当谈到零知识证明时，常用的技术包括基于交互式证明系统和非交互式证明系统。交互式证明系统需要证明者与验证者进行多轮通信，而非交互式证明系统则可以通过一次通信来完成证明过程。

在非交互式证明系统中，R1CS是一种经典的可证明算术电路，它使用线性约束系统来描述一个计算过程，并且可以通过计算出这些约束的解来证明该过程的正确性。由于它具有高效性、易于实现等优点，R1CS已经成为了构建零知识证明系统的核心组件之一。

R1CS模型由三部分组成：输入、输出和一组约束条件。输入是指证明者希望证明的私密信息，输出是他们想要证明的结论，约束条件是一组线性方程和不等式，这些方程和不等式描述了计算的限制条件。具体而言，每个约束都可以表示为形如“a·x + b·y + c·z = d”的线性方程，其中a、b、c是给定的系数，而x、y、z是证明者希望证明的变量。

假设我们有一个加法器电路，其中输入是两个整数，输出是它们的和。我们可以将这个电路表示为一个R1CS约束系统，其中包含三个变量（输入的两个整数和输出值）以及两个约束条件。第一个约束条件是“x + y = z”，表示加法器的运算规则，第二个约束条件是“x、y和z都必须是整数”，这是一种边界条件。

证明者可以使用R1CS约束系统来证明他们掌握着满足这些约束条件的私密信息，同时不必向验证者透露这些信息的实际值。证明者只需要将这些约束条件作为公共参数进行公开，并且提供一组响应来证明这些约束条件得到了满足。验证者可以使用这些响应来验证证明者的声明，并确定其是否具有正确性。

总之，R1CS是一种强大的可证明算术电路模型，它在零知识证明中扮演了重要角色。

////

2023/4/10 19:55:08

R1CS模型描述了一组基于约束的计算规则，可以通过一组公共参数和一个私密的输入序列来进行验证。

在这个模型中，计算被表示为一组约束条件，也就是线性方程和不等式。每个方程都有自己的一组系数，而每个变量代表一个输入或输出值。这些方程和不等式总体上描述了计算的限制条件，即满足这些条件就意味着给定的输入序列可以正确地计算出对应的输出结果。

实际上，一个R1CS约束可以看作一个三元组, 它由三个向量a, b, c组成, ，假设 R1CS 的解也是一个向量，记为s, 则s满足

s.a \* s.b – s.c =0

这就构成了一个一阶约束,这样的一个约束对应了电路中的一个乘法门。如果我们将所有的约束联立起来，就得到一个一阶约束系统

从本质上讲，R1CS约束系统使用广义线性规划技术来解决问题，其中最终目标是找到一组变量值，使得所有约束都满足，并且计算结果与期望的输出一致。在这个过程中，证明者需要提供一组响应，以便验证者可以验证其声明的正确性。

总之，R1CS是一种强大的计算模型，已经在许多实际应用中得到了广泛的应用。作为可验证算法设计领域的重要组成部分，它极大地扩展了我们对计算机科学和密码学方面的理解，也为各种隐私保护措施提供了关键支持。

////

当涉及到零知识证明技术时，Circom是一个重要的编程语言之一，它被设计用来编写基于约束系统（Constraint System）的算术电路。它提供了一种简单、声明式的方法来定义约束系统，并生成对应的R1CS（Rank-1 Constraint Systems）约束系统。在这个过程中，开发者可以使用Circom来描述不同的约束条件，例如线性方程和不等式。

Circom是一种领域特定语言（DSL），专门为构建零知识证明系统而设计。DSL是一种针对特定领域的编程语言，通常比通用的编程语言更具有表现力和易用性。Circom通过提供一组高级抽象概念，使得开发人员能够更加关注算法本身，而不必过多地考虑底层实现细节。

Circom的核心思想是将算术电路表示为约束系统，其中输入、输出和计算过程都表示为线性方程和不等式。通过这种方式，开发者可以定义任意复杂的计算过程，并从中生成相应的R1CS约束系统。这种约束系统可以用于实现各种隐私保护协议和零知识证明技术，包括数字资产交易、密码学签名、匿名投票等。

