Conversão da linguagem Solidity para redes de Petri visando a verificação de contratos inteligentes

Felipe Ariji[1], **Adilson Luiz Bonifácio**[1]

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

felipe.s.ariji@uel.br, bonifacio@uel.br

Abstract. *Even with the popularization blockchain and smart contracts, improvements in verifying this system still neded. This work seeks to develop a conversion of the Solidity language, used on the Etherium platform, to the formal Petri net model, which turns possible to use techniques already suggested in the literature for that validation.*

Resumo. *Mesmo com a popularização dos assuntos relacionados a blockchain e contratos inteligentes, melhorias ainda são necessárias na verificação desse tipo de sistema. Com tal foco, esse trabalho busca desenvolver uma conversão da linguagem Solidity utilizada na plataforma Etherium para o modelo formal de redes de Petri, sobre a qual é possível utilizar técnicas já sugeridas na literatura para validação.*

1. Introdução

Este trabalho visa aumentar a confiabilidade nos contratos inteligentes criados na plataforma *Blockchain* de propósito geral Ethereum(ETH), os quais são escritos na linguagem de programção de alto nível Turing completa *Solidity*.

Isso será efetivado com o desenvolvimento de um algoritmo de tradução dos códigos base desses documentos para a representação formal de redes de Petri coloridas. Feito isso as pesquisas futuras poderão utilizar os métodos de verificação previamente propostos para o formalismo, dessa forma certificando a confiabilidade dos acordos.

Não se pode duvidar que o conceito de contrato, é tão antigo quanto o próprio ser humano, pois nasceu a partir do momento em que as pessoas passaram a se relacionar e viver em sociedade. A palavra sociedade em si remete a ideia de contrato, da união de partes com interesses em comum [Tartuce 2016].

Como a citação anterior enfatiza, o contrato é algo enraizado e vital nas relações humana. De tal modo que apesar da evolução socio-cultural e tecnológica ainda se mostra necessário a utilização dele para a formalização de acordos entre duas ou mais pessoas. Porém como qualquer elemento social, esse documento deve se adequar aos novos tempos e atualizar a forma como é organizado, garantido e efetivado.

A partir da definição da cripto-moeda *Bitcoin* [Nakamoto 2008], uma nova forma de registrar transações entre pessoas foi criada, a *Blockchain*, a qual não necessitaria de um intermediário direto, o que simplifica e imparcializa o processo.

Apesar de a proposta inicial da tecnologia ser trocas financeira, uma aplicação é o uso para contratos inteligentes. Tais documentos eletrônicos, além de não necessitar

do intermédio de um terceiro, garantem a execução automática dos acordos, pois são códigos autoexecutáveis que podem ser programados para que, quando alguma condição seja atendida, realizem as ações especificadas.

Embora as vantagens sejam evidentes, o uso de contratos inteligentes ainda necessita de desenvolvimento de novos técnicas de segurança, pois costumam controlar movimentações monetárias consideráveis [Zupan et al. 2020]. Um exemplo das possíveis vulnerabilidade desse método é o ataque hacker de 2016 ao DAO (Decentralized Autonomous Organization), o qual era implementado em Ethereum [Dhillon et al. 2017].

Para que essa nova modalidade de contrato seja mais amplamente utilizada e revolucione o mundo de maneira geral, é essencial que ela seja ao menos tão confiável quanto a sua contraparte tradicional. Tal necessidade de segurança é o que motiva a criação desse artigo.

O restante desse projeto está organizado da seguinte forma. A Seção 2 descreve as bases teóricas essenciais para o seu desenvolvimento, como contratos, *Blockchain*, *Ethereum*, *Solidity* e redes de Petri. Já os objetivos dessa proposta estão descritos na Seção 3. Os procedimentos e o cronograma de estudo, escrita e desenvolvimento são apresentados na Seção 4. A Seção 5 lista algumas das contribuições esperadas, com ênfase na criação do algoritmo de tradução de *Solidity* para redes de Petri. A proposta termina com uma lista de referências usadas na elaboração desse projeto.

2. Fundamentação

Esta seção aborda os temas de contratos, blockchain, plataforma Ethereum e sua linguagem Solidity e redes de Petri. Os quais são necessários ao desenvolvimento do projeto final dessa tese.

2.1. Contratos

Contrato pode ser definido como um negócio jurídico bilateral ou plurilateral, o qual visa a criação, modificação ou extinção de direitos e deveres. Sendo, desse modo, uma convenção, criada pelo acordo de vontades e por outros fatores acessórios [Tartuce 2016].

Basicamente os contratos são documentos que gerenciam um acordo entre múltiplas pessoas ou empresas, a fim de garantir que todas as partes cumpram suas promessas. Eles podem possuir diferentes naturezas, desde acordos de negócios, envolvendo compra e vendas de ações de corporações, até acordos entre pessoas civis como o casamento [Szabo 1996].

