Modelling Bitcoins in Agda

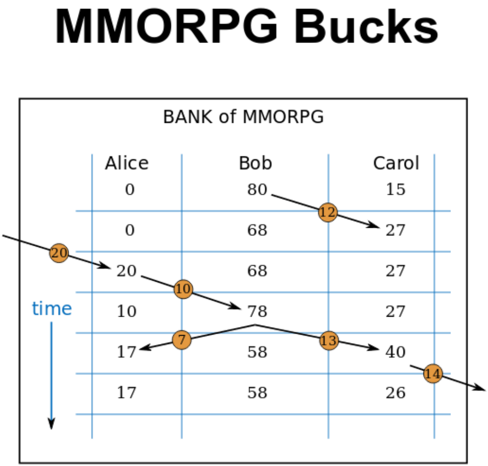
Anton Setzer
Swansea University, Swansea UK
Types 2017, Budapest, Hungary

1 June 2017

Introduction to Bitcoins based on Talk by Warner

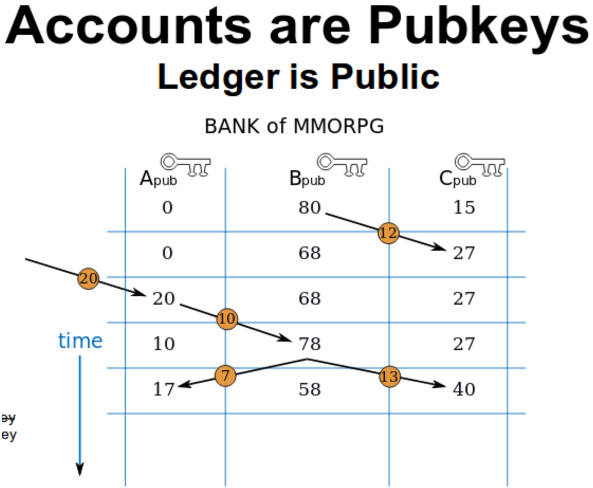
- Warner gave an excellent talk about bitcoins [1].
- He explained how one can obtain bitcoins starting from a simple model of a bank
- We will in the following show the keysteps of his talk.
- The screenshots are taken from his presentation.

Model of Bank



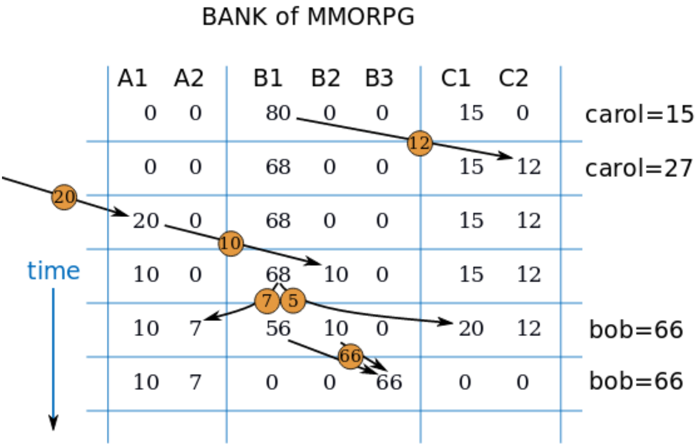
Source: [1]

Replace Names By Public Keys



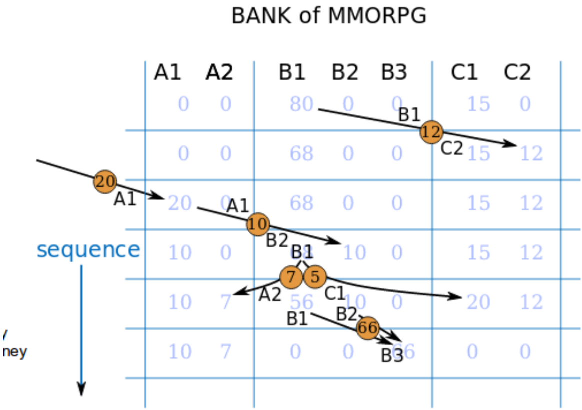
Source: [1]

