Fantastic Bugs

& how to automatically find them using reusable invariants

Mooly Sagiv

TEL AVIV UNIVERSITY אוניברסיטת תל אביב

CERTORA
Keeping your code secure forever

Free demo & user-guide at demo.certora.com

JANUARY 2020

2020

The Certora Team

Shelly Grossman
Chief Scientist

Dr. James Wilcox
CTO

Dr. John Toman
Formal Verification Expert

Dr. Nurit Dor
Head of Product

Dr. Alexander Nutz
Formal Verification Expert

Or Pistiner
Software Engineer

Anastasia Fedotov Software Engineer

Thomas Bernardi
Software Engineer

Marcelo Taube
Product Architect

Lior Oppenheim
Security Researcher

And also

Neil Immerman
UMASS

Daniel Jackson
MIT

Noam Rinetzky

Top tier customers

"...Certora's technology is used daily to locate **mind blowing** bugs"

Compound

Geoff Hayes, CTO

"The Certora-Prover has already surfaced **significant problems** missed by expensive and unscalable manual audits."

coinbase

Shamiq Islam, Head of Security

"Certora's Prover tool has already uncovered a number of **nuanced bugs** and also **mathematically proved** interesting properties of linked lists."

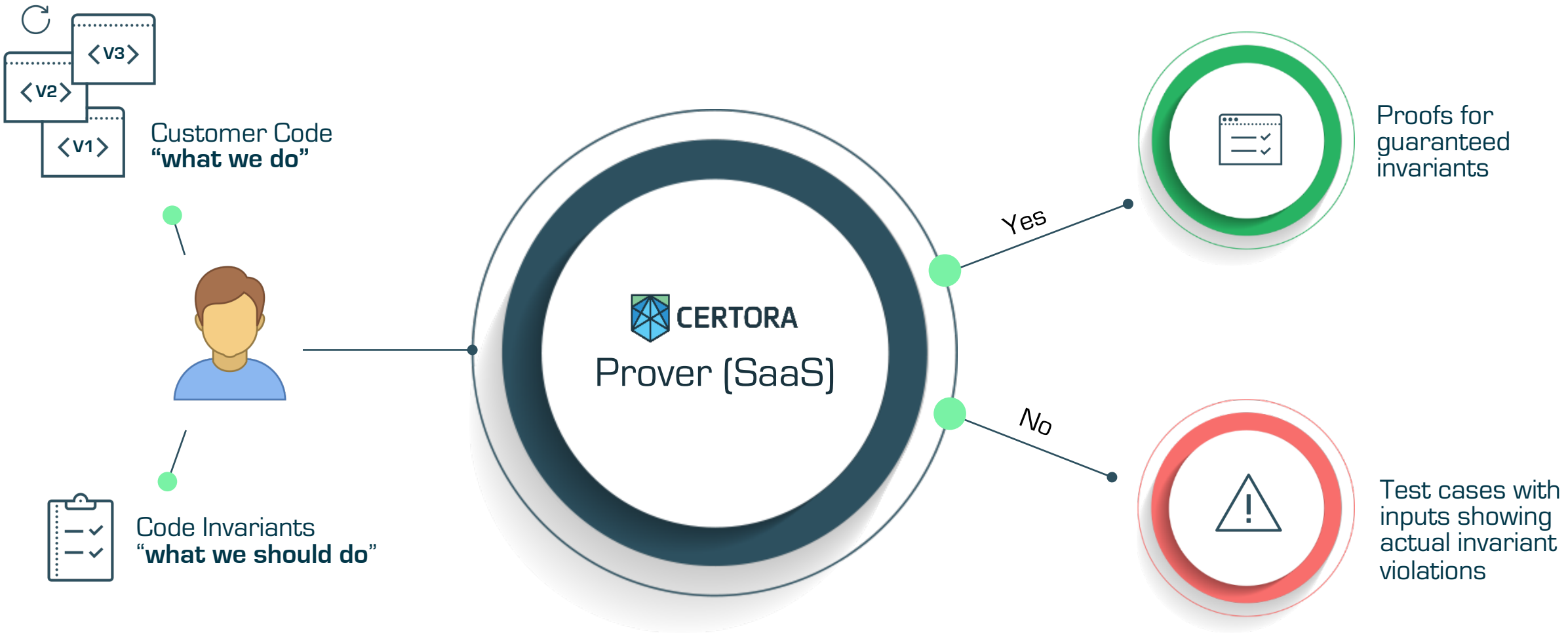
celo

Marek Olszweski, Founder

orchid

STARKWARE

Product: Continuous Code Verification



MOVE FAST AND BREAK NOTHING

Secret Sauce: Constraint Solving

Code:

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

Constraints:

```
balances@start[from] ≥ amount
balancesFrom = balances@start[from] – amount
balancesTo = balances@start[to] + amount
balances[from] = balancesFrom
balances[to] = balancesTo
balances[from] + balances[to]  ≠
        balances@start[from] + balances@start[to]
```

Invariant:

$\sum_{a:\ address} balances[a]$ ✗

Solution:

```
from="Alice"
to="Alice"
amount = 18
balances@start[Alice] = 20
balances[Alice] = 38
```

The Bug

```
transfer (Alice, Alice, 18) {
    require (balances[from] ≥ 18);
    balancesFrom := balances[from] - 18;
    balancesTo := balances[to] + 18;
    balances[from] := balancesFrom;
    balances[to] := balancesTo;
}
```

Balances [Alice]	balances[from]	balances[to]
