Verification Techniques for Smart Contracts in Agda

Fahad F. Alhabardi[1], Anton Setzer[2], Arnold Beckmann[3] ,and Bogdan Lazar[4]

Swansea University, Dept. of Computer Science, UK[1,2,3]
University of Bath, UK[4]

Types 2022 (28th International Conference on Types for Proofs and Programs), University of Nantes, France

June 22, 2022

Table of Contents

Smart Contracts

- What are smart contracts?
  Smart contracts are transactions that are defined through software and executed automatically when conditions in the blockchains are met.
- Smart contracts in the cryptocurrency Bitcoin are written in the language Script.

EVM vs Script

- Ethereum Virtual Machine and Bitcoin Script are similar.
- EVM [5]:
  - EVM extends and modifies Bitcoin Script, especially it
    - adds loops (jumps),
    - allows calls to other contracts,
    - adds cost of execution of instructions (gas) to guarantee termination.
- Bitcoin Script [5]:
  - without loops.
  - without possibility to calling other contracts.

Bitcoin Script Language

- The scripting language for Bitcoin is stack-based, and similar to Forth.
- The script in Bitcoin has a set of commands called Operation Codes such as OP_HASH160, OP_ ADD, OP_ EQUAL and OP_VERIFY.
- Several standards scripts [3] are used in Bitcoin such as the Pay-to-Public-Key-Hash (P2PKH) and Multi-signature (P2MS) scipts.

Contribution

- Operational semantics.
- Specification of correctness using weakest preconditions.
- Verification of example bitcoin scripts.
- An Agda library supporting it.

Bitcoin Script

Several opcodes have been introduced and formalised in Agda [1, 2] such as **OP_EQUAL**, **OP_IF**, **OP_ELSE**, and **OP_ENDIF**...

- Example of **OP_EQUAL**:
  $< 2 > < 3 >$ OP_ADD $< 5 >$ OP_EQUAL
  - The stack evolves as follows:
    []
    [$< 2 >$]
    [$< 2 >, < 3 >$]
    [$< 5 >$]
    [$< 5 > , < 5 >$]
    [1]

Cont.

- Example of **OP_IF**, **OP_ELSE**, and **OP_ENDIF**:

  OP_IF <Alice's PubKey> **OP_CHECKSIG**
  OP_ELSE <Bob's PubKey> **OP_CHECKSIG** **OP_ENDIF**

  - assume stack contains <sig> <1> (1 = True)
    The script will succeed if sig is a signature for the transaction using Alice's private key.
  - In case it conatins <sig> <0> signature need to be for Bob's Pub Key.

P2PKH

- All Bitcoin scripts consist of a locking script (provided by a person who sends coins) and an unlocking script (provided by someone who wants to access it).
- P2PKH has a locking script (scriptPubKey) and an unlocking script (scriptSig) [4].

```
scriptPubKey: OP_DUP OP_HASH160 <pubKeyHash> OP_EQUAL OP_VERIFY OP_CHECKSIG
scriptSig:    <sig> <pubKey>
```

  - success if running first scriptSig and then scriptPubKey succeeds with not false on top of stack.

Operational semantics

- The operational semantics of opcodes depends on Time × Msg × Stack . We define it in Agda as the record type StackState.
- All opcodes is given as InstructionBasic.
  - Opcodes can fail, for example if there are not enough elements on the stack as required by the operation.
- The operational semantics of p : InstructionBasic
  ⟦ p ⟧s : StackState → Maybe StackState
  - As an example, the semantics of ⟦ opEqual ⟧s:

⟦ opEqual ⟧s <t , msg, s > = nothing if s has height <= 1
⟦ opEqual ⟧s < t , msg , s0 :: s1 :: s > = just < t , msg , i :: s >
where i = 1 if s0 = s1 and i = 0 otherwise

Cont.

- Time: there are instructions for checking that a certain amount of time has passed, and time is used for checking against the current time.
- opCHECKLOCKTIMEVERIFY: allows to lock a resource until a certain amount of time has passed.
- Msg is the part of the transaction to be signed when a signature is required.

Hoare triple and weakest precondition

We define for $\Phi, \Psi \subseteq \mathrm{State}$ and p a Bitcoin Script the Hoare triple with weakest pre condition

For the unlocking script of P2PKH we show:

Therefore in order to unlock one needs to provide a script which computes the pubkey hashing to the pbkh and a corresponding signature.

Hoare triple and weakest precondition

We define for $\Phi, \Psi \subseteq \mathrm{State}$ and p a Bitcoin Script the Hoare triple with weakest pre condition

$$
\begin{aligned}
&\langle\ \Phi\ \rangle^{\leftrightarrow}\ p\ \langle\ \Psi\ \rangle :\Leftrightarrow \\
&\qquad (\forall s \in \mathrm{State1}.\Phi(s) \rightarrow \Psi([\![\, p\,]\!] s)) \\
&\qquad \wedge (\forall s \in \mathrm{State1}.\Psi([\![\, p\,]\!] s) \rightarrow \Phi(s))
\end{aligned}
$$

For the unlocking script of P2PKH we show:

Therefore in order to unlock one needs to provide a script which computes the pubkey hashing to the pbkh and a corresponding signature.

