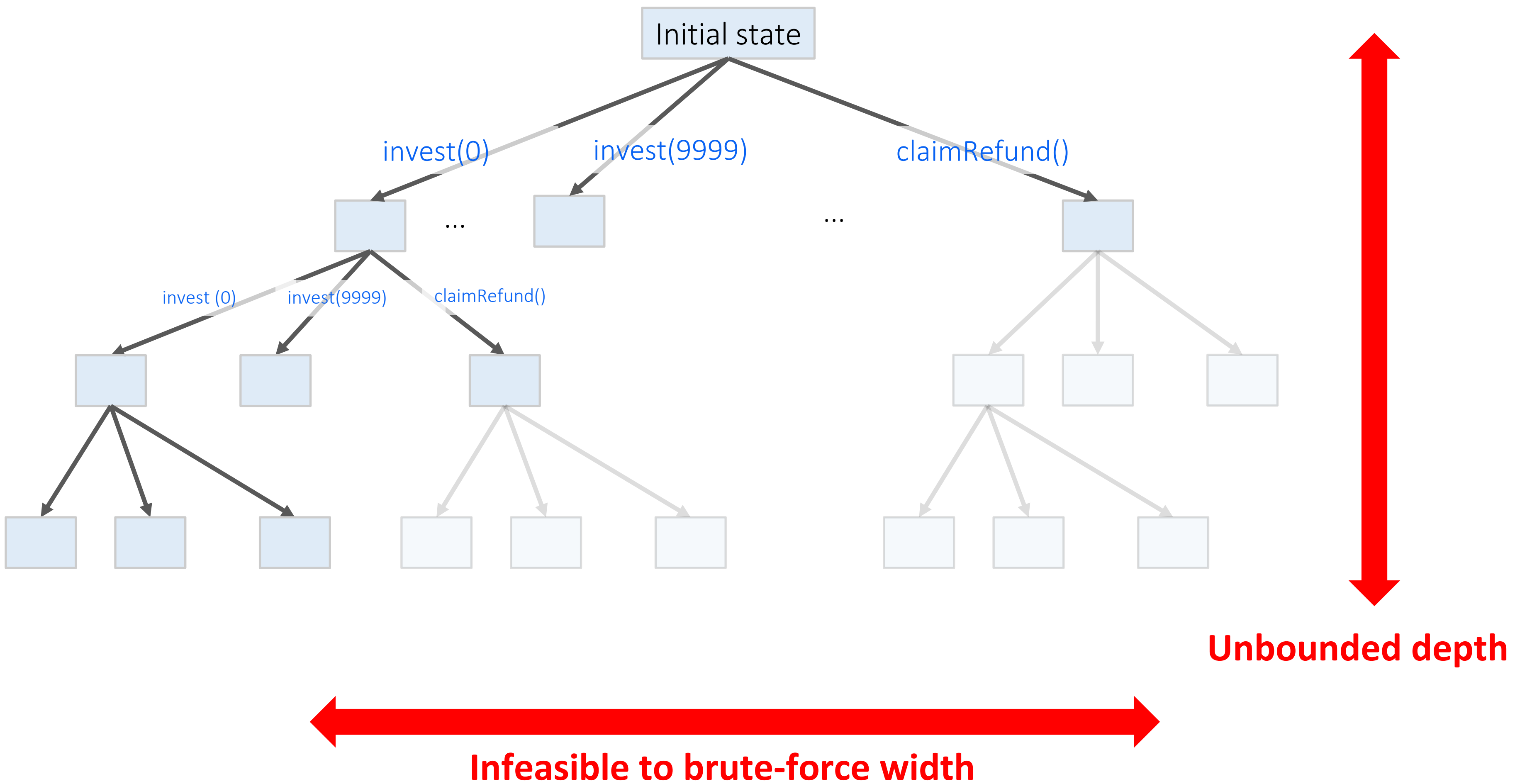
(Informal) requirement:

"Sum of all deposits equals the escrow's ether balance"

