BlockChain CA-3 Report

Name: Janyavula Mohan Krishna
Reg no: 12113621
Section: K21CS
Course code: CSC403
Roll no:26

Submitted to: **Piyush Gururani**

Using a **formal verification tool** like Dafny, verify a simple Solidity function that calculates the balance of each user after multiple deposits. Write the code for a function that accepts a deposit and returns the updated balance. Then, specify the properties that need to hold true for this function (e.g., that the balance can only increase after a deposit). Explain how formal verification could help prove the correctness of this function.

Objective

The goal is to ensure the correctness of a Solidity function implementing a deposit operation in a smart contract using formal verification with Dafny. The Solidity contract must satisfy properties such as ensuring that balances only increase after a deposit and that the updated balance is correctly calculated. This process involves:

1.Writing a Dafny function to formally verify the deposit logic.
2.Implementing the Solidity function with runtime checks (require and assert).
3.Discussing how formal verification ensures correctness and supplements runtime checks.

1. Solidity Code Analysis

1.1 Solidity Code Overview

The provided Solidity code defines a simple bank contract with the following features:

Mapping to Store Balances: mapping(address => uint256) private balances keeps track of each user's balance.
Deposit Function: deposit(uint256 amount) allows users to deposit funds and updates their balances.
Get Balance Function: getBalance() retrieves the user's current balance.
Verification Function: verifyBalanceIncreases(uint256 initialBalance, uint256 depositAmount) checks that deposits always increase balances and calculates the updated balance.

1.2 Solidity Code

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract Bank {
   mapping(address => uint256) private balances;

   event Deposit(address indexed user, uint256 amount, uint256
newBalance);

   function deposit(uint256 amount) public returns (uint256) {
      require(amount > 0, "Deposit amount must be greater than
0");

      balances[msg.sender] += amount;

      emit Deposit(msg.sender, amount, balances[msg.sender]);

      return balances[msg.sender];
   }

   function getBalance() public view returns (uint256) {
      return balances[msg.sender];
   }

   function verifyBalanceIncreases(uint256 initialBalance, uint256
depositAmount)
      public
      pure
      returns (bool)
   {
      require(depositAmount > 0, "Deposit amount must be greater
than 0");
```

```
        uint256 newBalance = initialBalance + depositAmount;

        assert(newBalance >= initialBalance);
        assert(newBalance == initialBalance + depositAmount);

        return true;
    }
}
```

1.3 Properties in Solidity

Balance Increases After Deposit: The deposit function ensures that the balance only increases (balances[msg.sender] += amount).
Correctness of Calculation: The function uses assertions (assert) in verifyBalanceIncreases to ensure correctness.

2. Dafny Code Analysis:

2.1 Dafny Code Overview

The Dafny code is written to mathematically verify the correctness of the deposit operation. The verification involves:

1.Ensuring that the new balance is greater than or equal to the initial balance.
2.Proving that the new balance is exactly the sum of the initial balance and the deposit amount.

2.2 Dafny Code

```
function deposit(balance: nat, amount: nat): nat
   ensures deposit(balance, amount) >= balance
   ensures deposit(balance, amount) == balance + amount
{
   balance + amount
}

method VerifyDepositIncreasesBalance()
{
   var initialBalance: nat := 100;
   var depositAmount: nat := 50;

   var newBalance: nat := deposit(initialBalance, depositAmount);
```

```
    assert newBalance >= initialBalance;
    assert newBalance == initialBalance + depositAmount;
}
```

2.3 Properties in Dafny:

1.Balance Does Not Decrease: Using ensures deposit(balance, amount) >= balance.
2.Correct Calculation: Using ensures deposit(balance, amount) == balance + amount.
3.Mathematical Proof: Dafny verifies these properties for all possible values of balance and amount that satisfy the preconditions.

3.Key Differences Between Dafny and Solidity Verificationamount

Aspect	Dafny	Solidity
Verification Type	Formal verification using mathematical proofs.	Runtime verification using require and assert.
Scope of Verification	Verifies properties for all possible inputs.	Verifies behavior for specific inputs during execution.
Error Detection	Errors are detected at compile time.	Errors are detected during contract execution.

4. How Formal Verification Ensures Correctness

Formal verification with Dafny:

A.Proves All Possible Scenarios:
Unlike Solidity's runtime checks, Dafny ensures correctness for all values of balance and amount that meet the requirements.
Example: It guarantees that deposit(balance, amount) always satisfies the ensures clauses.

B.Prevents Logical Errors:

1.Logical errors (e.g., integer overflow) are ruled out since Dafny uses mathematical reasoning.
2.Solidity uses runtime checks, but they might not catch all edge cases unless explicitly written.

C.Mathematical Precision:
Dafny enforces precise mathematical rules, avoiding common pitfalls like rounding errors or unexpected behavior.

D.Improves Reliability:
Verified code gives confidence to developers and users, reducing the risk of bugs.

5. Conclusion

By combining Dafny's formal verification and Solidity's runtime checks, the deposit operation is thoroughly validated:

Dafny proves that the deposit logic is correct for all inputs, ensuring balances increase and are correctly calculated.
Solidity implements the same logic with runtime checks to enforce behavior during execution.
This approach enhances the reliability and security of smart contracts, a critical requirement in blockchain applications where financial operations are irreversible.

THANKYOU