Circom提供了一组丰富的语法和语义，以帮助开发者快速定义约束系统。例如，它支持算术运算符（加、减、乘、除）和逻辑运算符（与、或、非），可以用于快速构建各种复杂的计算过程。此外，Circom还提供了一组标准库函数，例如hash、signature和merkle tree，可用于构建更高级别的应用程序。

Circom还可以与其他工具集成，例如SnarkJS和ZoKrates，以构建更复杂的零知识证明系统。这些工具可以将Circom代码编译为可执行的JavaScript代码，并提供一组API，用于验证证明和生成证据。

总之，Circom是一个强大的编程语言，可用于构建各种复杂的算术电路和R1CS约束系统。它提供了一种简单、声明式的方法来定义约束系统，使得开发者能够快速实现各种隐私保护协议和零知识证明技术。Circom已经被广泛用于密码学、区块链和其他安全应用程序，为保护用户隐私和数据安全提供了重要的支持。

////

zk-SNARKs 是一种用于验证信息真实性的数学工具。它可以让信息在不暴露任何相关内容的情况下进行验证，同时保护用户的隐私。Zk-SNARKS是一种特殊的零知识证明，这意味着证明者可以证明某些事情是正确的，但不需要透露任何关于证明的信息，即证明者不需要透露具体的数据，只需要证明这些数据满足某些条件即可。

zk-SNARKs 工作的原理比较复杂，但可以简单描述如下：首先，用户将其想要证明的内容转化为了解某些代数方程解的等价形式，并将该问题发送给证明者。这个过程具体如下:

**计算 → 算术电路 → R1CS → QAP → zk-SNARK**

工作原理比较复杂 但可以简单分为以下步骤首先，当用户将其需要验证的信息转换成一个数学问题时，这个过程通常称作计算（computation）。计算可以使用任意图灵完全的编程语言实现，比如 C、Python 和 Solidity（用于 Ethereum 智能合约编程），因为 zk-SNARKs 算法不依赖于特定的编程语言。

然后，计算的结果通常被转化成一个算术电路（arithmetic circuit），它是一种表示计算过程的数据结构。算术电路包含多个门（gate），每个门执行一个基本的计算操作，比如加法或乘法。整个算术电路可以用于生成可验证的算术电路（R1CS），它是基于约束的公式系统（rank-1 constraint system）的形式，这个公式系统表达了算术电路的约束条件。可验证的算术电路是 zk-SNARKs 算法的输入之一。

接下来是 QAP（Quadratic Arithmetic Program）阶段，其中可验证算术电路被进一步转换为 QAP 格式。QAP 是一种公式系统，其中多项式表示了算术电路的行为。QAP 使得 zk-SNARKs 算法的实现更加高效，同时提高了安全性。

最后是 zk-SNARK 阶段。在这个阶段，可验证算术电路和 QAP 被用来生成证明（proof）。

zk-SNARKs 是一种强大的隐私保护工具，可以用于数字支付、区块链技术和其他领域。它可以通过验证信息的真实性，同时保护用户的隐私。虽然 zk-SNARKs 的工作原理比较复杂，但这一技术已经被广泛应用，为数字化世界带来更高的安全性和隐私保护。

## Abstract

近年来，随着区块链技术的发展和应用，R1CS在这个领域中变得越来越重要。例如，在以太坊中，许多重要的协议和应用程序都需要使用R1CS进行状态转换的验证，包括DeFi、NFT等应用。然而由于等价R1CS的过于灵活,使得对其正确性的验证和可拓展性的研究都比较困难。本文介绍了一种基于数据流的R1CS范式生成算法。该算法将R1CS转换为数据流图，并在抽象化处理和权重计算等步骤后将等价的R1CS转化为相同的唯一范式。我们的模拟研究标明，面对常见的产生等价R1CS的场景下，我们的算法均能正确生成范式。 0

In recent years, with the development and application of blockchain technology, R1CS has become increasingly important in this field. For example, in Ethereum, many important protocols and applications require the use of R1CS for verifying state transitions, including DeFi, NFT, and other applications. However, due to the excessive flexibility of equivalent R1CS, it is difficult to verify its correctness and study its scalability. This paper introduces a data flow-based R1CS paradigm generation algorithm. The algorithm transforms R1CS into a data flow graph and converts equivalent R1CS into the same unique paradigm after abstract processing and weight calculation steps. Our simulation studies indicate that our algorithm can correctly generate paradigms in common scenarios that produce equivalent R1CS.

