

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 protocolo 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.

Benefícios

- 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

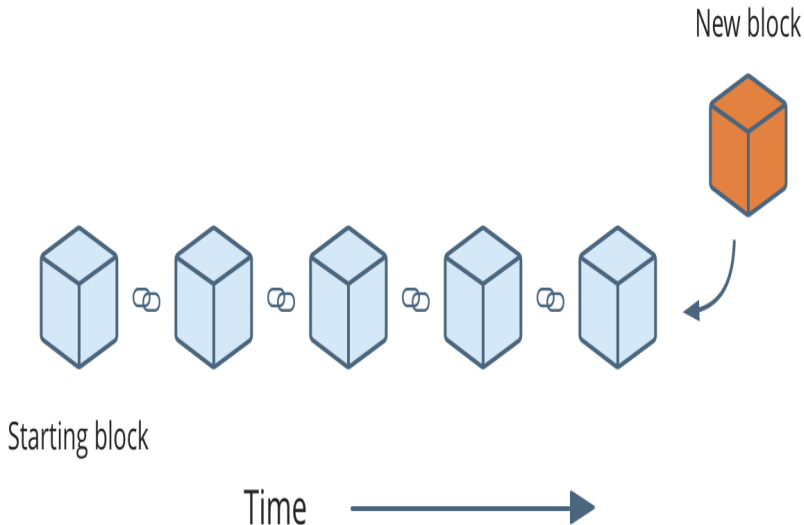
Criptomoedas



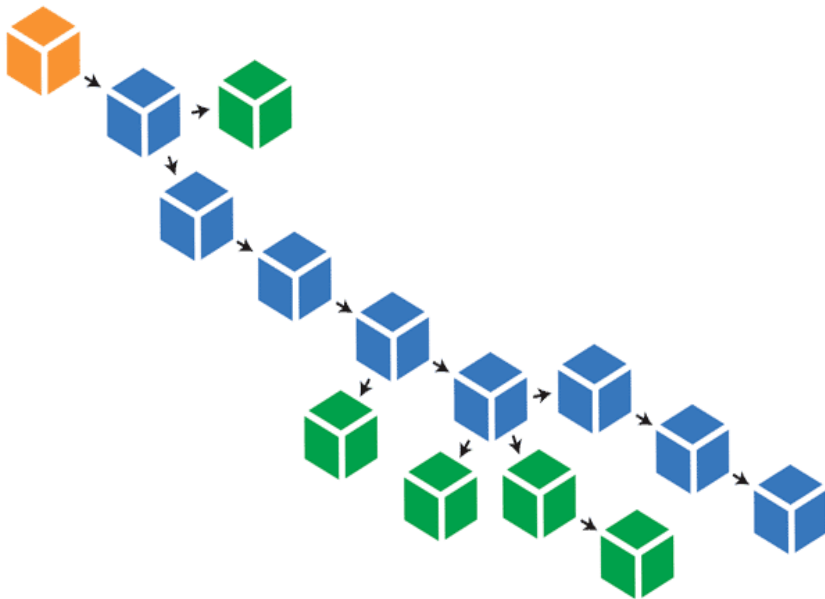
Criptomoeda

- 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*.

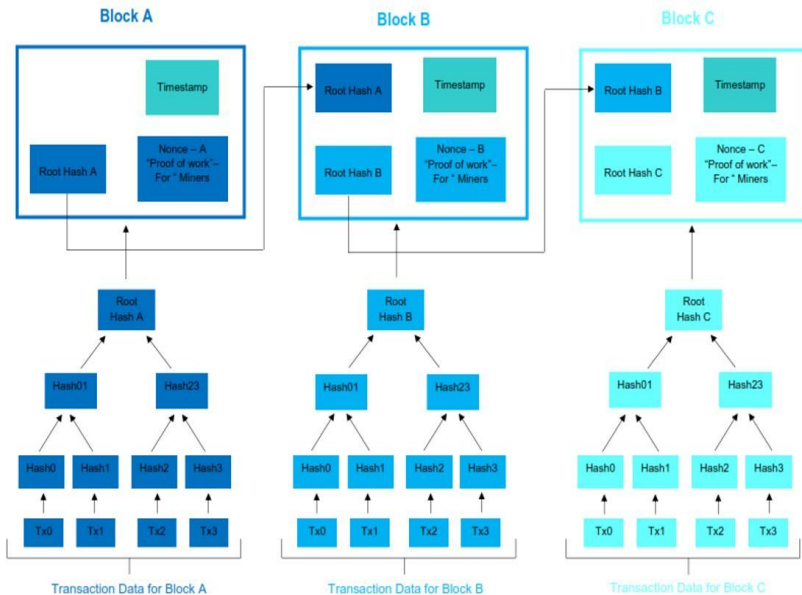
Blockchain



Blockchain



Blockchain



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 Lambda Calculus.

