Anais do Congresso Brasileiro de Iniciação Científica
(https://periodicos.unisanta.br/COB)

ISSN 2317-1316

USANDO AGDA PARA TESTAR CIRCUITOS ARITMÉTICOS

PURPOSING AGDA TO TEST ARITHMETHIC CIRCUITS

Gabrielle Lara Moricz de Souza, Alexandre Garcia de Oliveira

Faculdade de Tecnologia do Estado de São Paulo, Curso de Ciência de dados
E-mail para contato: gabrielle.souza@aluno.cps.sp.gov.br

RESUMO – *Um dos sistemas de restrições aritméticas utilizados na construção de zkSNARKs, protocolos de prova de conhecimento zero (zero-knowledge proofs) amplamente empregados em aplicações de blockchain e verificação criptográfica, é o Rank-One Constraint System (R1CS). Esse sistema permite representar relações polinomiais por meio de restrições expressas em termos de vetores. Neste trabalho, pretende-se implementar o R1CS em Agda, uma linguagem de programação com suporte à demonstração formal de propriedades matemáticas, com o objetivo de analisar, verificar formalmente e testar o sistema, além de explorar as funcionalidades e características da própria linguagem. Adicionalmente, busca-se investigar o potencial da Agda como linguagem de programação e ferramenta de verificação formal, contribuindo para a ampliação de seu uso no meio acadêmico.*

Palavras-chave: Circuito aritmético; Agda; zkSNARK; R1CS

ABSTRACT – *One of the arithmetic constraint systems used in the construction of zkSNARKs, zero-knowledge proof protocols widely employed in blockchain applications and cryptographic verification, is the Rank-One Constraint System (R1CS). This system enables the representation of polynomial relations through constraints expressed in terms of vectors. This study aims to implement R1CS in Agda, a programming language that supports the formal proof of mathematical properties, with the objective of analyzing, formally verifying, and testing the system, while also exploring the functionalities and characteristics of the language itself. Additionally, the study seeks to investigate the potential of Agda as a programming language and formal verification tool, contributing to the expansion of its use in academia.*

Keywords: Arithmethic circuits; Agda; zkSNARK; R1CS

ISSN 2317-1316

1 INTRODUÇÃO

O presente estudo tem como objetivos implementar o circuito aritmético *rank one constraint system, ou R1CS,* em um sistema escrito em Agda, assim possibilitando a prova de corretude matemática deste sistema e realizar o teste para ao menos um polinômio, provando a abstração deste como válida dentro do circuito. O estudo também visa desenvolver um entendimento mais profundo sobre Agda, quando apropriada para circuitos aritméticos, com o intuito de futura expansão.

Circuitos aritméticos são circuitos cujos portões operam dentro de um semi-anel, ao invés de utilizar conectivos booleanos convencionais, como diz Heribert Vollmer (1999), estes fazem parte da teoria da complexidade algébrica e são utilizados principalmente para realizar a computação de equações polinomiais. Um uso especializado de circuitos aritméticos é para sistemas de *Succinct Non-interactive Arguments of Knowledge* (SNARK), os quais segundo Anca Nitulescu (2020) por serem baseados em programas quadráticos se dão ao uso de instâncias de circuitos aritméticos, senão instâncias booleanas. Neste trabalho visa-se analisar um dos circuitos mais comumente utilizados no processo de SNARK, o R1CS.

O sistema de R1CS, como afirmam estudos recentes (Krishnamachari, 2024), é uma forma intermediaria para a representação de circuitos aritméticos que é representada conforme uma coleção de equações no formato

$$s \cdot A * s \cdot B = w \cdot C. \quad (1)$$

Onde s é o vetor de testemunha, com as variaveís do sistema, e A B e C são os vetores contendo cada parte. De maneira que equações sejam reescritas, genericamente, como uma equivalência entre o resultado de uma equação e a multiplicação vetorial de dois produtos escalares que se integram a ela. Esses vetores são abstrações das equações e confinados a um vetor testemunha contendo variáveis necessárias. O autor Krishnamachari (2024) também ressalta que esse processo deve ser realizado várias vezes para equações mais complexas, criando assim variáveis adicionais com o resultado de equações menores, como no exemplo $4xy^2 * 2y$, onde teríamos uma hierarquia.

