Uma versão simplificada do Bitcoin, implementada em Agda

Guilherme Horta Alvares da Silva
Orientador: Flávio Codeço Coelho

Fundação Getulio Vargas

2019

Uma versão simplificada do Bitcoin, implementada em 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)

Linguagem Funcional

- Em linguagens funcionais, toda função retorna o mesmo output para o mesmo input.
- Em Agda, toda função é total e ela sempre termina.

Efeitos Colaterais

- Para Agda ser utilizada como um programa, ela tem que lidar com o sistema operacional. Operação chamada de IO (Input Output).
- Várias pesquisas estão sendo feitas para minimizar os efeitos colaterais do IO.
- Lógica linear é usada para lidar com recursos, como abrir arquivos.
- Efeitos algébricos são utilizados para distinguir os diferentes tipos de efeitos relacionados ao sistema operacional.

Efeitos Colaterais

- Frontend e backend de serviços estão sendo programados na mesma linguagem funcional para minimizar os efeitos colaterais.
- Haxl, usado no Facebook, é uma ferramenta que faz o cache das operações IO para fácil debug, log e teste.
- Facebook também faz Hot Swapping de forma funcional. Ou seja, ele não reinicia o processo para iniciar um novo. Algo que já era comum em Lisp.
- Sistemas distribuidos já estão sendo completamente descritos por meio de linguagens funcionais. Whatsapp utiliza Elixir como linguagem.

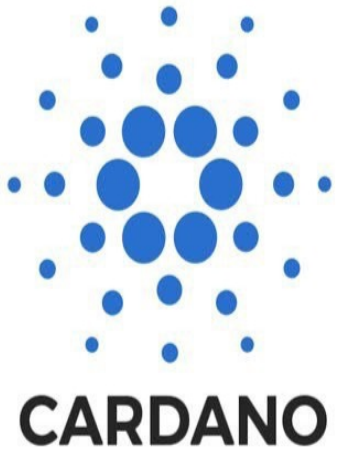
Beneficios

- Por causa dos inúmeros benefícios das linguagens funcionais, elas estão sendo cada vez mais utilizadas na indústria.
- Garantem corretude de software. É de fácil reproducibilidade, testabilidade e é mais fácil de debugar.
- Entretanto, elas são mais difíceis de serem aprendidas. Necessitam alto grau de abstração e é mais difícil de encontrar mão de obra.

