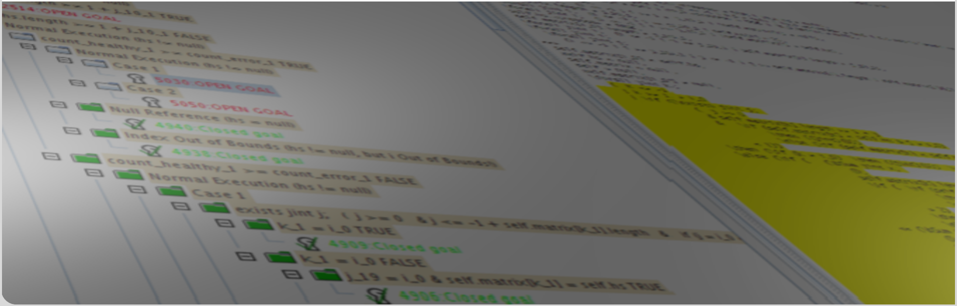
Karlsruhe Institute of Technology

Verification of Smart Contracts - Concrete, Intermediate, Abstract

Jonas Schiffl | July 13, 2022

KARLSRUHE INSTITUTE OF TECHNOLOGY



Correctness of Smart Contracts - Example Properties

```
contract Auction {

  address owner;
  mapping(address => uint) balances;
  address highestBidder;
  uint highestBid;

  function bid() public payable {}
  function withdraw() public {}
  function close() public {}
}
```

Correctness of Smart Contracts - Example Properties

```
contract Auction {

  address owner;
  mapping(address => uint) balances;
  address highestBidder;
  uint highestBid;

  function bid() public payable {}
  function withdraw() public {}
  function close() public {}
}
```

Desired property: “When the auction closes, the owner gets payed.”

Correctness of Smart Contracts - Example Properties

```
contract Auction {

  address owner;
  mapping(address => uint) balances;
  address highestBidder;
  uint highestBid;

  function bid() public payable {}
  function withdraw() public {}
  //@ ensures balance == \old(balance) - highestBid
  //@ ensures owner.balance ==
  //            \old(owner.balance) + highestBid
  function close() public {}
}
```

Desired property: “When the auction closes, the owner gets payed.”

Correctness of Smart Contracts - Example Properties

```
contract Auction {

  address owner;
  mapping(address => uint) balances;
  address highestBidder;
  uint highestBid;

  function bid() public payable {}
  function withdraw() public {}
  function close() public {}
}
```

Desired property: “The highest bidder cannot withdraw their bid.”

Correctness of Smart Contracts - Example Properties

```
contract Auction {

  address owner;
  mapping(address => uint) balances;
  address highestBidder;
  uint highestBid;

  function bid() public payable {}

  //@ modifies balances[msg.sender]
  //               if msg.sender != highestBidder
  function withdraw() public {}
  function close() public {}
}
```

Desired property: “The highest bidder cannot withdraw their bid.”

Correctness of Smart Contracts - Example Properties

```
contract Auction {
  ...
  address highestBidder;
  bool closed;

  function bid() public payable {}
  function withdraw() public {}
  function close() public {}
}
```

Desired property: “After bidding in the auction, I will either get my money back or win the auction.”

Correctness of Smart Contracts - Example Properties

```
contract Auction {
  ...
  address highestBidder;
  bool closed;

  function bid() public payable {}
  function withdraw() public {}
  function close() public {}
}
```

Desired property: “After bidding in the auction, I will either get my money back or win the auction.”

$\exists(Bidder, amt) : bid(Bidder, amt)$
$\rightarrow \Diamond(closed \wedge highestBidder = Bidder) \vee enabled(withdraw, Bidder, amt)$

Research questions

How to model smart contracts in a way that...

- is somewhat intuitive?
- (also) allows liveness specification and verification?

Research questions

How to model smart contracts in a way that...

- is somewhat intuitive?
- (also) allows liveness specification and verification?

Idea: State transition systems (e.g. TLA+)

Research questions

How to model smart contracts in a way that...

- is somewhat intuitive?
- (also) allows liveness specification and verification?

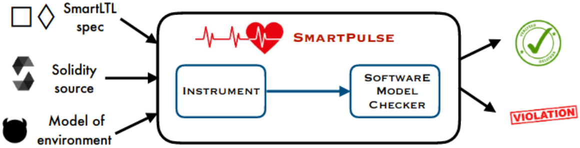
Idea: State transition systems (e.g. TLA+)

... more questions!

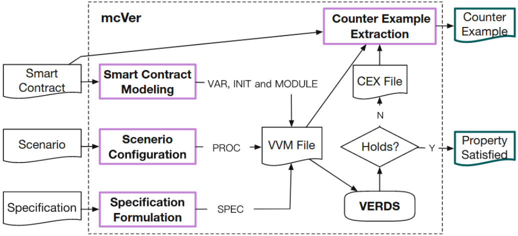
- How to translate from model to code / code to model?
- How to ensure correspondence (in a modular fashion)?