ISSN 2317-1316

Tabela 1. Exemplo R1CS

Equação	Vetores de R1CS
V1 = y^2	*A = (0,0,1,0,0,0) , B = (0,0,1,0,0,0), C = (0,0,0,1,0,0)*
V2 = 4xV1	*A = (0,4,0,0,0,0) , B = (0,0,0,1,0,0), C = (0,0,0,0,1,0)*
Z = V2 * 2y	*A = (0,0,0,0,1,0) , B = (0,0,2,0,0,0), C = (0,0,0,0,0,1)*

Um dos problemas que incita esta pesquisa seria a rápida transição das atividades diárias, principalmente transações, do âmbito físico para o digital. Esse cenário torna tópicos de cibersegurança cada vez mais relevantes. Como antes citado, o circuito R1CS é um possível componente no sistema de SNARK, o qual gera o interesse por ser, como citam Joancomartí e Rodríguez (2020), um dos métodos mais utilizados para provas *zero-knowledge* não interativas, usadas em blockchain e validação criptografia. Analisando um dos integrantes deste sistema, de uma maneira não utilizada anteriormente, cria-se melhor entendimento quanto a ferramentas para garantir integridade de transações e segurança do usuário.

O outro motivo, por trás da pesquisa, é aprofundar-se sobre linguagem de tipos dependentes e assistente de provas Agda, criada em 1999 como extensão da teoria de Martin-löf por Ulf Norell em sua tese de doutorado. É uma linguagem pouco utilizada, cuja base na teoria dos tipos vem ganhando popularidade apenas nas últimas décadas, apesar de ter seus princípios nos anos 1900 (Norell, 2007). É intuito da pesquisa desenvolver um entendimento melhor sobre a linguagem, mostrar usos para a mesma e, por extensão, uma alternativa à teoria dos conjuntos que a compõe. Cogita-se, também, que o trabalho incite que mais acadêmicos vejam o valor na linguagem

Os resultados esperados da pesquisa são tão simples quanto um módulo de Agda que compile sem erros e que seja capaz de provar como correta uma implementação de R1CS.

2 MATERIAIS E MÉTODOS

Essa pesquisa tomou em primeiro momento uma abordagem de revisão de literatura e pesquisa exploratória, sendo estudado o circuito de R1CS tanto teoricamente quanto na prática, utilizando guias como o *R1CS to Quadratic Arithmetic Program over a Finite Field in Python* disponibilizados no RareSkills para testar o circuito em python. Nesta fase também houve um estudo exploratório da linguagem Agda, sendo desenvolvidos interpretadores e teoremas preliminares com o propósito de familiarização assim como uma versão simplificada do código.

ISSN 2317-1316

Todos os códigos para este artigo foram escritos no ambiente de programação em Agdapad, o qual opera em browser usando uma integração com o Emacs Agda-mode. O programa disponibiliza na sua biblioteca os módulos importados no trabalho, como o Algebra.Bundles, que teve o papel de construir o anel com tipo Carrier e todas suas operações.

Um anel R é um conjunto com duas operações binárias + e x (chamadas adção e multiplicação) satisfazendo os segundos axiomas:

(i) (R, +) é um grupo Abeliano

(ii) x é associativo: (a x b) x c = a x (b x c) para todo a, b, c E R,

(iii) As leis da distributividade se mantém em R : para todo a, b, c E R

(a + b) x c = (a x c) + (b x c) and a x (b + c) = (a x b) + (a x c).

(Dummit; Foote, 2008. tradução minha)

Observa-se que um grupo Abeliano é uma estrutura matemática composta de um conjunto R e uma operação binária de soma que seja associativa, comutativa e que todo elemento possua um oposto e um neutro.

tendo-se o anel, genérico no escopo dessa pesquisa, o próximo passo foi definir vetores e produtos escalares dentro dele que possam ser utilizados como restrições do circuito e um interpretador capaz de interpretar os processos passo a passo utilizando um processo de validação em matriz fazendo uso de uma função que avalia a igualdade para cada linha individual *CheckCons.* Observa-se que, como na versão de teste, no plano original iria ser utilizado o modo *Cubical-compatible,* um modo feito para suportar teoria dos tipos cúbica e, especialmente, univalência computacional e tipos indutivos superiores, o que foi revocado na versão final por motivos mais bem descritos nos resultados e discussão.

