Verificação de Contratos Inteligentes para Solidity/Ethereum

Danilo Yudi Futata Kassuya[1], **Adilson Luiz Bonifacio**[1]

[1]Departamento de Computação – Universidade Estadual de Londrina (UEL)
Caixa Postal 10.011 – CEP 86057-970 – Londrina – PR – Brasil

danilo.yudi.futata@uel.br, bonifacio@uel.br

Abstract. *Blockchain technology and cryptocurrencies have allowed for an automatic and decentralized execution of digital contracts, giving rise to the concept of Smart Contracts. A smart contract can be written using programming languages to then be stored and executed in a blockchain. They need no third-party to manage their execution which occurs automatically according to their agreements. In this setting, methods and practical tools are deemed important when verifying contracts of this nature in order to guarantee their correctness. Therefore, this work aims to propose methods and tools to support the verification process of smart contracts.*

Palavras-chave: *Blockchain*, *smart contracts*, Ethereum, Solidity.

Resumo. *A tecnologia* Blockchain *e das criptomoedas permitiu que contratos, de um modelo geral, também pudessem ser executados de forma digital, autônoma e descentralizada, dando origem ao conceito de contratos inteligentes. Um contrato inteligente pode ser escrito numa linguagem de programação, para então ser armazenado e executado numa* Blockchain*. A execução de um contrato inteligente então ocorre de forma automática sem o envolvimento de terceiros conforme os termos firmados. Nesse cenário, métodos de verificação e ferramentas automáticas são de extrema importância para garantir as propriedades desses contratos. Dessa forma, este trabalho pretende propor técnicas e ferramentas para a verificação de contratos inteligentes.*

Palavras-chave: *Blockchain*, Contratos inteligentes, Ethereum, Solidity.

1. Introdução

Os contratos inteligentes[1] [5] surgiram com a tecnologia *Blockchain* [7], inicialmente concebida como um sistema criptográfico para armazenar dados. Posteriormente, Satoshi Nakamoto [11] introduziu o conceito de criptomoeda mostrando o potencial da tecnologia para sistemas peer-to-peer. As *Blockchains* têm a função de armazenar informações, como no caso do *Bitcoin*, dados de transações da criptomoeda, ou como em plataformas de propósito geral, onde informações e transações mais complexas podem ser tratadas. Uma Blockchain pode ser definida pela composição de diversos blocos de dados, ligados uns aos outros, com as informações desejadas. Essa disposição da Blockchain é garantida pelos códigos *hash* [13], compostos de símbolos para representar os dados armazenados. . Uma das vantagens dos contratos inteligentes é seu funcionamento descentralizado, onde sua execução ocorre sem o envolvimento de terceiros, ainda assim garantindo a segurança das informações e transações realizadas na *Blockchain*. Além disso, os contratos inteligentes evitam os processos burocráticos que envolvem a criação de um contrato convencional e diminuem os custos associados a sua construção, já que tais contratos são concebidos basicamente por programadores com o apoio de advogados.

Em plataformas mais gerais, como o Ethereum [3], os contratos inteligentes podem implementar contratos legais mais complexos, não apenas transações sobre criptomoedas como é o caso do *Bitcoin*. A Solidity [1] é uma dessas linguagens de programação, baseada em C++, Python e JavaScript, que implementa contratos inteligentes para serem executados automaticamente no Ethereum.

A escrita de um contrato depende de diversas cláusulas ou condições que devem ser cumpridas antes que este possa ser executado na prática. Pelo fato de um contrato inteligente ser escrito em linguagem de programação para uma Blockchain, o seu entendimento, de uma forma geral, ainda permanece de difícil análise, exigindo um suporte técnico e automático para auxiliar o processo de verificação e correção. A ausência de métodos e ferramentas automáticas abre a possibilidade de brechas em tais contratos, resultando em contratos com execuções contraditórias e/ou com resultados indesejados. Por isso, nesse processo é desejável que o programador possua um método de verificação ou construção para que o contrato seja concebido sem falhas, ou que sua execução não traga resultados inesperados, onde alguma das partes envolvidas venha ser prejudicada.

Essas situações podem ser evitadas com uma verificação mais precisa através de ferramentas que visam minimizar tais problemas. A ferramenta proposta em [9], por exemplo, tem o objetivo de transformar um contrato expresso por um autômato finito num contrato inteligente em Solidity. Já em outro trabalho [14], um método de verificação usando Redes de Petri é proposto de forma a garantir a segurança de um contrato inteligente antes de depositá-lo na Blockchain. Este trabalho então pretende oferecer alternativas práticas para verificar contratos inteligentes de maneira mais automática, sem perdas de precisão usando o apoio de mecanismos formais.

