Isabelle/Solidity for Smart Contracts

Jane Open Access
Dummy University Computing Laboratory, [optional: Address], Country
My second affiliation, Country

Joan R. Public[1]
Department of Informatics, Dummy College, [optional: Address], Country

Abstract

2012 ACM Subject Classification Replace ccsdesc macro with valid one

Keywords and phrases Program Verification, Smart Contracts, Isabelle, Solidity

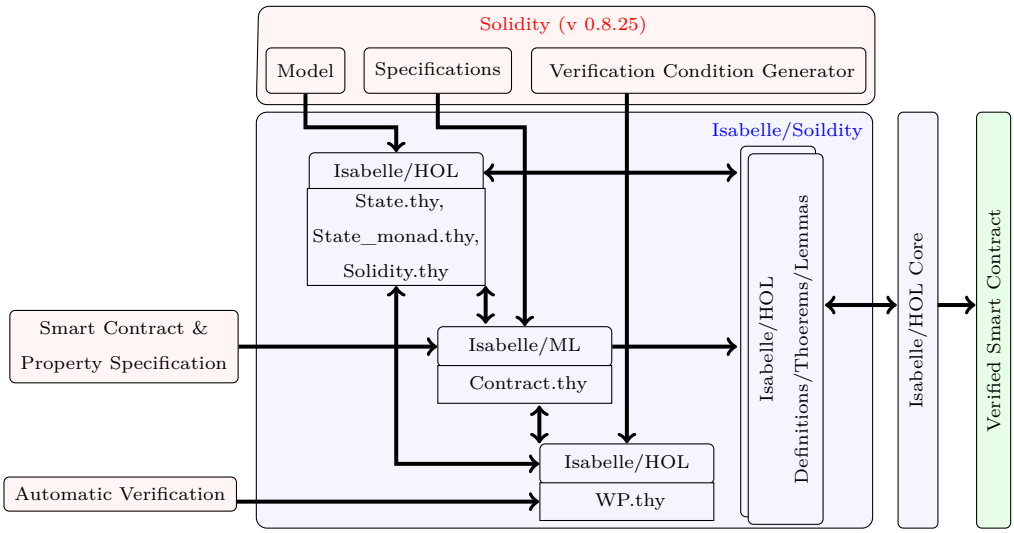
Digital Object Identifier 10.4230/OASIcs.CVIT.2016.23

Funding *Jane Open Access*: (Optional) author-specific funding acknowledgements
Joan R. Public: [funding]

Acknowledgements I want to thank ...

1 Introduction

2 Overview



[1] Optional footnote, e.g. to mark corresponding author

3 Case Study

Listing 1: Solidity source code for the Casino

```
1  contract Casino {
2    enum Coin { HEADS, TAILS } ;
3    enum State { IDLE, GAME_AVAILABLE, BET_PLACED }
4    State private state;
5    address public operator, player;
6    uint public pot;
7    bytes32 public hashedNumber;
8    uint public bet;
9    Coin guess;
10
11   function createGame(bytes32 hashNum)
12   public byOperator, inState(IDLE) {
13   hashedNumber = hashNum;
14   state = GAME_AVAILABLE;
15   }
16
17   function placeBet(Coin _guess) public payable inState(GAME_AVAILABLE) {
18   require (msg.sender != operator);
19   require (msg.value <= pot);
20   state = BET_PLACED;
21   player = msg.sender;
22   bet = msg.value;
23   guess = _guess;
24 }
25
26   function decideBet(uint secretNumber)
27   public byOperator, inState(BET_PLACED) {
28     require (hashedNumber == keccak256(secretNumber));
29     Coin secret = (secretNumber % 2 == 0)? HEADS : TAILS;
30     if (secret == guess) { pot = pot - bet; player.transfer(bet*2); bet =
           0;}
31     else {
32     pot = pot + bet; bet = 0;
33       }
34   state = IDLE;}
35   function addToPot() public payable byOperator { pot = pot + msg.value;}
36
37   function removeFromPot(uint amount) public byOperator, noActiveBet {
         operator.transfer(amount); pot = pot - amount;}
38   }
```

Casino (Listing 1) implements a bet game based on the flip-coin (Line 2) using Solidity syntex. This game has three explicit state: `IDLE`, `GAME_AVAILABLE`, `BET_PLACED` (Line 3). An operator may create a new game by calling `creatGame` function (Line 11-15). The operator provides a `hashNum` (Line 11) to ensure unbiased and verifiable bet (Line 26-30). The function, `creatGame`, uses two modifiers (`byOperator` and `inState(s)`) to implement only operator access and state-flow control, i.e., new game can only be created in `IDLE` state. The `creatGame` function changes the state to `BET_PLACED` which allows players to place bet on `HEADS` or `TAILS`, as `_guess`, by calling `betPlaced` function. The `betPlaced` function uses `require` to safe guard possible manipulation from operator by restricting its access (Line 18) and payout safety by capping the maximum bet amount with pot balance (Line 19) in the

game.

Once the game is in `BET_PLACED`, the operator may decide the bet (`decideBet`) by passing `secretNumber` (Line 26). The secret number is used to verify the bet against `hashNumber` (Line 28) to ensure fairness. Then, secret number is used to reveal `HEADS` (even) or `TAILS` (odd) (Line 29) which is further utilized to resolve the bet (Line 30-33). In case, player wins, then double amount of the original bet is transferred to the player's account (Line 30), otherwise, amount equivalent to original bet is added to the pot (Line 32).

Finally, in `IDLE` and `GAME_AVAILABLE` states, an operator may add or remove any amount of money from pot by invoking `addToPot` or `removeFromPot` function. However, in `BET_PLACED` state, an operator is allowed to only remove the money which is ensured by the modifier `noActiveBet`.

4 Specification

In this section, we present a Solidity equivalent specification of Casino smart contract in Isabelle/Solidity.

Storage Variables

5 Related Work

6 Conclusion