//0423修改

在零知识证明中，证明者需要构建一个包含许多约束条件的系统，并证明该系统满足这些约束条件, 而Circom和R1CS 是其底层工具链中的重要部件，使得用户能够轻松地创建、优化和验证零知识证明系统。用户可以通过Circom轻松地定义各种约束条件，并通过一些编译器将其转换为 R1CS 格式，再将其转换为可验证的形式。但是由于R1CS表达形式的灵活性，相同语义的Circom程序编译出的R1CS之间往往大相径庭。这也导致对R1CS正确性的验证和可拓展性的研究都比较困难. 本文介绍了一种基于数据流的R1CS范式生成算法。该算法将R1CS转换为数据流图，并在抽象化处理和权重计算等步骤后将等价的R1CS转化为相同的唯一范式。我们的模拟研究标明，面对常见的产生等价R1CS的场景下，我们的算法均能正确生成范式。

In zero-knowledge proofs, the prover needs to construct a system that contains many constraints and prove that the system satisfies these constraints. Circom and R1CS are important components of the underlying toolchain that allow users to easily create, optimize, and verify zero-knowledge proof systems. Users can define various constraints easily with Circom and convert them to the R1CS format using a compiler, and then convert them to a verifiable form. However, due to the flexibility of the R1CS representation, R1CS compiled from Circom programs with the same semantics often differ significantly. This also makes it difficult to verify the correctness and scalability of R1CS. In this paper, we propose a data flow-based R1CS paradigm generation algorithm. The algorithm converts R1CS to a data flow graph and transforms equivalent R1CS into the same unique paradigm after abstract processing and weight calculation. Our simulation studies indicate that our algorithm can correctly generate paradigms in scenarios where equivalent R1CS are commonly produced.

## Introduction

Zero-knowledge proof is increasingly recognized for its importance in modern society, as more and more cryptographic communities seek to address some of the blockchain's biggest challenges: privacy and scalability[paper]. The heightened emphasis on information privacy and security, from both user and developer perspectives, has led to a greater appreciation for the privacy advantages offered by zero-knowledge proofs. As decentralized finance (DeFi) usage continues to grow, zero-knowledge applications that offer scalability and privacy advantages will have more opportunities to increase industry-wide adoption. However, not all computational problems can be directly addressed using zero-knowledge proofs. Instead, we must transform the problem into the correct form of computation, known as a "Quadratic Arithmetic Program (QAP)." In the specific process of a first-order zero-knowledge proof, we first convert the problem into Circom language, then into R1CS constraints, and finally from constraints to the QAP form.

However, the conversion from Circom to R1CS constraints in the underlying toolchain of zero-knowledge proofs faces many limitations, with the primary issue being poor mergeability of R1CS. When merging A and B, the resulting R1CS has no formal relationship with the independently generated R1CS of A and B. This limitation is related to the inherent expressive power constraint of R1CS, where the program can generate multiple equivalent R1CS constraints. Therefore, it is necessary to propose a canonical form for R1CS constraints to facilitate the determination of equivalence and correctness for different R1CS constraints. This proposal would greatly benefit us in verifying program equivalence and correctness, including further research into the mergeability of R1CS.

因此,本论文旨在提出一种R1CS范式生成的算法, 使得对于不同的R1CS约束，我们可以将其转化至唯一的范式, 从而更简单地判断其等价性和正确性。我们以数据流为基础, 将R1CS转化为语法树并分割成瓦片, 还提出了一系列排序的规则,以便范式的生成. 我们还在自己通过分析等价R1CS生成规律整理出的benchmark上进行了测试. 结果表明,在不同情形下,等价的R1CS均能被转换至唯一且相同的范式.

//0423修改,一二段保留, 第三段分两段, 再加contribution 和 organization

本论文提出了一种基于数据流的R1CS范式生成算法,使得对于不同的R1CS约束,我们可以将其转化至唯一的范式, 从而更加简单地判断期等价性和正确性. 在该算法中,我们首先将R1CS转化为类似于算式树的数据流图结构, 再对该数据流图进行分割与抽象,消除了等价R1CS之间产生的数据流图的区别, 然后根据我们提出的一系列排序规则,对R1CS内部的约束和变量进行排序,最终为等价的R1CS生成唯一的范式.