2. Fundamentação Teórico-Metodológica e Estado da Arte

Essa seção descreve os conceitos fundamentais que serão abordados neste trabalho, tais como as blockchains, os contratos inteligentes, a plataforma Ethereum e a linguagem de

[1] do inglês, *smart contracts*

programação associada para contratos inteligentes Solidity, bem como algumas direções de trabalhos na área de verificação de contratos.

2.1. Blockchain

A *Blockchain* foi inicialmente proposto por Stuart Haber e W. Scott Stornetta em 1991 [7], era um sistema de criptografia que armazenava os dados em o que é hoje conhecido como *blockchain*.

A tecnologia só foi se tornar mais conhecida em 2008 quando criou o primeiro conceito do que seria um *Blockchain* e em 2009 publicaria seu whitepaper [11] demonstrando como a tecnologia poderia ser usada para criação de sistemas peer-to-peer que mais tarde resultaria na criação do bitcoin.

A *blockchain* funciona como um histórico onde ficam armazenados os históricos de transações de um banco, em casos como o bitcoin ficam salvos as transações realizadas com o bitcoin.

O banco funciona mantendo diversos blocos e em cada bloco ficam registradas as transações aceitas pela rede. Cada bloco possui uma *hash*, hash seria uma cadeia de caracteres única que possui um tamanho padrão gerado a partir de uma função que criptografa um grupo de dados.

Cada bloco representa um conjunto de dados que a *blockchain* armazena, os conteúdos do bloco variam com cada sistema, o bitcoin mantem um bloco bem simples que possui o *timestamp* da transação, a chave pública do dono atual da criptomoeda, uma assinatura feita com a chave privada do usuário anterior, com essa assinatura sendo possível verificar se a transação é válida usando a chave pública do bloco anterior e o valor *hash* do bloco.

A *hash* de um bloco é feita a partir da *hash* do bloco anterior, os dados das transações armazenadas no bloco e um nonce, um valor qualquer. Para a geração de um bloco considerado valido para a blockchain cada rede decide em uma condição para o formato da hash, normalmente algo como um certo número de zeros no começo da blockchain. Com a geração da hash do bloco sendo dependente tanto do bloco anterior como de um valor arbitrário para um hash considerada válida torna impossível alterar o valor de uma hash em qualquer bloco sem tornar os blocos seguintes inválidos por consequência garantindo a segurança na imutabilidade dos dados na blockchain.

Os blocos são verificados pelos usuários da rede blockchain, e para garantir a segurança a blockchain pode usar alguns métodos, o bitcoin utiliza o *proof-of-work*, esse método faz com que os membros da rede que verificam os blocos precisem criar uma hash válida para tal bloco. Usando o *proof-of-work* o sistema seleciona a primeira hash válida criada e recompensa o usuário com a criptomoeda.

O Ethereum diferente do bitcoin utiliza o *proof-of-stake*, nesse método diferente do *proof-of-work* a hash não calculado pelo usuário mais rápido, os usuários depositam uma quantidade de criptomoeda na rede para então o sistema selecionar aleatoriamente algum membro da blockchain para verificar o novo bloco da cadeia, caos o usuário tente aprovar um bloco invalido as criptomoedas depositadas pelo usuário são tomadas. Usuário que depositam mais moedas na rede tem mais chance de serem escolhidos para criarem o bloco, pois se forem pegos tem mais moedas que podem ser perdidas e, portanto, são

mais confiáveis.

O sistema *proof-of-stake* não é perfeito, um dos problemas que são questionados é o fato de a rede recompensar com mais frequência usuário com mais criptomoedas gerando uma espécie de monopólio nas redes com esse sistema, no presente momento o sistema *proof-of-stake* ainda apresenta mais riscos que o *proof-of-work*, mais como vimos antes sistemas como bitcoin que usam *proof-of-work* apresentam problemas de escalabilidade onde o consumo enérgico do usuários competindo se tornam um problema e por esse motivo *proof-of-stake* ainda precisa ser estudado para entender seus benefícios e corrigir seus riscos.