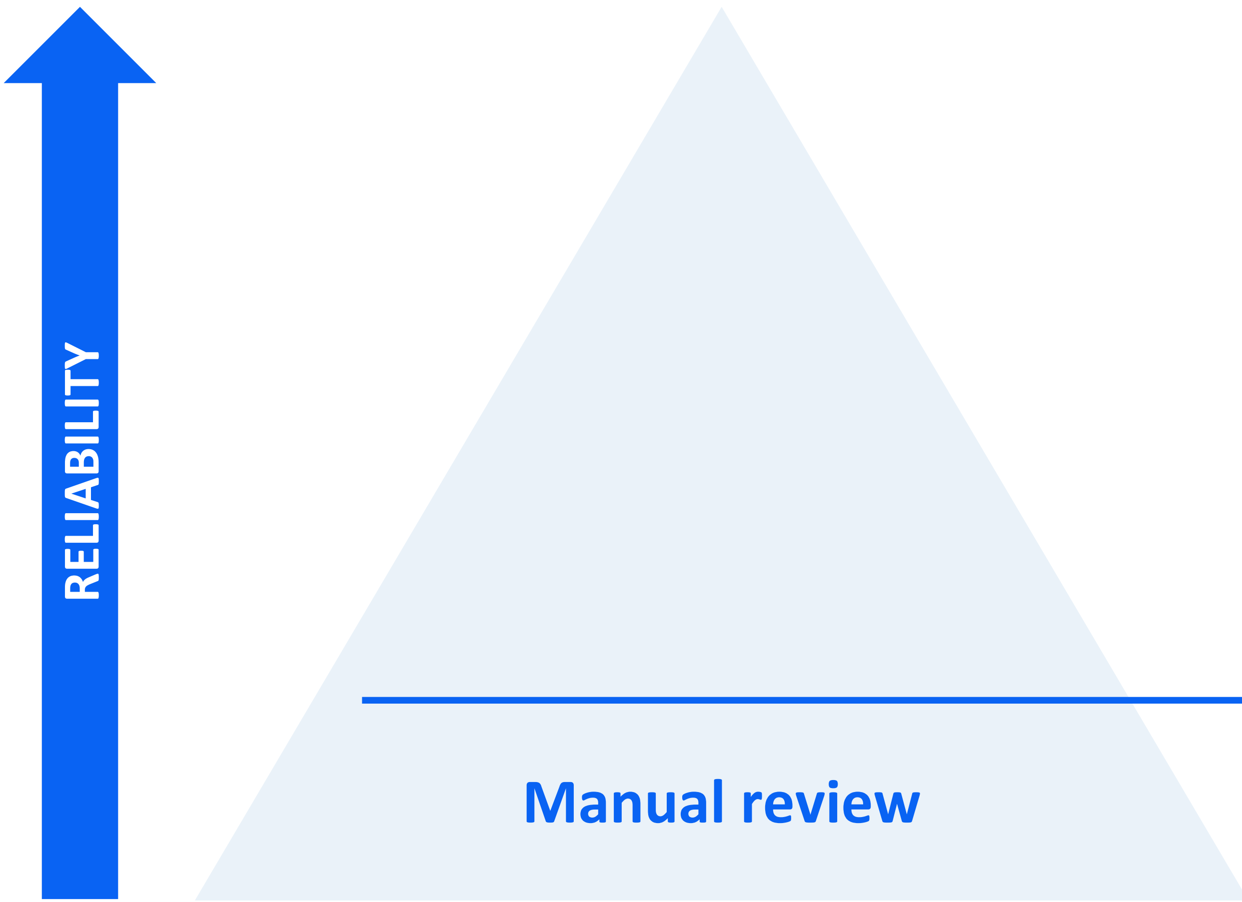
Formal property

```
always sum(Escrow.deposits) == Escrow.balance)
```

