A Simple Model of Smart Contracts in Agda

Fahad Alhabardi[1] and Anton Setzer[2]
[1,2]Swansea University, Blockchain Lab, Dept. of Computer Science, UK

Types 2023
Universitat Politècnica de València, València, Spain

June 12, 2023

Table of Contents

Ethereum

- **Ethereum** = A **second-generation Blockchain** technology [7].
- Launched by Vitalik Buterin [4] in 2013.
- Main difference to Bitcoin is in the use of smart contracts:
  - Ethereum [9]:
    - **Turing complete** language which includes loops;
    - allows **calls to other contracts**;
    - adds **cost of execution of instructions** (gas) to guarantee termination.
- Recently switch from **proof of work** to **proof of stake** [5], **solving the waste of energy problem**.

Smart Contracts

- **Smart contract** = **program** which is **automatically executed** when conditions in the blockchains are satisfied [8].
- Smart contracts are **immutable programs** [3].
- Smart contracts in the cryptocurrency Ethereum are usually written in the high-level language **Solidity** [6] which compiles into the low-level **Ethereum Virtual Machine (EVM)** [4].
- **World State Machine** with essentially immutable history.
- **Example applications:**
  - Tracing of goods (using we have an immutable database),
  - Electronic voting,
  - NFT (ownership of electronic items),
  - Investment fonds (DAO).
- Because of **high monetary impact**, **immutability**, and **shortness of programs**, **prime candidate for verification**.

Smart Contracts

- Blockchain is roughly speaking a **data base** which **determines for each address its current state** (amount of money, other data).
- In Ethereum **smart contracts = objects deployed to addresses**, with methods which can be called by
  - ordinary (externally owned) accounts,
  - other smart contracts.
- Toy example (Solidity):

```
pragma solidity ^0.8.17;

contract testLedger {
     function f (int n) public pure returns (int){
       return g(n);
     }

    function g (int n) public pure returns (int){
        if (n > 0) {return f(n - 1);}
        else       {return 0;}
    }
}
```

Contribution

- **Previous work:** Verification of Bitcoin smart contracts using **weakest preconditions** of Hoare logic [2, 1] in Agda.
- Goal of this and follow up papers is adaption to Solidity style smart contracts.
- **First Step Here:** develop model of Solidity-style smart contracts in Agda.
- More complex, because of use of **objects**.
- We cover execution of contracts including **calling of other contracts** and contracts having **multiple functions (methods)**.

Messages

- EVM allows calling functions with **serialised parameters**.
- Parameters represent elements of **arrays**, **maps**, **enumerations**, **integers**, etc.
- In our model, we abstract from this encoding by defining a **message data type**:

```
data Msg : Set where nat  : (n : ℕ) → Msg
                     list : (l : List Msg) → Msg
```

- Arrays are represented as lists of messages.
- Maps are represented as lists of pairs (represented as lists) of messages.

Programs (SmartContractExec)

```
data SmartContractExec : Set where
  return : Msg → SmartContractExec
  call   : SmartContractExecStep → SmartContractExec
  error  : ErrorMsg → SmartContractExec
```

- SmartContractExec consists of three constructors:

return	=	terminates execution and return its argument;
call	=	calls SmartContractExecStep
		then continues as defined by its continuation argument
error	=	raises an error.

SmartContractExecStep

```
record SmartContractExecStep : Set where
  coinductive
  field calledAddress  : Address
        calledFunction : FunctionName
        calledMsg      : Msg
        cont           : Msg → SmartContractExec
```

calledAddress	=	address of contract being called;
calledFunction	=	function name called;
calledMsg	=	argument of the function (a message);
cont	=	continuation, depends on the result of executed function.

- SmartContractExec and SmartContractExecStep are defined coinductively, so loops and even non-terminating programs are allowed.

Ledger and ExecutionStack

- A ledger determines for any address function name and msg argument the smart contract function to be executed:

  $\mathsf{Ledger} = \mathsf{Address} \to \mathsf{FunctionName} \to \mathsf{Msg} \to \mathsf{SmartContractExec}$

- $\mathsf{ExecutionStack}$ = stack of continuations
  - continuation are executed once the result of the execution above it has finished giving an element of Msg.

  $\mathsf{ExecutionStack} = \mathsf{List}\ (\mathsf{Msg} \to \mathsf{SmartContractExec})$

StateExecFun

- The state of execution is given by

```
record StateExecFun : Set where
  constructor stateEF
  field executionStack : ExecutionStack
        nextstep       : SmartContractExec
```

i.e. having two fields:

- executionStack is the current execution stack;
- nextstep is the current code to be executed.

stepEF and stepEFntimes

- stepEF, is the one-step execution of a smart contract.
- stepEFntimes, which iterates it n times, corresponding to execution with a simple form of gas limit.

$$\begin{array}{lcl} \text{stepEF} & : & \text{Ledger} \to \text{StateExecFun} \to \text{StateExecFun} \\ \text{stepEFntimes} & : & \text{Ledger} \to \text{StateExecFun} \to \mathbb{N} \to \text{StateExecFun} \end{array}$$

Evaluation

$$\text{evaluateNonTerminating} : \text{Ledger} \to \text{Address} \to \text{FunctionName} \\ \to \text{Msg} \to \text{NatOrError}$$

We can define as well a terminating version with additional parameter

$$\mathit{gas} : \mathbb{N}$$

which restricts evaluation to gas many steps.

