Qualificação

Guilherme Horta Alvares da Silva
Orientador: Flávio Codeço Coelho

Fundação Getulio Vargas

2019

Programação de uma criptomoeda em linguagem Agda

- Objetivo
- Justificativa
- Introdução
  - Criptomoedas
  - Agda
  - Bugs em criptomoedas
- Trabalho executado
- Próximos passos
- Referências Bibliográficas

Objetivo

- Programar uma criptomoeda (similar ao Bitcoin) em Agda, que é uma linguagem com tipos dependentes.

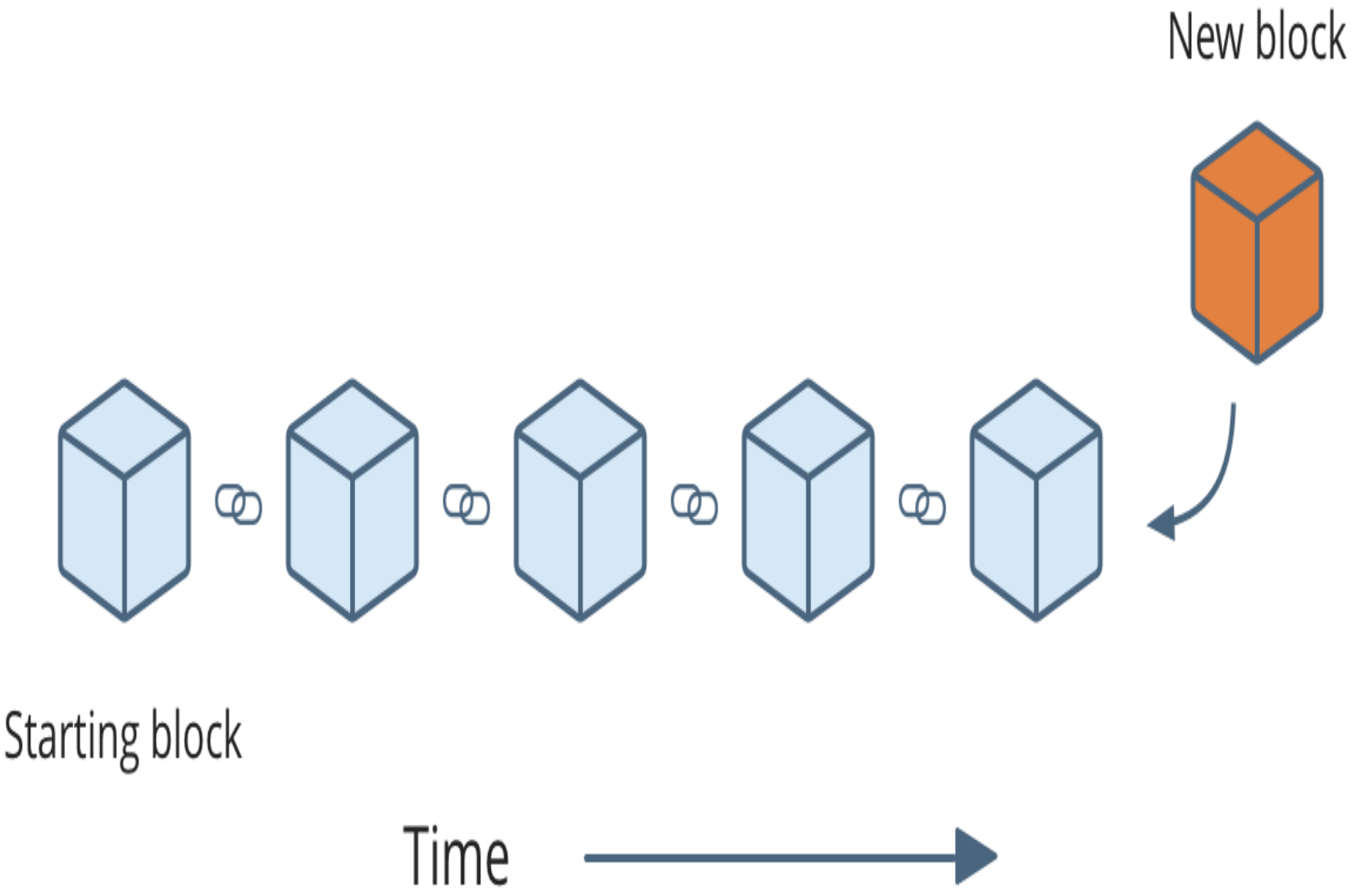
Justificativa

- Programar um protocólo de criptomoedas livre de erros (bugs)
- Utilizar Agda permite, além da programação da criptomoeda, especificar como ela deve funcionar (Norell, 2008)

Criptomoeda