Step 2: *Check* formal property

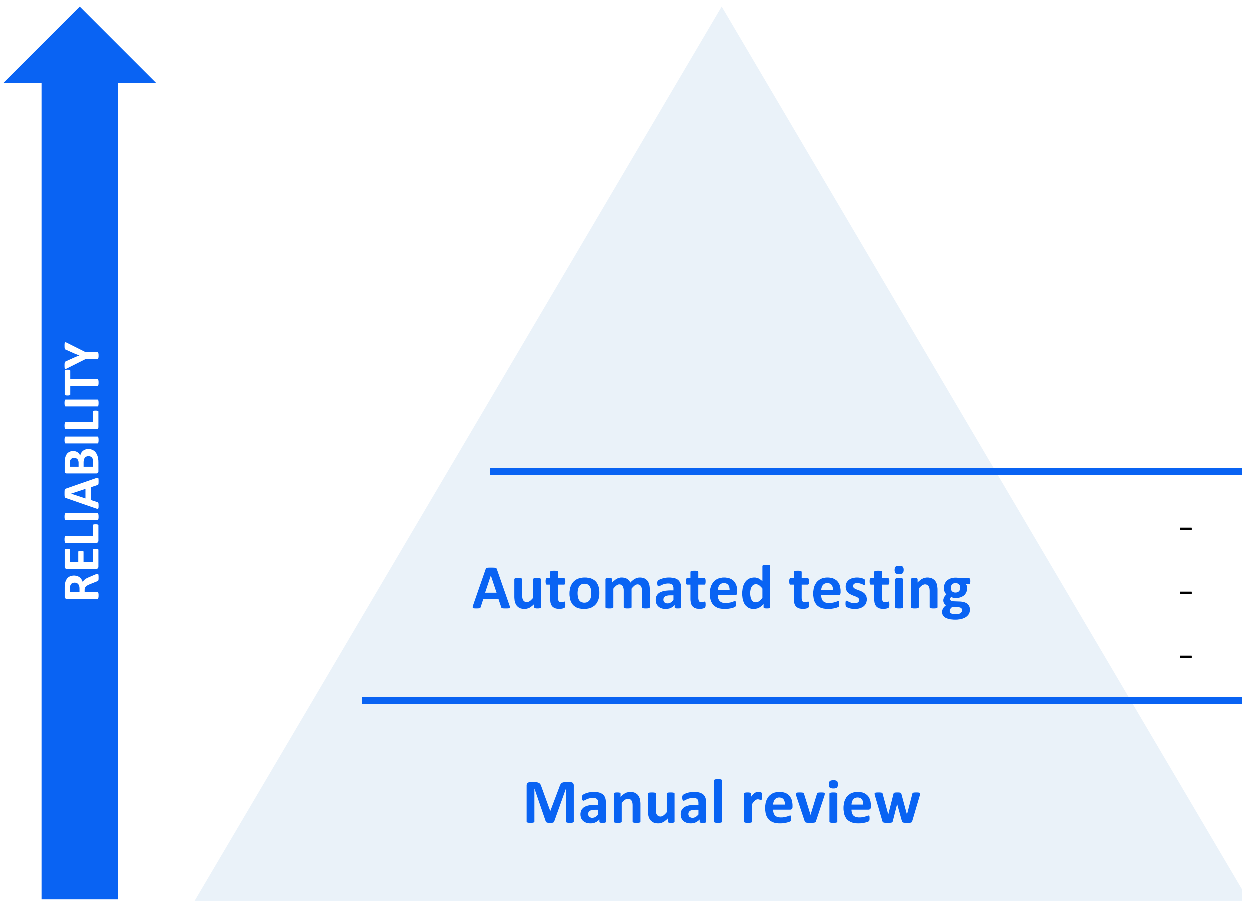


Methods and guarantees



- Time consuming
- *Can miss errors*

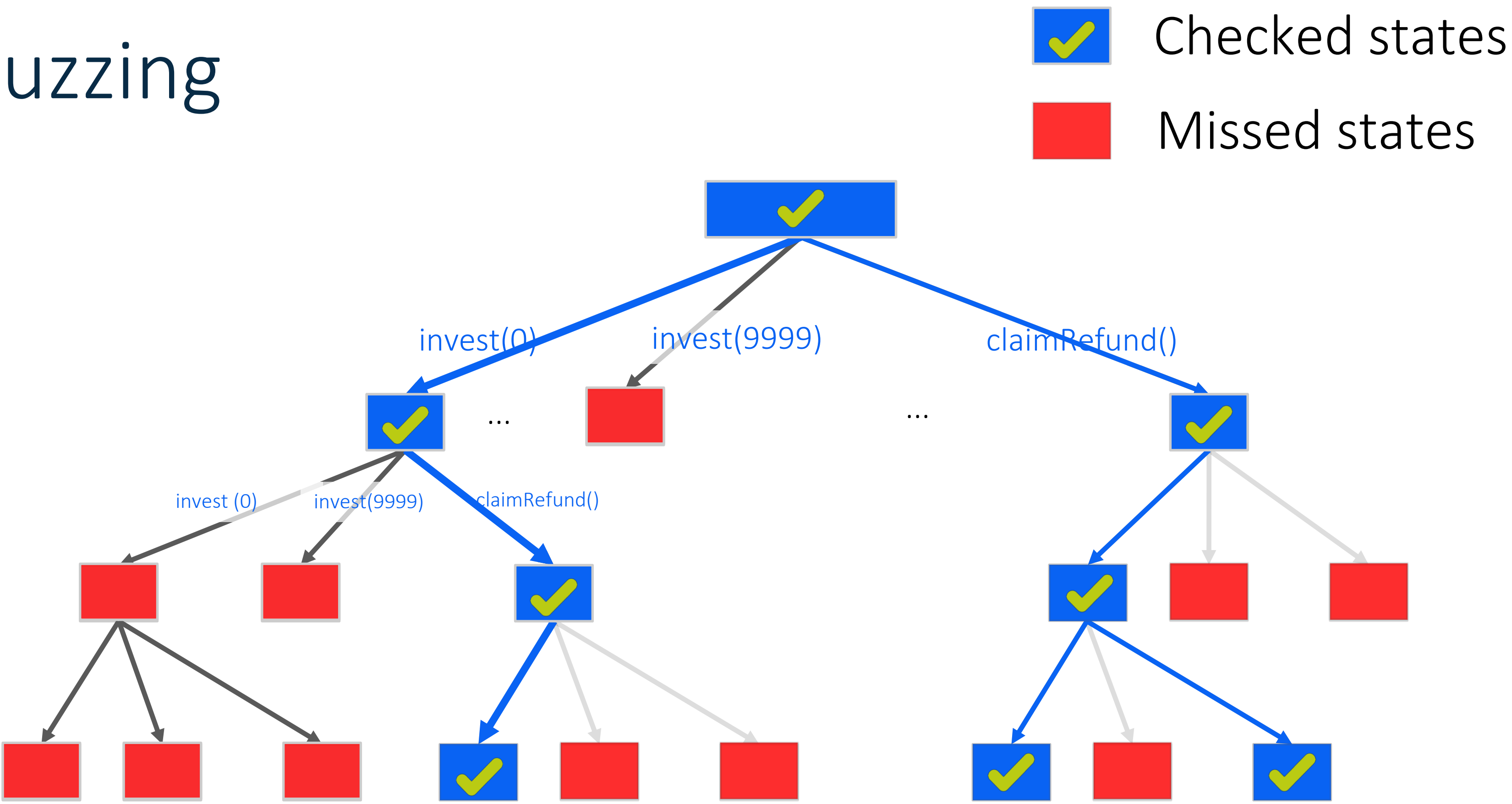
Methods and guarantees



- Fuzzing