2.1.1. Contratos eletrônicos

Devido às novas tecnologias, os contratos e seus processos evoluíram de forma a facilitar e oferecer melhorias aos usuários, dando origem aos contratos eletrônicos ou *e-contratos* [Governatori et al. 2018]. Esses se caracterizam por serem criados e administrados computacionalmente.

Os contratos eletrônicos são documentos análogos à sua versão em papel, possuindo inclusive a mesma validade legal, mas criados, acompanhados e assinados através

da rede de internet. Eles podem ser criados utilizando ferramentas próprias de geração de contratos, e-mail ou outros métodos computacionais [Alden 2022].

Apesar de a primeira vista não demonstrar o mesmo formalismo de uma assinatura em um documento físico, ações como marcar a opção de concordância com os termos de serviço de algum site e utilizar um *token* de assinatura eletrônica utilizando seus dados, possuem a validade legal que a versão tradicional de demonstração de consentimento, quando respeitando os processos burrocráticos [Ferreira 2023].

Uma evolução do contrato eletrônico que surge da associação desse conceito à *Blockchain* é o contrato inteligente, o qual será abordado a seguir.

2.1.2. Contratos inteligentes

A ideia central dos contratos inteligentes é inserir as clausulas dentro de *hardwares* e *softwares* cotidianos, de forma a informar as movimentações contratuais à dois ou mais envolvidos de forma síncrona. Como um antepassado desse conceito é possível mencionar as máquinas de venda automáticas, que utilizam leitores de cartão de crédito, notas e moedas e distribuidores mecânicos para realizar uma compra sem o contato direto entre comprador e vendedor [Szabo 1996].

Contratos inteligentes costumam ser escritos para garantir o devido cumprimento de suas clausulas e seu possível cancelamento, caso necessário, sem a necessidade de terceiros, substituindo o mediador pela *Blockchain*. Para que isso ocorra esses códigos seguem dois princípios básicos. Primeiro, as condições escritas no SC não podem ser alteradas nem mesmo pelo proprietário após publicá-lo no *blockchain*. Segundo, as condições do SC são impostas pelo código escrito e pelo mecanismo de consenso do sistema blockchain subjacente. [Zupan et al. 2020].

Em sua estrutura básica um contrato inteligente é um código, escrito em uma linguagem de alto nível, o qual é implantado na *Blockchain*, para que, quando as clausulas forem cumpridas, seja executado e seus resultados sejam verificados pelos elos da rede *Blockchain*. [Zupan et al. 2020]

2.2. Blockchain

A *Blockchain*, como o nome sugere, é formada por blocos de informação formando uma corrente, ou seja, uma sequência organizada com encadeamento simples sequencial. Ela tem como objetivo manter a integridade e imutabilidade dos dados que estão dentro dos blocos.

O seu conceito inicial foi proposto por Stuart Haber e W. Scott Stornetta 1991. Inicialmente era numa cadeia de blocos criptograficamente protegida, através da qual ninguém poderia adulterar os registos de data e hora dos documentos, conceito que se mantem até hoje. Porém somente em 2008, com a proposta do *Bitcoin* [Nakamoto 2008], a *Blockchain* se tornou o que conhecemos hoje e se popularizou [Iredale 2020].

Cada bloco dessa rede é vinculado a outro usando um *hash* criptográfico do bloco anterior, um registro de data e hora e dados de transação [Zupan et al. 2020]. Através desse *hash* é possível verificar se um bloco foi adulterado, pois caso isso ocorra o apontamento dos blocos adjacentes ficará incorreta. Além disso os famosos mineradores de

cripto-moeda, os quais nada mais são do que usuários verificadores de blocos, que costumam receber cripto-moedas como recompensa, sendo essenciais para manter a integridade do sistema.

O bitcoin usa o método *proof-of-work*, onde os membros da rede competem para ser o primeiro a criar uma *hash* válida para um novo bloco. O mais rápido define a chave e recebe a recompensa em cripto-moeda [Kassuya 2023].

Já o Ethereum utiliza o *proof-of-stake*, no qual o hash calculada aleatoriamente. Usuários que depositam mais moedas na rede possuem mais chance de serem escolhidos. Caso o usuário selecionado aprove um bloco inválido as cripto-moedas depositadas pelo usuário são tomadas. Esse mecanismo visa punir os que tentam burlar o sistema com blocos inválidos, pois se alguma trapaça é detectada esse usuário pode perder muito dinheiro. [Kassuya 2023]

A Figura 1 mostra o funcionamento da *Blockchain*, quando um usuário faz uma requisição de transação, um bloco é criado para representá-la, contendo os respectivos dados. Posteriormente esses conjunto será difundido pela rede e os nós já presentes nela validam-no, caso seja aprovado é adicionado à *Blockchain*.