Empresas

Jane Street

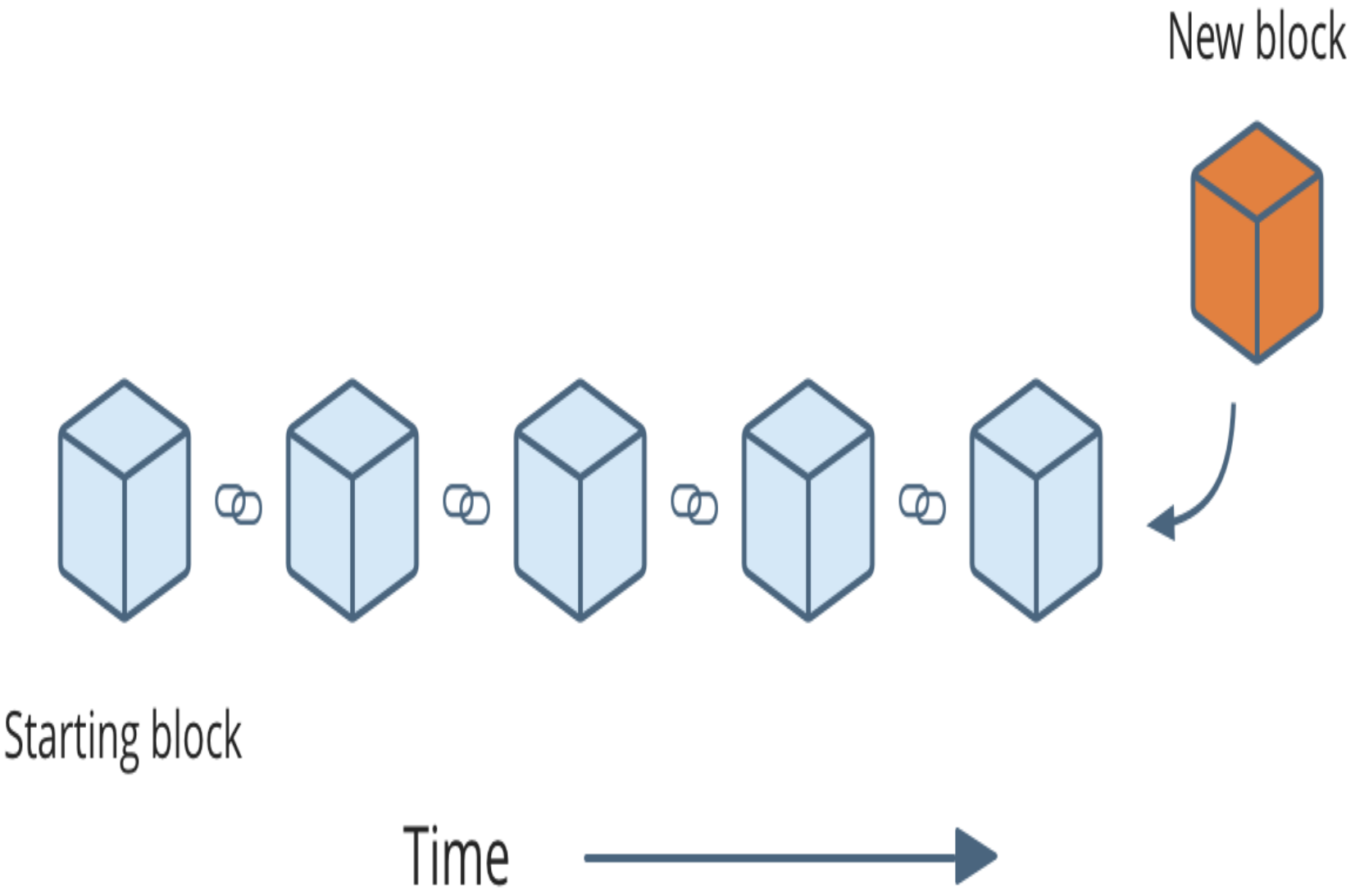
Criptomoedas