- Symbolic execution
- *Can miss errors*

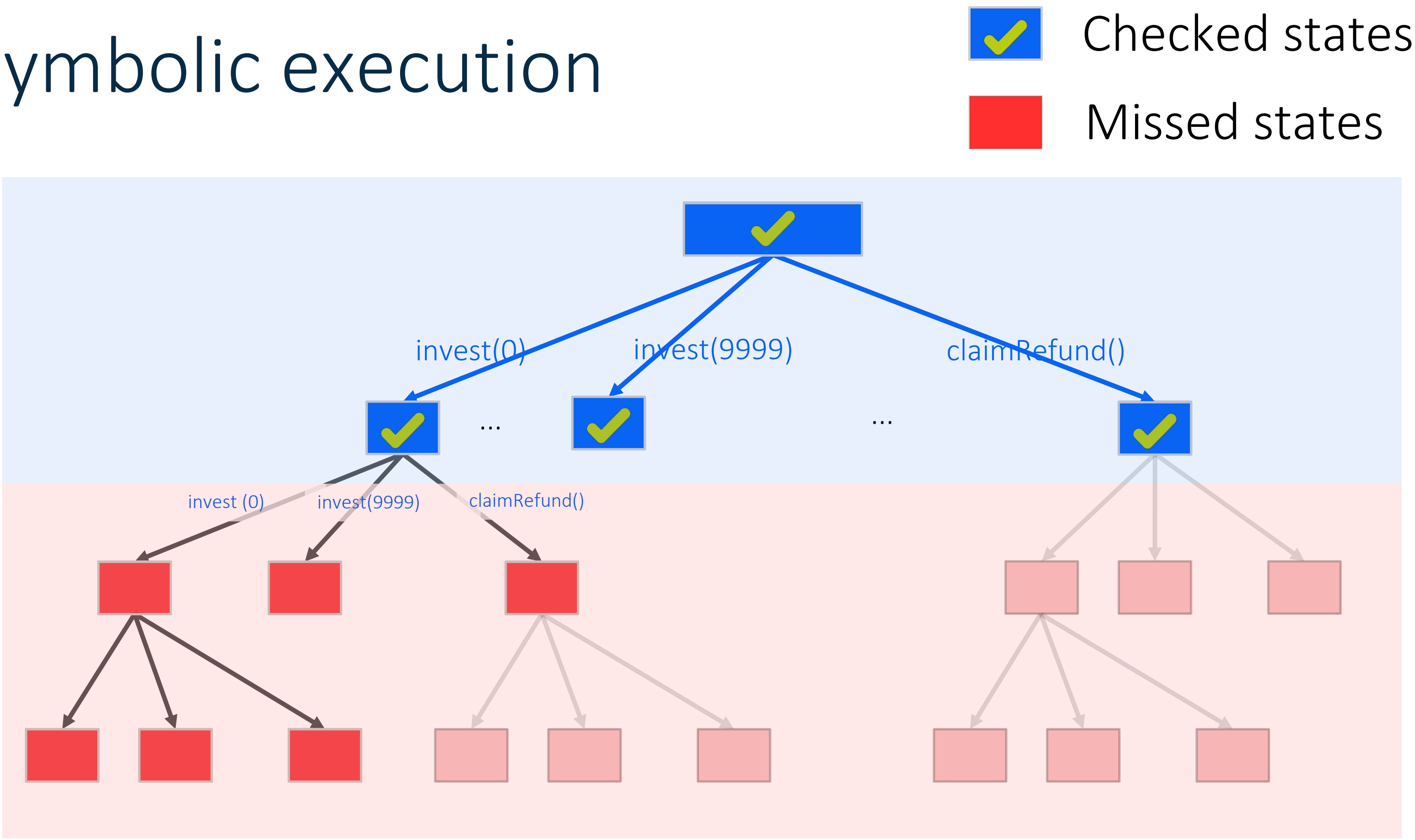
- Time consuming
- *Can miss errors*

Fuzzing



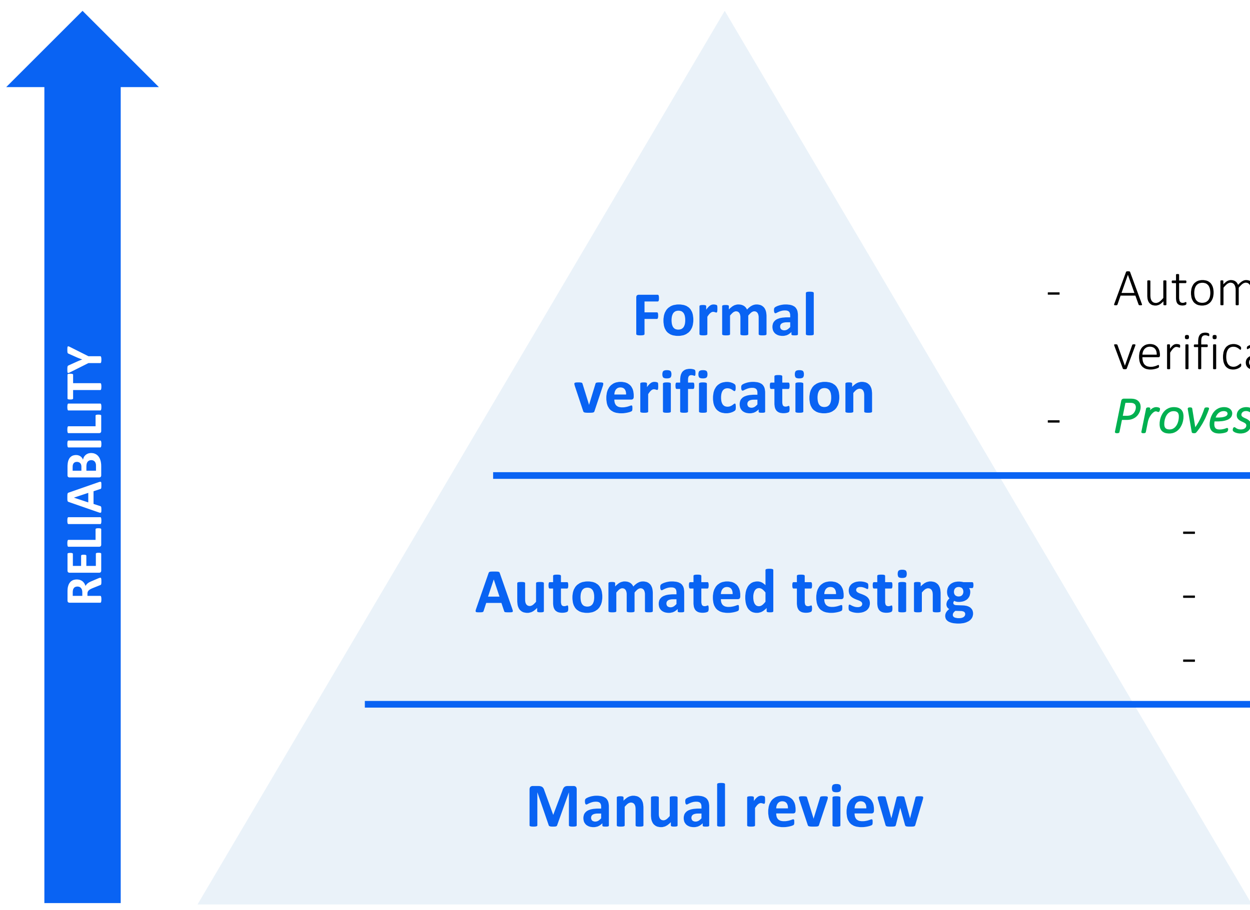
Tools: ChainFuzz, Echidna, ContractFuzzer, Harvey, ...