$\text{id} : \{A : \text{Set}\} \rightarrow A \rightarrow A$

$\text{id} = \lambda x \rightarrow x$

$\text{id}' : \{A : \text{Set}\} \rightarrow A \rightarrow A$

$\text{id}' x = x$

Booleans

- Definição de booleanos

$\text{true} : \{A : \text{Set}\} \rightarrow A \rightarrow A \rightarrow A$

$\text{true } x y = x$

$\text{false} : \{A : \text{Set}\} \rightarrow A \rightarrow A \rightarrow A$

$\text{false } x y = y$

Números naturais

- Definição de números naturais

`zero` : $\{A : \text{Set}\} \rightarrow (A \rightarrow A) \rightarrow A \rightarrow A$
`zero` *suc* *z* = *z*

`one` : $\{A : \text{Set}\} \rightarrow (A \rightarrow A) \rightarrow A \rightarrow A$
`one` *suc* *z* = *suc* *z*

`two` : $\{A : \text{Set}\} \rightarrow (A \rightarrow A) \rightarrow A \rightarrow A$
`two` *suc* *z* = *suc* (*suc* *z*)

Zero

`isZero` : $\{A : \text{Set}\} \rightarrow ((A \rightarrow A) \rightarrow A \rightarrow A) \rightarrow (A \rightarrow A \rightarrow A)$

`isZero` n *true false* = $n (\lambda _ \rightarrow \text{false})$ *true*

`isZero-zero` : $\{A : \text{Set}\} \rightarrow \text{Result } (\text{isZero } \{A\} \text{ zero})$

`isZero-zero` = *res* $(\lambda \text{ true false} \rightarrow \text{true})$

`isZero-two` : $\{A : \text{Set}\} \rightarrow \text{Result } (\text{isZero } \{A\} \text{ two})$

`isZero-two` = *res* $(\lambda \text{ true false} \rightarrow \text{false})$

Soma

$\text{plus} : \{A : \text{Set}\} \rightarrow ((A \rightarrow A) \rightarrow A \rightarrow A)$

$\rightarrow ((A \rightarrow A) \rightarrow A \rightarrow A)$

$\rightarrow ((A \rightarrow A) \rightarrow A \rightarrow A)$

$\text{plus } n \ m = \lambda \ \text{suc } z \rightarrow n \ \text{suc } (m \ \text{suc } z)$

$_+ _ : \{A : \text{Set}\} \rightarrow ((A \rightarrow A) \rightarrow A \rightarrow A)$

$\rightarrow ((A \rightarrow A) \rightarrow A \rightarrow A)$

$\rightarrow ((A \rightarrow A) \rightarrow A \rightarrow A)$

$_+ _ \ n \ m \ \text{suc } z = n \ \text{suc } (m \ \text{suc } z)$

$\text{one+one} : \{A : \text{Set}\} \rightarrow \text{Result } (_+ _ \ \{A\} \ \text{one} \ \text{one})$

$\text{one+one} = \text{res } (\lambda \ \text{suc } z \rightarrow \text{suc } (\text{suc } z))$

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

`left` : $\{A\ B\ C : \text{Set}\} \rightarrow A \rightarrow (A \rightarrow C) \rightarrow (B \rightarrow C) \rightarrow C$

`left` $x\ f\ g = f\ x$

`right` : $\{A\ B\ C : \text{Set}\} \rightarrow B \rightarrow (A \rightarrow C) \rightarrow (B \rightarrow C) \rightarrow C$

`right` $x\ f\ g = g\ x$

Tipo soma

$\text{zero-isZero} : \forall \{A\}$
 $\rightarrow \text{Result } (\text{zero-left } \{A\} \text{ isZero id})$
 $\text{zero-isZero} = \text{res } (\lambda \text{ true false} \rightarrow \text{true})$

$\text{one-isZero} : \forall \{A\}$
 $\rightarrow \text{Result } (\text{one-left } \{A\} \text{ isZero id})$
 $\text{one-isZero} = \text{res } (\lambda \text{ true false} \rightarrow \text{false})$

$\text{false-id} : \forall \{A\}$
 $\rightarrow \text{Result } (\text{false-right } \{(A \rightarrow A) \rightarrow A \rightarrow A\} \text{ isZero id})$
 $\text{false-id} = \text{res } (\lambda \text{ true false} \rightarrow \text{false})$

$\text{true-id} : \forall \{A\}$
 $\rightarrow \text{Result } (\text{false-right } \{(A \rightarrow A) \rightarrow A \rightarrow A\} \text{ isZero id})$
 $\text{true-id} = \text{res } (\lambda \text{ true false} \rightarrow \text{false})$

