# Secure Smart Contracts with Isabelle/Solidity

on 2025-09-08


Diego Marmsoler

Department of Computer Science
University of Exeter


d.marmsoler@exeter.ac.uk


www.marmsoler.com


@DiegoMarmsoler


@dmarmsoler.bsky.social


Joint work with Achim D. Brucker and Asad Ahmed


This work was supported by EPSRC [grant number EP/X027619/1].

Diego Marmsoler

UNIVERSITY OF EXETER



Diego Marmsoler

UNIVERSITY OF EXETER



# Blockchain and Smart Contracts

**Blockchain** Novel technology to store data in *decentralized* and *immutable* manner

- Main application: Cryptocurrencies
- Other: Finance, Healthcare, Identity Management, ...



# Blockchain and Smart Contracts

**Blockchain** Novel technology to store data in *decentralized* and *immutable* manner

- Main application: Cryptocurrencies
- Other: Finance, Healthcare, Identity Management, ...

**Smart Contracts** Digital contracts which are automatically executed once certain conditions are met

- Example: Payment release
- Every day hundreds of thousands of contracts are deployed managing millions of dollars in assets



# Smart Contracts and Solidity

Technically, a smart contract (SC) is *code which is deployed to a blockchain*, and which can be executed by sending special transactions to it

- Usually developed in a high-level programming language
- Most popular language: Solidity



# Smart Contracts and Solidity

Technically, a smart contract (SC) is *code which is deployed to a blockchain*, and which can be executed by sending special transactions to it

- Usually developed in a high-level programming language
- Most popular language: Solidity

## Solidity

- Works on all EVM-based platforms, such as Ethereum, Polygon, ...
- More than 90% of all smart contracts are developed using Solidity



# A Simple Banking Contract in Solidity

Solidity

```
1  contract Bank {
2    mapping(address => uint256) balances;
3
4    function deposit() external payable {
5      balances[msg.sender] = balances[msg.sender] + msg.value;
6    }
7
8    function reset() internal {
9      balances[msg.sender] = 0;
10   }
11
12   function withdraw() external {
13     uint256 bal = balances[msg.sender];
14     reset();
15     msg.sender.transfer(bal);
16   }
17 }
```



# The Problem With Smart Contracts

As with every computer program, *SCs may contain bugs which can be exploited*



# The Problem With Smart Contracts

As with every computer program, *SCs may contain bugs which can be exploited*

Since SCs are used to automate financial transactions, such exploits may result in *high economic losses*

- Example: DAO attack in 2016 resulted in a loss of approximately $60M
- Since 2019, more than $5B have been lost due to vulnerabilities in SCs



# The Problem With Smart Contracts

As with every computer program, *SCs may contain bugs which can be exploited*

Since SCs are used to automate financial transactions, such exploits may result in *high economic losses*

- Example: DAO attack in 2016 resulted in a loss of approximately $60M
- Since 2019, more than $5B have been lost due to vulnerabilities in SCs

Together with the fact that SCs are only difficult to update/remove it is important to *"get them right"* before deployment



## Smart Contract Verification

Popular approaches to verify Solidity smart contracts

| | |
|---|---|
| Certora | Chandrakana Nandi, Mooly Sagiv, and Daniel Jackson (2022) |
| SolCMC | Leonardo Alt (2022) |
| solc-verify | Ákos Hajdu and Dejan Jovanović (2020) |



## Smart Contract Verification

Popular approaches to verify Solidity smart contracts

| | |
|---|---|
| Certora | Chandrakana Nandi, Mooly Sagiv, and Daniel Jackson (2022) |
| SolCMC | Leonardo Alt (2022) |
| solc-verify | Ákos Hajdu and Dejan Jovanović (2020) |

All based on SMT solvers

- Axiomatic: Easy to introduce inconsistencies (soundness)
- Automatic: Fail to verify more complex properties (completeness)



## Smart Contract Verification

Popular approaches to verify Solidity smart contracts

- Certora Chandrakana Nandi, Mooly Sagiv, and Daniel Jackson (2022)
- SolCMC Leonardo Alt (2022)
- solc-verify Ákos Hajdu and Dejan Jovanović (2020)