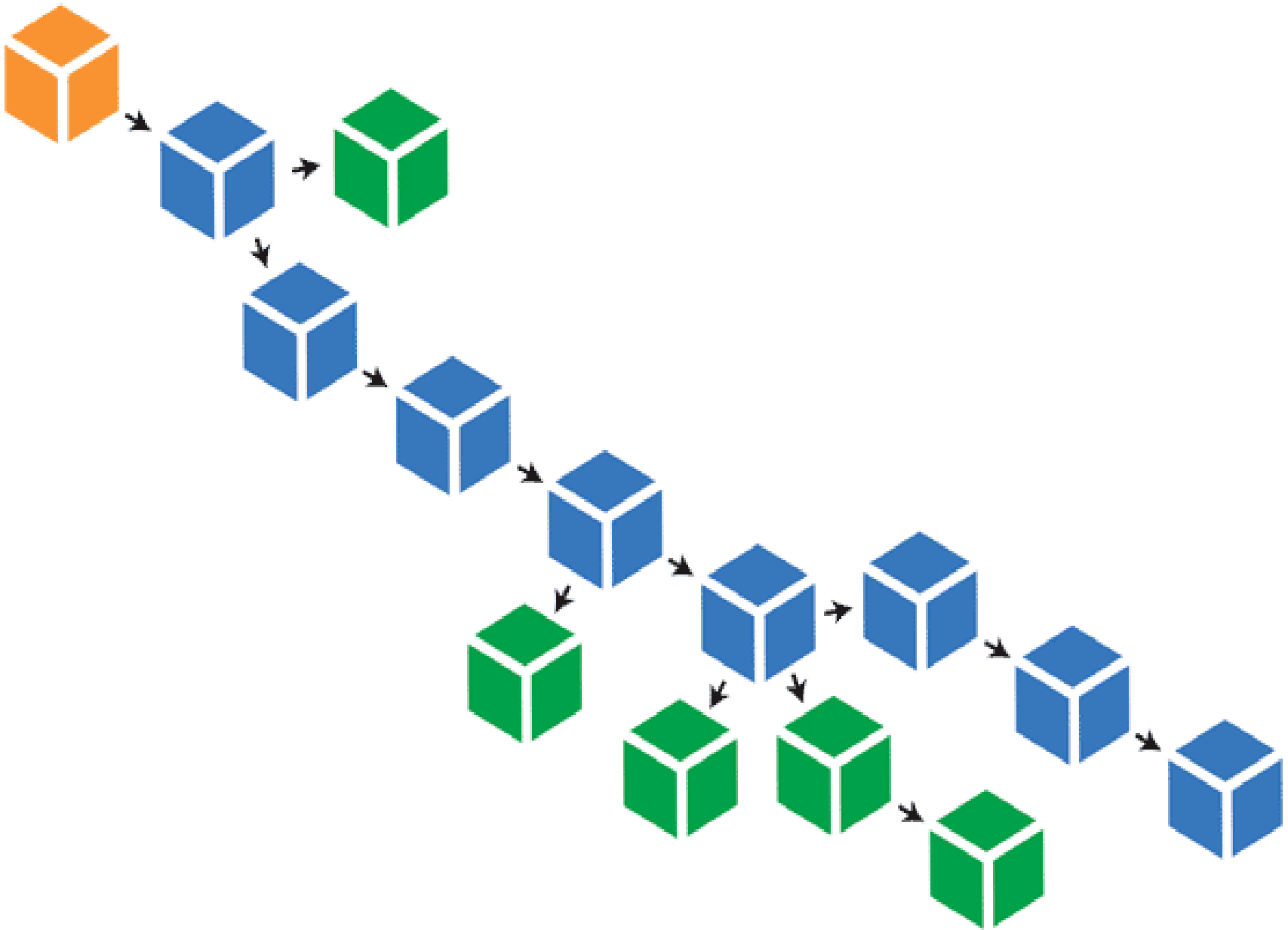
- Uma criptomoeda é um meio de troca descentralizado que se utiliza da tecnologia de blockchain e da criptografia para assegurar a validade das transações e a criação de novas unidades da moeda
- O bitcoin é considerado a primeira moeda digital mundial descentralizada, constituindo um sistema econômico alternativo (*peer-to-peer electronic cash system*) e responsável pelo ressurgimento do sistema bancário livre (Nakamoto et al., 2008)
- O Bitcoin permite transações financeiras sem intermediários, mas verificadas por todos usuários (nodos da rede). Estas transações são gravadas em um banco de dados distribuídos (uma rede descentralizada), chamado de *blockchain*.