Tuplas

- Definição de tupla

$\text{tuple} : \{A \ B \ C : \text{Set}\} \rightarrow A \rightarrow B \rightarrow (A \rightarrow B \rightarrow C) \rightarrow C$
 $\text{tuple } x \ y \ f = f \ x \ y$

$\text{zero-false} : \{A \ B \ C : \text{Set}\} \rightarrow (((A \rightarrow A) \rightarrow A \rightarrow A) \rightarrow (B \rightarrow B \rightarrow B) \rightarrow C) \rightarrow C$
 $\text{zero-false} = \text{tuple zero false}$

$\text{one-true} : \{A \ B \ C : \text{Set}\} \rightarrow (((A \rightarrow A) \rightarrow A \rightarrow A) \rightarrow (B \rightarrow B \rightarrow B) \rightarrow C) \rightarrow C$
 $\text{one-true} = \text{tuple one true}$

Adição de tuplas

$\text{add-true} : \{A : \text{Set}\} \rightarrow ((A \rightarrow A) \rightarrow A \rightarrow A)$
 $\rightarrow (A \rightarrow A \rightarrow A) \rightarrow ((A \rightarrow A) \rightarrow A \rightarrow A)$
 $\text{add-true } n \ b \ \text{suc } z = b \ (\text{suc } (n \ \text{suc } z)) \ (n \ \text{suc } z)$

$\text{add-zero-false} : \{A : \text{Set}\}$
 $\rightarrow \text{Result } (\text{zero-false } \{(A \rightarrow A) \rightarrow A \rightarrow A\} \ \text{add-true})$
 $\text{add-zero-false} = \text{res } (\lambda \ \text{suc } z \rightarrow z)$

$\text{add-one-true} : \forall \{A\}$
 $\rightarrow \text{Result } (\text{one-true } \{(A \rightarrow A) \rightarrow A \rightarrow A\} \ \text{add-true})$
 $\text{add-one-true} = \text{res } (\lambda \ \text{suc } z \rightarrow \text{suc } (\text{suc } z))$

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

Funções criptográficas

- No bitcoin, existe a chave privada e a pública.
- A chave privada serve para assinar a transação.
- A chave pública, derivada da chave privada, serve para demonstrar que você é o dono da chave privada.
- Essas funções utilizam como base a função hash SHA-256.

Função hash

- Uma função hash é uma função que serve para comprimir dados grandes, como um vídeo, em um número pequeno. De forma que dois arquivos diferentes sempre terão hashes diferentes (injetividade)
- Pelo princípio da casa dos pombos, isso é impossível. Porém, nunca foi encontrado nenhum dois arquivos que possuem o mesmo hash.
- O Google encontrou dois arquivos que possuíam o mesmo hash no SHA-1. Logo em seguida, essa função hash parou de ser usada.

Funções criptográficas

```

postulate _priv≡pub_ : PrivateKey → PublicKey → Set
postulate publicKey2Address : PublicKey → Address
postulate Signed : Msg → PublicKey → Signature → Set
postulate Signed? : (msg : Msg) (pk : PublicKey)
                    (sig : Signature) → Dec $ Signed msg pk sig
postulate hashMsg : Msg → Hashed
postulate hash-inj : ∀ m n → hashMsg m ≡ hashMsg n
                    → m ≡ n

```

```

record SignedWithSigPbk (msg : Msg)(address : Address)
  : Set where
  field
    publicKey : PublicKey
    pbkCorrect : publicKey2Address publicKey ≡ address
    signature : Signature
    signed : Signed msg publicKey signature

```

Próximos passos

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

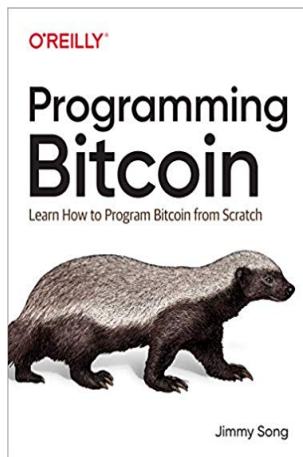
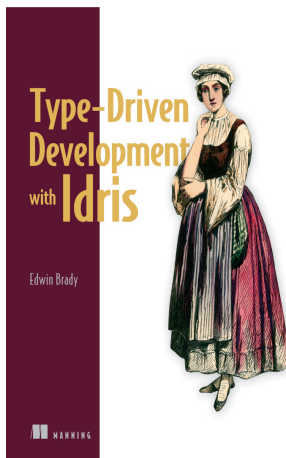
Teoremas

- 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



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