2.2. Contratos Inteligentes

Os contratos inteligentes, inicialmente proposto por Nick Szabo [12], foram desenvolvidos com o objetivo de garantir que acordos pudessem ser executados de forma independente, sem o envolvimento de um agente certificador. A ausência de um agente certificador é a diferença básica entre um contrato inteligente de um contrato convencional. Por isso, uma das principais vantagens dos contratos inteligentes serem executados numa Blockchain é, justamente, promover maior segurança para os envolvidos e imutabilidade sobre a rede. Entretanto, o uso de contratos inteligentes ainda assim possui muitas limitações, como por exemplo, a necessidade de treinamento específico para que estes contratos possam ser devidamente implementados através de uma linguagem de programação adequada.

Uma das possibilidades de implementação prática de um contrato inteligente é através da linguagem Solidity [1] para serem executados na plataforma Ethereum [3].

2.2.1. Solidity

A linguagem de programação Solidity [1], fortemente influenciada por linguagens como C++, Python e JavaScript, é uma linguagem orientada a objetos criada com o objetivo de escrever contratos inteligentes no ambiente Ethereum. A Figura 1 ilustra um exemplo simples de um contrato inteligente em Solidity, onde a primeira linha define a versão utilizada. O contrato ainda declara uma variável storedData do tipo unsigned integer e duas funções. A primeira função “set” define o valor de storedData, enquanto a função “get” retorna o valor de storedData.

Na Figura 2 observa-se a similaridade de um contrato em Solidity com as variáveis e funções de programas em Java. Em Solidity, por exemplo, a palavra chave “public” permite que funções sejam chamadas por funções de outros contratos, implementados como classes distintas. Outra semelhança é a presença de construtores que são funções executadas apenas durante a criação do contrato. Note que no exemplo apresentado a função “mint” realiza um envio de moedas recém criadas para algum endereço. Para garantir que apenas o proprietário do contrato seja capaz de chamar essa função, a verificação “require(msg.sender == minter)” é realizada.

```
pragma solidity >=0.4.0 <0.6.0;

contract SimpleStorage {
    uint storedData;

    function set(uint x) public {
        storedData = x;
    }

    function get() public view returns (uint) {
        return storedData;
    }
}
```

Figura 1. Exemplo simples de contrato inteligente [1]

2.2.2. Ethereum

A plataforma Ethereum [3] foi desenvolvida com objetivo de dar mais liberdade para a escrita de contratos inteligentes, permitindo com que fosse possível criar contratos mais complexos que em outras plataformas, com mais do que apenas transações monetárias.

Parar criar um ambiente que permita a existência de contratos inteligentes mais complexo o Ethereum possui algumas diferenças com relação a outras plataformas, como o Bitcoin. No Ethereum existem entidades chamadas "accounts" ou contas, que armazenam valores referentes a quantidade de *ether*, a criptomoeda utilizada pelo *Ethereum*, que cada parte possui dentro da rede. As contas podem ser divididas em contas externas e de contrato. As contas externas não possuem código associado, mas podem enviar mensagens criando contratos e assinando transações. Já as contas de contrato possuem código associado, ou seja, as linhas de código que representam o contrato inteligente, e quando recebem uma mensagem executam seu código para então enviar uma mensagem ou criar novos contratos [3].

As mensagens enviadas pelas contas no Ethereum funcionam de forma similar com transações no Bitcoin, no qual as transações armazenam apenas o valor da transação e as chaves de seu antigo e novo dono. Já, no Ethereum as mensagens também possuem as chaves de reconhecimento do remetente e do destinatário, mas também contem um valor de ether, gás e um tipo de retorno.

O Ethereum roda em uma máquina virtual chamada Ethereum Virtual Machine ou EVM. Nesse ambiente são criados os contratos inteligentes que funcionam como transações similares ao Bitcoin. Porém, nesse ambiente condições ou operações podem ser especificadas para que uma transação seja executada, tal como a data para uma transferência de recursos seja executada. O Ethereum cobra um valor (Wei ou ether) para executar uma transação toda vez que um contrato precisa realizar uma operação. Esses valores podem ser recarregados durante o tempo de vida do contrato. Porém, se não houver um valor suficiente para a execução de uma transação antes do fim do contrato, todas as alterações realizadas são revertidas. De outra forma, caso sobre algum valor após as transações do contrato, a parte envolvida recebe de volta o valor excedente. Após a execução do contrato, uma função é chamada para finalizá-lo para então salvar as transações realizadas de forma permanente.