同时, 根据目前主流编译器的约束生成逻辑以及R1CS的约束表达特性, 我们对等价R1CS生成的不同原因及特点进行了划分与总结. 并且根据所总结出的等价R1CS产生原因,我们制作了相对完备的benchmark. 目前,本文所提出的算法能通过benchmark中所有的用例, 也就是说, ,在不同情形下,等价的R1CS均能被转换至唯一且相同的范式.

This paper proposes a data flow-based algorithm for generating normalization of R1CS, which enables the conversion of different R1CS constraints into a unique normal form, facilitating the determination of equivalence and correctness. In this algorithm, we first transform R1CS into a data flow graph structure resembling an expression tree. We then segment and abstract the data flow graph, eliminating differences between equivalent R1CS constraints that may arise from the generation process. Next, we propose a set of sorting rules to sort the constraints and variables within R1CS, ultimately generating a unique normal form for equivalent R1CS.

In addition, based on the constraint generation logic of mainstream compilers and the expressiveness of R1CS, we classify and summarize the reasons and characteristics of the different equivalent R1CS generated. Moreover, based on the identified reasons for producing equivalent R1CS, we create a relatively complete benchmark. Currently, our proposed algorithm can pass all test cases in the benchmark, meaning that equivalent R1CS can be converted into a unique and identical canonical form under various circumstances.

//0423修改 contribution,

本研究提出了一种新颖的算法，用于生成等价R1CS约束的规范形式，为R1CS优化领域做出了贡献。我们的算法通过消除不必要的冗余和规范化表达，改进了现有方法，从而方便了等价性和正确性分析。此外，我们的基准测试对所提出的算法性能进行了全面评估，证明了其有效性和实用性。

This work contributes to the field of R1CS optimization by providing a novel algorithm for generating canonical forms of equivalent R1CS constraints. Our algorithm improves upon existing methods by eliminating unnecessary redundancy and normalizing representation, thus facilitating the analysis of equivalence and correctness. Furthermore, our benchmark provides a comprehensive evaluation of the proposed algorithm, demonstrating its effectiveness and practicality.

//0423修改 organization

The rest of this paper is organized as follows. A brief background review of zero-knowledge proof and related components in its underlying toolchain is presented in the next section. Section 3 presents the process of the proposed algorithm in this paper. The detailed logic of the key steps, along with their implementation in code, is elucidated in Section 4 using a technical exposition. The specific categories of benchmarks and their corresponding experimental results can be found in Section 5. Finally, Section 6 and Section 7 summarize the related work and conclusions of the present study correspondingly.

## Background

### Zk-SNARKS

zk-SNARKs is a specific type of zero-knowledge proof, with a working principle that can be simplified into several steps. First, when a user needs to verify their information, they convert it into a mathematical problem, which is called computation. Computation can be implemented using any Turing-complete programming language, such as C, Python, and Solidity (used for Ethereum smart contract programming) because the zk-SNARKs algorithm does not depend on any specific programming language.

Next, the result of the computation is usually transformed into an arithmetic circuit, which is a data structure representing the calculation process. An arithmetic circuit consists of multiple gates, with each gate performing a basic arithmetic operation such as addition or multiplication. The entire arithmetic circuit can be used to generate a verifiable arithmetic circuit (R1CS), which takes the form of a constraint-based formula system (rank-1 constraint system) expressing the constraints of the arithmetic circuit. A verifiable arithmetic circuit is one of the inputs of the zk-SNARKs algorithm.

The next stage is the Quadratic Arithmetic Program (QAP), where the verifiable arithmetic circuit is further converted into QAP format. QAP is a formula system in which polynomials represent the behavior of the arithmetic circuit. QAP makes the implementation of the zk-SNARKs algorithm more efficient while also enhancing security.

