Deductive Verification of Smart Contracts

Mooly Sagiv

TEL AVIV UNIVERSITY אוניברסיטת תל אביב

vmware

CERTORA

OCTOBER 2018

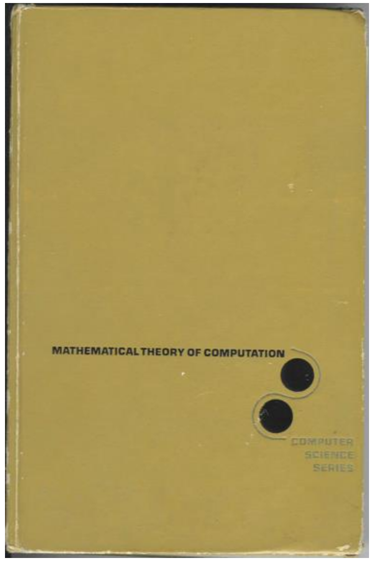
Deductive Verification of Smart Contracts

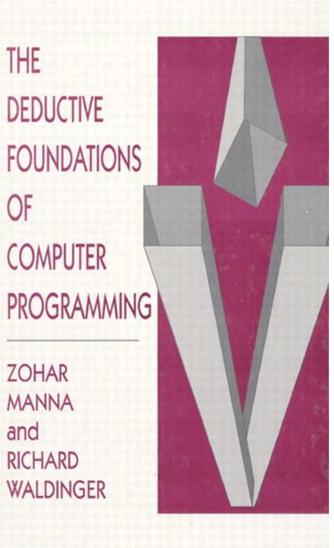
A Tribute to the Legacy of Prof. **Zohar Manna** 1939-2018

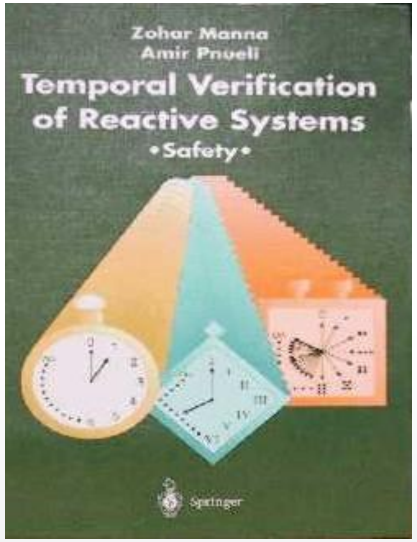
Zohar Manna

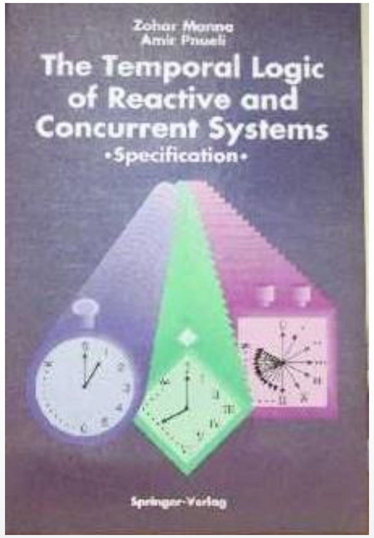
Weizmann Inst. & Stanford Uni.

1939-2018



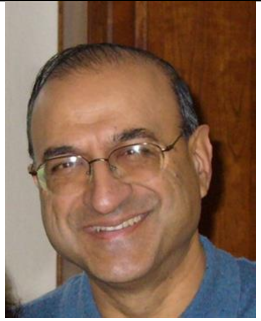
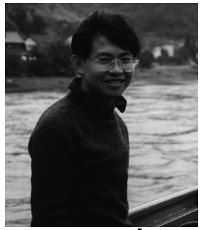






Milestones

- Thesis (CMU, 1968)
- Mathematical Theory of Computation (1974)
- Logical Basis for Computer Programming (with Waldinger, 1985-8)
- STeP: A tool for deductive verification of systems
- Consummate teacher and advisor

 Jean-Marie Cadiou	 Ashok Chandra	 Jean Vuillemin	 Shmuel Katz	 Adi Shamir	 Nachum Dershowitz	 William Scherlis	 Pierre Wolper
Ben Moszkowski	 Yoni Malachi	 Martín Abadi	 Marianne Baudinet	 Rajeev Alur	 Tom Henzinger	Eddie Chang	Hugh McGuire
 Anuchit Anuchitanukul	Arjun Kapur	 Luca de Alfaro	 Nikolaj S. Bjorner	Tomás E. Uribe	 Henny B. Sipma	 Bernd E. Finkbeiner	Michael Colón
 Calogero Zarba	 Sriram Sankaranarayanan	Ting Zhang	Matteo Slanina	 César Sánchez	 Aaron Bradley		

Software verification

- The programmer defines what is the desired behavior
- Ensures there is a proof of correctness
- Proof covers all scenarios

Why verify smart contracts?

Smart Contracts are hard to get right

alex van de sande @avsa
Follow

I repeat. There was an attack on the DAO so we launched our white hat counter attack. More updates will follow

RETWEETS 46 LIKES 51

8:11 PM - 21 Jun 2016

Manuel Aráoz @maraoz
Follow

Someone stole ~$32M (~153k ether) from three multisig wallets. More info and blog post coming soon. etherscan.io/address/0xb376 …

12:08 PM - 19 Jul 2017

Parity Technologies @ParityTech
Follow

UPDATE: A user exploited an issue and thus removed the library code, as it seems unaware of the consequences.

2:51 AM - 7 Nov 2017

SpankChain @SpankChain
Follow

At 6pm PST Saturday, an unknown attacker drained 165.38 ETH (~$38k) from our payment channel smart contract which also resulted in $4,000 worth of BOOTY on the contract becoming immobilized. Here is what we know so far:

We Got Spanked: What We Know So Far – SpankChain – M...
At 6pm PST Saturday, an unknown attacker drained 165.38 ETH (~$38,000) from our payment channel smart contract which also resulted in…
medium.com

8:49 PM - 8 Oct 2018

A barrier to trust!

Correctness is essential for Smart Contracts

Traditional software

- Buggy code is a reality
- Mechanisms for reverting effects of erroneous code execution
- Continuous code maintenance is standard practice