All based on SMT solvers

- Axiomatic: Easy to introduce inconsistencies (soundness)
- Automatic: Fail to verify more complex properties (completeness)

Isabelle/Solidity

- Foundational approach guarantees correctness by construction
- Based on HOL allows verification of more complex properties

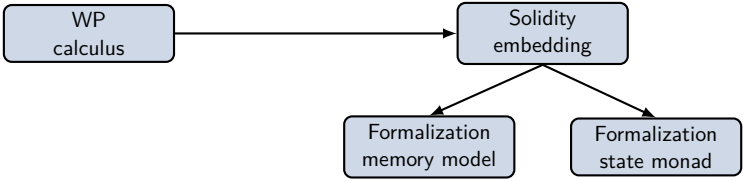


Diego Marmsoler

UNIVERSITY OF EXETER

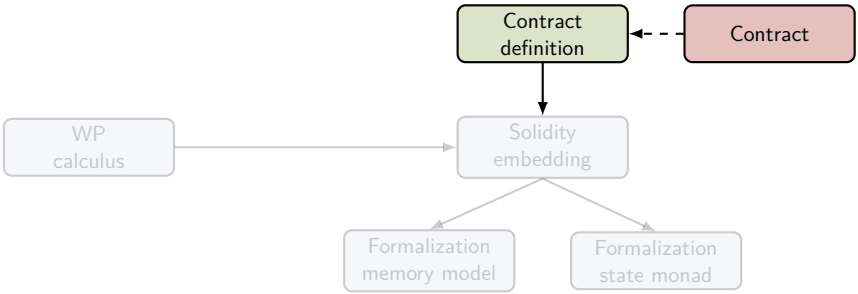


# Architecture of Isabelle/Solidity



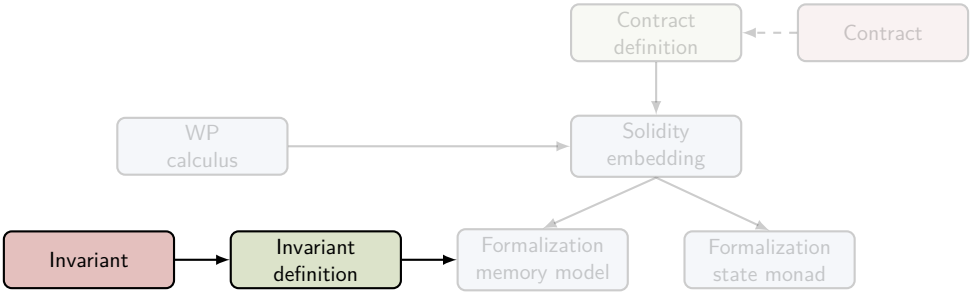



# Architecture of Isabelle/Solidity



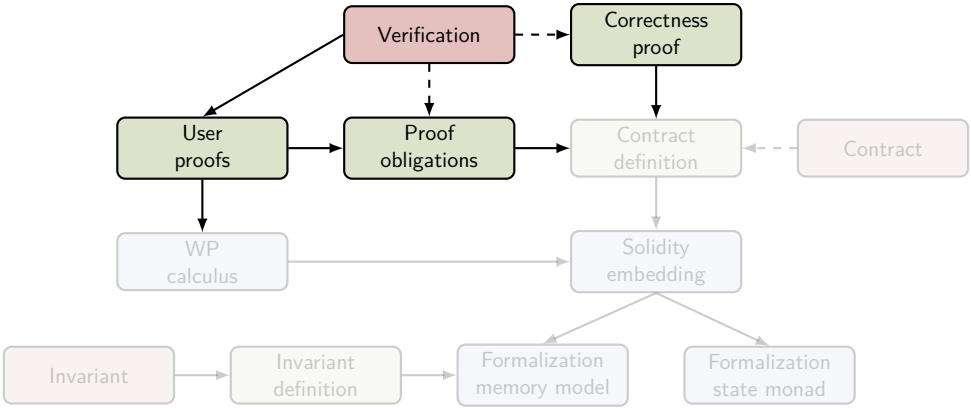



# Architecture of Isabelle/Solidity






# Architecture of Isabelle/Solidity






# Banking Contract In Isabelle/Solidity

Isabelle

```
1  contract Bank
2    for balances: TMap (TValue TAddress) (TValue TSint)
3
4    constructor where
5      ⟨skip⟩
6
7    cfunction deposit external payable where
8      balances [⟨sender⟩] ::=s balances ∼s [⟨sender⟩] ⟨+⟩ ⟨value⟩ ,
9
10   cfunction reset where
11     balances [⟨sender⟩] ::=s ⟨sint⟩ 0 ,
12
13   cfunction withdraw external where
14     do {
15       bal :: TSint;
16       bal [] ::= balances ∼s [⟨sender⟩];
17       icall reset;
18       ⟨transfer⟩ ⟨sender⟩ (bal ∼ [])
19     }
```



# Banking Contract In Isabelle/Solidity

Isabelle

```
1  contract Bank
2    for balances: TMap (TValue TAddress) (TValue TSint)
3
4    constructor where
5      ⟨skip⟩
6
7    cfunction deposit external payable where
8      balances [⟨sender⟩] ::=ₛ balances ∼ₛ [⟨sender⟩] ⟨+⟩ ⟨value⟩ ,
9
10   ...
```

## Generated artifacts

- Mutual recursive, partial function definitions
- Inductive proof rule



# Invariant For Banking Contract In Isabelle/Solidity

$P(\texttt{balances}, \mathit{balance})$?



# Invariant For Banking Contract In Isabelle/Solidity

$$\sum_{ad} \texttt{balances}(ad) \leq \mathit{balance}$$