20		

The Bug

```
transfer (Alice, Alice, 18) {
    require (balances[from] ≥ 18);
    balancesFrom := balances[from] - 18;
    balancesTo := balances[to] + 18;
    balances[from] := balancesFrom;
    balances[to] := balancesTo;
}
```

Balances [Alice]	balances[from]	balances[to]
20	2	

The Bug

```
transfer (Alice, Alice, 18) {
    require (balances[from] ≥ 18);
    balancesFrom := balances[from] - 18;
    balancesTo := balances[to] + 18;
    balances[from] := balancesFrom;
    balances[to] := balancesTo;
}
```

Balances [Alice]	balances[from]	balances[to]
20	2	38

The Bug

```
transfer (Alice, Alice, 18) {
    require (balances[from] ≥ 18);
    balancesFrom := balances[from] - 18;
    balancesTo := balances[to] + 18;
→   balances[from] := balancesFrom;
    balances[to] := balancesTo;
}
```

Balances [Alice]	balances[from]	balances[to]
2	2	38

The Bug

```
transfer (Alice, Alice, 18) {
    require (balances[from] ≥ 18);
    balancesFrom := balances[from] - 18;
    balancesTo := balances[to] + 18;
    balances[from] := balancesFrom;
    balances[to] := balancesTo;
}
```

Balances [Alice]	balances[from]	balances[to]
38	2	38

Invariant:

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

Secret Sauce: Constraint Solving

Code:

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

Constraints:

$from \neq to$

$balances@start[from] \geq amount$

$balancesFrom = balances@start[from] - amount$

$balancesTo = balances@start[to] + amount$

$balances[from] = balancesFrom$

$balances[to] = balancesTo$

$balances[from] + balances[to] \neq balances@start[from] + balances@start[to]$

Invariant:

$\sum_{a:\ address} balances[a]$ ✓

A mathematical proof that the invariant is maintained

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 has to be done at accurate machine level	Abstraction is key to scalability: - Natural vs. bit-vector arithmetic - Memory abstraction - Loop abstraction - Ignore gas
Must consider all objects at once	Modularity concept enable scalability
FV is one-time deal	FV guarantees code upgrade safety