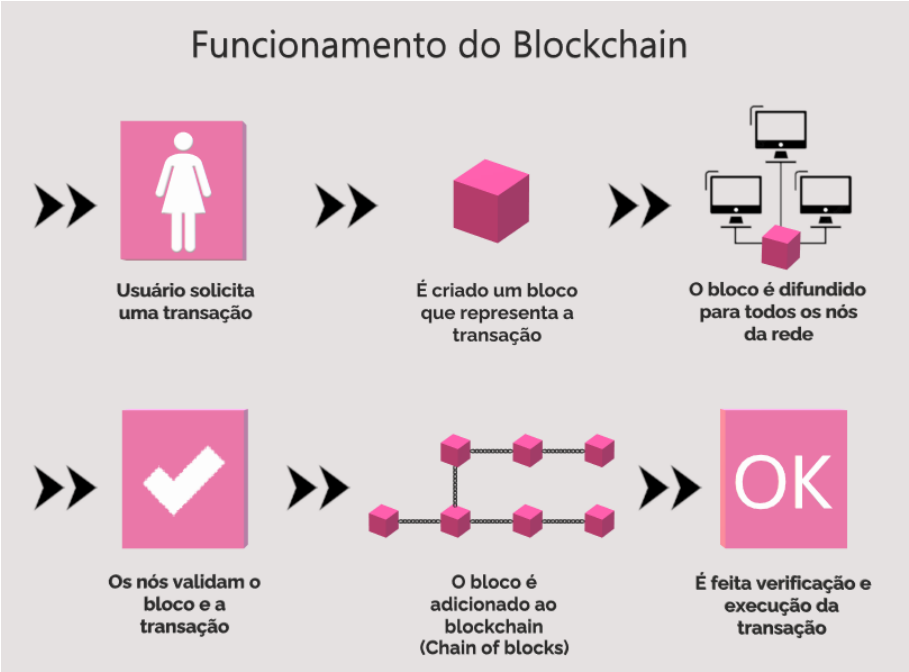

Figure 1. Como funciona o *Blockchain*[Ferreira 2023]

Alguns exemplos de implementações e plataformas de *Blockchain*:

1. ***Ethereum*** (Plataforma utilizada nesse projeto);
2. *Binance Smart Chain*;
3. *Polkadot*;
4. *Cardano*;
5. *Tezos*;
6. *EOS*;
7. *Hyperledger*;

8. *Hyperledger Sawtooth*;
9. *Fabric*;

2.3. Uma Plataforma Blockchain

Essa seção descreve a plataforma *Blockchain* a qual será utilizada como base para esse projeto, a *Ethereum*, juntamente com a linguagem alto nível que ela utiliza para escrever contratos inteligentes, a *Solidity*.

2.3.1. Ethereum

A *Ethereum* foi criada em 2013, por Vitalik Buterin, com a ideia de usar a *Blockchain* para diversas funções, e não somente para a criação e movimentação de cripto-moedas. A plataforma foi lançada oficialmente em 2015 na versão *Frontier*, *software* de contrato inteligente que aprimorou o ecossistema da rede [Bitso 2022].

Multiplicador	Nome
10^0	*Wei*
10^{12}	*Szabo*
10^{15}	*Finney*
10^{18}	*Ether*

Table 1. As subunidades do *Ether* e seus pesos [Wood et al. 2014]

Para incentivar as verificações necessárias ao funcionamento da rede, é necessário uma cripto-moeda como recompensa. A cripto-moeda da *Ethereum* se chama*Ether* ou *ETH*. Devido ao altíssimo valor monetário de uma única dessa cripto-moeda, existem subunidades dela, como mostra a Tabela 1, sendo a principal delas a *Wei*, da qual 10^{18} unidades valem 1 Ether [Wood et al. 2014].

O bloco na *Ethereum* é a coleção de peças relevantes de informação, conhecidas como cabeçalho do bloco, juntamente com informações correspondentes às transações compreendidas e um *ommer*, o conjunto de outros cabeçalhos de bloco que são conhecidos por terem pais equivalentes aos do bloco atual [Wood et al. 2014]. Em relação ao *Bitcoin*, a quantidade de dados transmitidos pela *Ethereum* é maior, de acordo com [Wood et al. 2014] esses dados são:

- ***parentHash***: Um *hash* de 256 *bits* do cabeçalho do bloco pai, em sua totalidade;
- ***ommersHash***: Um *hash* de 256 *bits* da parte da lista de *ommers* deste bloco;
- ***beneficiary***: Um endereço de 160 *bits* para o qual todas as taxas coletadas da mineração bem-sucedida deste bloco serão transferidas;
- ***stateRoot***: Um *hash* de 256 *bits* do nó raiz do estado testado, após todas as transações serem executadas e as finalizações aplicadas;
- ***transactionsRoot***: Um *hash* de 256 *bits* do nó raiz da estrutura trie preenchida com cada transação na parte da lista de transações do bloco;
- ***receiptsRoot***: Um *hash* de 256 *bits* do nó raiz da estrutura trie preenchida com os recibos de cada transação na parte da lista de transações do bloco;
- ***logsBloom***: Composto de informações indexáveis contidas em cada entrada de *log* do recebimento de cada transação na lista de transações;