Para testar o sistema com um exemplo real, foi escolhida uma das equações que foram escritas na fase exploratória em python. A equacão escolhida foi $xy + 8$, A qual foi reescrita com a igualdade $xy = z\text{-}8$. Esse sistema pode ser escrito da seguinte maneira:

Tabela 2. Vetores da pesquisa

Equação	**Vetores de R1CS**
Z - 8 = xy	*A = (0,1,0,0) , B = (0,0,1,0), C = (-8,0,0,1)*

A equação foi escolhida por demonstrar tanto a soma quando a multiplicação e por ser de natureza mais simplista. Apesar do interpretador ter sido idealizado com a intenção de rodar

ISSN 2317-1316

com matrizes, foi decidido utilizar um caso simples de um passo só para a fase de teste como este.

3 RESULTADOS E DISCUSSÃO

Em primeiro momento, o código se concentra em preparar o ambiente para trabalhar dentro de um anel, importando as propriedades do Algebra.Bundles e Algebra.Structures e definindo elementos fundamentais do circuito R1CS, como vetores e matrizes, para uso futuro. Um destes elementos necessários é a definição do produto escalar. Neste passo ocorreu um conflito com o modo Cubical-compatible que originalmente se planejava utilizar. Enquanto o código não deixava de compilar, ele avisa enfaticamente quanto a um problema no *pattern-matching utilizado*, o processo que avalia uma sequência de constituintes de um padrão. Neste caso o problema se encontra no fato da função utilizar matching por índice, o que necessita da injetividade que ainda não é suportada no modo cúbico do Agda e causará falha em caso da aplicação de transportes.

Enquanto neste trabalho se optou por descartar o uso do Cubical-Agda, isso pode interferir em pesquisas futuras na área, principalmente em análise de univalência, uma vez que o modo foi feito para servir este propósito e a falta de transportes, que são uma propriedade chave em teoria dos tipos (Vezzosi, A., Mörtberg, A., Abel, A; 2021) pode trazer dificuldades em futuros estudos.

Feito o necessário para o interpretador passou-se a formalização deste, a qual foi feita utilizando uma função verificadora para um único passo e aplicado a uma matriz por divisão do problema até o caso vazio:

Código 1. Interpretador

```
CheckCons : {n : ℕ} → Vector n → Vector n → Vector n → Vector n → Set l
CheckCons A B C witness = (DotProduct A witness) * (DotProduct B witness) ≈
(DotProduct                                C                                witness)


data SatisfyAll : {n m : ℕ} →  Mat n m → Mat n m → Mat n m → Vector n → Set l where
   null : {n : ℕ} → {witness : Vector n} → SatisfyAll [] [] [] witness
   cons : {n m : ℕ} → {As Bs Cs : Mat n m} → {A B C witness : Vector n}
      →           CheckCons           A           B           C           witness
      →          SatisfyAll          As          Bs          Cs          witness
      → SatisfyAll (A ∷ As) (B ∷ Bs) (C ∷ Cs) witness
```

ISSN 2317-1316

Com as variáveis e os vetores definidos, o estudo segue para a parte que apresentou o maior desafio no decorrer do código, em grande parte por conta de conflitos de importação da biblioteca e pela escolha de trabalhar no anel.

O método mais comum para escrita de provas em Agda é a partir de teoremas e preenchimento de buracos, com blocos utilizando '*begin*' e '*end*' para provas mais extensas. No entanto houve conflitos de precedência das operações com o solver e estes outros recursos de prova de teorema de maneira que as partes individuais tiveram que ser escritas de maneira extensas para a prova. A decisão de não deixar espaço para provas interativas com buraco também se justifica pela necessidade de ter um resultado concretizado. Para assistência nesta fase foram adicionados trechos de código explicitando propriedades multiplicativas e aditivas dentro do anel, como a multiplicação de identidade com o Carrier 1 ou soma por Carrier 0, assim como importadas propriedades de simetria, transitividade e identidade (refl) dentro de um anel—as quais foram amplamente utilizadas para reduzir equações.