Finally, there is the zk-SNARK stage, where the verifiable arithmetic circuit and QAP are used to generate proof.

zk-SNARKs are powerful privacy protection tools that can be utilized in digital payments, blockchain technology, and other fields. They can verify the authenticity of information while protecting the user's privacy. Although the working principle of zk-SNARKs is relatively complex, this technology has been widely applied, bringing higher security and privacy protection to the digital world.

### Circuit Language

When it comes to zero-knowledge proof technology, Circom is one of the most important programming languages designed for building zero-knowledge applications[paper]. It provides a simple, declarative way to define constraint systems and generates corresponding R1CS (Rank-1 Constraint Systems) constraint systems. In this process, developers can use Circom to describe different constraints such as linear equations and inequalities.

Circom is a domain-specific language (DSL) specifically designed for building zero-knowledge proof systems. DSL programming languages targeted at specific domains are typically more expressive and user-friendly than general-purpose programming languages. By providing a set of high-level abstract concepts, Circom allows developers to focus more on the algorithm itself without worrying too much about low-level implementation details.

The core idea of Circom is to represent an arithmetic circuit as a constraint system, where inputs, outputs, and computation processes are all represented as linear equations and inequalities. This approach allows developers to define arbitrarily complex computation processes and generate the corresponding R1CS constraint system. This constraint system can be used to implement various privacy-preserving protocols and zero-knowledge proof techniques, including digital asset transactions, cryptographic signatures, anonymous voting, and more.

A rich set of syntax and semantics was provided in Circom to help developers quickly define constraint systems. For example, it supports arithmetic operators (addition, subtraction, multiplication, division) and logical operators (AND, OR, NOT), which can rapidly construct complex computation processes. Additionally, Circom provides a standard library of functions, such as hash, signature, and Merkle tree, that can be used to build higher-level applications.

Circom can also be integrated with other tools such as SnarkJS and ZoKrates to build more complex zero-knowledge proof systems. These tools can compile Circom code into executable JavaScript code and provide a set of APIs for verifying proofs and generating evidence.

In summary, Circom is a powerful programming language that can be used to build various complex arithmetic circuits and R1CS constraint systems. It provides a simple, declarative way to define constraint systems, allowing developers to quickly implement various privacy-preserving protocols and zero-knowledge proof techniques. Circom has been widely used in cryptography, blockchain, and other security applications, providing critical support for protecting user privacy and data security.

### R1CS

R1CS model describes a set of constraint-based computation rules that can be verified using a set of public parameters and a private input sequence. In this model, computations are represented as a set of constraint conditions, namely linear equations and inequalities. Each equation has its own set of coefficients, while each variable represents an input or output value. These equations and inequalities overall describe the limiting conditions of the computation, meaning that satisfying these conditions implies that the corresponding output result can be correctly calculated for the given input sequence.

In essence, an R1CS constraint can be viewed as a triple, consisting of three vectors a, b, and c, where s is assumed to be the solution of the R1CS and satisfies

s.a \* s.b - s.c = 0.

This constitutes a first-order constraint, which corresponds to a multiplication gate in the circuit. If we combine all constraints, we obtain a first-order constraint system.

Fundamentally, the R1CS constraint system employs the technique of generalized linear programming to solve problems, where the ultimate goal is to find a set of variable values that satisfy all constraints and the computed result matches the expected output. In this process, the prover needs to provide a set of responses so that the verifier can verify their claimed correctness.

Overall, R1CS is a powerful computational model that has been widely used in many practical applications. As an important component of verifiable algorithm design, it greatly expands our understanding of computer science and cryptography and provides critical support for various privacy protection measures.

### Weighted pagerank

In this paper, we adopt the weighted PageRank algorithm to compute the weight of each node in the data flow graph.

Pagerank algorithm is a method used for computing the ranking of web pages in search engine results. It was originally proposed by Larry Page and Sergey Brin, co-founders of Google, in 1998 and has since become one of the most important algorithms in the field of search engines.

The algorithm analyzes web pages on the internet to determine their weight values and uses these values to rank search results. The core idea behind Pagerank is that the weight of a web page depends on the number and quality of other web pages linking to it.

The main steps of the Pagerank algorithm are as follows:

1. Building the graph structure: Firstly, the web pages and links on the internet need to be converted into a graph structure. In this structure, each web page corresponds to a node and each link to a directed edge pointing to the linked web page.