- ***difficulty***: Um valor escalar correspondente ao nível de dificuldade deste bloco. Isso pode ser calculado a partir do nível de dificuldade do bloco anterior e do registro de data e hora;
- ***number***: Um valor escalar igual ao número de blocos ancestrais. O bloco origem tem um número zero;
- ***gasLimit***: Um valor escalar igual ao limite atual de gasto de gás por bloco;
- ***gasUsed***: Um valor escalar igual ao gás total usado em transações neste bloco;
- ***timestamp***: Um valor escalar igual à saída razoável do time() do Unix no início deste bloco;
- ***extraData***: Uma matriz de bytes arbitrária, de no máximo 32 bytes, contendo dados relevantes para este bloco;
- ***mixHash***: Um *hash* de 256 *bits* que, combinado com o *nonce*, prova que uma quantidade suficiente de computação foi realizada neste bloco;
- ***nonce***: Um valor de 64 *bits* que, combinado com o *mixhash*, prova que uma quantidade suficiente de computação foi realizada neste bloco;

No *Ethereum* as contas que compões os estados possuem um endereço de 20 bytes e uma lista de transições de estado, as quais são transferências de valores e informações diretas. Cada uma dessas contas possuem 4 campos: *nonce*, armazenamento, quantidade de *Ether* e contratos [Buterin et al. 2014].

A plataforma funciona utilizando uma máquina virtual nomeada com a sigla *EVM*, do inglês *Ethereum Virtual Machine*, um ambiente virtual descentralizado que garante a padronização de execução dos códigos dentro do sistema, facilitando manter a segurança dele [Otto-AA 2024].

Essa virtualização utiliza o parâmetro *gas* como forma de abstração da capacidade de processamento e armazenamento do equipamento físico que controla determinado nó da rede, levando isso em consideração ao distribuir as operações e não causar falhas por sobrecarga de *hardware* [Otto-AA 2024]. A Figura 2, demonstra a arquitetura computacional da EVM.

2.3.2. Solidity

A *Solidity* como dito anteriormente é uma linguagem de programação de alto nível, na qual os contratos inteligentes da plataforma *Ethereum* são escritos. Esses códigos, podem servir para votações, arrecadação de fundos, leilões cegos e são executados através da máquina virtual *Ethereum*.

A *Solidity* segue o paradigma orientado a objeto e utiliza chaveamento para definir os limites dos escopos do código. Além disso é estaticamente tipada, suporta herança, bibliotecas e tipos complexos definidos pelo usuário e esses são apenas alguns de seus recursos básicos. Sua criação foi influenciada por linguagens como *C++*, *Python* e *JavaScript* [team Ethereum.org 2023].

Na Figura 3 temos um exemplo simples de armazenamento de dados utilizando a linguagem Solidity. A seguir será explicado como o código opera e como cada comando funciona.

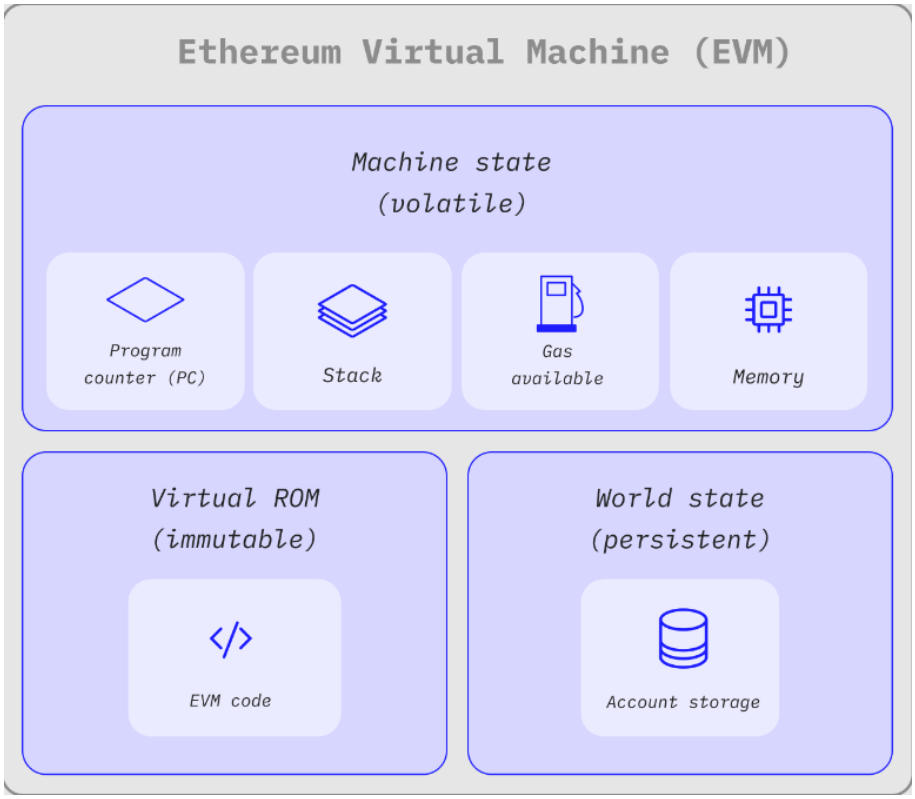

Figure 2. Arquitetura da *EVM*[Otto-AA 2024]

A primeira linha do código indica algumas instruções e restrições de execução do código, como quantas vezes deve ser executado e/ou para quais versões do *Solidity* ele é compatível. Isso é essencial em todo código dessa linguagem, uma vez que utilizar o código de maneira errada pode causar problemas de segurança. no exemplo da Figura 3, a restrição é que o código foi projetado para as versões desde a 0.4.16 até antes da 0.9.0.