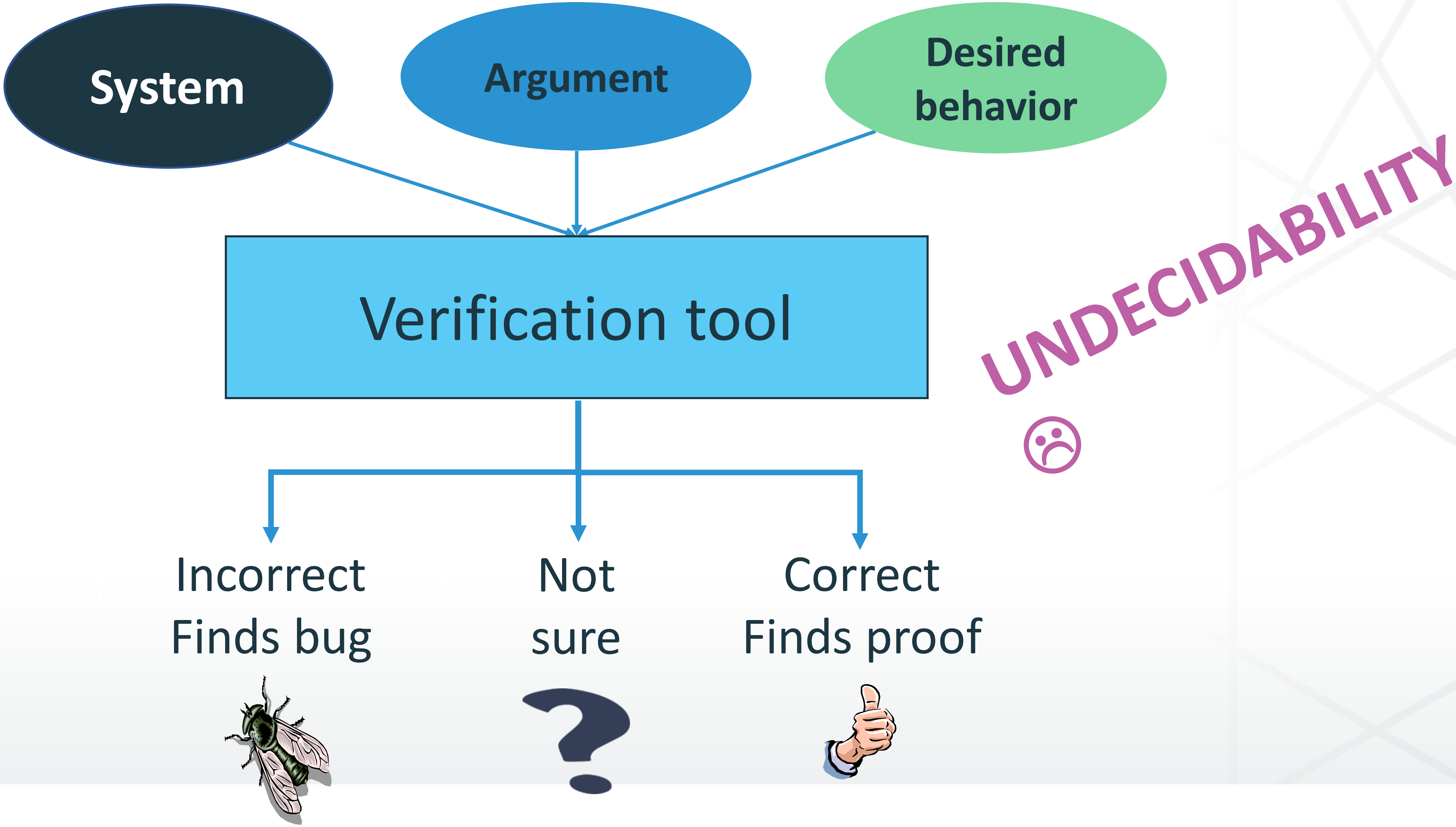
Smart Contracts on Blockchains

- Code as law
- Transactions are irreversible, often anonymous
- Smart Contracts are unpatchable
  - Upgrade is tricky

Auditing

- The standard procedure for checking contracts
- Expensive $$$
- Quality depends on the auditors
- Miss bugs
- Not decentralized
  - Auditor reputation is the trust authority

Semi-automatic deductive verification



What We Do: Technology for certifying contracts

Find bugs or prove their absence

- No false alarms or missed errors

Define what is required from contracts

- Generic properties
  - No overflow
  - Isolation between contracts [POPL'18]

- Standard requirements
  - ERC20, ERC721
    Money market, Exchanges…

- Contract-specific correctness
  - Wallet should have sufficient number of signers
  - Correct libraries

[POPL'18] S. Grossman et. al. Online Detection of Effectively Callback Free Objects with Applications to Smart Contracts

What Customers Say

“Compound worked with Certora to verify the correctness of a preliminary-version of a core contract.

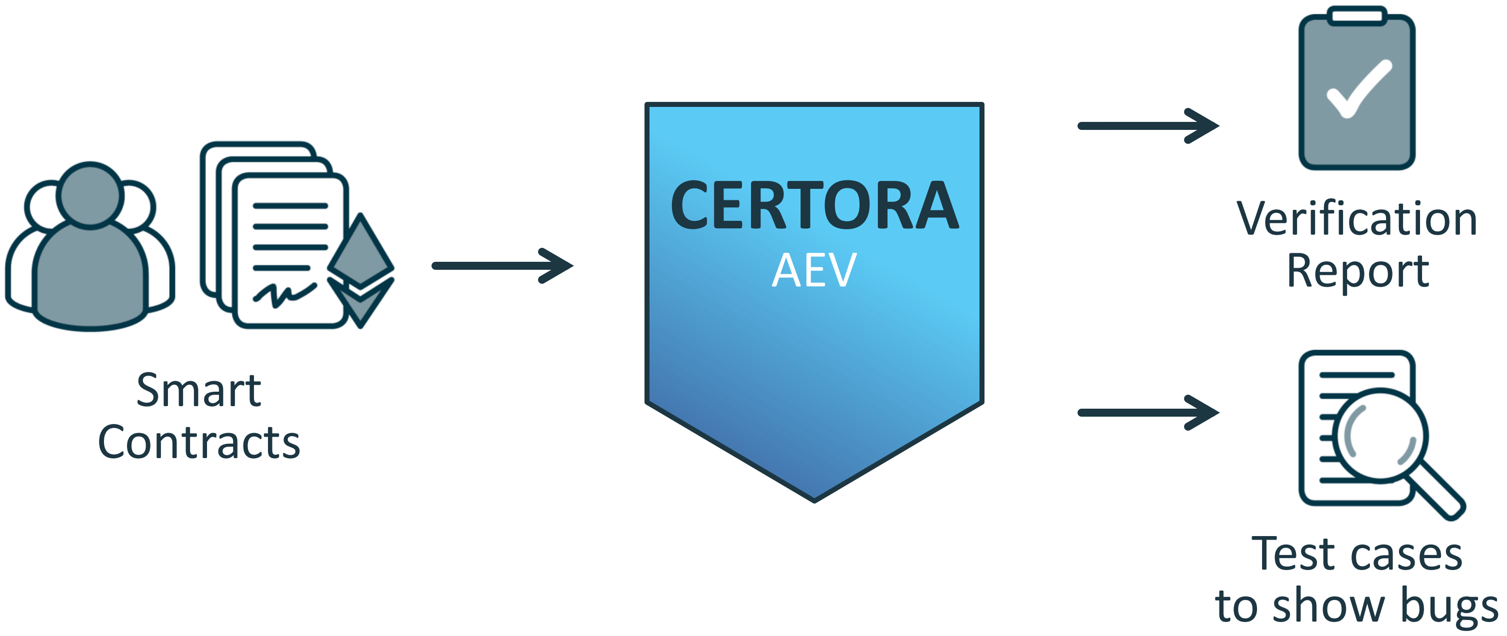
The tool demonstrated a unique capability to discover not just the obvious corner-cases, but also subtle cases that would have been difficult, if not impossible, to find through standard unit-testing.

The Certora team discovered two subtle bugs in the contract which were patched, as well definitively proving a conjecture which influenced an important design decision.

Certora's collection of properties proven to hold for all inputs and environments greatly increased our confidence in the correctness of our contract.”

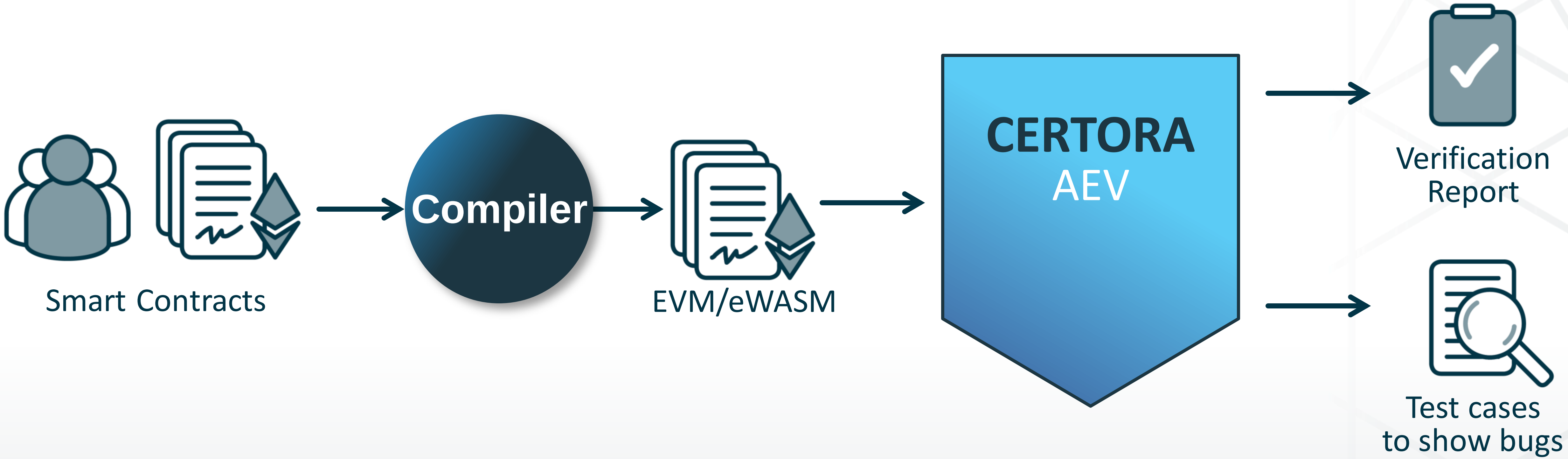
Geoff Hayes | CTO Compound

Certora - Automatic Exact Verification (AEV)