Symbolic execution



Tools: Oyente, Manticore, Mythril, MAIAN, …

Methods and guarantees

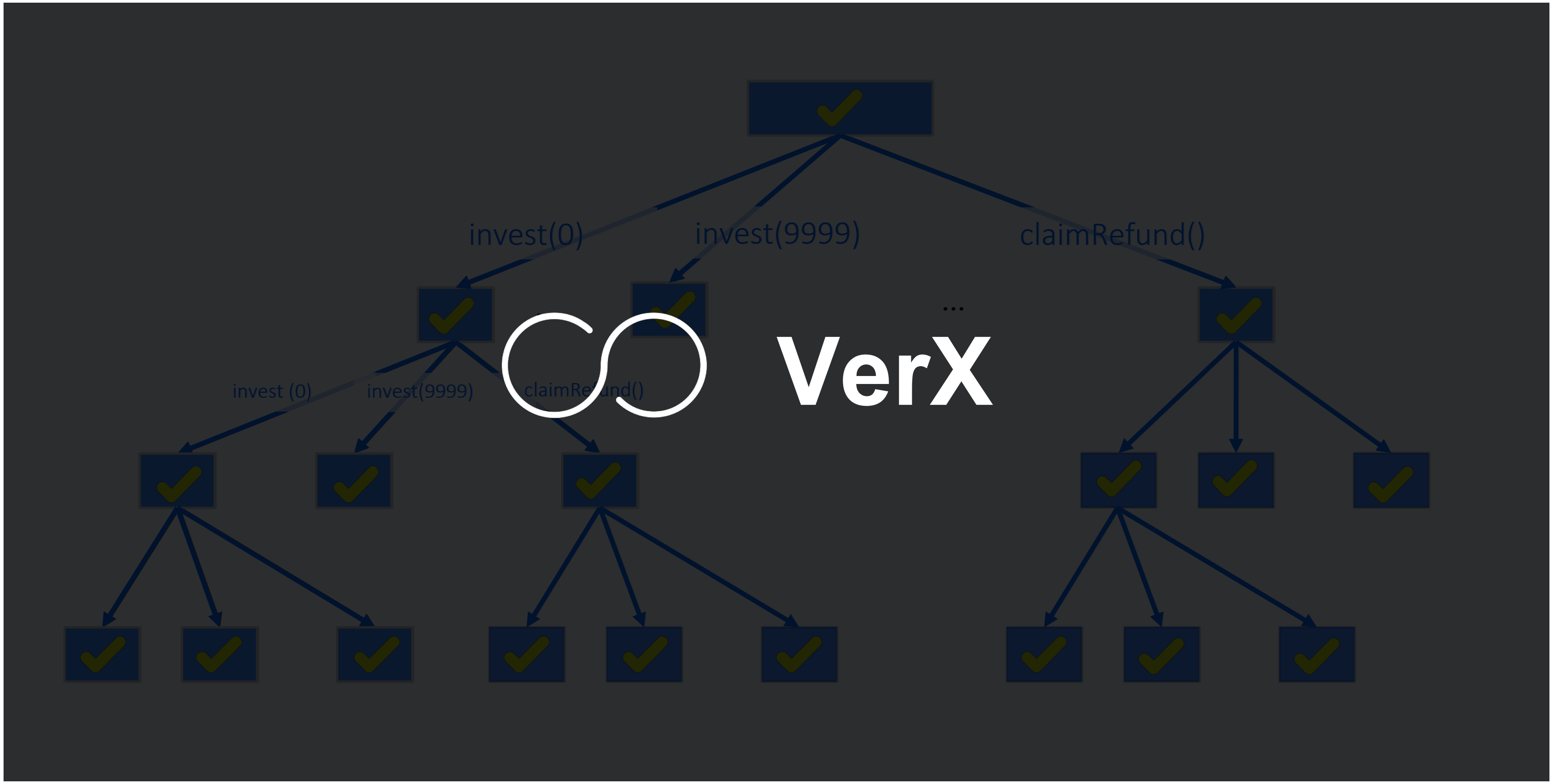


- Automated program verification
- *Proves absence of errors*

- Fuzzing
- Symbolic execution
- *Can miss errors*

- Time consuming
- *Can miss errors*

Formal verification



Automated formal verification with VerX

"Investors can claim refunds only if the sum of deposits never exceeded 10,000 ether "