2. Computing the initial scores of each page: In Pagerank, the initial score of each page is set to 1. This means that initially, each node has an equal score.

3. Iteratively computing the scores of each page: Each node's score is iteratively computed based on its incoming links and averaged onto its outgoing links at each iteration.

4. Considering the number and quality of links: In addition to the relationships between nodes, Pagerank considers the number and quality of links pointing to a web page. Links from high-quality websites may carry more value than those from low-quality sites. Therefore, when computing scores, the algorithm weights links according to their number and quality.

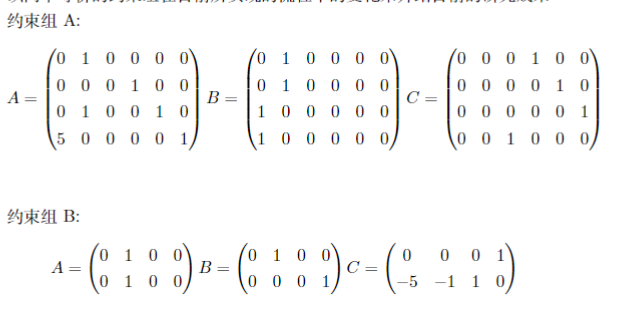
5. Iterating until convergence: When the score of a node stabilizes, the algorithm stops iterating. This indicates that the final scores of all nodes have been determined and can be used to rank search results.

The Weighted PageRank algorithm differs from the standard PageRank algorithm in that it incorporates the weight of each link as a factor, resulting in a more precise evaluation of a webpage's importance. In this paper, we aim to use this algorithm to obtain more accurate weight values for each node in the data flow graph.

## Overview

在这个部分中,我们通过两个等价R1CS约束组在范式生成算法中的转换过程来介绍整个算法的运作方式. 我们将这两个约束组分别记为约束组A 与 约束组B

In this section, we will introduce the operation of the entire algorithm through the process of converting two equivalent R1CS constraint groups in the paradigm generation algorithm. We refer to these two constraint groups as Constraint Group A and Constraint Group B, respectively.



其中, 约束组由约束组A将自己的后三个约束合并, 再经过一些变量顺序的交换而生成的.

The Constraint Group B is generated by merging the last three constraints of Constraint Group A and performing some variable order swapping.

### Construction of RNode Graph

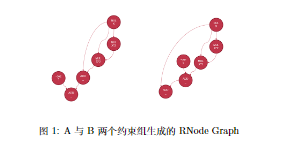
RNode graph的生成主要分为三个阶段:

1. 将每一个约束按照R1CS约束需要满足的等式a\*b=c转换为算式

2. 将原R1CS 约束组中每一个约束, 都化成这样的算式

3. 将得到的得到一个以DAG 形式存储的含有公共子式的算式树

两个约束组生成的RNode Graph的结构如图1所示

The generation of the RNode Graph involves three main stages:

Each constraint is transformed into an equation in the form of a\*b=c, as required by the R1CS constraints.

Each constraint in the original R1CS constraint set is then converted into such an equation.

The resulting equations, which contain common subexpressions, are organized into a DAG-structured expression tree.

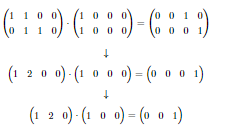
The structure of the RNode Graph generated by two sets of constraints is shown in Figure 1.

RNode是我们在建立RNode Graph过程中用以储存R1CS约束组中各个变量的信息的数据结构. 在RNode中, 与一般的算式树中的节点不同, 同时存储了变量和操作符的信息,这样也使每一个运算符的结果都能被看作是中间变量, 这也与R1CS约束组的性质更加贴近.

约束组 B 中的第二个约束, 实际上是将约束组 A 的后三个约束合并在一起,而在实际情况中这种约束之间的合并于拆分正是因为 R1CS 约束组等价性难以判别的原因之一. 但是当我们观察两个生成的 RNodeGraph, 可以发现这样的合并并没有带来明显的不同. 这是由于当约束被合并时, 在原约束组中会减去一个变量, 比如

The RNode is a data structure used to store information about the variables in an R1CS during the construction of the RNode Graph. Unlike typical nodes in an expression tree, each RNode stores information about both the variable and operator involved in a given operation, allowing each operator's output to be considered an intermediate variable and making it more closely aligned with the properties of an R1CS constraint set.

