Termination-checked Solidity-style smart contracts in Agda in the presence of Turing completeness

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

Types 2024
IT University of Copenhagen, Copenhagen, Denmark

June 10, 2024

Termination-checked Solidity-style smart contracts in Agda in the presence of Turing completeness

- J.w.w. Fahad Alhabardi
  - 552 p. PhD thesis expected soon to become public
- **Smart contract** = **program** placed on the blockchain which is **automatically executed** when conditions in the blockchains are satisfied [10].
- Topic of this talk are
  - **Smart Contracts** of the cryptocurrency **Ethereum**
  - written in the **object-oriented** language **Solidity**
  - which is **Turing complete**
  - and how to deal with the **termination** problem.
- Based on the **model of objects** using coalgebras in Agda [1].

Table of Contents

1. Background
2. Model of Smart Contracts in Agda
3. Execution of Smart Contracts
4. Conclusion

Ethereum

- **Ethereum** = A **second-generation Blockchain** [9].
- Main difference to Bitcoin is in the use of smart contracts:
  - Bitcoin Smart Contracts (Bitcoin Script)
    - no loops, which guarantees termination,
    - functionality as smart contract language very limited, many say not enough to call it Smart Contract.
  - Ethereum [11]:
    - **Turing complete** language which includes **loops**;
    - allows **calls to other contracts**;
    - problem that validators need to execute smart contracts without knowing whether they terminate;
    - Ethereum solves this by adding a **cost of execution of instructions** (gas) to guarantee termination.
- Recent switch from **proof of work** to **proof of stake** [7], **solving the waste of energy problem** for Ethereum.

Smart Contracts

- Smart contracts are **immutable programs** [5].
- Smart contracts in the cryptocurrency Ethereum are usually written in the high-level language **Solidity** [8] which compiles into the low-level **Ethereum Virtual Machine (EVM)** [6].
- Ethereum is a **World State Machine** with essentially immutable history.
- **Example applications:**
  - **Non-monetary** applications
    - Tracing of goods (e.g. tracing organic apples in super market through intermediate vendors to farmers)
    - Electronic voting,
  - **Monetary** applications
    - investment fonds (DAO).
- Because of **immutability**, **high monetary impact**, 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, smart contracts).
- In Ethereum **smart contracts = objects deployed to addresses**, with methods (called **functions**), which can be called by
  - non-smart-contract accounts (called externally-owned)
  - other smart contracts.

Toy example (Solidity)

```
contract Test1 {
      Test2 test2;
      // code for setting test2 omitted.

      function f (int n) public view returns (int){
        return test2.g(n);
      }
}

contract Test2{
      Test1 test1;
      // code for setting test1 omitted.

      function g (int n) public view returns (int){
          if (n > 0) {return test1.f(n - 1);}
          else       {return 0;}
     }
}
```

Contribution

- **Previous work:**
  - Verification of Bitcoin smart contracts using **weakest preconditions** of Hoare logic for **access control** [4, 2] in **Agda**.
  - Introduction of a **simple model** [3] of **Solidity-style smart contracts** which doesn't involve gas.
- **This Talk:**
  - Addition of **gas cost** to solve the **termination problem** (modeling implementation in Ethereum).
  - Resulting code **termination checks** in Agda.
- **Goal of the Project:**
  - Verify Solidity style smart contracts using **weakest precondition semantics**.

Messages

- In Solidity functions have arguments and result types originating from a rich type structure.
- The EVM is untyped, and uses **serialised** arguments and return values.
- 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
```

- This data type allows to represent elements of data types such as **lists** (called arrays), finite **maps**, **enumerations**, **integers**.

Programs (SmartContractExec)

The body of a function is represented as an element of SmartContractExec:

```
data SmartContractExec (A : Set) : Set where
  return : A → (gascost : ℕ) → SmartContractExec A
  error  : ErrorMsg → DebugInfo
           → SmartContractExec A
  exec   : (c : CCommands)
           → (cont : CResponse c → SmartContractExec A)
           → (gascostCont : CResponse c → ℕ)
           → SmartContractExec A
```

Contract