Smart contract

```
mapping(address => uint) deposits;
function claimRefund(){..}
```

Formal property

```
(always Escrow.claimRefund
    ==> !before(sum(deposits) >= 10000)
```

VerX

Verified

May not hold

Expressive and intuitive specifications

Access control

```
always Escrow.deposit(address)
        ==> (msg.sender == Escrow.owner)
```

Expressive and intuitive specifications

Access control	`always Escrow.deposit(address) ==> (msg.sender == Escrow.owner)`
State-based properties	`always (now > Vault.refundTime + 1 week) ==> ! Vault.refund(uint256)`

Expressive and intuitive specifications

Access control	`always Escrow.deposit(address) ==> (msg.sender == Escrow.owner)`
State-based properties	`always (now > Vault.refundTime + 1 week) ==> ! Vault.refund(uint256)`
State machine properties	`always !(once(state == REFUND) && once(state == FINALIZED)`

CHAINSECURITY

ICE center@ETH

Expressive and intuitive specifications

Access control	`always Escrow.deposit(address) ==> (msg.sender == Escrow.owner)`
State-based properties	`always (now > Vault.refundTime + 1 week) ==> ! Vault.refund(uint256)`
State machine properties	`always !(once(state == REFUND) && once(state == FINALIZED)`
Invariants over aggregates	`always totalSupply == sum(balances)`

Expressive and intuitive specifications

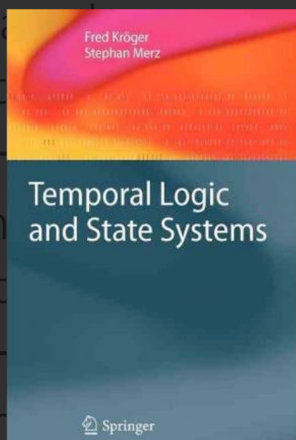
Access control	`always Escrow.deposit(address) ==> (msg.sender == Escrow.owner)`
State-based properties	`always (now > Vault.refundTime + 1 week) ==> ! Vault.refund(uint256)`
State machine properties	`always !(once(state == REFUND) && once(state == FINALIZED))`
Invariants over aggregates	`always totalSupply == sum(balances)`
Multi-contract invariants	`always Token.totalSupply >= Sale.issuance`