A prova foi dividida em duas partes, o teste de que a multiplicação dos produtos escalares de vetor era o mesmo que a multiplicação das duas variáveis do sistema (a prova a esquerda) e o teste de que a multiplicação entre variáveis do sistema é equivalente ao produto escalar do vetor final (a prova a direita). A transitividade entre as duas foi usada de prova para a validade das restrições, o que preenche a lacuna para a prova final em matriz. Para manter a integridade da prova foi optado por evitar o uso da função de Agda que possibilita a criação de postulados, ou seja, suposições não comprovadas matematicamente ou axiomas. O único postulado utilizado foi em parte da prova a direita, mostrando a equivalência entre - (- 8) e 8 no anel, conforme representado pela multiplicação equivalente. A maior pare dos passos de prova foram feitos no seguinte formato:

Código 2. Demonstração igualdade

```
SimplificaA : ((0# * 1#) + ((1# * x) + ((0# * y) + ((0# * z) + 0#)))) ≈ x
SimplificaA                                                             =
      ≈-trans (+-cong P0 (+-cong idA  (+-cong P0 (+-cong P0 (Ring.refl R))))) (
      ≈-trans (+-cong (Ring.refl R) (+-cong (Ring.refl R)  (+-cong (Ring.refl R)
0+))) (≈-trans (+-cong (Ring.refl R) (+-cong (Ring.refl R) +0)) (≈-trans (+-cong
            (Ring.refl R) +0) 0+)))
```

Esse trecho da prova servindo para reduzir o produto escalar à variável x do qual provém, ele vem da prova esquerda e é acompanhado de uma prova similar para y.

ISSN 2317-1316

Em uma linguagem de assistência de prova como o Agda, a compilação correta do sistema é o que indica a corretude matemática, logo tendo sido compilado o código obtém-se a prova de que o sistema é plausível de maneira integrada ao anel. A pesquisa agora serve de base para experimentação futura, com os conhecimentos adquiridos servindo para orientação em vistas futuras em relação ao tema., principalmente pelas dificuldades. Para o restante do código, foi disponibilizado acesso ao repositório do trabalho no Github.

4 CONCLUSÃO

Criar o sistema de R1CS em si se mostrou, por maior parte, trivial, no entanto provas específicas foram mais difíceis. Após o sistema rodar propriamente, foi possível garantir corretude matemática tanto do R1CS em si quando da abstração. Com a conclusão deste trabalho, planeja-se utilizar os conhecimentos desenvolvidos e sistema criado em um estudo mais abrangente de zkML como um todo, sendo incorporado o programa de R1CS juntamente de um outro circuito *plonkish encoding* com intuito de validação e comparação.

5 DECLARAÇÃO DE USO DE IA

Durante a elaboração dos códigos abordados neste trabalho, utilizou-se a ferramenta Gemini 3.0 do google para responder dúvidas e auxiliar na programação. O conteúdo gerado foi revisado, editado e validado pela autora, que assume total responsabilidade pelo produto final. Não foram utilizadas ferramentas de IA na escrita desse artigo.

6 REFERÊNCIAS

Bove, A., Dybjer, P., Norell, U. A Brief Overview of Agda – A Functional Language with Dependent Types. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds) Theorem Proving in Higher Order Logics. TPHOLs 2009. Lecture Notes in Computer Science, vol 5674. Springer, Berlin, Heidelberg. 2009 disponível em: https://link.springer.com/chapter/10.1007/978-3-642-03359-9_6

Dummit, S, D. Foote R, M. Abstract algebra: third edition. 2004 disponível em: https://rksmvv.ac.in/wp-content/uploads/2021/04/David_S_Dummit_Richard_M_Foote_Abstract_Algeb_230928_225848.pdf

ISSN 2317-1316

Joancomartí, J.H; Rodríguez, A.B zk-SNARKs Analysis and Implementation on Ethereum, 2020. disponível em: https://openaccess.uoc.edu/server/api/core/bitstreams/13e30744-ae31-4214-bde9-be95b7482250/content

Krishnamachari, B. Understanding Zero-Knowledge Proofs: Part 5-Arithmetic Circuits and R1CS**,** 2024. disponível em: https://medium.com/@bhaskark2/understanding-zero-knowledge-proofs-part-5-arithmetic-circuits-and-r1cs-be66347096b8

Nitulescu, A.. zk-SNARKs: A Gentle Introduction. 2020. disponível em: https://www.semanticscholar.org/paper/zk-SNARKs%3A-A-Gentle-Introduction-Nitulescu/18a17e5f50c43e694c6d9fce5ec565e61cf544a2

Vollmer, H.. Arithmetic Circuits. In: Introduction to Circuit Complexity. Texts in Theoretical Computer Science An EATCS Series. Springer, Berlin, Heidelberg. 1999 disponível em: https://link.springer.com/chapter/10.1007/978-3-662-03927-4_6

Github do projeto: https://github.com/Gabi444444/R1CS_Agda