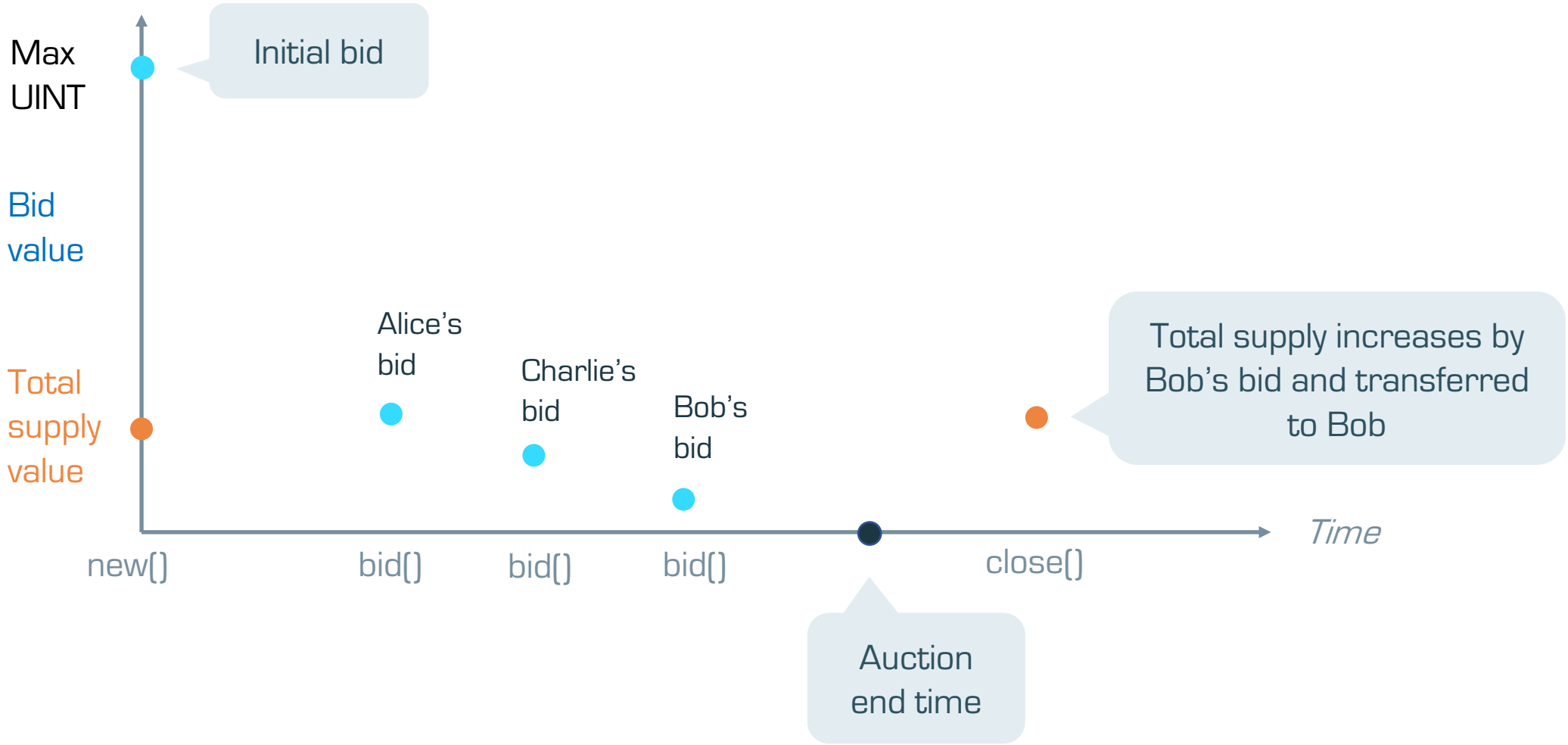
The Bounded Supply Invariant (ERC20 tokens)

"Nobody should be able to mint unbounded number of tokens"