A linha 3 do código está declarando o identificador, nesse caso *SimpleStorage*, do contrato inteligente, essa operação é análoga a definição de uma classe de objeto. A linha 4 define uma variável chamada *storedData*, a qual pertence ao tipo *uint (unsigned integer)*, ou seja um inteiro sem sinal positivo ou negativo.

Para finalizar o algoritmo, o bloco da linha 6 até a 8 define uma função *set* e o bloco da linha 10 até a 12 define uma função *get*, ambas relacionadas ao *storedData* e possuem funcionamento parecido com a suas contrapartes em outras linguagens orientadas a objeto.

2.4. O Formalismo Redes de Petri

Redes de Petri são representações formais, as quais foram apresentadas em 1962 e levam o nome de seu autor Carl Adam Petri. A motivação para a criação desse modelo foi a necessidade de representar conceitos, os quais eram difíceis ou impossíveis de serem demonstrados através de autômatos, como sistemas paralelos, concorrentes, assíncronos e não-determinísticos [Petri 1962].

```solidity
pragma solidity >=0.4.16 <0.9.0;

contract SimpleStorage {
    uint storedData;

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

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

Figure 3. Exemplo de código Solidity [team Ethereum.org 2023].

2.4.1. Redes de Petri Ordinárias

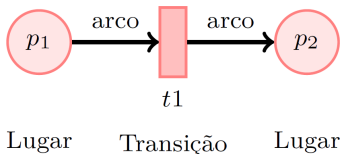

Figure 4. RdP simples

As Redes de Petri ordinárias também são conhecidas como Redes e Lugar/Transição, pois são constituídas por lugares, representados por círculos, transições, representadas por retângulos ou barras verticais e arcos, representados como setas. Os arcos conectam lugares a transições, ou o contrário, mas nunca podem ser conectados 2 elementos do mesmo tipo [Gehlot 2019]. Esse arranjo é demonstrado na Figura 4

- **Descrição 1:** De acordo com [Francês 2003], uma Rede de Petri pode ser formalmente definida por sendo **R**, tal que, R = (P, T, I, O, K). Onde P, T, I, O significam respectivamente *places*, *transitions*, *in*, *out*, traduzindo para o português, lugares, transições, entradas e saídas. Sendo representadas pelas formalidades:
    1. P = {p1, p2, ..., pn}, é um conjunto finito não vazio de lugares, da rede;
    2. T = {t1, t2, ..., tn}, é um conjunto finito não vazio de transições, da rede;
    3. I: T $\rightarrow$ P é o conjunto finito não vazio de arcos de entrada da rede;
    4. O: T $\rightarrow$ P é o conjunto finito não vazio de arcos de saída da rede;
    5. K: P $\rightarrow$ N é o conjunto possivelmente infinito de propriedades de cada lugar.

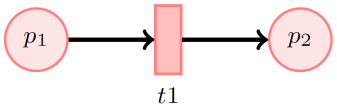

Figure 5. Exemplo de sequenciamento

Uma Rede de Petri permite uma análise mais detalhada e um melhor entendimento de um sistema modelado, permitindo assim identificar pontos de falha ou otimizar o seu

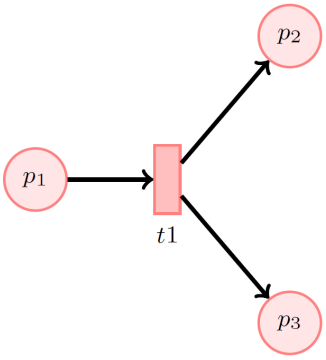

Figure 6. Exemplo de distribuição

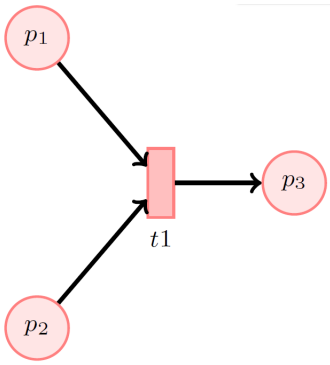

Figure 7. Exemplo de junção