Replace Single Public Keys by Multiple Ones



Source: [1]

Ledger Can be Derived From Transactions

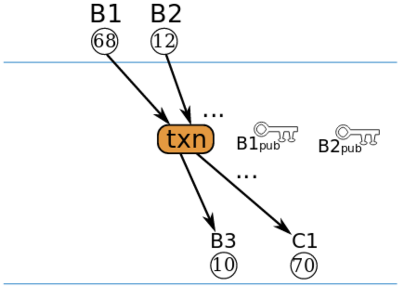


Source: [1]

Form of Single Transaction

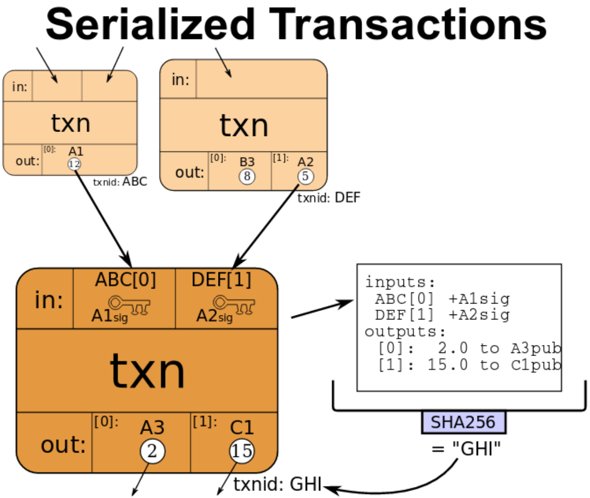
Transactions

- ≥0 inputs, signature for each
- ≥1 outputs, recipient pubkey for each



Source: [1]

Merkle Trees



Source: [1]

Model in Agda

- We will model transactions in Agda.
- Forming Merkle trees is next step.
- In picture the signatures need to apply to the whole transaction not just the input.

Postulated Cryptography

- $\mathsf{Amount} = \mathbb{N}$

- Messages formed from iterated lists of numbers

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

- We postulate Public keys:

```
postulate PublicKey : Set
postulate publicKeytoℕ : (pubk : PublicKey) → ℕ
```

- Messages signed by private key corresponding to public key:

```
postulate Signed : (msg : Message)(pubk : PublicKey) → Set
```

Inputs and Outputs for Transaction

```
record TransactField : Set where
  field
    amount    : Amount
    publicKey : PublicKey

transactFieldToMessage : (inp : TransactField) → Message

transactFieldListToAmount : (inp : List TransactField) → Amount
transactFieldListToAmount [] = 0
transactFieldListToAmount (x :: inp) =
  amount x + transactFieldListToAmount inp
```

Unsigned Transactions

```
record TransactionUnsigned : Set where
  field
    inputs  : List TransactField
    outputs : List TransactField
```

```
transactUnsignedToMessage : (transac : TransactionUnsigned)
                            → Message
```

```
transactionsToPublicKeys : (transac : TransactionUnsigned)
                           → List PublicKey
```

Signed Transaction

```
record Transaction : Set where
  field
    transactions : TransactionUnsigned
    cor : transactFieldListToAmount (inputs transactions) ≥
          transactFieldListToAmount (outputs transactions)
    sig : publicKeysToSignatures
            (transactUnsignedToMessage transactions)
            (transactionsToPublicKeys transactions)
```

Ledger

$$
\begin{aligned}
&\text{Ledger} : \text{Set} \\
&\text{Ledger} = (\mathit{pubk} : \text{PublicKey}) \to \text{Amount}
\end{aligned}
$$