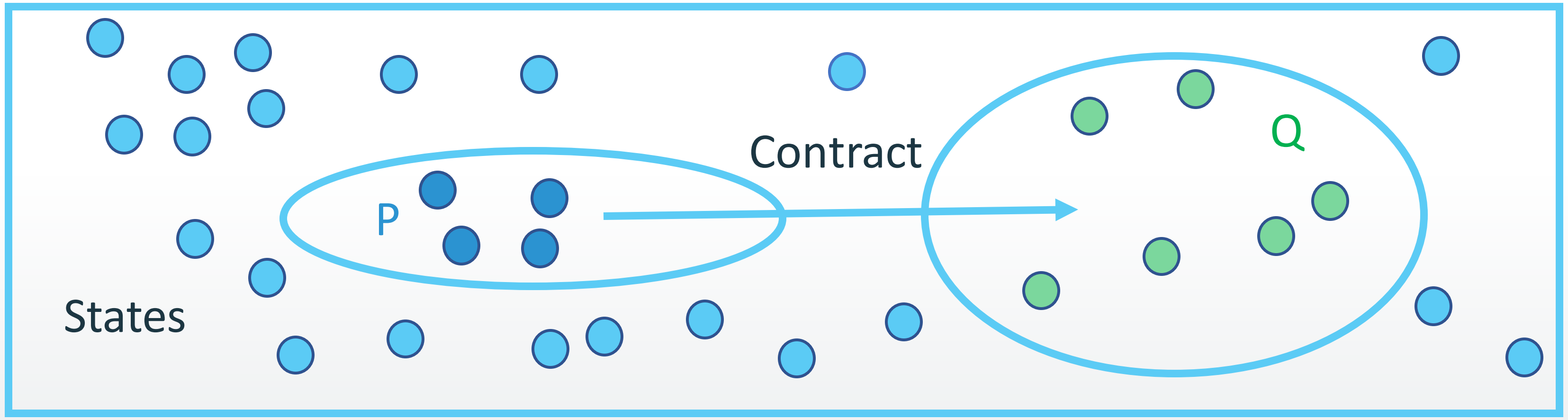
Benefits	Superior Accuracy	Automatic	Zero False Alarms	Zero Missed Errors
	Most accurate method to detect bugs	No customization per contract or services are required	All reported errors are real and come with risk explanation	All errors are eventually detected and come with formal checkable proofs

How does Certora-AEV work?



Hoare Triples

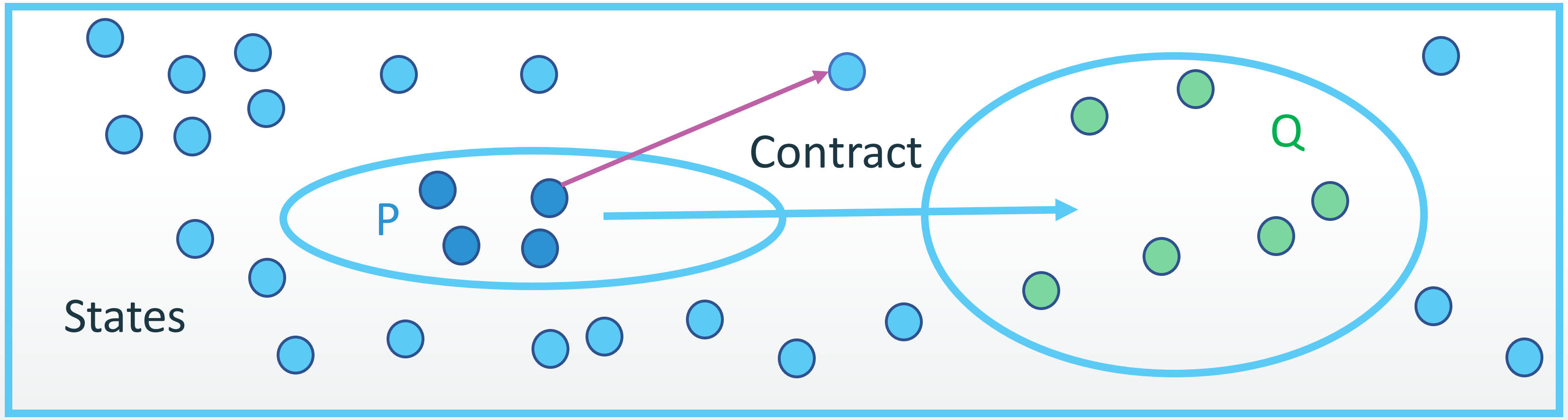
- Useful to explain verification
- Annotate the code with assertions
- {P} Contract {Q}
  - Every execution of the contract starting in a state in P results in a state in Q




```
if P then {
        Contract;
        assert Q;
}
```

Hoare Triples

- Useful to explain verification
- Annotate the code with assertions
- **{P}** Contract **{Q}**
  - Every execution of the contract starting in a state in P results in a state in Q




```
if P then {
        Contract;
        assert Q;
}
```

Example Hoare Triples

- { **x=0** } x := x+2 { **x=2** } ✓*valid*
- { **x=0** } x := x+2 { **x>0** } ✓*valid*
- { **x=1** } x := x+y { **x>0** } ✗ *invalid* *Test y = -3*
- { **true** } if x<0 then y:=-x else y:=x { **y≥0** } ✓*valid*
- { **y≥0** } t:=y; z:=1; while t>0 do z:=z*x; t:=t-1; done { **z=x^y** } ✓*valid*

Composing Operations

- Prove that

  {P}
  command_1 ;
  command_2
  {Q}

- Find an intermediate assertion R and show
  - **{P}** command_1 **{R}**
  - **{R}** command_2 **{Q}**?
- Can be found automatically

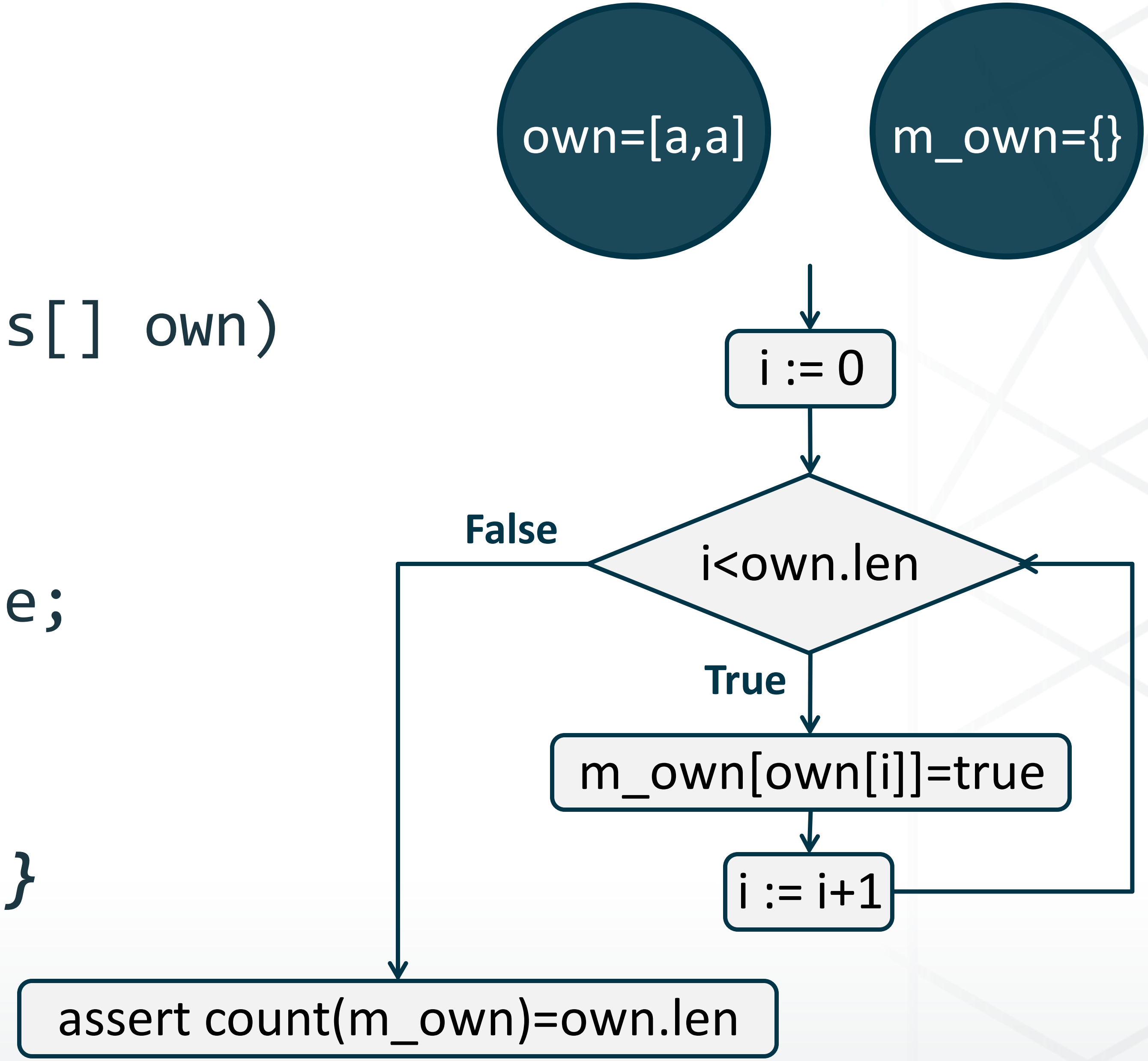
Composing Operations Example

- How to prove that **{ x=0 }** x:=5; **{?}** y:=x+1 **{ y>0 }**
- Prove that
  - **{ x=0 }** x:=5 **{ x≥0 }**
  - **{ x≥0 }** y:=x+1 **{ y>0 }**