funcionamento. Também vale ressaltar que uma rede mais complexa pode ser composta por redes mais básicas, assim como um código de programação. Tais redes básicas são definidas a seguir [Kassuya 2023]:

- **Sequenciamento:** a rede começa em um lugar, que representa uma condição, seguido pela transição que representa uma ação, que quando disparada avança para um novo lugar. Veja a Figura 5 [Kassuya 2023].
- **Distribuição:** similar a um sequenciamento, mas o disparo da transição, ou seja a realização da ação, dá origem a processos paralelos, ou seja avança para dois lugares de saída simultaneamente. Veja a Figura 6 [Kassuya 2023].
- **Junção:** processo inverso à distribuição, onde os lugares de entrada devem habilitar simultaneamente a transição, gerando recursos apenas no lugar de saída. Veja a Figura 7 [Kassuya 2023].
- **Não-determinismo:** Um único lugar de entrada possui uma marcação para habilitar duas transições, porém suficiente para disparar apenas uma delas. Veja a Figura 8 [Kassuya 2023].

A Figura 9 representa um sistema R de envio e recebimento de pacotes, nesse caso a definição formal R = (P, T, I, O, K), tal que:

1. P = {Enviar, Pacotes recebidos, Proximo envio, A, B, C, D};
2. T = {Enviar Pacotes, Transmitir pacote, Receber pacote, Transmitir ACK, Receber ACK};
3. I: {I(Enviar Pacotes) = [Enviar, Próximo envio], I(Transmitir pacote) = [A], I(Receber pacote) = [B], I(Transmitir ACK) = [C], I(Receber ACK) = [D]};

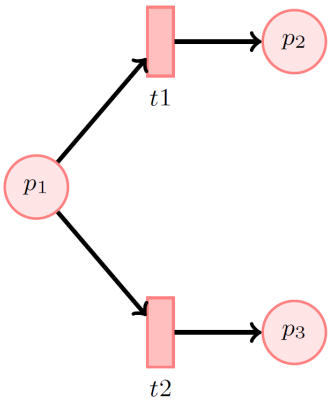

Figure 8. Exemplo de Não-determinismo

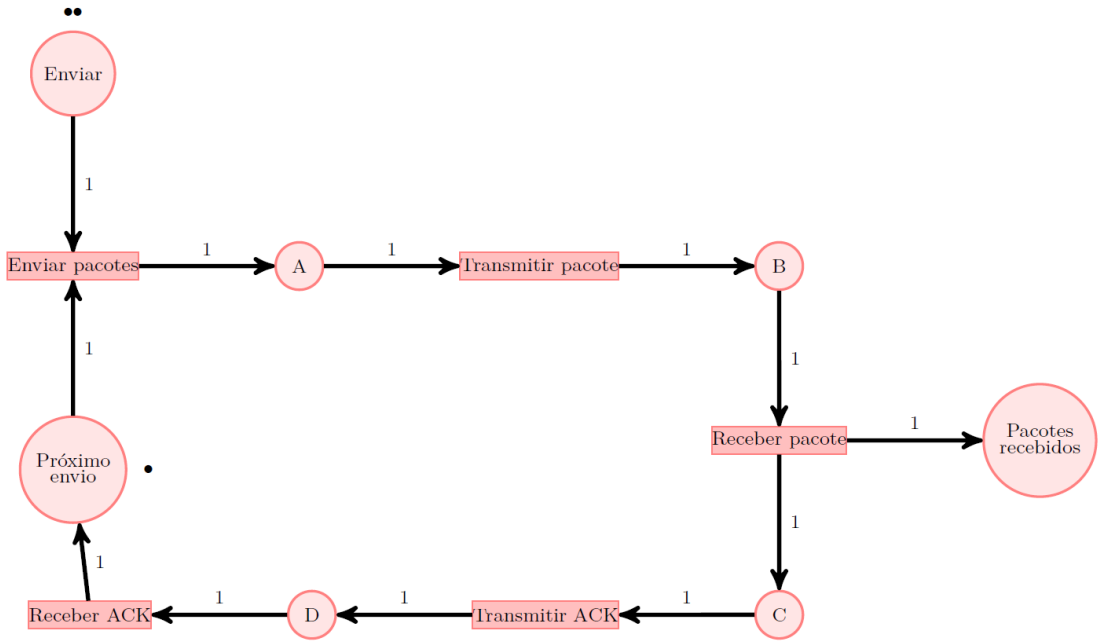

Figure 9. Exemplo de Rede de Petri Ordinária [Ferreira 2023]

4. O: {O(Enviar Pacotes) = [A], O(Transmitir pacote) = [B], O(Receber pacote) = [C, Pacotes recebidos], O(Transmitir ACK) = [D], O(Receber ACK) = [Próximo envio]};
5. K = {}

2.4.2. Redes de Petri Coloridas

As Redes de Petri Coloridas, conceituadas em 2009, são uma derivação da versão tradicional do formalismo com vocabulário e estendido e recursos adicionados visando aplicações de maior escala. Esse nome Coloridas vem do recurso mais importante delas a capacidade de usar cores para representar a atribuição de valores [Gehlot 2019].

- **Descrição 2:** uma Rede de Petri Colorida pode ser formalmente definida por

sendo **N**, tal que, N = (P, T, Σ, C, G, A, E, I). Sendo representadas pelas formalidades:

1. P = {p1, p2, ..., pn}, é um conjunto finito de lugares;
2. T = {t1, t2, ..., tn}, é um conjunto finito de transições;
3. Σ: é um conjunto finito de tipos não nulos chamados de conjuntos de cores;
4. $C : L \rightarrow \Sigma$ é uma função de atribuição de cores que associa cada local l a uma cor $C(l)$, onde $C(l) \subseteq \Sigma$ e $\bigcup_{l \in L} C(l) = \Sigma$;
5. $G : T \rightarrow expr$ é uma função de restrição que associa a cada transição t uma expressão booleana, onde $expr$ é o conjunto de todas as expressões possíveis.
6. $A \subseteq (L \times T) \cup (T \times L)$ é um conjunto finito de arcos;
7. $E : A \rightarrow expr$ é uma função de expressão de arco que associa a cada arco a uma expressão, onde $expr$ é o conjunto de todas as expressões possíveis