CHAINSECURITY

ICE center@ETH

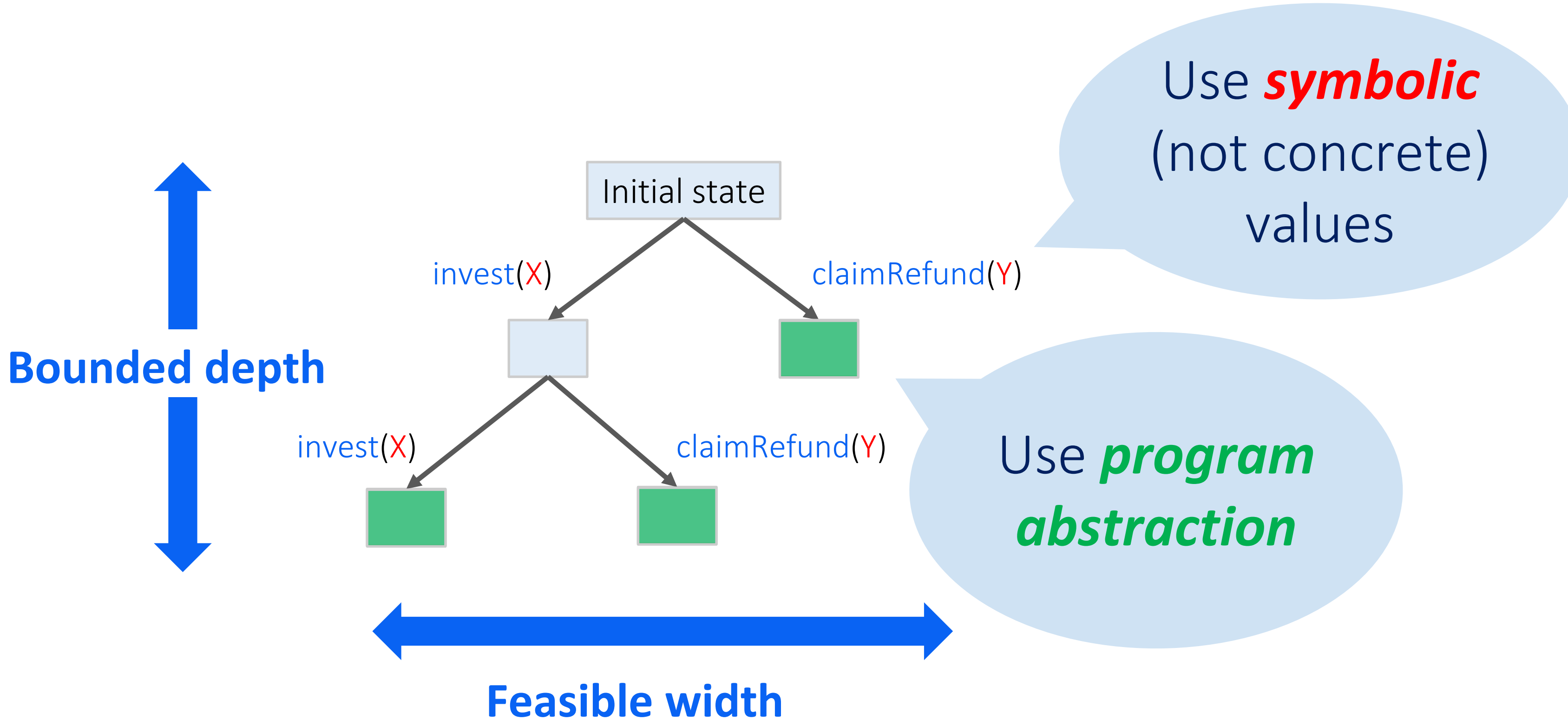
Expressive and intuitive specifications

Access control	`always Escrow.deposit(address) ==> (msg.sender == Escrow.owner)`
State-ba... propert...	`always (now > Vault.refundTime + 1 week) ==> ! Vault.refund(uint256)`
State m... propert...	`always (! once(state == REFUND) && once(state == FINALIZED))`
Invarian... aggregates	`always totalSupply == sum(balances)`
Multi-contract invariants	`always Token.totalSupply >= Sale.issuance`



Solid formal foundation (Temporal logic)

Dealing with *unbounded* state spaces



CHAINSECURITY

ICE center@ETH

Sound symbolic reasoning

- Hash-based storage allocation
- Gas mechanics
- Calls to untrusted contracts
- Dynamically constructed contracts

Impact and experience

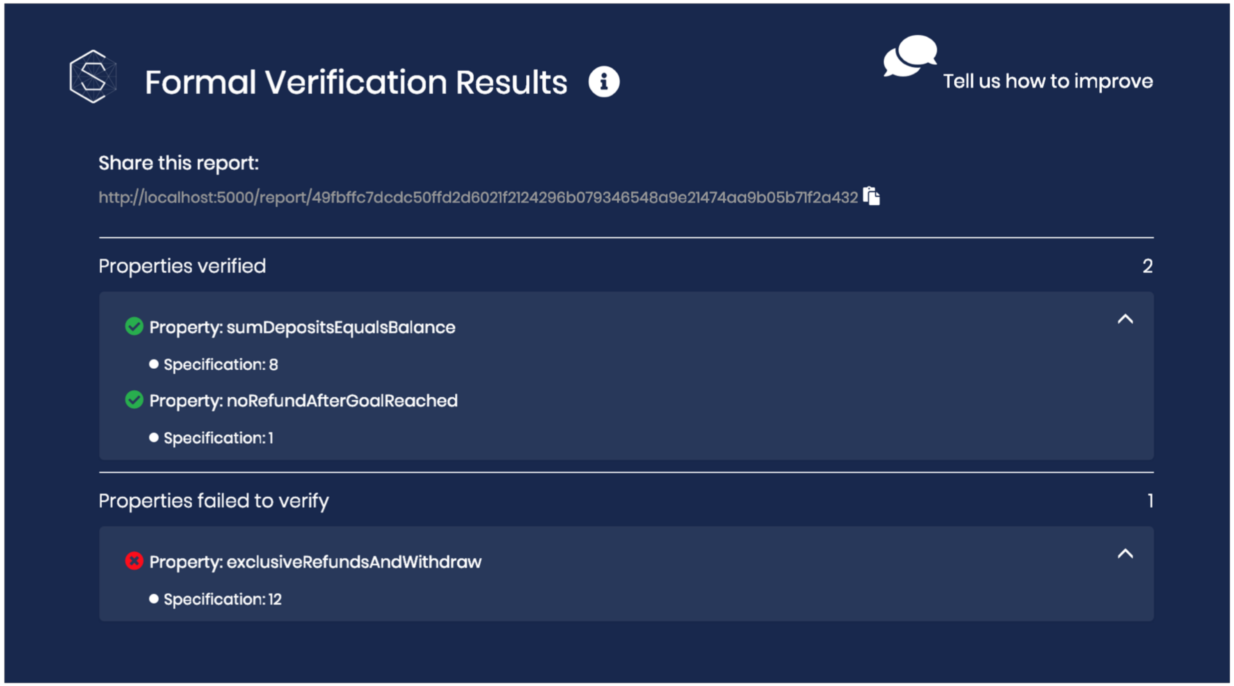
Fast and *scalable* formal verification of Ethereum contracts

(157+ contracts, 100+ properties, ~1 min / property)

Benefits:

- *Certify* what works (go beyond bug finding)
- *Re-use* libraries of common specifications
- *Re-certification* is cheap

How to get access to VerX?