# Invariant For Banking Contract In Isabelle/Solidity

$$\sum_{ad} \texttt{balances}(ad) \leq \mathit{balance}$$

Isabelle

```
1 invariant sum_bal sb where
2 snd sb ≥
3   ∑ad. unat (sint (vt ((mp (fst sb balances)) (Address ad))))
4 for Bank
```



# Invariant For Banking Contract In Isabelle/Solidity

$$\sum_{ad} \texttt{balances}(ad) \leq \mathit{balance}$$

Isabelle

```
1 invariant sum_bal sb where
2 snd sb ≥
3   ∑ad. unat (sint (vt ((mp (fst sb balances)) (Address ad))))
4 for Bank
```

## Generated artifacts

- Definition for invariant
- Specification and proofs for introduction and elimination rules



# Verifying Banking Contract In Isabelle/Solidity

Isabelle

```
verification sum_bal:
  sum_bal
  K (K (K True))
  deposit K (K (K True)) and
  withdraw K (K (K True)) and
  reset reset_post
  for Bank
```



# Verifying Banking Contract In Isabelle/Solidity

Isabelle

```
verification sum_bal:
  sum_bal
  K (K (K True))
  deposit K (K (K True)) and
  withdraw K (K (K True)) and
  reset reset_post
  for Bank
```

## Generated artifacts

- Proof obligations
- Correctness proof by fixed-point induction



# Verifying Banking Contract In Isabelle/Solidity

Isabelle/Solidity generates one proof obligation for each method

- Constructor: Establishes invariant and post-condition
- Internal: Establishes post-condition
- External: Preserves invariant and establishes post-condition



# Verifying Banking Contract In Isabelle/Solidity

Isabelle/Solidity generates one proof obligation for each method

- Constructor: Establishes invariant and post-condition
- Internal: Establishes post-condition
- External: Preserves invariant and establishes post-condition

Isabelle

```
1 ⋀ call.
2  (⋀ x h r. effect (call x) h r ⟹ vcond x h r) ⟹
3  effect (deposit call) s r ⟹
4  inv_state sum_bal s ⟹
5  post s r sum_bal (K True) (K (K (K True)))
```