8. $I : L \rightarrow expr_{fechada}$ é a função que estabelece a configuração inicial dos locais em termos de expressões fechadas (ou seja, sem variáveis), onde $expr_{fechada}$ é o conjunto de todas as expressões fechadas;

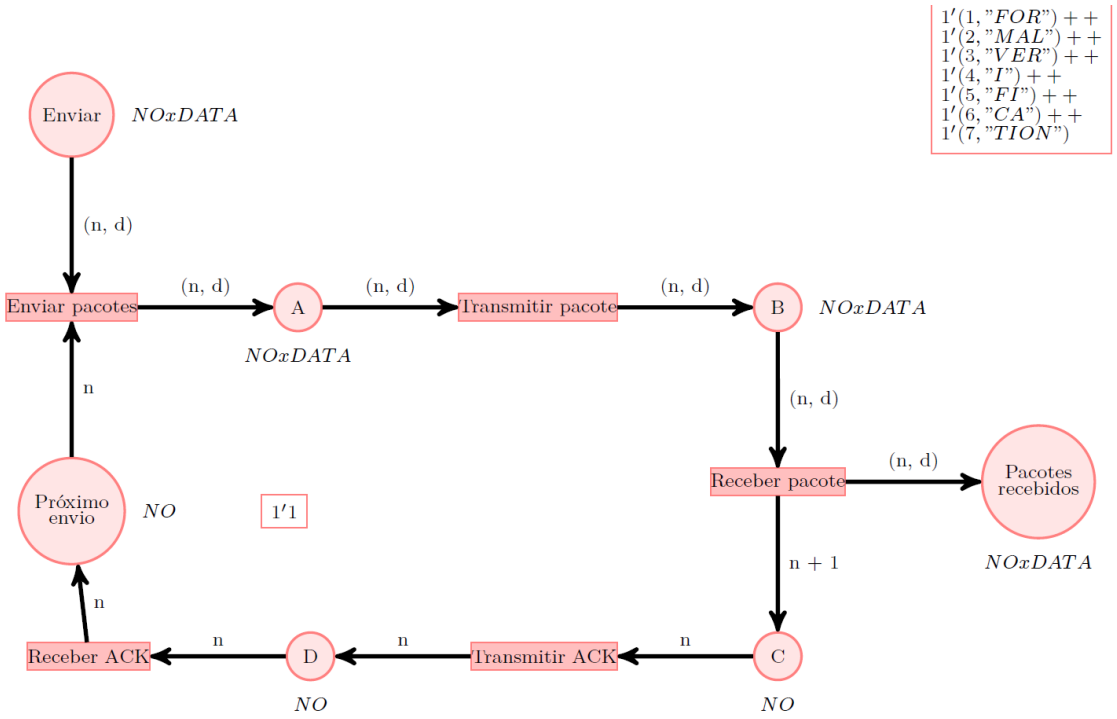

Figure 10. Exemplo de Rede de Petri Colorida [Ferreira 2023]

A Figura 10 representa uma Rede de Petri Colorida equivalente a Rede de Petri Ordinária da Figura 9, porém com o acréscimo dos elementos próprios desse tipo de representação, o que aumenta a complexidade e capacidade descritiva.

3. Objetivos

O objetivo desse projeto é criar um algoritmo capaz de traduzir códigos de contratos inteligentes na linguagem de alto nível *Solidity* para as redes de Petri coloridas, a fim de

utilizar as técnicas de verificação já pesquisadas para essa representação formal. E dessa forma aumentar a confiabilidade dos contratos inteligentes.

Os objetivos específicos do projeto são:

1. Estudar conceitos de contratos legais e inteligentes, *Blockchain*, *Ethereum*, *Solidity* e redes de Petri tradicionais e coloridas
2. Pesquisar sobre os desdobramentos de uma maior aplicação de contratos inteligentes no cotidiano
3. Estudar e avaliar técnicas de validação de contratos inteligentes
4. Propor uma arquitetura de teste para o algoritmo
5. Implementar um algoritmo que traduza códigos *Solidity* para redes de Petri
6. Testar o algoritmo desenvolvido ao longo dessa tese através de uma análise formal, utilizando uma gramatica que gere essa transformação. Desse modo as derivações da gramatica poderiam garantir uma transformação confiável.

4. Procedimentos metodológicos

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 2 e descritas abaixo:

1. Revisão bibliográfica sobre contratos tradicionais e inteligentes, *Blockchain* e *Ethereum*;
2. Estudo e levantamento bibliográfico de *Solidity* e redes de Petri tradicionais e coloridas;
3. Análise e avaliação de técnicas possíveis para a tradução de *Solidity* para redes de Petri coloridas;
4. Elaboração de mecanismos de verificação e teste de efetividade para o algoritmo a ser desenvolvido;
5. Implementação do algoritmo de tradução de *Solidity* para redes de Petri coloridas;
6. Teste do algoritmo implementado;
7. Divulgação dos resultados através de publicações.

Atividade	2024						2025	
	Jul	Ago	Set	Out	Nov	Dez	Jan	Fev
1	•							
2	•	•						
3		•	•					
4			•	•				
5				•	•	•	•	
6				•	•	•	•	
7								•

Table 2. Cronograma de execução

5. Contribuições e/ou Resultados esperados

Entre os resultados esperados desse projeto estão: os estudos aprofundados sobre contratos legais e inteligentes, *Blockchain*, *Ethereum*, *Solidity* e redes de Petri tradicionais

e coloridas; a proposta de um algoritmo capaz de transformar um contrato inteligente em *Solidity* para Redes de Petri; a implementação do sistema proposto; e avaliação da performance do programa com base em testes.