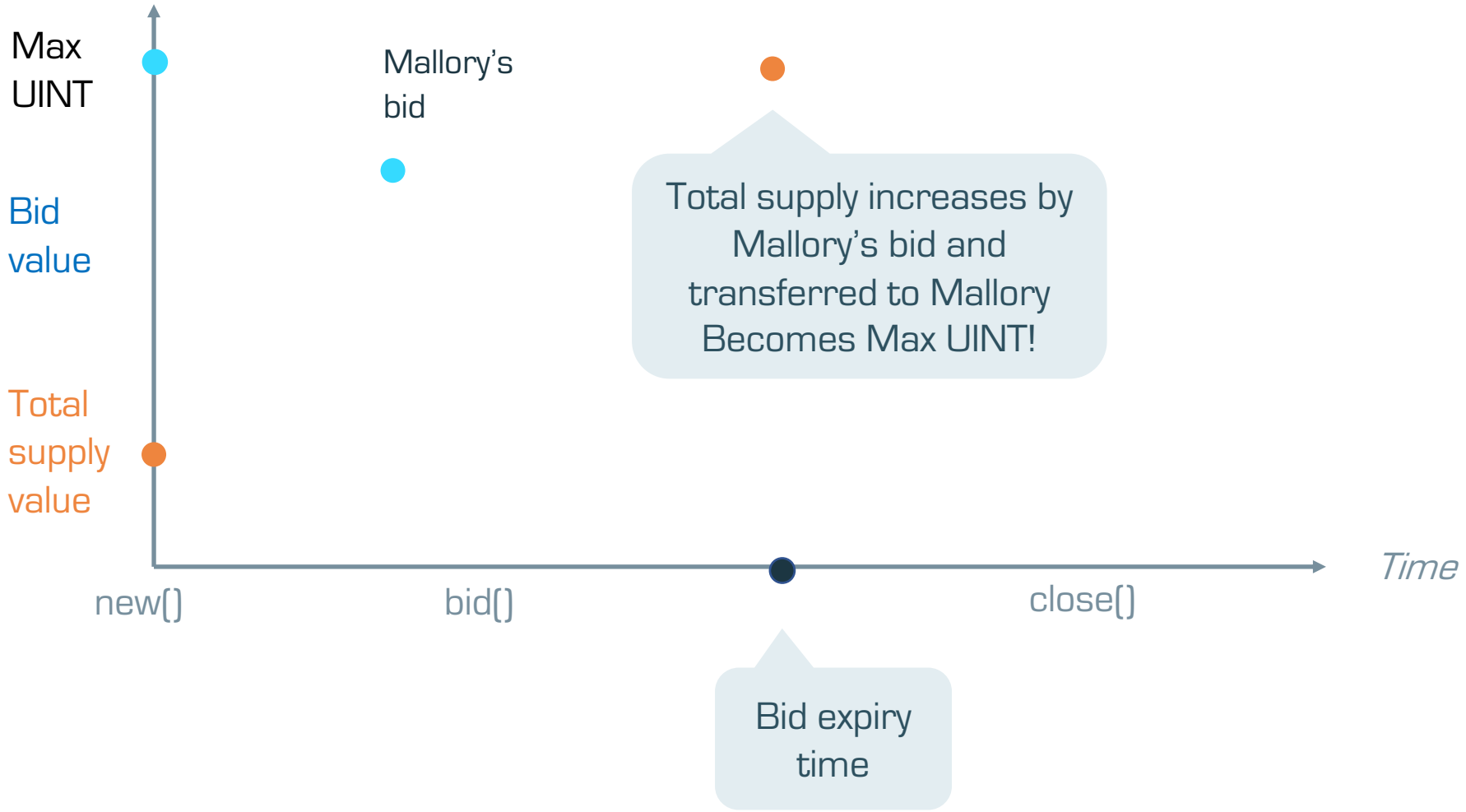
Shamiq Islam
Head of Security, coinbase

Minted tokens < predefined amount

MakerDao Test Version



MakerDao Test Version



Checking the bounded supply invariant

Results for boundedSupply:

Go back to top

Function name	Result	Time(secs)	Dump
balanceOf(address)	👍	3	
close(uint256)	👎	1	
totalSupply()	👍	0	
transferTo(address,uint256)	👍	0	
bid(uint256,uint256)	👍	0	
getAuction(uint256)	👍	0	
owner()	👍	0	
newAuction(uint256,uint256)	👍	0	
getTotalSupply()	👍	0	

close(uint256)

Assertion message:

Cannot increase to MAX_UINT256

Call Trace

boundedSupply(f=close(uint256))

System.totalSupply() / 0 SUCCESS

System.close(uint256=0) SUCCESS

System.totalSupply() / ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff SUCCESS

Variables

e.block.number=1
e.block.timestamp=1
e.msg.address=ce4604a00000000000000000010002
e.msg.sender=1
e.msg.value=0
e.tx.origin=0
totalSupply(e)18:5=0

Fixing the Code

```
function close(uint id)  public {
    require(auctions[id].bid_expiry != 0
                && (auctions[id].bid_expiry < now || auction
                 s[id].end_time < now));
       require(auctions[id].prize.safeAdd(auctions[id].prize)
       + getTotalSupply() >= getTotalSupply());
    mint(auctions[id].winner, auctions[id].prize);
    delete auctions[id];
    }
```

Checking the bounded supply invariant

Results for System:

Go to available contracts listing

Test name	Result	Time(sec)	Dump
boundedSupply	👍	5	
senderCanOnlyIncreaseOthersBalance	👍	2	
transferWithIllegalValue	👍	0	

Results for boundedSupply:

Go back to top

Function name	Result	Time(secs)	Dump
balanceOf(address)	👍	1	
close(uint256)	👍	1	
totalSupply()	👍	0	
transferTo(address,uint256)	👍	0	
bid(uint256,uint256)	👍	1	
getAuction(uint256)	👍	1	

High Level Smart Contract Invariants

Informal Property:	Bugs Found:
Immunity to reentrancy attack	DAO, SpankChain, Constantinople fork
Robustness	Compound V1 Price Oracle
Bounded Token Supply	Maker MCD
Proportional token distribution	Compound V2, Maker MCD
Any loan can be fully repaid	Compound V2
Sufficient reserves	Several tokens, Maker MCD

Shelly @ Certora Geoff Hayes @ Compound Shamiq Islam @ Coinbase Jared Flatow @ Compound