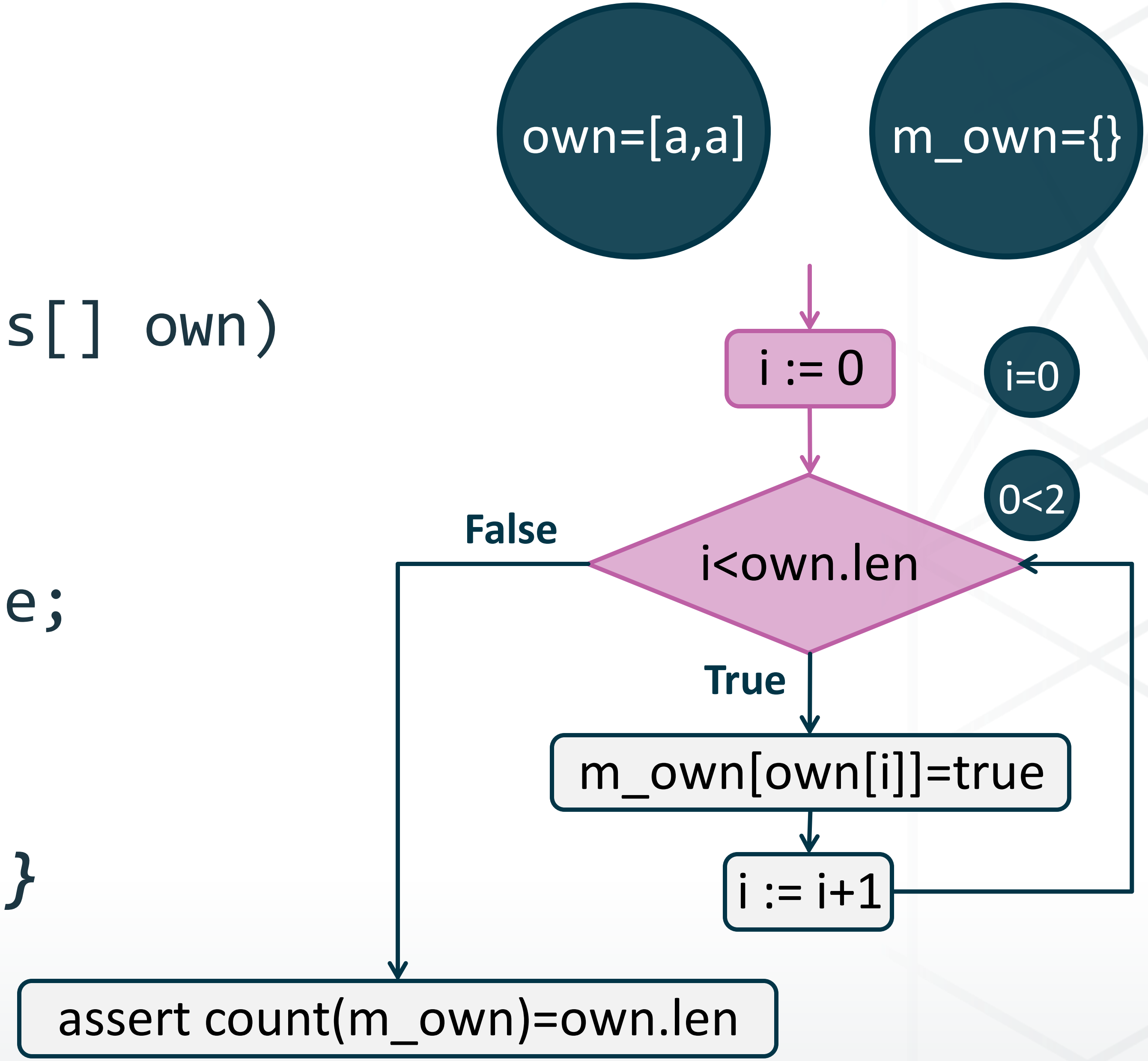
Example Wallet

```
{ count(m_own) = 0 }

wallet_constructor(address[] own)
    int i = 0;
    while (i < own.len)
        m_own[own[i]] = true;
        ++i;

{ count(m_own) = own.len }
```

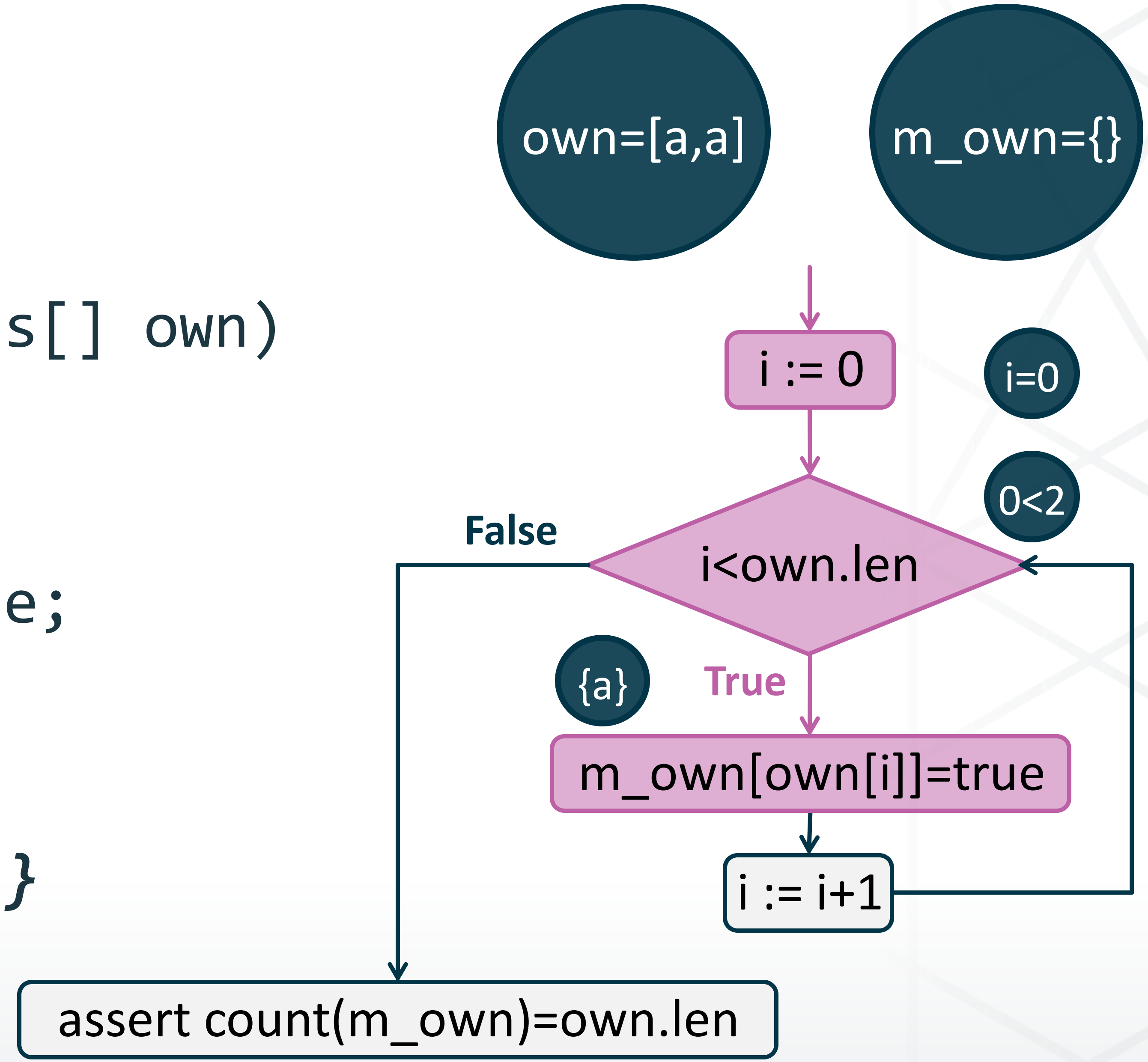
```
{ count(m_own) = 0 }

wallet_constructor(address[] own)
   int i = 0;
   while (i < own.len)
      m_own[own[i]] = true;
      ++i;

{ count(m_own) = own.len }
```