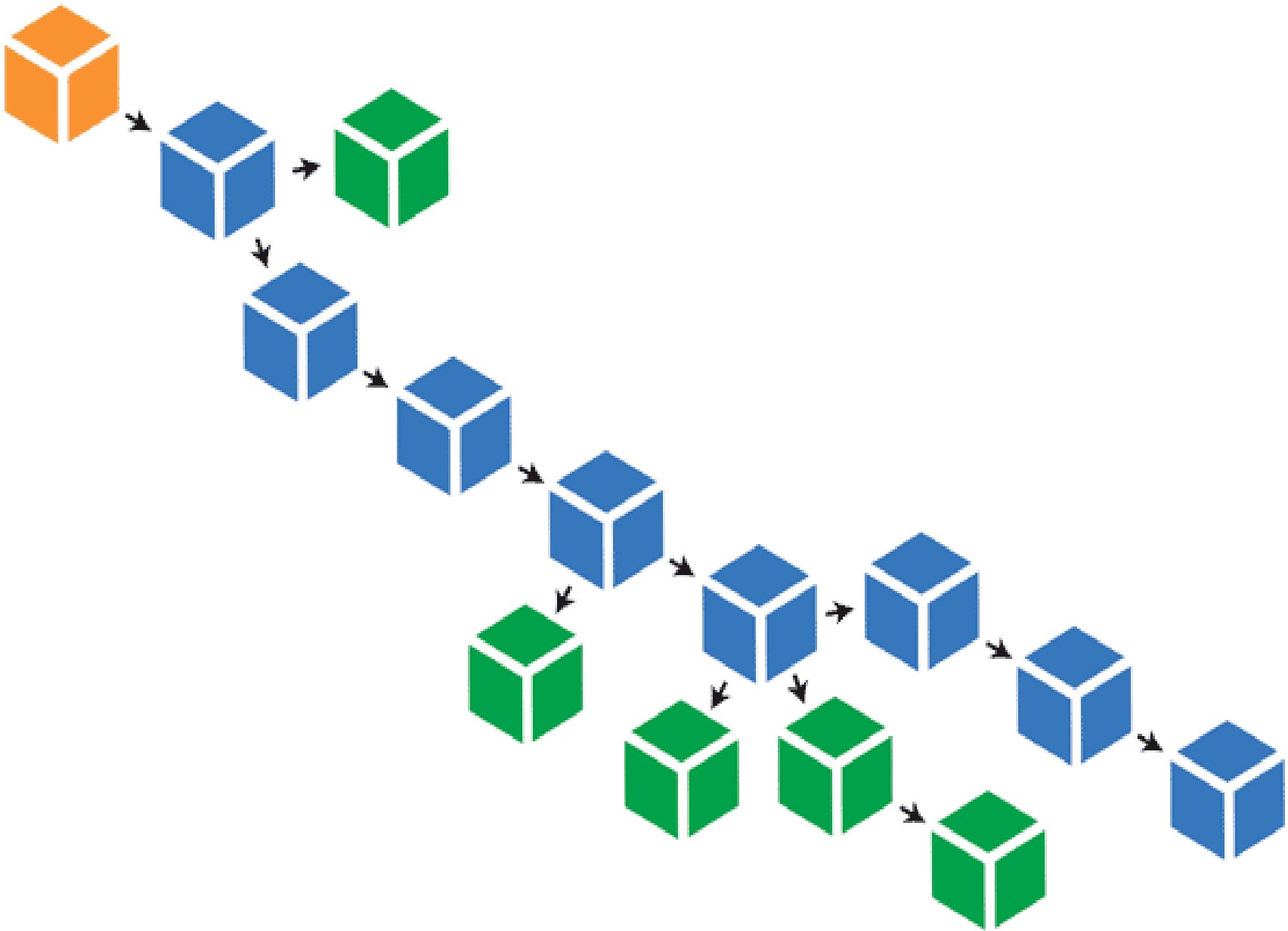
Criptomoeda

- 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*.