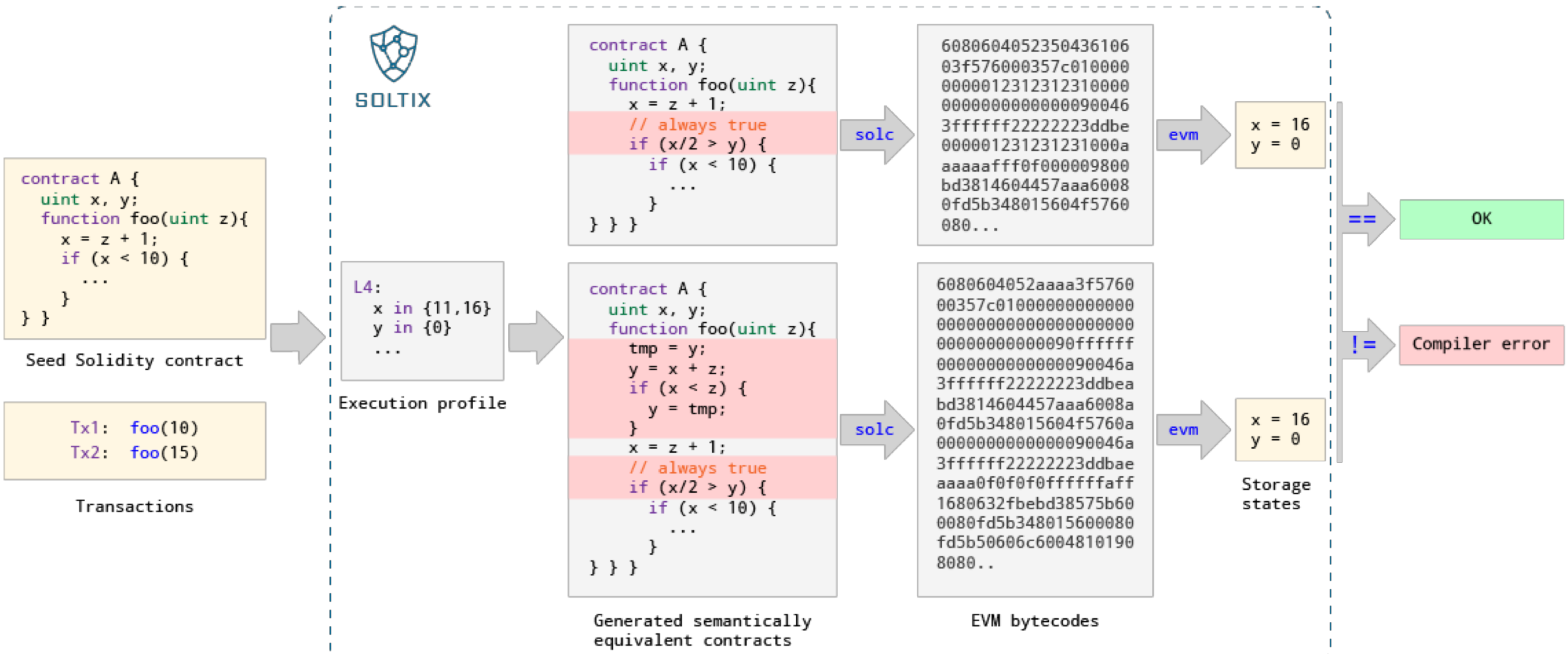
Demo: http://verx.ch

VerX as a service: contact@chainsecurity.com

One more announcement…

SOLTIX

First automated framework for testing Solidity compilers



CHAINSECURITY

ethereum foundation grants

ICE center@ETH

First automated framework for testing Solidity compilers

Internal compiler error for call to unimplemented "super" function #5130

Closed **nweller** opened this issue on 2 Oct 2018 · 10 comments

nweller commented on 2 Oct 2018 • edited

The follow
recent buil

```
contrac
    fun
}
contrac
    fun
}
```

It does cor

(This repor
the Ethere

Exponentiation producing inconsistent results #4893

Closed **nweller** opened this issue on 4 Sep 2018 · 6 comments

nweller commented on 4 Sep 2018 • edited

Across multiple unsigned integer types I've tried - e.g. uint8 - the exponentiation operator can produce a result which is outside of the range of that type in some contexts, but exhibits the truncation behavior I'd expect in other contexts.

I get different failure modes in the truffle-based test framework I'm using (with ganache-cli, Linux, solc-js 0.4.24 - this produces unexpected values) and http://remix.ethereum.org/ (this produces a VM error).

See:

First automated framework for testing Solidity compilers

SOLTIX

https://github.com/eth-sri/soltix

https://discord.gg/XKSVavS

ethereum foundation grants

CHAINSECURITY

ICE center@ETH

Safety certification of contracts

VerX: Automated formal verification

"Investors can claim refunds only if the sum of deposits never exceeded 10,000 ether "

Smart contract

```
mapping(address => uint) deposits;
function claimRefund(){..}
```

Formal property

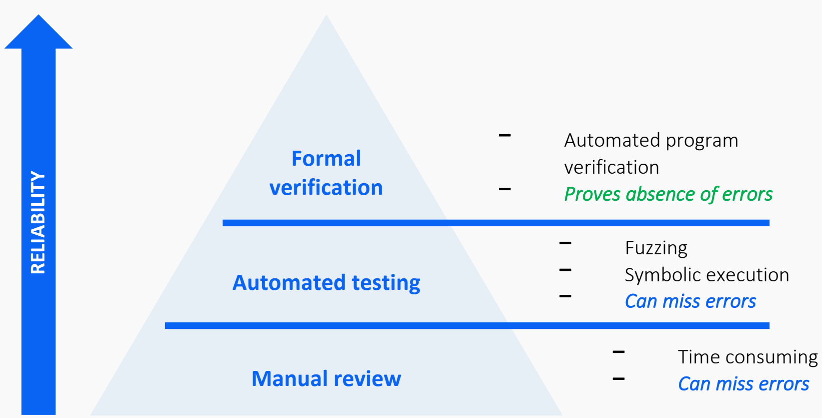
```
(always Escrow.claimRefund
  ==> !before(sum(deposits) >= 10000)
```

VerX

Verified

May not hold

Methods and techniques



Symbolic reasoning + abstraction