```
record Contract : Set where
  field
    amount : Amount
    fun : FunctionName → Msg → SmartContractExec Msg
    viewFunction : FunctionName → Msg → MsgOrError
    viewFunctionCost : FunctionName → Msg → ℕ
```

Which includes the fields:

- The balance of a contract (amount),
- its functions (fun);
- its view functions (viewFunction);
- the estimated gas cost for executing a view function (viewFunctionCost).

Normal functions can modify **view functions**.
We use **view functions** to represent **variables**.

Ledger

- The state of a ledger determines for any address function name and msg argument the smart contract function to be executed:

  $\text{Ledger} = \text{Address} \rightarrow \text{Contract}$

- Strictly speaking the Ledger should be called LedgerState, since it excludes the history of the ledger.
  - Execution of a smart contracts depends only on the current state of the ledger.

Intermediate State of Execution

```
record StateExecFun : Set where
field ledger            : Ledger
      executionStack : ExecutionStack
      initialAddr lastCallAddr calledAddr : Address
      nextstep          : SmartContractExec Msg
      gasLeft           : ℕ
      funNameevalState : FunctionName
      msgevalState : Msg
```

The state of the execution (StateExecFun) include the following fields:

- The ledger;
- the execution stack (executionStack);
- the initial address that initiated the current sequence (initialAddr);
- the last called made (lastCallAddr);

State Execution Function (Cont.)

- the address which is called (calledAddr);
- the current code to be executed (nextstep);
- the gas left (gasLeft);
- two extra fields that we use with debug information: funcNameexecStackE and msgexecStackEl.

Execution Stack Element

- The elements of ExecutionStack are given by

```
record ExecStackEl : Set where
  field lastCallAddress calledAddress : Address
        continuation : Msg → SmartContractExec Msg
        costCont                    : Msg → ℕ
        funcNameexecStackEl : FunctionName
        msgexecStackEl              : Msg
```

with the following fields:

- The address that made the last call (lastCallAddress);
- the address that was called (calledAddress);
- continuation which determines the next execution step to be executed depending on the message returned after the call to the function has been completed;
- funcNameexecStackE which is the last function called and the argument of the last function call (msgexecStackEl).

Evaluation of Function Calls

evaluateTerminatingfinal :
  Ledger
  → (*initialAddr lastCallAddr calledAddr*: Address)
  → (*funName* : FunctionName)
  → (*msg* : Msg)
  → (*gaslimit* : $\mathbb{N}$)
  → Ledger × MsgOrErrorWithGas

- Evaluates a call
  - from ***lastCallAddr***
  - to function ***funName*** applied to ***msg*** in contract ***calledAddr***
  - using gas limit ***gaslimit***
  - assuming the chain of calls was initiated from externally owned account ***initialAddr***
- evaluateTerminatingfinal termination checks in Agda.

Conclusion

- Solidity is **object oriented** and functions can call each other mutually.
- Use of **gas** to **guarantee termination**.
- Development of a **model** in Agda which allows to execute smart contract functions.
- **Gas cost** is added as **explicit parameters** to commands.
- Execution of smart contracts **termination checks** in **Agda**.
- Development of a **simulator**, which allows to simulate Ethereum including smart contracts interactively in Agda.
- Aim is to use **weakest precondition semantics** to verify Solidity smart contracts.

Thank you for listening.

[1] Andreas Abel, Stephan Adelsberger, and Anton Setzer.
Interactive programming in Agda – Objects and graphical user interfaces.
Journal of Functional Programming, 27:e8, Jan 2017.
https://doi.org/10.1017/S0956796816000319.
doi:10.1017/S0956796816000319.

[2] 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.

[3] Fahad Alhabardi and Anton Setzer.
A simple model of smart contracts in Agda, January 2023.
In Abstracts for Types 2023.
URL: https://types2023.webs.upv.es/TYPES2023.pdf.

[4] 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`.

[5] 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`.

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

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

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

[9] 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.

[10] 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.

[11] 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.