# Verifying Banking Contract In Isabelle/Solidity

Isabelle

```
show "⋀call.
  (⋀x h r. effect (call x) h r ⟹ vcond x h r) ⟹
  effect (deposit call) s r ⟹
  inv_state sum_bal s ⟹
  post s r sum_bal (K True) (K (K (K True)))"
unfolding deposit_def
apply (erule post_exc_true, erule_tac post_wp)
unfolding inv_state_def deposit_post_def
apply vcg
apply (auto simp add: wpsimps)
apply (rule bal_msg_sender, assumption)
apply vcg
apply (auto simp add: wpsimps intro!: sum_balI 1)
apply vcg
apply (auto simp add: wpsimps)
apply (rule bal_msg_sender, assumption)
by vcg
```



# Isabelle/Solidity Language Features

Isabelle/Solidity supports a large subset of Solidity

- Domain specific language features: payable, transfer, balance, ...
- Advanced storage model: Storage, Memory, Calldata, Stack
- Semantic intricacies: fallback functions, safe/unsafe arithmetic, array assignments



Diego Marmsoler

UNIVERSITY OF EXETER



# Semantic Conformance

| Test | # tests |
|---|---|
| **State Updates** | 09 |
| **Basic Operators** | 19 |
| **Storage Lookups** | 07 |
| **Stack Lookups** | 14 |
| **Conditionals** | 2 |
| **Store Assignment** | 15 |
| **Variable Declarations** | 04 |
| **Total** | **70** |



# Case Studies

## Banking contract

- User to deposit and withdraw funds
- Based on the idea of ERC-20 Tokens
- Property: The funds associated with our contract on the blockchain covers at least the sum of all internal balances



# Case Studies

## Banking contract

- User to deposit and withdraw funds
- Based on the idea of ERC-20 Tokens
- Property: The funds associated with our contract on the blockchain covers at least the sum of all internal balances

## Casino contract

- Betting contract based on the idea of a flipping a coin
- VerifyThis long-term verification challenge[a]
- Property: Casino has always enough funds to cover the pot

[a] https://verifythis.github.io/02casino/



# Case Studies

## Voting contract

- Implements delegated voting idea
- Official example from Solidity documentation[a]
- Property: Number of votes is always bound by the number of eligible voters

[a] https://docs.soliditylang.org/en/v0.8.25/solidity-by-example.html#voting



# Case Studies

## Voting contract

- Implements delegated voting idea
- Official example from Solidity documentation[a]
- Property: Number of votes is always bound by the number of eligible voters

[a] https://docs.soliditylang.org/en/v0.8.25/solidity-by-example.html#voting

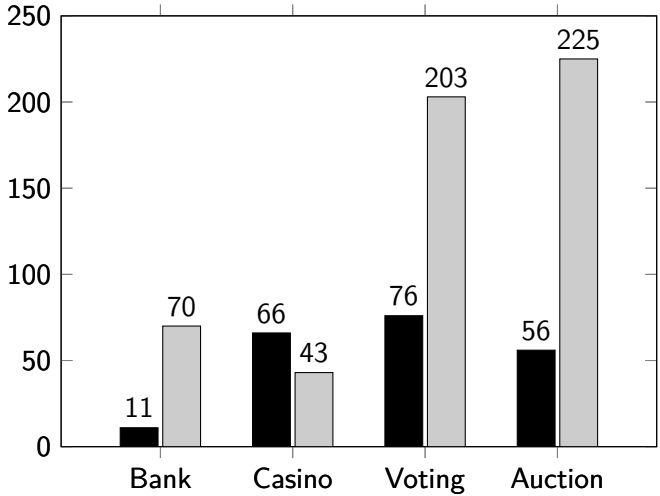
## Auction contract

- Open auction with bidding and automatic determination of highest bidder
- Official example from Solidity documentation[a]
- Property: Beneficiary and bidders will always be able to get their funds

[a] https://docs.soliditylang.org/en/v0.8.25/solidity-by-example.html#simple-open-auction