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.4;

contract Coin {
    // The keyword "public" makes variables
    // accessible from other contracts
    address public minter;
    mapping (address => uint) public balances;

    // Events allow clients to react to specific
    // contract changes you declare
    event Sent(address from, address to, uint amount);

    // Constructor code is only run when the contract
    // is created
    constructor() {
        minter = msg.sender;
    }

    // Sends an amount of newly created coins to an address
    // Can only be called by the contract creator
    function mint(address receiver, uint amount) public {
        require(msg.sender == minter);
        balances[receiver] += amount;
    }

    // Errors allow you to provide information about
    // why an operation failed. They are returned
    // to the caller of the function.
    error InsufficientBalance(uint requested, uint available);

    // Sends an amount of existing coins
    // from any caller to an address
    function send(address receiver, uint amount) public {
        if (amount > balances[msg.sender])
            revert InsufficientBalance({
                requested: amount,
                available: balances[msg.sender]
            });

        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        emit Sent(msg.sender, receiver, amount);
    }
}
```

Figura 2. Outro exemplo de contrato inteligente [1]

2.3. Verificação de Contratos

Para a correção de contratos inteligentes é necessário que se tenha um sistema ou processo que permita a transformação desse contrato inteligente em algo mais próximo da linguagem natural de contratos convencionais ou algo de mais fácil compreensão que linguagem de programação como autômatos [6].

Uma das alternativas para esse processo seria a criação de um método para extrair de um contrato convencional e suas condições, transformando-o em um lista de mais fácil compreensão [2]. Em seguida, o contrato deveria ser transformado num modelo mais preciso, que permitisse uma forma clara de expressar as condições do contrato. Nesse sentido, seria possível verificar se as condições de um contrato inteligente estão de acordo com as regras pretendidas, sem contradições ou inconsistências.

Uma das formas de se representar contratos de maneira precisa é através da RCL [2], que permite a especificação de contratos multilaterais. Assim, um alternativa para se verificar contratos inteligentes seria propor uma transformação de contratos inteligentes para contratos em RCL, para então se utilizar de uma ferramenta prática de verificação [4].

Outro trabalho [8], autômatos finitos foram aplicados para permitir uma análise das condições e interações dentro de um contrato. O autômato permite a verificação de inconsistências no contrato, garantindo o seu funcionamento. Entretanto esse método possui falhas, pois o sistema do contratos inteligentes funciona em um ambiente distribuído, podendo ocorrer uma quantidade alta de eventos em qualquer ordem.

Já em outra vertente, o modelo de Redes de Petri [14] foi proposto para modelar contratos inteligentes em Solidity. A ideia é transformar contratos inteligentes em modelos de Redes de Petri para daí realizar uma verificação mais precisa, buscando possíveis conflitos. Um apoio nesse direção seria uma implementação para realizar essa transformação de forma mais automatizada.

3. Objetivos

O objetivo desse projeto é pesquisar, agrupar e analisar métodos [9] [14] e ferramentas de verificação para contratos inteligentes [10]. A ideia é avaliar como tais contratos se comportam na linguagem Solidity, quais são as ferramentas utilizadas atualmente com esse objetivo e de que maneira essas ferramentas poderiam ser aprimoradas.

Os objetivos específicos do projeto são listados como segue:

1. Estudar como modelar e especificar contratos inteligentes de forma precisa.
2. Pesquisar sobre as possíveis transformações entre contratos legais e inteligentes.
3. Estudar e avaliar as técnicas de verificação existentes para contratos inteligentes.
4. Propor novas estratégias para uma verificação precisa de contratos inteligentes.
5. Implementar e testar os métodos propostos.
6. Realizar experimentos práticos.

4. Procedimentos metodológicos/Métodos e Técnicas