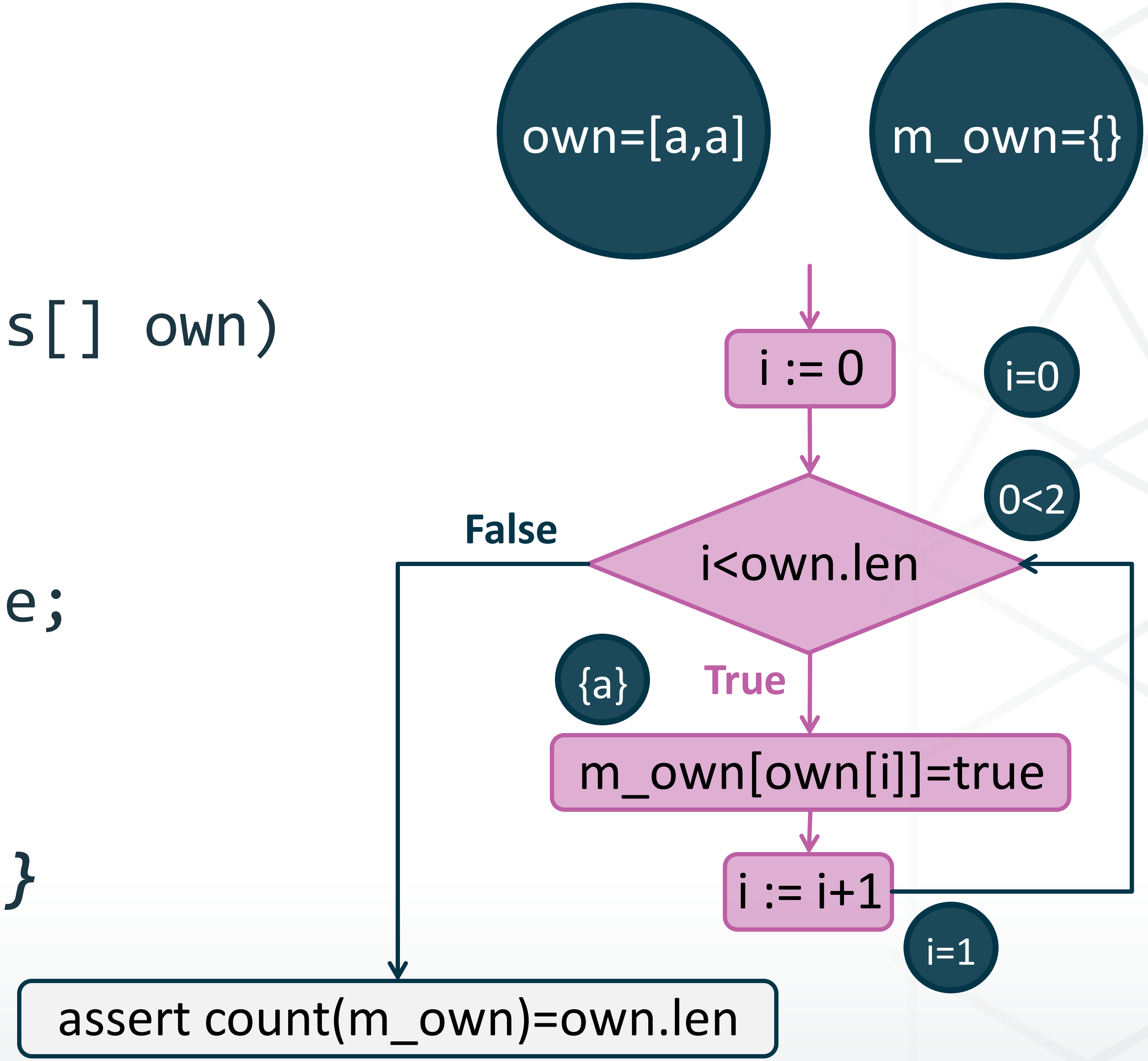
```
{ count(m_own) = 0 }

wallet_constructor(address[] own)
    int i = 0;
    while (i < own.len)
        m_own[own[i]] = true;
        ++i;

{ count(m_own) = own.len }
```



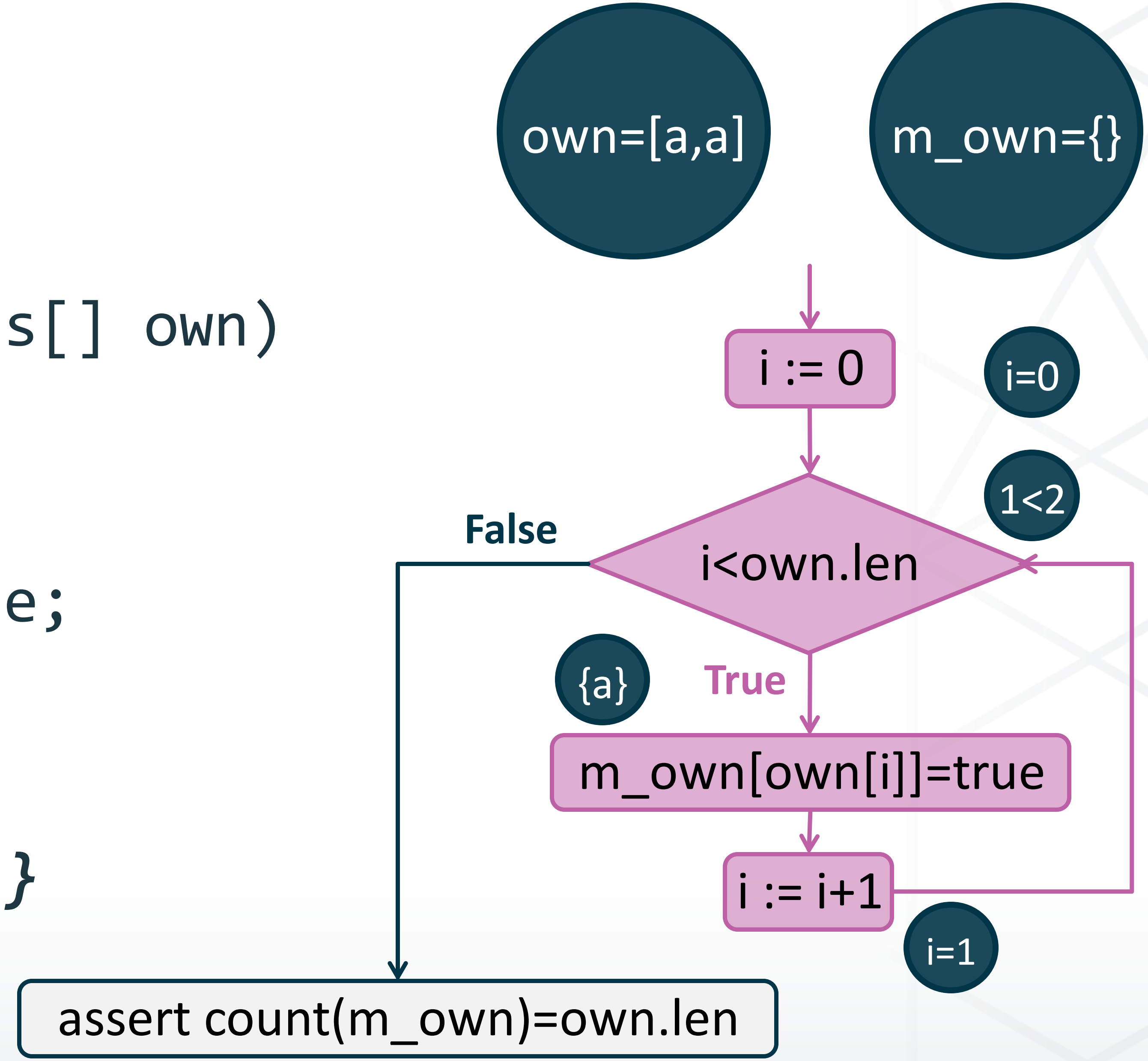
```
{ count(m_own) = 0 }

wallet_constructor(address[] own)
   int i = 0;
   while (i < own.len)
      m_own[own[i]] = true;
      ++i;

{ count(m_own) = own.len }
```