Example of simple Solidity-style of smart contract in Agda

- Example recursively decrementing by 1 until 0:

```
testLedger : Ledger
testLedger 0 "f" (nat n)
  = call (smartContractExecStep 0 "g" (nat n) return)
testLedger 0 "g" (nat (suc n))
  = call (smartContractExecStep 0 "f" (nat n) return)
testLedger 0 "g" (nat 0)
  = return (nat 0)
testLedger ow ow' ow''
  = error (strErr " Error undefined")
```

- evaluateNonTerminating testLedger 0 "f" (nat 5)
  evaluates to nat 0

Example in Solidity language

- Corresponding Solidity code:

```
pragma solidity ^0.8.17;

contract testLedger {
     function f (int n) public pure returns (int){
       return g(n);
     }

    function g (int n) public pure returns (int){
        if (n > 0) {return f(n - 1);}
        else       {return 0;}
    }
}
```

Example Run in Solidity

When applying "f" to 7 and "g" to 4 we obtain the following results:

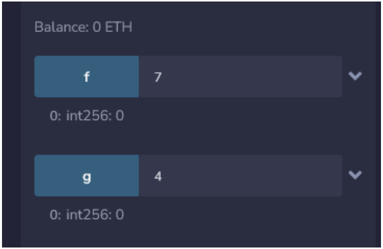

Figure: Result using Solidity language

Conclusion

- We developed a **simple model** of **Solidity-style smart contracts** in Agda.
- Dealt with **execution** and **calling of other contracts**.
- Not yet support of **gas cost**, **amount of money**, **transfer of money**, **state**.
- **Work in progress:**
  - Extend the simple model by the not yet supported items.
  - Develop an **interactive program in Agda** which allows to execute calls of functions in contracts with a corresponding ledger.
- **Future work:**
  - Adapt the verification of bitcoin using weakest preconditions [2] to verifying contracts in this model.

Thank you for listening.

[1] Fahad Alhabardi, Bogdan Lazar, and Anton Setzer.
Verifying correctness of smart contracts with conditionals.
In *2022 IEEE 1st Global Emerging Technology Blockchain Forum: Blockchain & Beyond (iGETblockchain)*, pages 1–6, 2022.
doi: `https://doi.org/10.1109/iGETblockchain56591.2022.10087054`.

[2] Fahad F. Alhabardi, Arnold Beckmann, Bogdan Lazar, and Anton Setzer.
Verification of Bitcoin Script in Agda Using Weakest Preconditions for Access Control.
In *27th International Conference on Types for Proofs and Programs (TYPES 2021)*, volume 239 of *LIPIcs*, pages 1:1–1:25, Dagstuhl, Germany, 2022. Leibniz-Zentrum für Informatik.
doi: `https://doi.org/10.4230/LIPIcs.TYPES.2021.1`.

[3] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli.
A Survey of Attacks on Ethereum Smart Contracts (SoK).
In *Principles of Security and Trust*, pages 164–186, Berlin, Heidelberg, 2017. Springer.
doi: https://doi.org/10.1007/978-3-662-54455-6_8.

[4] Vitalik Buterin.
Ethereum: A next-generation smart contract and decentralized application platform, 2014.
Availabe from https://ethereum.org/en/whitepaper.

[5] Ethereum community.
Proof-of-Stake (POS), Retrieved 03 May 2023.
Availabe from https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/.

[6] Ethereum Community.
Solidity documentation, Retrieved 15 April 2023.
Availabe from https://docs.soliditylang.org/en/v0.8.16/.

[7] Han-Min Kim, Gee-Woo Bock, and Gunwoong Lee.
Predicting ethereum prices with machine learning based on blockchain information.
Expert Systems with Applications, 184:115480, 2021.
doi:https://doi.org/10.1016/j.eswa.2021.115480.

[8] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor.
Making Smart Contracts Smarter.
In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 254–269, New York, NY, USA, 2016. Association for Computing Machinery.
doi:http://dx.doi.org/10.1145/2976749.2978309.

[9] Dejan Vujičić, Dijana Jagodić, and Siniša Ranđić.
Blockchain technology, Bitcoin, and Ethereum: A brief overview.
In *2018 17th International Symposium INFOTEH-JAHORINA (INFOTEH)*, pages 1–6, 2018.
doi:http://dx.doi.org/10.1109/INFOTEH.2018.8345547.