Blockchain

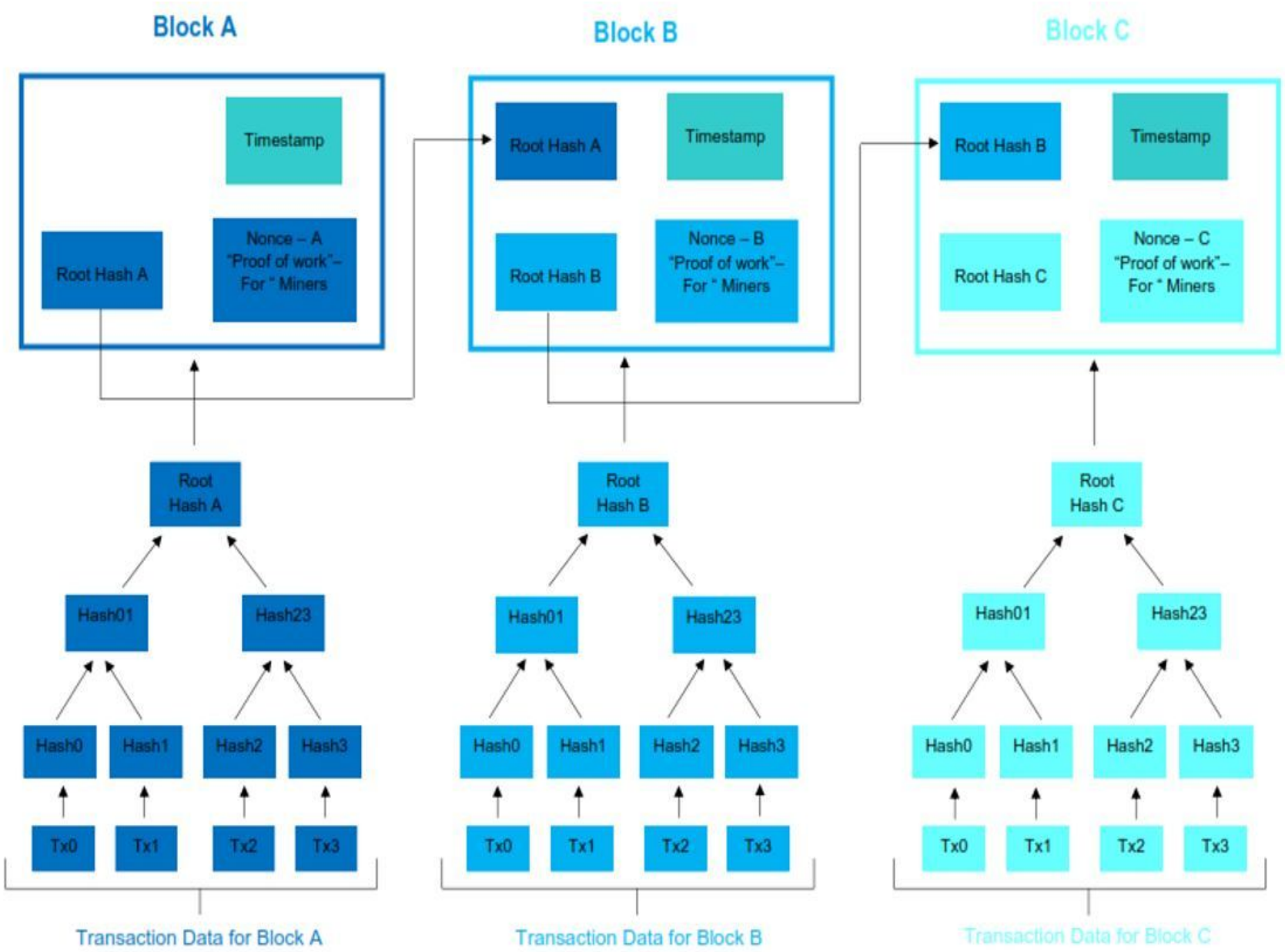


Crypto Beginners

Blockchain



Blockchain


Block A
Timestamp
Root Hash A
Nonce – A
"Proof of work"–
For " Miners
Root
Hash A
Hash01
Hash23
Hash0
Hash1
Hash2
Hash3
Tx0
Tx1
Tx2
Tx3
Transaction Data for Block A
Block B
Root Hash A
Timestamp
Root Hash B
Nonce – B
"Proof of work"–
For " Miners
Root
Hash B
Hash01
Hash23
Hash0
Hash1
Hash2
Hash3
Tx0
Tx1
Tx2
Tx3
Transaction Data for Block B
Block C
Root Hash B
Timestamp
Root Hash C
Nonce – C
"Proof of work"–
For " Miners
Root
Hash C
Hash01
Hash23
Hash0
Hash1
Hash2
Hash3
Tx0
Tx1
Tx2
Tx3
Transaction Data for Block C

Agda

- Linguagem funcional com sistema de tipos expressivos capazes de representar as especificações
- Possibilita especificar e programar em um único lugar
- O processo de verificação acontece no compilador

Agda — II

- A linguagem não possui *Built-in* como em Python