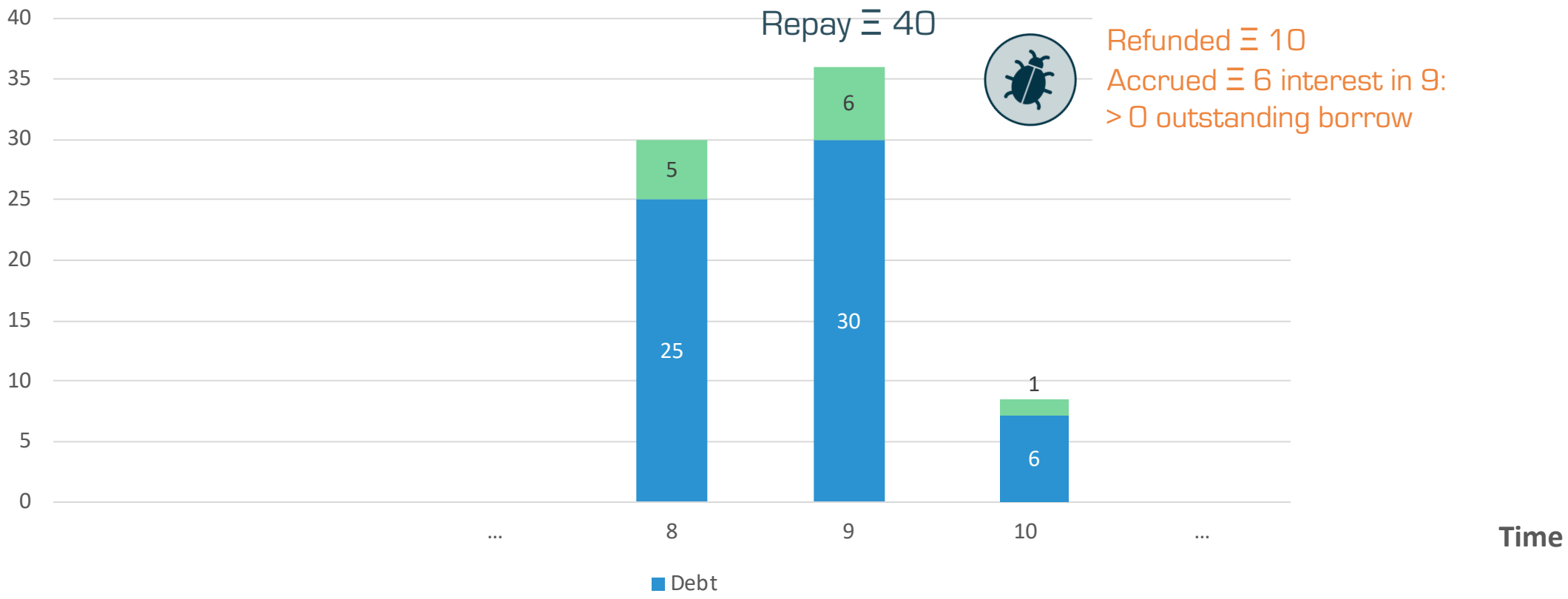
Correctness rules for Debt (Compound Finance)

Any debt can be paid off: repayAmount ≥ borrowed ➔ newBorrowBalance = 0

```
function repayBehalfExplicit(address borrower, CEther cEther_)
                public
                payable {
        uint received = msg.value;
        uint borrows = cEther_.borrowBalanceStored(borrower);
        if (received > borrows) {
            cEther_.repayBorrowBehalf.value(borrows)(borrower);
            msg.sender.transfer(received - borrows);
        } else {
            cEther_.repayBorrowBehalf.value(received)(borrower);
        }
    }
```

Correctness rules for Debt (Compound Finance)

Any debt can be paid off: repayAmount ≥ borrowed ➔ newBorrowBalance = 0



Correctness rules for Debt (Compound Finance)

Any debt can be paid off: repayAmount ≥ borrowed ➔ newBorrowBalance = 0

```
function repayBehalfExplicit(address borrower, CEther cEther_)
                public
                payable {
        uint received = msg.value;
        uint borrows = cEther_.borrowBalanceStored(borrower);
        if (received > borrows) {
            cEther_.repayBorrowBehalf.value(borrows)(borrower);
            msg.sender.transfer(received - borrows);
        } else {
            cEther_.repayBorrowBehalf.value(received)(borrower);
        }
    }
```

Correctness rules for Debt (Compound Finance)

Any debt can be paid off: repayAmount ≥ borrowed ➔ newBorrowBalance = 0

```
function repayBehalfExplicit(address borrower, CEther cEther_)
                public
                payable {
            uint received = msg.value;
            uint borrows = cEther_.borrowBalanceCurrent(borrower);
            if (received > borrows) {
                cEther_.repayBorrowBehalf.value(borrows)(borrower);
                msg.sender.transfer(received - borrows);
            } else {
                cEther_.repayBorrowBehalf.value(received)(borrower);
            }
        }
```

Summary

1

Software specification is the holy grail of computer science

2

Community effort

3

Reusable invariants provide a layer of security used for early bug detection

4

Code is the law ➔ Spec is the law

Thank you!

CERTORA

Mooly Sagiv
mooly@certora.com
Twitter@SagivMooly
www.certora.com
+1-617-650-4612
+972-548-303-111