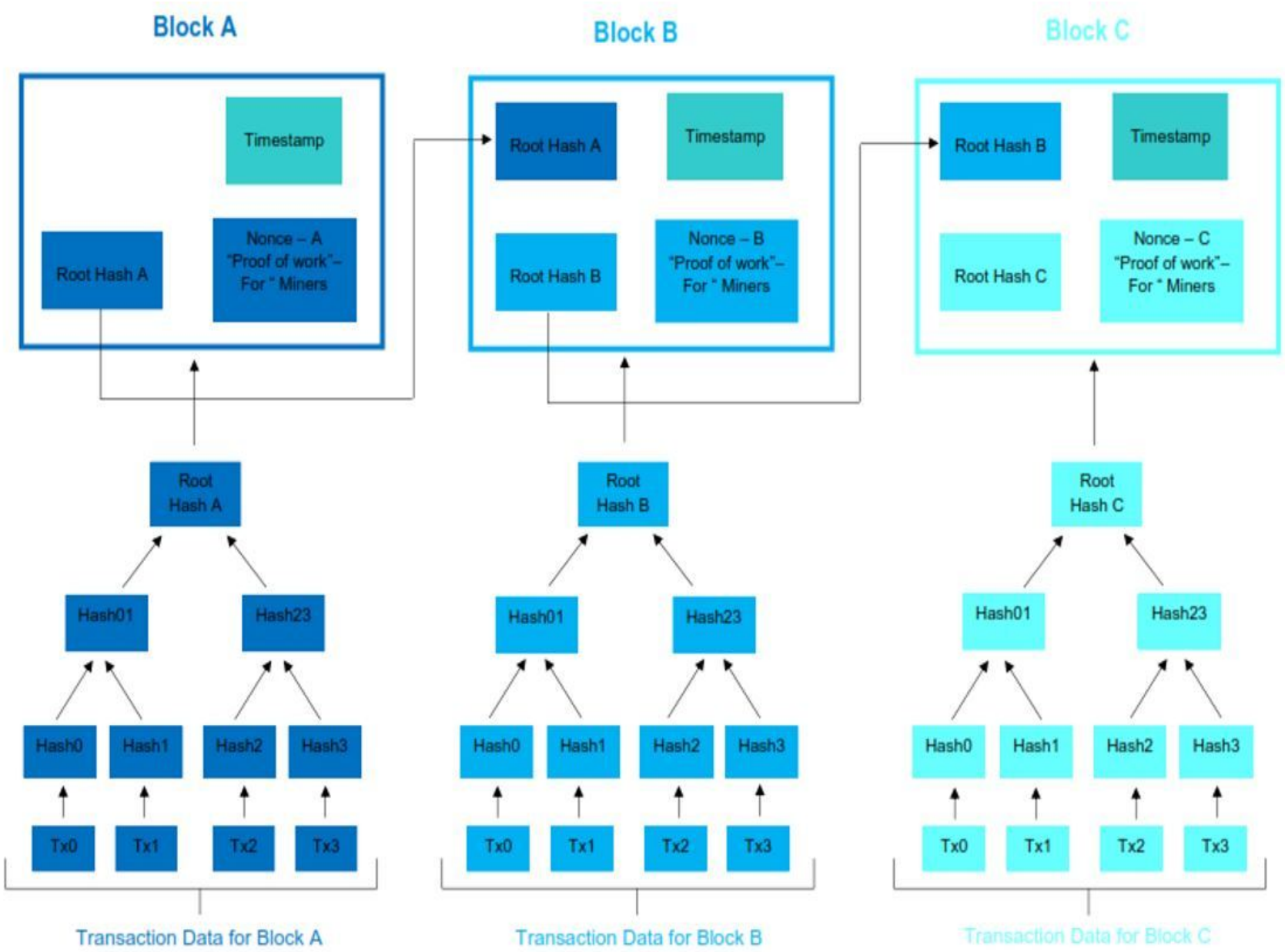
Blockchain



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

Identidade

- Agda utiliza alguns conceitos do Lamdba Calculus.

$$\begin{aligned} &\mathsf{id} : \{A : \mathsf{Set}\} \to A \to A \\ &\mathsf{id} = \lambda\ x \to x \end{aligned}$$

$$\begin{aligned} &\mathsf{id'} : \{A : \mathsf{Set}\} \to A \to A \\ &\mathsf{id'}\ x = x \end{aligned}$$

Booleanos

- Definição de booleanos

$$\mathsf{true} : \{A : \mathsf{Set}\} \to A \to A \to A$$
$$\mathsf{true}\ x\ y = x$$

$$\mathsf{false} : \{A : \mathsf{Set}\} \to A \to A \to A$$
$$\mathsf{false}\ x\ y = y$$

Números naturais

- Definição de números naturais

$$\begin{aligned}&\text{zero} : \{A : \text{Set}\} \to (A \to A) \to A \to A\\&\text{zero}\ \mathit{suc}\ z = z\end{aligned}$$

$$\begin{aligned}&\text{one} : \{A : \text{Set}\} \to (A \to A) \to A \to A\\&\text{one}\ \mathit{suc}\ z = \mathit{suc}\ z\end{aligned}$$

$$\begin{aligned}&\text{two} : \{A : \text{Set}\} \to (A \to A) \to A \to A\\&\text{two}\ \mathit{suc}\ z = \mathit{suc}\ (\mathit{suc}\ z)\end{aligned}$$