- Tipos como inteiros, ponto flutuantes, *strings* e vetores devem ser definidos pelo próprio usuário
- Tipos em Agda são uma generalização de tipos de dados algébricos encontrados em Haskell e ML

Agda — III

- Definição dos números naturais como axiomas de Peano:

```
data ℕ : Set where
  zero : ℕ
  suc  : ℕ → ℕ
```

Agda — IV

- Adição em Agda:

$$
\begin{aligned}
&\_+\_ : \mathbb{N} \to \mathbb{N} \to \mathbb{N} \\
&\mathsf{zero} \quad + \; n = n \\
&\mathsf{suc}\; m + \; n = \mathsf{suc}\; (m + \; n)
\end{aligned}
$$

Agda — V

- Um tipo é dependente se este depende de um valor.
- Exemplo — Listas indexadas por seu tamanho:

```
data Vec (A : Set) : ℕ → Set where
  []  : Vec A 0
  _::_ : ∀ {n} → A → Vec A n → Vec A (suc n)
```

Agda — VI

- Modo seguro de remover primeiro elemento do vetor:

$$\begin{aligned}&\mathsf{head} : \forall\ \{A : \mathsf{Set}\}\{n : \mathbb{N}\} \to \mathsf{Vec}\ A\ (\mathsf{suc}\ n) \to A\\&\mathsf{head}\ (x :: xs) = x\end{aligned}$$

