Modelling Bitcoins in Agda

Anton Setzer

Dept. of Computer Science, Swansea University, Swansea, UK
a.g.setzer@swan.ac.uk

Abstract

We present a model of a block chain in Agda. We deal with Cryptographic operations and their correctness by postulating corresponding operations and their correctness. We determine correctness of blockchain transactions and show how to translate the blockchain back into a traditional ledger of a bank.

Since its introduction in November 2008, the market capitalisation of bitcoins has risen to over 18 Billion US-$. Other bitcoins such as Ethereum are following its lead. Cryptocurrencies have been proposed for introducing smart contracts. In its simplest form the buyer reserves money for the seller on the blockchain, and the seller only receives it once the seller has signed on time that she has received the goods. Bitcoins can be considered as the true cloud: whereas in normal cloud applications, data is stored on one server, and therefore everything relies on that service, Cryptocurrencies allow to store data on a peer-to-peer network. The block chain can then be used to certify which data is genuine, and determine the order and times when data was added.

In this project we use Agda as a modelling language for modelling the block chain. The goal is to obtain a deeper understanding of how the block chain operates and to prove correctness of certain aspects of the block chain. This project is the result of a series of third year and MSc projects supervised by the author at Swansea University. We follow the brown-bag talk by Warner [1], in which he shows how to obtain the blockchain starting from simple ledger.

In order to avoid having to introduce and verify cryptographic functions in Agda, we axiomatise those functions and their properties using Agda's postulates:

```
postulate Message : Set
postulate PublicKey : Set
postulate checkKey : (m : Message) (p : PublicKey) → Bool
postulate Names : Set
postulate messageToNat : (m : Message) → ℕ
postulate nameToPublicKey : (n : Names) → PublicKey
```

A message is here supposed to be a message with a signature and containing a value, namely the amount of bitcoins being represented in this message. checkKey checks whether a message has been signed by the private key corresponding to the public key of the name, and messageToNat determines the number contained in a message.

A bitcoin transaction consists of sequence of messages, and public keys, together with a proof that the messages have been signed by the private keys of the public keys:

```
data Input : Set where
  input : (message : Message) (publicKey : PublicKey)
          (cor : IsTrue (checkKey message publicKey)) → Input
```

similarly one can define outputs of a transaction. A transaction is now given by a list of inputs and a list of outputs:

```
data Transaction : Set where
  transaction : (input : List Input)(output : List Output) → Transaction
```

Time and amount of bitcoins are defined as natural numbers and the ledger is a function which assigns for every time and name the amount amount of bitcoins attributed to that person:

```
Time = ℕ
Amount = ℕ
Ledger = (t : Time)(n : Names) → Amount
```

We can now express the correctness of a transaction w.r.t. the state of the ledger before it is executed:

```
correctSingleTransaction : (oldLedger : Names → Amount)(trans : Transaction) → Set
correctSingleTransaction oldLedger (transaction inputlist outputlist)
  = IsTrue (checkKeysInInput inputlist)
    ∧ ((name : Names) →
                IsTrue (oldLedger name ≥ sumOfInputs inputlist (nameToPublicKey name)))
    ∧ IsTrue (sumOfInputsTotal inputlist ≥ sumOfOutputsTotal outputlist )
    ∧ inputPublicKeysAreProper inputlist
    ∧ outputPublicKeysAreProper outputlist
```

We can compute now the ledger after one transaction:

```
updateSingleTransaction : (oldLedger : Names → Amount)(trans : Transaction)
                          (n : Names) → Amount
updateSingleTransaction oldLedger (transaction inputlist outputlist) n =
  oldLedger n - sumOfInputs inputlist (nameToPublicKey n)
              + sumOfOutputs outputlist (nameToPublicKey n)
```

and can compute from this the complete ledger from a sequence of transactions:

```
transactionsToLedger : (initialLedger : Names → Amount)(trans : Time → Transaction)
                       → Ledger
```

Modifications are needed in order to deal with mining and fees. We are currently working on extending this model to adding smart contracts. Modelling simple smart contracts is straightforward, the challenge is to introduce a language for more generalised smart contracts.

References

[1] B. Warner. Bitcoin: A technical introdution. Available from http://www.lothar.com/presentations/bitcoin-brownbag/, July 2011.