Hoare triple and weakest precondition

We define for $\Phi, \Psi \subseteq \mathrm{State}$ and p a Bitcoin Script the Hoare triple with weakest pre condition

$$\langle\ \Phi\ \rangle^{\leftrightarrow}\ p\ \langle\ \Psi\ \rangle :\Leftrightarrow$$
$$(\forall s \in \mathrm{State1}.\Phi(s) \to \Psi([\![\ p\ ]\!]s))$$
$$\wedge (\forall s \in \mathrm{State1}.\Psi([\![\ p\ ]\!]s) \to \Phi(s))$$

For the unlocking script of P2PKH we show:

$(\langle\Phi\rangle^{\leftrightarrow}\ \mathrm{scriptSig}\ \langle\mathrm{accept}\rangle)$
$\Longleftrightarrow$ the two top elements of the stack consist of a pubkey hashing to the pbkh and a corresponding signature.

Therefore in order to unlock one needs to provide a script which computes the pubkey hashing to the pbkh and a corresponding signature.

Our library

- Develop a library in Agda and prove correctness of smart contracts [1, 2].

We prove the theorem for the Hoare triple for prog1 ++ (prog2 ++ prog3) is given as follows:

```
theorem : < precondition >iff prog1 ++ (prog2 ++ prog3) < postcondition >
theorem = precondition       <><>⟨ prog1  ⟩⟨ proof1   ⟩
          intermediateCond1 <><>⟨ prog2  ⟩⟨ proof2   ⟩
          intermediateCond2 <=>⟨   proof3 ⟩
          intermediateCond3 <><>⟨ prog3  ⟩⟨ proof4   ⟩e postcondition ∎p
```

Proof of Correctness of the P2PKH script using the Step-by-Step approach

- P2PKH script:

$\text{scriptP2PKH}^b \;:\; (pbkh \;:\; \mathbb{N}) \rightarrow \text{BitcoinScriptBasic}$

$\text{scriptP2PKH}^b \; pbkh = \text{opDup} :: \text{opHash} :: (\text{opPush} \; pbkh) :: \text{opEqual} :: \text{opVerify} :: [\; \text{opCheckSig} \;]$

- Intermediate conditions accept_1, accept_2, etc
  - For example:
    - ★ $\text{accept}^s_1 \; m \; t \; st \Leftrightarrow \exists \; pbk, sig, st'.st \equiv pbk :: sig :: st' \wedge \text{IsSigned} \; m \; sig \; pbk$
    - ★ $\text{accept}^s_2 \; m \; t \; st \Leftrightarrow \exists \; x, pbk, sig, st'.st \equiv x :: pbk :: sig :: st' \wedge x > 0 \wedge \text{IsSigned} \; m \; sig \; pbk$
- Proofs correct-1,correct-2, etc

$\text{correct-1} : < \text{accept}_1 > \text{iff}([\; \text{opCheckSig} \;]) < \text{acceptState} >$

$\text{correct-2} : < \text{accept}_2 > \text{iff}([\; \text{opVerify} \;]) < \text{accept}_1 >$

Cont.

- Weakest precondition

```
wPreCondP2PKHˢ : (pbkh : ℕ) → StackPredicate
wPreCondP2PKHˢ pbkh time m []                    = ⊥
wPreCondP2PKHˢ pbkh time m (x :: [])             = ⊥
wPreCondP2PKHˢ pbkh time m ( pubKey :: sig :: st) =
               (hashFun pubKey ≡ pbkh ) ∧ IsSigned m sig pubKey
```

- Prove the weakest precondition for the P2PKH script as follows

```
theoremP2PKH : (pbkh : ℕ) → < wPreCondP2PKH pbkh >iff scriptP2PKHᵇ pbkh < acceptState >
theoremP2PKH pbkh = wPreCondP2PKH pbkh <><>⟨ [ opDup ] ⟩⟨ correct-6 pbkh ⟩
                    accept₅ pbkh <><>⟨ [ opHash ]      ⟩⟨ correct-5 pbkh ⟩
                    accept₄ pbkh <><>⟨ [ opPush pbkh ] ⟩⟨ correct-4 pbkh ⟩
                    accept₃      <><>⟨ [ opEqual ]     ⟩⟨ correct-3 ⟩
                    accept₂      <><>⟨ [ opVerify ]    ⟩⟨ correct-2 ⟩
                    accept₁      <><>⟨ [ opCheckSig ]  ⟩⟨ correct-1 ⟩e acceptState ∎p
```

Conclusion

- Differences between precondition and weakest precondition.
- Implemented theorems for verifying Bitcoin script using conditions.
- Our goal is to develop our approach into a framework for developing smart contracts that are correct by construction.
- Applied our approaches to P2PKH and P2MS.

Thank you for listening.

Fahad F. Alhabardi, Arnold Beckmann, Bogdan Lazar, and Anton Setzer.
Verification of Bitcoin's Smart Contracts in Agda using Weakest Preconditions for Access Control, 2022.
arXiv:arXiv:2203.03054,
doi:https://doi.org/10.48550/arXiv.2203.03054.

Fahad F. Alhabardi, Arnold Beckmann, Bogdan Lazar, and Anton Setzer.
Verification of Bitcoin's Smart Contracts in Agda using Weakest Preconditions for Access Control, 2022.
To appear in proceedings of Types 2021, LIPIcs, 2022. Preprint see arXiv:2203.03054 where
https://doi.org/10.48550/arXiv.2203.03054.

Stefano Bistarelli, Ivan Mercanti, and Francesco Santini.
An Analysis of Non-standard Bitcoin Transactions.
In *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, pages 93–96, 2018.
doi:http://dx.doi.org/10.1109/CVCBT.2018.00016.

Bitcoin Community.
Welcome to the Bitcoin Wiki.
Availabe from https://en.bitcoin.it/wiki/Bitcoin, 2010.

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.