- Função zip com 2 vetores de mesmo tamanho:

$$\begin{aligned}&\mathsf{zipWith} : \forall\ \{A\ B\ C : \mathsf{Set}\}\{n : \mathbb{N}\} \to (A \to B \to C)\\&\qquad\qquad \to \mathsf{Vec}\ A\ n \to \mathsf{Vec}\ B\ n \to \mathsf{Vec}\ C\ n\\&\mathsf{zipWith}\ \_\ []\ [] = []\\&\mathsf{zipWith}\ f\ (x :: xs)\ (y :: ys) = f\ x\ y :: \mathsf{zipWith}\ f\ xs\ ys\end{aligned}$$

Maleabilidade de transacao

- Nesse tipo de bug, é possível alterar o hash da transação depois que ela foi enviada
- Todos os dados para calcular do hash não eram previamente calculados. Assim, o minerador poderia alterar o hash da transação
- O ataque consistiria de um usuário enviar uma transação e ela não ser confirmada pelo sistema. Logo em seguida, este mesmo usuário enviaria uma outra transação. Desta forma, ele faria duas transações com a mesma moeda
- Este tipo de BUG pode ser evitado usando tipos dependentes. Colocando como característica da transação, o fato de seu ID ser único

DAO Bug

- *Bug* que aconteceu em um cripto-contrato da rede Ethereum com um prejuízo de mais do que 250 milhões de dólares (Wood et al., 2014)
- No cripto-contrato, existia uma função recursiva que não terminava. Ou seja, o usuário enviava uma quantidade de ethereum, depois acontecia um *loop* infinito e só depois era feito a atualização do seu balanço
- Em Agda, esse tipo de bug seria evitado, pois é necessário provar que a função termina. Logo, *loops* infinitos não são possíveis em Agda

Trabalho já executado

- A criptomoeda já foi programada em Python
- Parte da *blockchain* já foi programada em Agda
- As transações já foram descritas na literatura: UTXO (*Unspent transaction output*)
  (Setzer, 2018)

Blockchain em Agda

```
data GenesisBlock : ℕ → Set where
  block : (n : ℕ) → (sb : SimpleBlock) → n ≡ hashBlock sb → GenesisBlock n

data Block : ℕ → ℕ → Set where
  block : (n : ℕ) → (m : ℕ) → (sb : SimpleBlock) → m ≡ hashBlock sb → Block n m

data Blockchain : ℕ → Set where
  gen : {n : ℕ} → GenesisBlock n → Blockchain n
  cons : {n m : ℕ} → Block n m → Blockchain n → Blockchain m
```

Próximos passos

- Anexar a *blockchain* às transações já programadas em Agda
- Provar alguns teoremas relacionados à criptomoeda