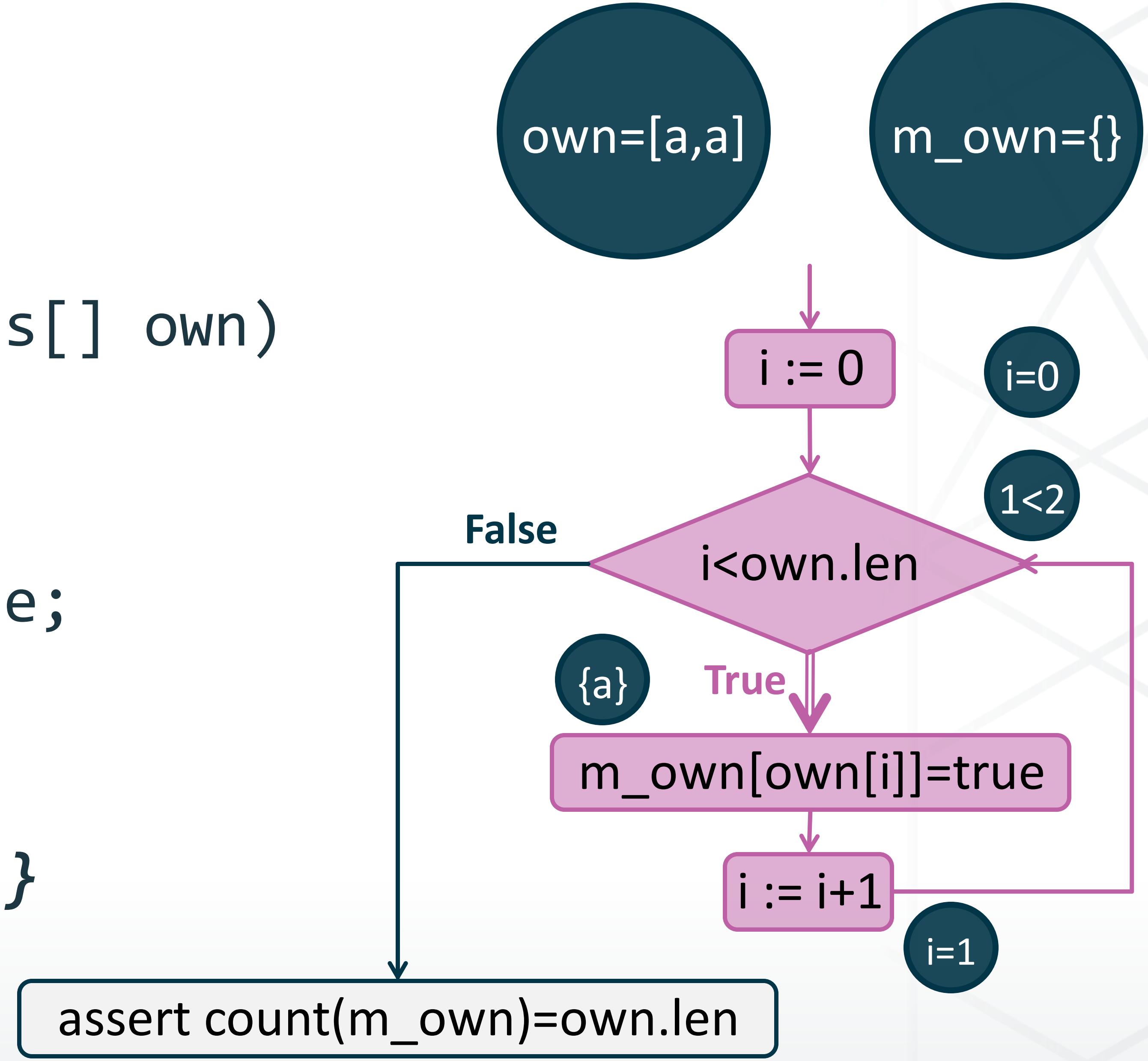
```
{ count(m_own) = 0 }

wallet_constructor(address[] own)
   int i = 0;
   while (i < own.len)
      m_own[own[i]] = true;
      ++i;

{ count(m_own) = own.len }
```



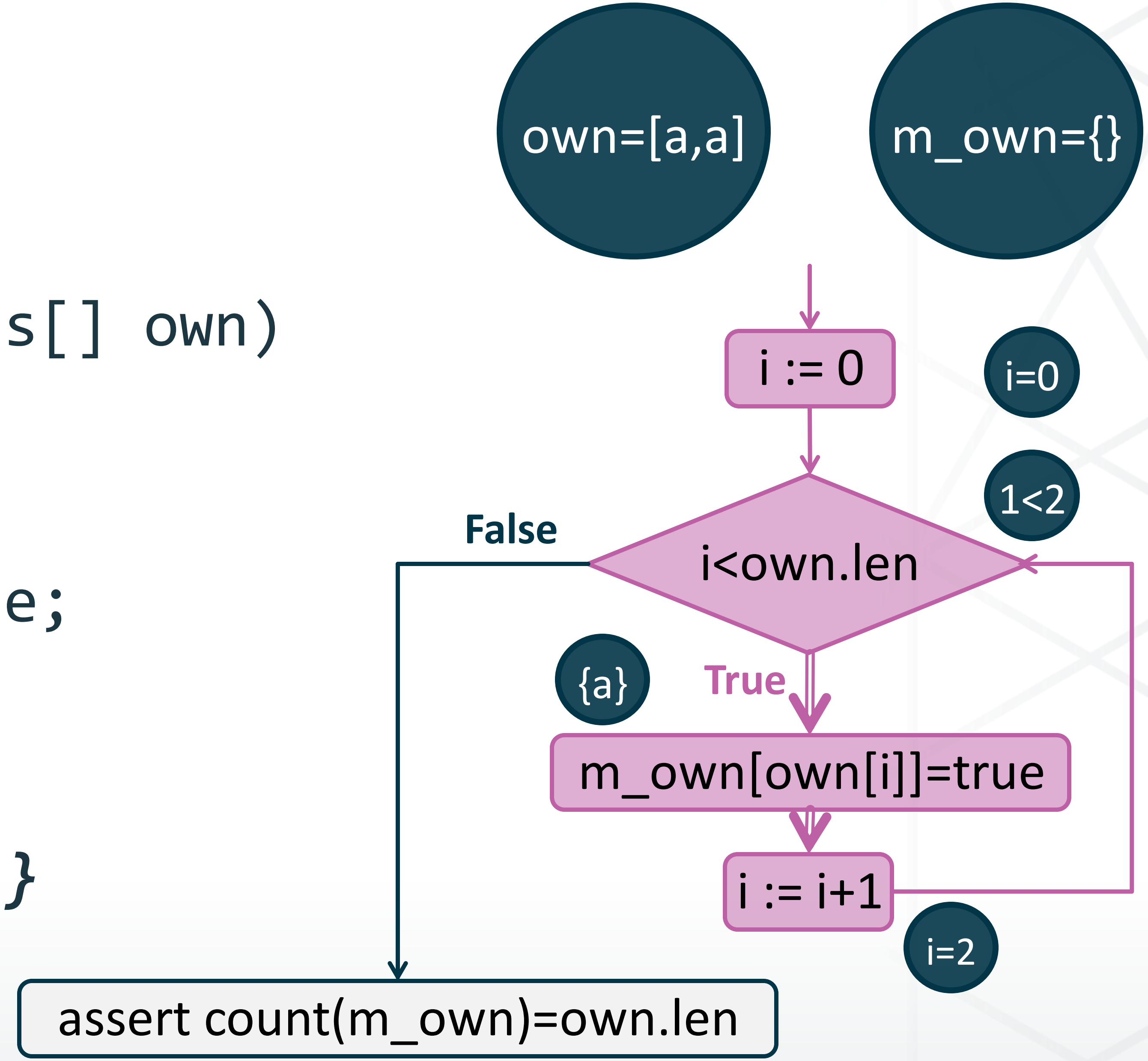
```
{ count(m_own) = 0 }

wallet_constructor(address[] own)
    int i = 0;
    while (i < own.len)
        m_own[own[i]] = true;
        ++i;

{ count(m_own) = own.len }
```