# Case Studies



Diego Marmsoler

UNIVERSITY OF EXETER



# Summary

## A foundational approach

- Specification and verification from first principles
- Correct by construction



# Summary

## A foundational approach

- Specification and verification from first principles
- Correct by construction

## Supports a large subset of Solidity features

- Domain specific language features and semantic intricacies
- Advanced storage model with support for storage, memory, calldata, stack



# Summary

## A foundational approach

- Specification and verification from first principles
- Correct by construction

## Supports a large subset of Solidity features

- Domain specific language features and semantic intricacies
- Advanced storage model with support for storage, memory, calldata, stack

## High semantic conformance

- Large set of unit tests
- Fuzzy testing framework in development



# Summary

## A foundational approach

- Specification and verification from first principles
- Correct by construction

## Supports a large subset of Solidity features

- Domain specific language features and semantic intricacies
- Advanced storage model with support for storage, memory, calldata, stack

## High semantic conformance

- Large set of unit tests
- Fuzzy testing framework in development

## Used to verify popular contracts

- Used for the verification of four popular contracts
- Results are promising



# Limitations and Future Work

## Memory Arrays

- Reasoning about memory arrays is difficult
- Calculus for Solidity-like memory arrays (Asad Ahmed)



# Limitations and Future Work

## Memory Arrays

- Reasoning about memory arrays is difficult
- Calculus for Solidity-like memory arrays (Asad Ahmed)

## Correctness of Bytecode

- Compiler could introduce bugs
- Verified compilation (Mark Utting, Naipend Dong, Horacio M. A. Quiles, and Achim D. Brucker)



# Limitations and Future Work

## Memory Arrays

- Reasoning about memory arrays is difficult
- Calculus for Solidity-like memory arrays (Asad Ahmed)

## Correctness of Bytecode

- Compiler could introduce bugs
- Verified compilation (Mark Utting, Naipend Dong, Horacio M. A. Quiles, and Achim D. Brucker)

## Missing of Advanced Features

- Inheritance, Libraries, Inline Assembly, ...
- Additional Case Studies (Asad Ahmed and Filip Maric)
- Verification Competition (Massimo Bartoletti and Enrico Lipparini)



# References I


Diego Marmsoler, Asad Ahmed, and Achim D. Brucker.
Secure Smart Contracts with Isabelle/Solidity.
In Alexandre Madeira and Alexander Knapp, editors, *Software Engineering and Formal Methods - 22nd International Conference, SEFM 2024, Aveiro, Portugal, November 6-8, 2024, Proceedings*, volume 15280 of *Lecture Notes in Computer Science*, pages 162–181. Springer, 2024.

Asad Ahmed and Diego Marmsoler.
Isabelle/Solidity: A Tool for the Verification of Solidity Smart Contracts (Tool Paper).
In Diego Marmsoler and Meng Xu, editors, *6th International Workshop on Formal Methods for Blockchains, FMBC 2025, May 4, 2025, Hamilton, Canada*, volume 129 of *OASIcs*, pages 12:1–12:9. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025.




# References II


Diego Marmsoler and Billy Thornton.
Deductive Verification of Solidity Smart Contracts with SSCalc.
*Sci. Comput. Program.*, 243:103267, 2025.

Diego Marmsoler and Achim D. Brucker.
Isabelle/Solidity: A deep embedding of Solidity in Isabelle/HOL.
*Formal Aspects Comput.*, 37(2):15:1–15:56, 2025.




# A Simple Banking Contract in Solidity

Solidity

```
1  contract Attacker {
2    uint8 iterations;
3    address bank;
4    constructor (address _ba, uint8 _it) payable public {
5      bank = _ba;
6      iterations = _it;
7    }
8    function deposit() public {
9      bank.call.value(1 ether).gas(20764) (bytes4(sha3("deposit()")));
10   }
11   function withdraw() public {
12     bank.call(bytes4(sha3("withdraw()")));
13   }
14   function () payable public {
15     if (iterations > 0) {
16       iterations --;
17       bank.call(bytes4(sha3("withdraw()")));
18     }
19   }
20 }
```