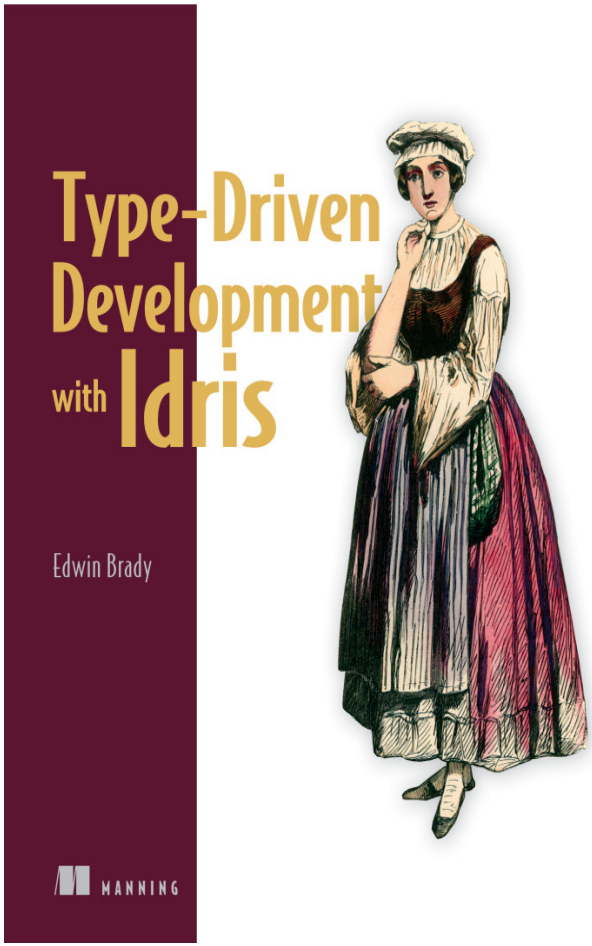
Teoremas

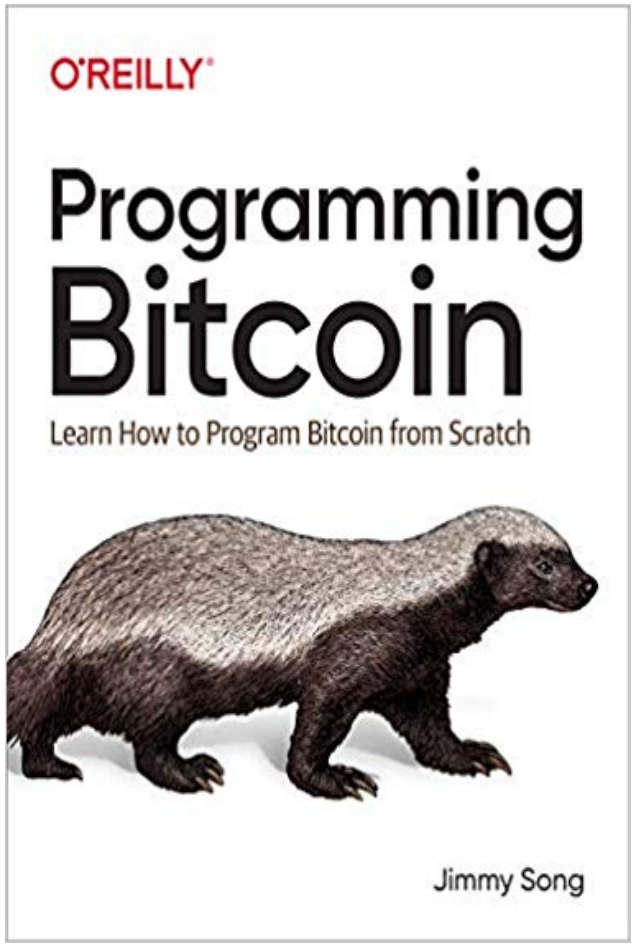
- Se uma transação tem algum *output* que não foi usado em nenhuma outra transação, então ela deve estar na lista de *outputs transactions* não usados
- Se uma transação tem algum *output* que foi gasto, ele não pode ser usado novamente
- Provar que transações e mensagens ids são únicos

O que não será realizado

- Modelo de criptomoeda em que é possível algum tipo de *fork*. Por exemplo, no bitcoin, é possível que exista algum tipo de *fork* temporário
- *Pool* de transações. Sua utilidade é apenas para guardar as transações que ainda não foram adicionadas a *blockchain* Isso pode ser feito fora do protocolo principal
- Otimização e protocolos RPC (*Remote Procedure Call*). O objetivo do projeto é definir as propriedades da criptomoeda, não como ela será implementada e usada

Livros


Type-Driven Development with Idris
Edwin Brady
MANNING


O'REILLY
Programming Bitcoin
Learn How to Program Bitcoin from Scratch
Jimmy Song

Referências Bibliográficas

Nakamoto, S., et al. (2008). Bitcoin: A peer-to-peer electronic cash system.

Norell, U. (2008). Dependently typed programming in agda. In *International school on advanced functional programming* (pp. 230–266).

Setzer, A. (2018). Modelling bitcoin in agda. *arXiv preprint arXiv:1804.06398*.

Wood, G., et al. (2014). Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, *151*, 1–32.