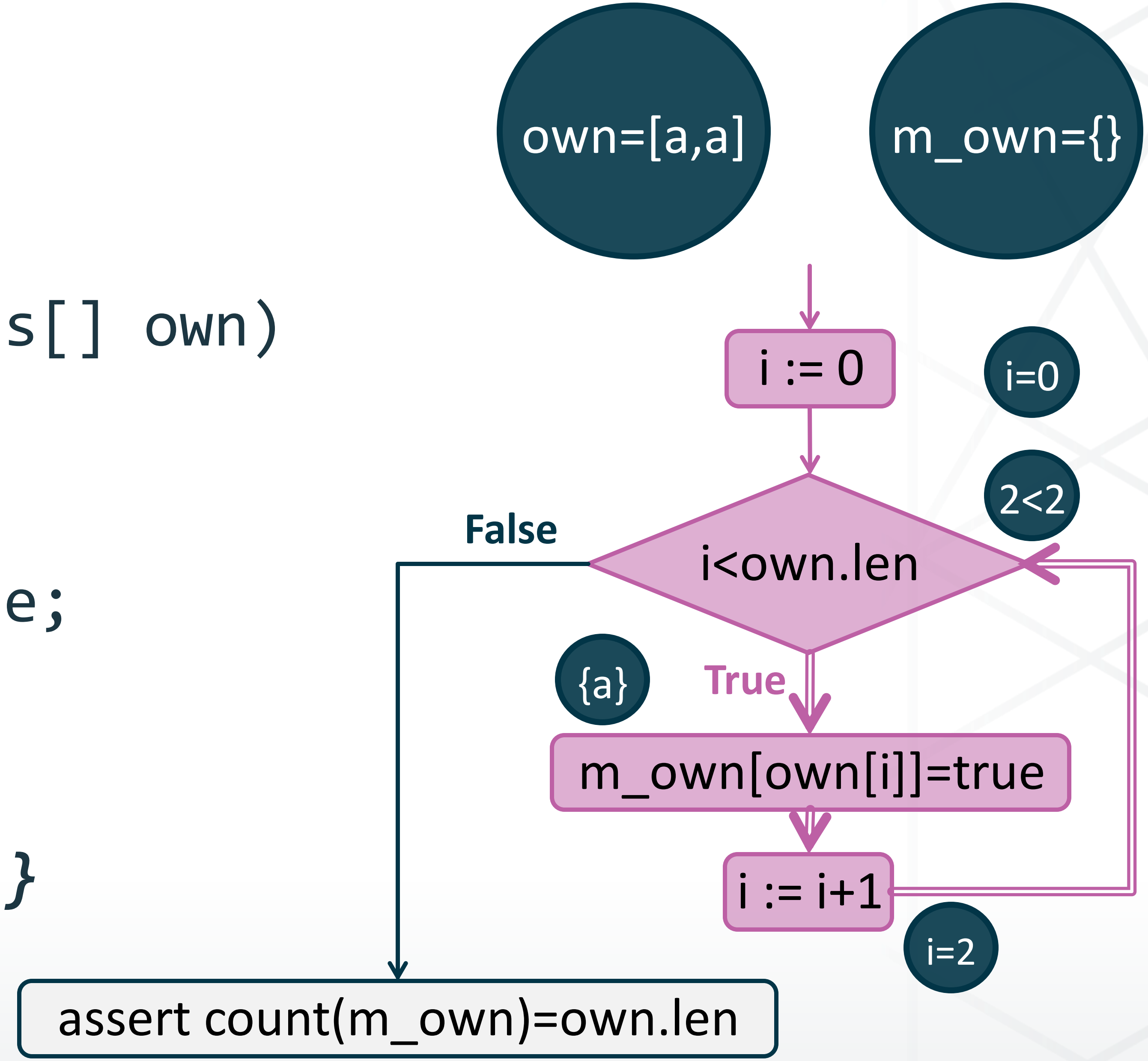
```
{ count(m_own) = 0 }

wallet_constructor(address[] own)
   int i = 0;
   while (i < own.len)
      m_own[own[i]] = true;
      ++i;

{ count(m_own) = own.len }
```



```
{ count(m_own) = 0 }

wallet_constructor(address[] own)
    int i = 0;
    while (i < own.len)
        m_own[own[i]] = true;
        ++i;

{ count(m_own) = own.len }
```



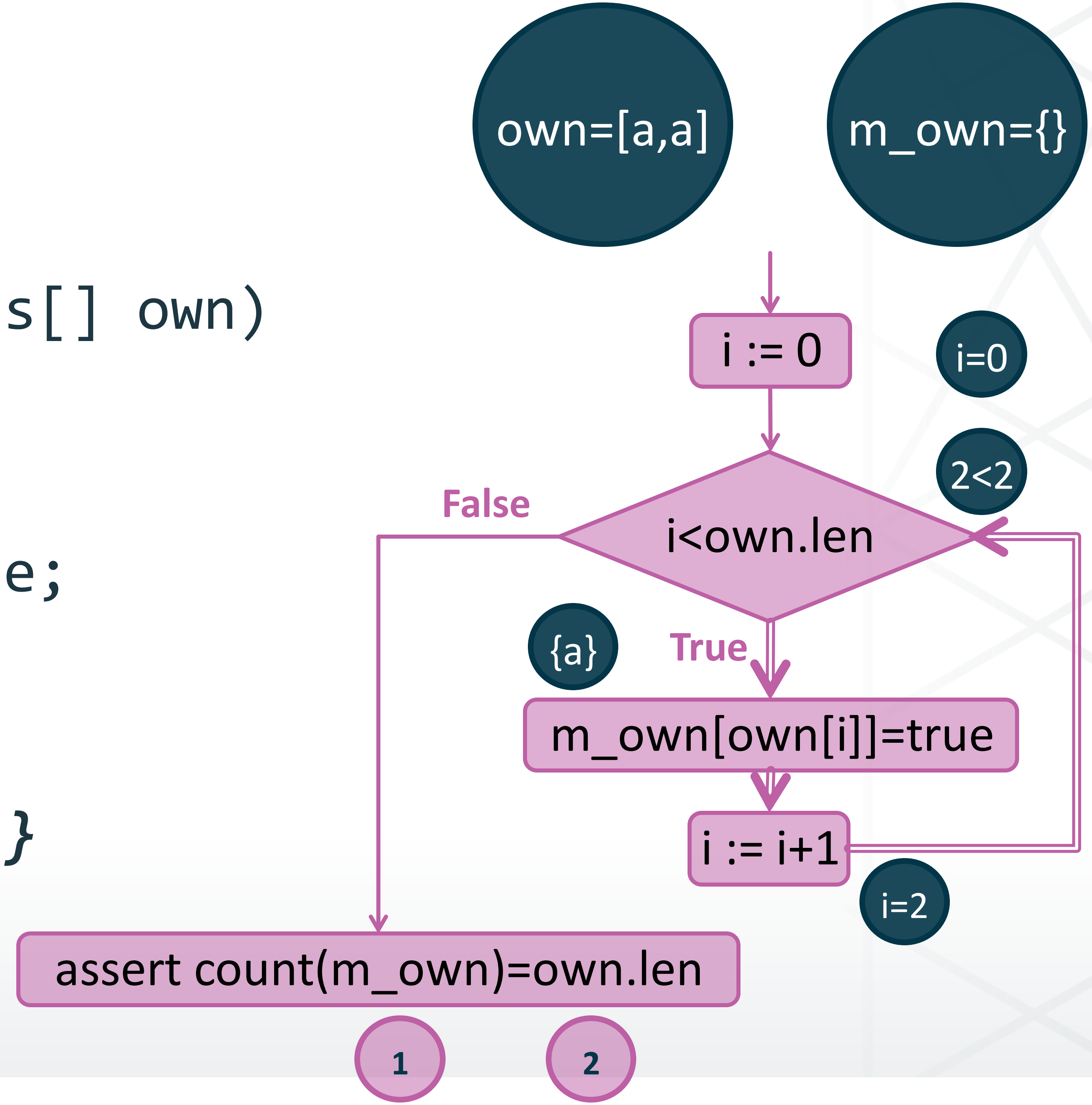
```
{ count(m_own) = 0 }

wallet_constructor(address[] own)
    int i = 0;
    while (i < own.len)
        m_own[own[i]] = true;
        ++i;

{ count(m_own) = own.len }
```