O desenvolvimento deste trabalho seguirá um conjunto de atividades estabelecidas com o intuito de alcançar os objetivos propostos. As atividades estão previstas para serem realizadas durante o período planejado no cronograma da Tabela 1 e descritas abaixo:

1. Revisão bibliográfica sobre as blockchain e contratos inteligentes;
2. Estudo e levantamento bibliográfico dos métodos de verificação de contratos inteligentes;
3. Estudo aprofundado de plataformas com propósito geral para contratos inteligentes, como o Ethereum, e a linguagem de implementação desses contratos, como a Solidity;
4. Estudo de falhas ou deficiências mais comuns em contratos inteligentes;
5. Análise e avaliação dos métodos para suprir as deficiências de ferramentas atuais;
6. Elaboração de mecanismos para garantir contratos inteligentes corretos;
7. Implementação dessa proposta para verificar contratos inteligentes;
8. Experimentos práticos e/ou estudo de caso envolvendo a proposta desenvolvida;
9. Divulgação dos resultados através de publicações.

5. Contribuições e/ou Resultados esperados

Entre os resultados esperados dessa Tese de Conclusão de Curso estão: os estudos aprofundados sobre os conceitos de blockchain e contratos inteligentes; os estudos e avaliação

Atividade	2022				2023				
	Set	Out	Nov	Dez	Jan	Fev	Mar	Abr	Mai
1	•								
2	•								
3		•	•						
4			•	•					
5				•	•				
6					•	•	•		
7						•	•	•	
8							•	•	•

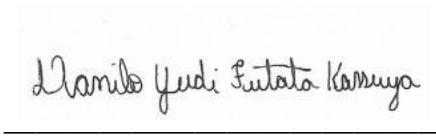
Tabela 1. Cronograma de execução

de técnicas de verificação desses contratos; a elaboração de um método preciso para garantir as propriedades sobre contratos inteligentes; o estudo de ferramentas existentes para tal tarefa; bem como a implementação, mesmo que parcial, dessa proposta.

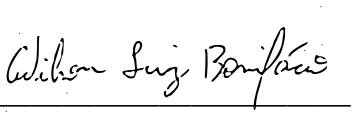
Espera-se que métodos e ferramentas existentes sejam avaliadas, bem como sua possível aplicação prática em contratos reais. Também se espera que seja gerada uma ferramenta de apoio computacional para detectar erros ou resultados inesperados em contratos inteligentes, antes que esses sejam efetivamente implementados numa plataforma, evitando que alguma das partes envolvidas no contrato seja prejudicada.

6. Espaço para assinaturas

Londrina, *12/09/2022*.



Aluno



Orientador

Referências

[1] Solidity, 2016. `https://docs.soliditylang.org/en/v0.8.13/index.html#`, Last accessed on 2022-07-25;.

[2] Adilson Luiz Bonifacio and Wellington Aparecido Della Mura. Automatically running experiments on checking multi-party contracts. *Artif. Intell. Law*, 29(3):7, sep 2021.

[3] Vitalik Buterin. A next generation smart contract & decentralized application platform. 2015.

[4] Wellington Aparecido Della Mura and Adilson Luiz Bonifacio. Devising a conflict detection method for multi-party contracts. In *2015 34th International Conference of the Chilean Computer Science Society (SCCC)*, pages 1–6, 2015.

[5] Guido Governatori, Florian Idelberger, Zoran Milosevic, Regis Riveret, Giovanni Sartor, and Xiwei Xu. On legal contracts, imperative and declarative smart contracts, and blockchain systems. *Artificial Intelligence and Law*, 26, 12 2018.

[6] Guido Governatori, Florian Idelberger, Zoran Milosevic, Regis Riveret, Giovanni Sartor, and Xiwei Xu. On legal contracts, imperative and declarative smart contracts, and blockchain systems. *Artificial Intelligence and Law*, 26:2, 12 2018.

[7] Gwyneth Iredale. History of blockchain technology: A detailed guide, 2020.

[8] Anastasia Mavridou and Aron Laszka. Designing secure ethereum smart contracts: A finite state machine based approach, 2017.

[9] Anastasia Mavridou and Aron Laszka. Designing secure ethereum smart contracts: A finite state machine based approach. In *Financial Cryptography*, 2018.

[10] Anastasia Mavridou and Aron Laszka. Tool demonstration: Fsolidm for designing secure ethereum smart contracts. In Lujo Bauer and Ralf Küsters, editors, *POST*, volume 10804 of *Lecture Notes in Computer Science*, pages 270–277. Springer, 2018.

[11] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. May 2009.

[12] Nick Szabo. Formalizing and securing relationships on public networks. *First Monday*, 2(9), Sep. 1997.

[13] Dejan Vujičić, Dijana Jagodic, and Siniša Ranđić. Blockchain technology, bitcoin, and ethereum: A brief overview. pages 1–6, 03 2018.

[14] Nejc Zupan, Prabhakaran Kasinathan, Jorge Cuellar, and Markus Sauer. *Secure Smart Contract Generation Based on Petri Nets*, pages 73–98. 01 2020.