Espera-se que os métodos e ferramentas criadas no decorrer desse artigo possibilitem pesquisas futuras mais avançadas no campo de verificação de contratos inteligentes. Criando dessa forma um ambiente propício para o uso cotidiano dessa tecnologia, facilitando processos importantes.

References

[Alden 2022] Alden, J. (2022). Everything you need to know about econtracts. `https://www.contracts365.com/blog/everything-you-need-to-know-about-econtracts`. Accessed: 28/07/2024.

[Bitso 2022] Bitso, B. (2022). História da rede ethereum: Confira como surgiu a segunda maior blockchain do mundo! `https://blog.bitso.com/pt-br/criptomoedas/historia-da-rede-ethereum`. Accessed: 28/07/2024.

[Buterin et al. 2014] Buterin, V. et al. (2014). A next-generation smart contract and decentralized application platform. *white paper*, 3(37).

[Dhillon et al. 2017] Dhillon, V., Metcalf, D., Hooper, M., Dhillon, V., Metcalf, D., and Hooper, M. (2017). The dao hacked. *blockchain enabled applications: Understand the blockchain Ecosystem and How to Make it work for you*, pages 67–78.

[Ferreira 2023] Ferreira, T. S. (2023). Embusca de uma verificação ao mais rigorosa para contratos inteligentes. *Universidade Estadual de Londrina*.

[Francês 2003] Francês, C. R. L. (2003). Introdução às redes de petri. *Laboratório de Computação Aplicada, Universidade Federal do Pará*.

[Gehlot 2019] Gehlot, V. (2019). From petri nets to colored petri nets: A tutorial introduction to nets based formalism for modeling and simulation. In *2019 Winter Simulation Conference (WSC)*, pages 1519–1533. IEEE.

[Governatori et al. 2018] Governatori, G., Idelberger, F., Milosevic, Z., Riveret, R., Sartor, G., and Xu, X. (2018). On legal contracts, imperative and declarative smart contracts, and blockchain systems. *Artificial Intelligence and Law*, 26:377–409.

[Iredale 2020] Iredale, G. (2020). History of blockchain technology: A detailed guide. 101 blockchains. *Retrieved from*.

[Kassuya 2023] Kassuya, D. Y. F. (2023). Transformação entre contratos inteligentes e redes de petri. *Universidade Estadual de Londrina*.

[Nakamoto 2008] Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. Accessed: 2015-07-01.

[Otto-AA 2024] Otto-AA (2024). Ehtereum virtual machine (evm). `https://ethereum.org/en/developers/docs/evm/`. Accessed: 29/07/2024.

[Petri 1962] Petri, C. A. (1962). Kommunikation mit automaten. *Universitat Hamburg*.

[Szabo 1996] Szabo, N. (1996). Smart contracts: building blocks for digital markets. *EXTROPY: The Journal of Transhumanist Thought,(16)*, 18(2).

[Tartuce 2016] Tartuce, F. (2016). *Manual de direito civil*, volume único 6º edição. Editora Método.

[team Ethereum.org 2023] team Ethereum.org (2023). Solidity. `hhttps://docs.soliditylang.org/en/v0.8.26/index.html`. Accessed: 29/07/2024.

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

[Zupan et al. 2020] Zupan, N., Kasinathan, P., Cuellar, J., and Sauer, M. (2020). Secure smart contract generation based on petri nets. In *Blockchain Technology for Industry 4.0: Secure, Decentralized, Distributed and Trusted Industry Environment*, pages 73–98. Springer.