We update a ledger by subtracting the amounts from input fields and adding the amounts from output fields:

$$
\begin{aligned}
\text{addTransactFieldToLedger} : \; & (\mathit{tr} : \text{TransactField}) \\
& (\mathit{oldLedger} : \text{Ledger}) \\
& \to \text{Ledger}
\end{aligned}
$$

$$
\begin{aligned}
\text{subtrTransactFieldFromLedger} : \; & (\mathit{tr} : \text{TransactField}) \\
& (\mathit{oldLedger} : \text{Ledger}) \\
& \to \text{Ledger}
\end{aligned}
$$

Update of Ledger after Transaction

$$
\begin{aligned}
\mathrm{updateLedgerByTransaction} : \ & (tr : \mathrm{Transaction}) \\
& (oldLedger : \mathrm{Ledger}) \\
& \rightarrow \mathrm{Ledger}
\end{aligned}
$$

$$
\begin{aligned}
& \mathrm{updateLedgerByTransaction}\ tr\ oldLedger = \\
& \quad \mathrm{addTransactFieldListToLedger}\ (\mathrm{outputs}\ (\mathrm{transactions}\ tr)) \\
& \quad (\mathrm{subtrTransactFieldListFromLedger}\ (\mathrm{inputs}\ (\mathrm{transactions}\ tr)) \\
& \qquad oldLedger\ )
\end{aligned}
$$

Correctness of Transactions

$$
\begin{array}{l}
\mathsf{correctInput} : (tr : \mathsf{TransactField}) \\
\qquad\qquad\qquad (ledger : \mathsf{Ledger}) \\
\qquad\qquad\qquad \to \mathsf{Set} \\
\mathsf{correctInput}\ tr\ ledger = ledger\ (\mathsf{publicKey}\ tr) \geq \mathsf{amount}\ tr
\end{array}
$$

$$
\begin{array}{l}
\mathsf{correctInputs} : (tr : \mathsf{List}\ \mathsf{TransactField}) \\
\qquad\qquad\qquad (ledger : \mathsf{Ledger}) \\
\qquad\qquad\qquad \to \mathsf{Set}
\end{array}
$$

$$
\begin{array}{l}
\mathsf{correctTransaction} : (tr : \mathsf{Transaction}) \\
\qquad\qquad\qquad\qquad (ledger : \mathsf{Ledger}) \\
\qquad\qquad\qquad\qquad \to \mathsf{Set} \\
\mathsf{correctTransaction}\ tr\ ledger \\
\quad = \mathsf{correctInputs}\ (\mathsf{outputs}\ (\mathsf{transactions}\ tr))\ ledger
\end{array}
$$

Blocks

$$
\begin{array}{l}
\mathsf{Block} : \mathsf{Set} \\
\mathsf{Block} = \mathsf{List}\ \mathsf{Transaction}
\end{array}
$$

$$
\begin{array}{l}
\mathsf{correctBlock} : \ (block : \mathsf{Block}) \\
\qquad\qquad\qquad\ \ (oldLedger : \mathsf{Ledger}) \\
\qquad\qquad\qquad\ \ \to \mathsf{Set} \\
\mathsf{correctBlock}\ []\ oldLedger = \top \\
\mathsf{correctBlock}\ (tr :: block)\ oldLedger = \\
\quad \mathsf{correctTransaction}\ tr\ oldLedger \times \\
\quad \mathsf{correctBlock}\ block\ (\mathsf{updateLedgerByTransaction}\ tr\ oldLedger)
\end{array}
$$

Conclusion

- Introduction to Bitcoin Protocol.
- Model of bitcoins in Agda.
- Next steps:
  - Formalise Merkle trees in Agda
    - Instead of a ledger refer to a list of open parts of transactions.
  - Formalise smart contracts.
    - Simplest form transaction relesed when signed for but after timeout returned to sender.
    - New transactions:
      smart contract established
      smart contract fulfilled.
  - Specify correctness properties (not easy!).
  - Prove correctness.
- These slides were "programmed" using lagda and Adelsberger/Abel's "lagdaLight".

Bibliography

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