Existing Approaches for Liveness Analysis

- SmartPulse (2021)

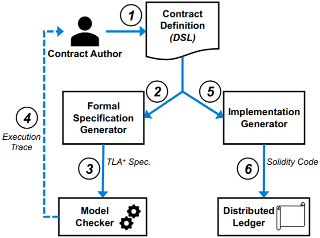


- mcVer (July 2022)

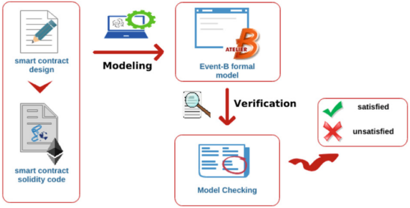


Existing Approaches for STS Modeling

- Quartz (2020)



- Event-B based approach (2020)



This project

Work both ways?

- Source Code -> State transition system
- State transition system -> Source Code

This project

Work both ways?

- Source Code -> State transition system
- State transition system -> Source Code

Idea: An intermediate representation of smart contracts

This project

Work both ways?

- Source Code -> State transition system
- State transition system -> Source Code

Idea: An intermediate representation of smart contracts

... more questions!

- How to translate from model to code / code to model?
- How to ensure correspondence (in a modular fashion)?

This project

Karlsruhe Institute of Technology

Work both ways?

- Source Code -> State transition system
- State transition system -> Source Code

Idea: An intermediate representation of smart contracts

... more questions!

- How to translate from model to code / code to model?
- How to ensure correspondence (in a modular fashion)?

Idea: Use functional verification as base for abstraction!

Functional Verification Tools

Assumption: Invariants and function contracts (postconditions + frame conditions) can be reliably proven

```
contract Auction {
  uint highestBid;
  address highestBidder;
  //@ invariant highestBid >= 0

  //@ ensures msg.value > highestBid
  //            => bids[msg.sender] == msg.value
  //@ ensures msg.value > highestBid
  //            => highestBidder == msg.sender
  //@ modifies balances[msg.sender]
  function bid() public payable {...}
}
```

Tools: Solc-verify, Celestial, VeriSolid

What is needed - IR

Idea: An intermediate representation of smart contracts

What is needed - IR

Idea: An intermediate representation of smart contracts

- Definition of IR
  - State Variables
  - Functions (names + parameters)
  - Invariants
  - Function contracts (postcondition + frame)
- Translation from IR to STS (e.g. TLA+)
- Translation from STS to IR

What is needed - SML

Idea: Use functional verification as base for abstraction!

What is needed - SML

Karlsruhe Institute of Technology

Idea: Use functional verification as base for abstraction!

A functional specification language for Solidity

- Specification language ("Solidity Modeling Language" (SML))
- Translation from SML to/from tool-specific languages
- Translation from SML-annotated source to abstract representation

Application I - proving properties about verified Source Code

- Source code
-> Annotations (tool-specific)
-> Verification

```
uint i = 0;
//@ ensures i == \old(i) + 1
function inc() public {...}
```

Application I - proving properties about verified Source Code

- Source code

-> Annotations (tool-specific)

-> Verification

-> Translation to IR

```
uint i = 0;
//@ ensures i == \old(i) + 1
function inc() public {...}
```

```
state: uint i
functions: inc (pre={}, post={i=\old(i)+1})
```

Application I - proving properties about verified Source Code

- Source code
- -> Annotations (tool-specific)
- -> Verification
- -> Translation to IR

```
uint i = 0;
//@ ensures i == \old(i) + 1
function inc() public {...}
```

```
state: uint i
functions: inc (pre={}, post={i=\old(i)+1})
```

Desired property: $\Diamond i == 42$

Application I - proving properties about verified Source Code

- Source code

-> Annotations (tool-specific)

-> Verification

-> Translation to IR

-> Translation to model

-> Proof of liveness property

```
uint i = 0;
//@ ensures i == \old(i) + 1
function inc() public {...}
```

```
state: uint i
functions: inc (pre={}, post={i=\old(i)+1})
```

Desired property: $\Diamond i == 42$

```
Integer i;
Init == i = 0;
inc == i' = i + 1
Spec == Init /\ []inc /\ WF(inc)
Desired == <> i = 42
```

Disclaimer

Liveness: Caution!

- Overapproximation of methods leads to wrong reachability results
- Further steps are required for dependable liveness results
- See Mattias' talk for some thoughts on this

Application II - Generating Code for Correctness

- STS model

-> Proof of desired property

-> Translation to IR

-> Translation to annotated code skeleton (functions not implemented)

-> Implementation

-> Proof of (functional) correctness

Benefits

Robustness

- Solidity version changes: Easier to adapt translation than to adapt entire tool (Caution: Assumption about deductive verification tools needs to hold!)
- Leverage more powerful verification tools (e.g. model checkers) without having to build new tool

Benefits

Robustness

- Solidity version changes: Easier to adapt translation than to adapt entire tool (Caution: Assumption about deductive verification tools needs to hold!)
- Leverage more powerful verification tools (e.g. model checkers) without having to build new tool

Flexibility

- (As shown) Verified Source -> Properties on Model
- (As shown) Properties on Model -> Correct Contracts
- But intermediate representation could also be used for
  - Relational Verification (on execution traces)
  - Access Control Modeling (IEEEDapps '22)
  - ...