{ count(m_own) = 0 }

```
wallet_constructor(address[] own)
    int i = 0;
    while (i < own.len)
        m_own[own[i]] = true;
        ++i;
```

{ count(m_own) = own.len }



Fixed Wallet

```
{ count(m_own) = 0 }

wallet_constructor_fixed(address[] own)
    int i = 0;
    while (i < own.len)
        if (m_own[own[i]])
            abort;
        m_own[own[i]] = true;
        ++i;

{ count(m_own) = own.len }
```

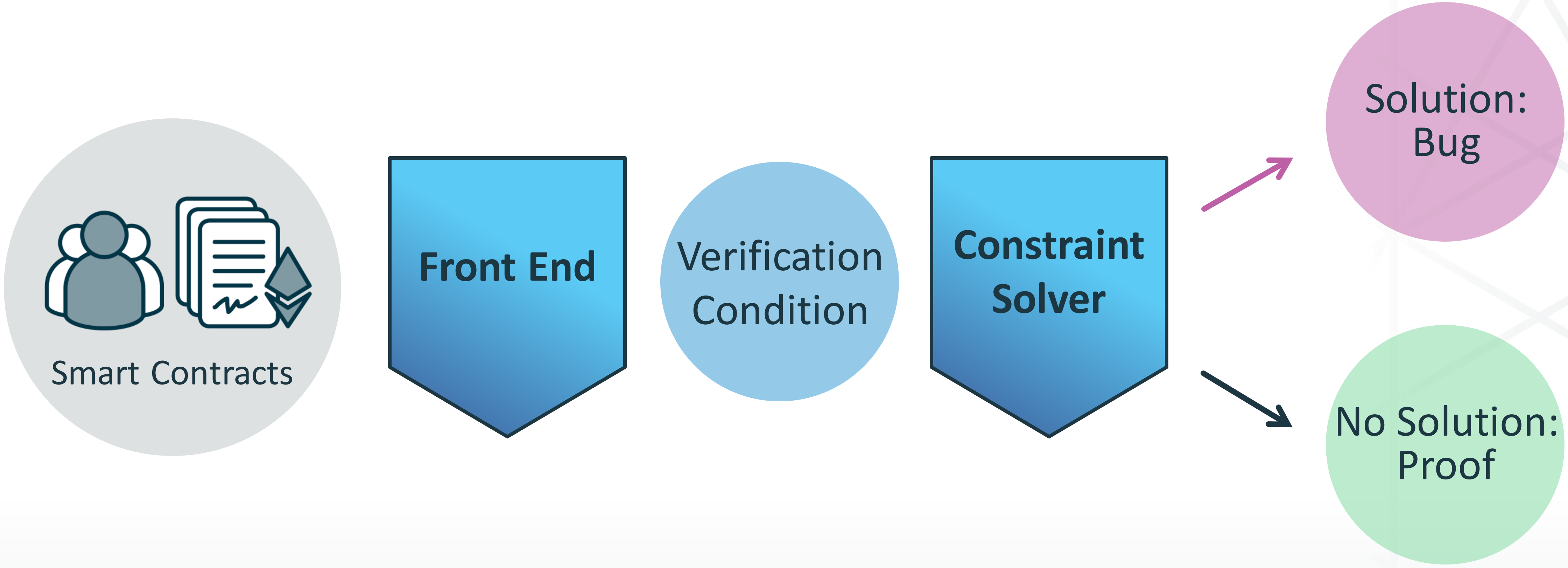
Fixed Wallet

```
{ count(m_own) = 0 }

wallet_constructor_fixed(address[] own)
    int i = 0;
    while (i < own.len)
        if (m_own[own[i]])
            abort;
        m_own[own[i]] = true;
        ++i;

{ count(m_own) = own.len }
```

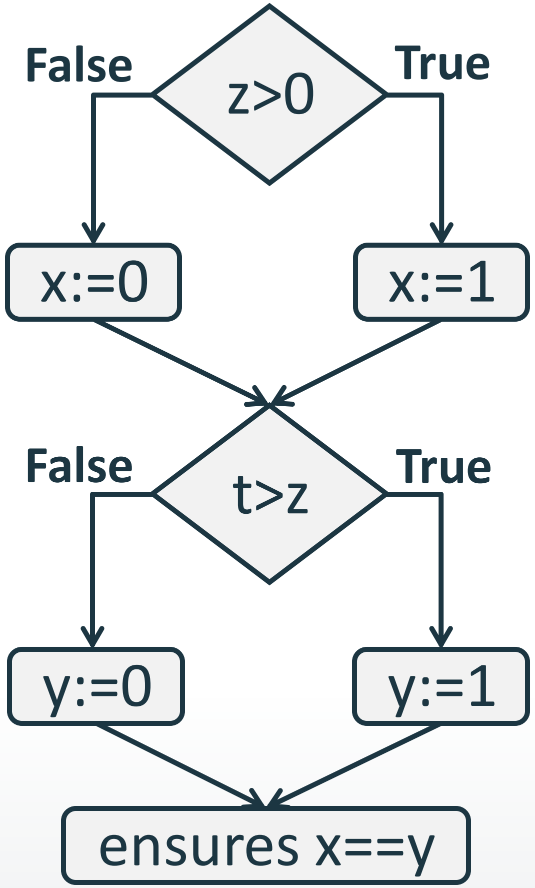
```
{ count(m_own) = 0 }
    int i = 0;
    while (i < own.len) { own.len>i≥ 0 ∧ count(m_own) = i }
        if (m_own[own[i]])
            abort;
        { own.len>i ≥ 0 ∧ count(m_own) = i ∧ ¬m_own[own[i]] }
        m_own[own[i]] = true;
        { own.len>i ≥ 0 ∧ count(m_own) = i+1 }
        ++i;
        { own.len ≥ i ≥ 0 ∧ count(m_own) = i }
{ count(m_own) ≥ own.len }
```

How does Certora-AEV automatically check correctness?

Secret Sauce – Compilation and Constraint Solving



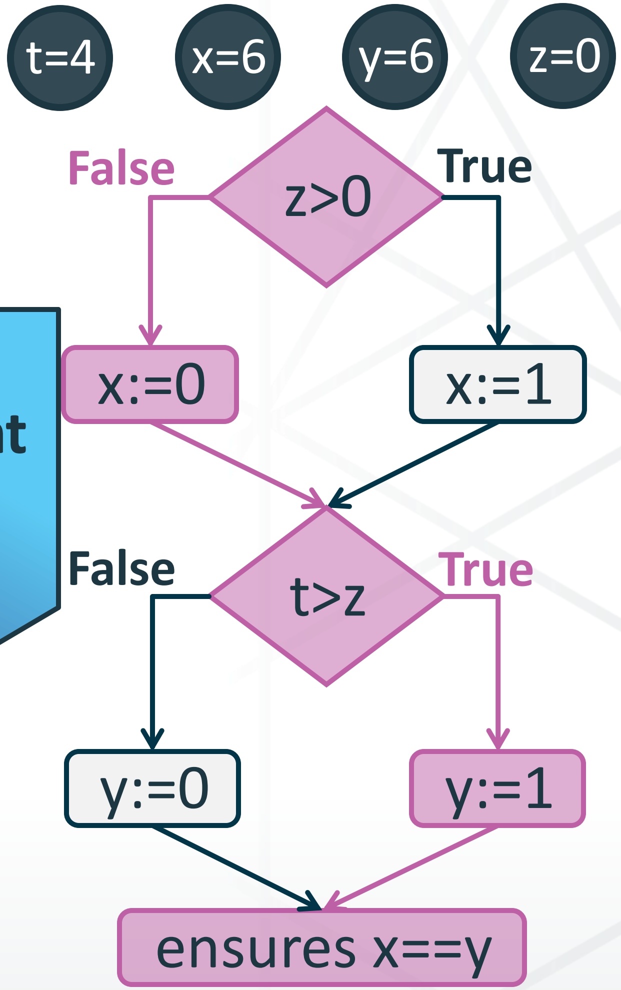
Secret Sauce – Compilation and Constraint Solving



Front End

$$(z>0 \wedge x=1) \vee (z \leq 0 \wedge x=1)$$
$$(t>z \wedge y=1) \vee (z \leq 0 \wedge y=1)$$
$$x \neq y$$

Constraint Solver



Summary

- Ensured correctness is critical for the adoption of Smart Contracts
- Formal verification is the tool we have
- Enabling technologies
  - Modularity
  - Mature tools