Zero

```
isZero : {A : Set} → ((A → A) → A → A) → (A → A → A)
isZero n true false = n (λ _ → false) true

isZero-zero : {A : Set} → Result (isZero {A} zero)
isZero-zero = res (λ true false → true)

isZero-two : {A : Set} → Result (isZero {A} two)
isZero-two = res (λ true false → false)
```

Soma

```
plus : {A : Set} → ((A → A) → A → A)
  → ((A → A) → A → A)
  → ((A → A) → A → A)
plus n m = λ suc z → n suc (m suc z)
```

```
_+_ : {A : Set} → ((A → A) → A → A)
  → ((A → A) → A → A)
  → ((A → A) → A → A)
_+_ n m suc z = n suc (m suc z)
```

```
one+one : {A : Set} → Result (_+_ {A} one one)
one+one = res (λ suc z → suc (suc z))
```

Lista

```
emptyList : {A List : Set}
  → (A → List → List) → List → List
emptyList _::_ nil = nil

natList : {A List : Set}
  → (((A → A) → A → A) → List → List) → List → List
natList _::_ nil = one :: (two :: nil)

sumList : {A List : Set}
  → Result (natList {A} {(A → A) → A → A} _+_ zero)
sumList = res (λ suc z → suc (suc (suc z)))
```

Soma

- Definição de tipos de soma

$$\text{left} : \{A\ B\ C : \text{Set}\} \to A \to (A \to C) \to (B \to C) \to C$$
$$\text{left}\ x\ f\ g = f\ x$$

$$\text{right} : \{A\ B\ C : \text{Set}\} \to B \to (A \to C) \to (B \to C) \to C$$
$$\text{right}\ x\ f\ g = g\ x$$

Tipo soma

```
zero-isZero : ∀ {A}
  → Result (zero-left {A} isZero id)
zero-isZero = res (λ true false → true)

one-isZero : ∀ {A}
  → Result (one-left {A} isZero id)
one-isZero = res (λ true false → false)

false-id : ∀ {A}
  → Result (false-right {(A → A) → A → A} isZero id)
false-id = res (λ true false → false)

true-id : ∀ {A}
  → Result (false-right {(A → A) → A → A} isZero id)
true-id = res (λ true false → false)
```

Tuplas

- Definição de tupla

$$\begin{aligned}&\text{tuple} : \{A\ B\ C : \text{Set}\} \to A \to B \to (A \to B \to C) \to C\\&\text{tuple}\ x\ y\ f = f\ x\ y\end{aligned}$$

$$\begin{aligned}&\text{zero-false} : \{A\ B\ C : \text{Set}\} \to (((A \to A) \to A \to A)\\&\quad \to (B \to B \to B) \to C) \to C\\&\text{zero-false} = \text{tuple zero false}\end{aligned}$$

$$\begin{aligned}&\text{one-true} : \{A\ B\ C : \text{Set}\} \to (((A \to A) \to A \to A)\\&\quad \to (B \to B \to B) \to C) \to C\\&\text{one-true} = \text{tuple one true}\end{aligned}$$

Adição de tuplas

```
add-true : {A : Set} → ((A → A) → A → A)
  → (A → A → A) → ((A → A) → A → A)
add-true n b suc z = b (suc (n suc z)) (n suc z)

add-zero-false : {A : Set}
  → Result (zero-false {(A → A) → A → A} add-true)
add-zero-false = res (λ suc z → z)

add-one-true : ∀ {A}
  → Result (one-true {(A → A) → A → A} add-true)
add-one-true = res (λ suc z → suc (suc z))
```

Agda — IV

- Adição em Agda:

Agda — V

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

Agda — VI

- Modo seguro de remover primeiro elemento do vetor:
- Função zip com 2 vetores de mesmo tamanho:

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

Próximos passos

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

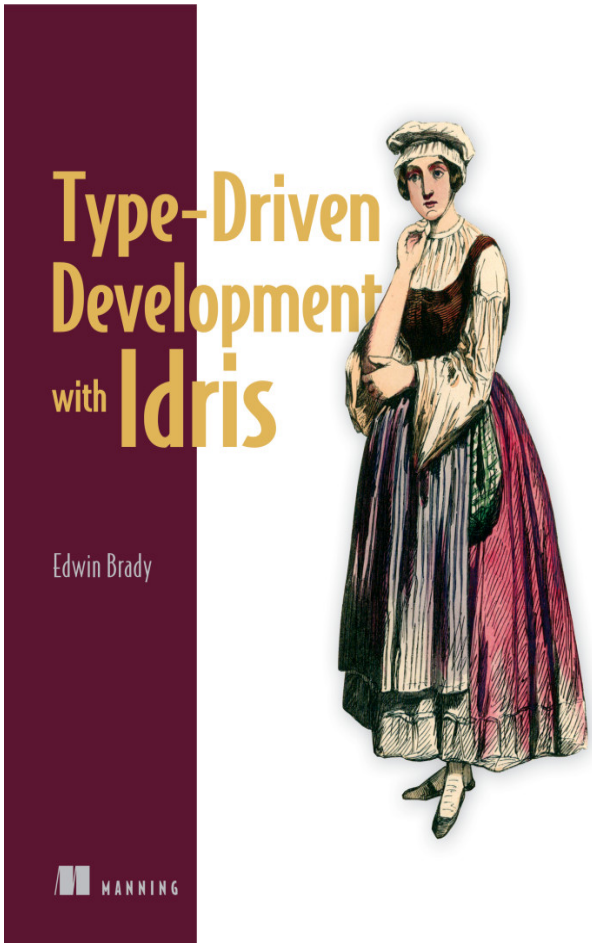
Teoremas

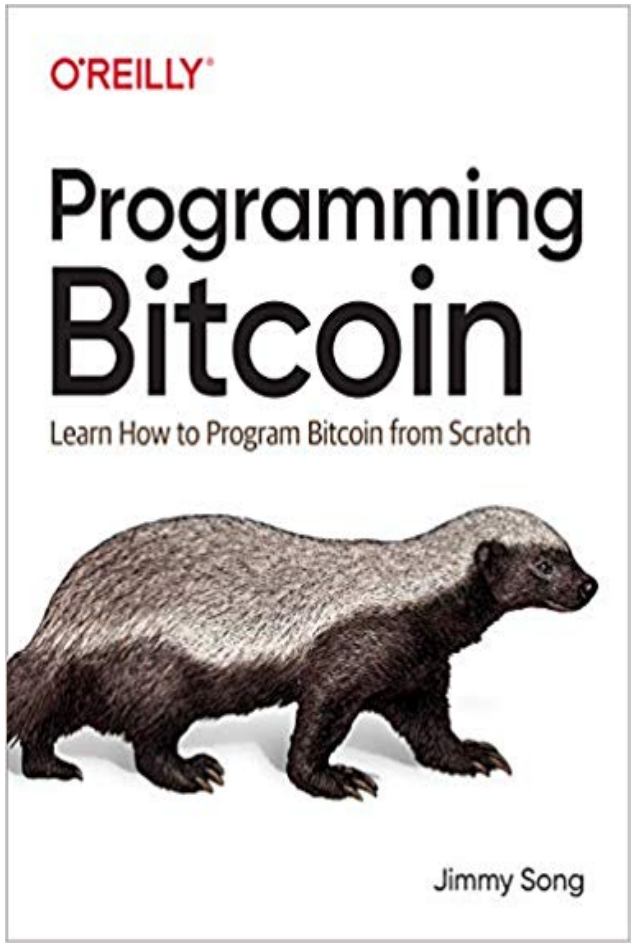
- Se uma transação tem algum *output* que não foi usado em nenhuma outra transação, então ela deve estar na lista de *outputs transactions* não usados
- Se uma transação tem algum *output* que foi gasto, ele não pode ser usado novamente
- Provar que transações e mensagens ids são únicos

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.