The second constraint in constraint set B is actually a combination of the last three constraints in constraint set A. In practice, the merging and splitting of constraints is one of the reasons why it is difficult to determine the equivalence of R1CS constraint sets. However, upon examining the two generated RNode Graphs, it is observed that this merging does not result in any significant differences. This is because when constraints are merged, one variable is subtracted from the original constraint set, for example



但是删去的这个变量会在建立 RNodeGraph 时, 作为连加式中的中间节点被加入到 RNodeGraph 中. 反之亦然.

观察图 1中的两个 RNodeGraph, 发现其主要的不同在于, 在在构造 x3 +x + 5 = out 这一连加式时, 相加的前后顺序不同, 这是由于约束组中变量排序不同, 而在这一阶段我们没有足够的信息去判别假发执行的顺序. 约束组 A 中为(x3 + x) + 5 = out, 而约束组 B 中为 (x3 + 5) + x = out. 因此对于这种情况, 我们需要对 RNodeGraph 进行进一步的抽象,以消除这一不同之处

However, the variable that is subtracted will be added back into the RNode Graph as an intermediate node in the sum-product expression during the construction of the RNode Graph. The reverse is also true.

Upon examining the two RNode Graphs shown in Figure 1, the main difference observed is in the order of addition in constructing the sum-product expression x3 + x + 5 = out. This is due to the different variable ordering in the constraint set, and we do not have sufficient information at this stage to determine the sequential execution order. In constraint set A, the expression is (x3 + x) + 5 = out, while in constraint set B, it is (x3 + 5) + x = out. Therefore, for such cases, further abstraction of the RNode Graph is required to eliminate this difference.

### Tile Selection

这里我主要将瓦片分成三个类型：

1. Quadratic：x ∗ y = z 的瓦片，其中 x，y，z 均为变量。

2. MullLinear：根节点由其两个父亲相乘得到的瓦片，两个父亲中至少有一个为常数, 比如 (5 ∗ x) ∗ 7 = z。

3. AddLinear：根节点由其两个父亲相加得到的瓦片，比如 x3 + 5 + x = z。

三种瓦片的示意图如图 2所示.其中 AddLinear 和 MullLinear 瓦片本质上都由线性约束产生, 都是 线性瓦片, 但是由于在算法处理上逻辑完全不同, 所以在此将其分为两个种类讨论。在瓦片选择时, 我们将上一个步骤中的数据流图分割成上述三种类型, 这样选取有几个考虑方面:

1. 将约束合并步骤暂时搁置, 待后续步骤获取树中的更多信息后再进行。

2. 产生未合并的范式后, 如有产生合并范式的需求, 只需在未合并的范式上应用固定算法即可, 相对简单。

3. 瓦片选取的算法实现较为简单。

前面我们提到过, 等价的R1CS约束组所生成的数据流图其不同之处在于在处理线性瓦片时，节点之间相加的顺序不同，但是在一个线性约束中所相加的节点看成一个集合的话，他们其实是等价的。也就是说，相加顺序的不同，在瓦片选取的过程中仅仅意味着各节点遍历顺序的不同，如果将选取好的线性瓦片视为其相加节点与其系数的乘积的集合，那么等价R1CS约束组产生的数据流图中所选出的线性瓦片之间显然是相同的。也即是说, 对于等价的R1CS约束组,所选出的瓦片集合之间并不会存在不同之处.

已选出的瓦片为基础,我们对数据流图进行再一次的抽象，进一步消除了各个等价R1CS约束组在数据流图层面的不同。对线性瓦片进行进一步的抽象，将他在数据流图中用一个抽象出的新节点代替。通过对线性瓦片的抽象, 我们在RNodeGraph屏蔽线性瓦片内部相加顺序的不同的差异，让外部节点到线性瓦片中具体节点的联系, 变成到这个节点具体所属的瓦片的联系。而在这一抽象后的数据流图中，边的类型有以下几种：

1. 非线性瓦片抽象节点到非线性瓦片抽象节点: 两个顶点在抽象前的数据流图中便已经存在。与抽象前的数据流图保持一致。

2. 非线性瓦片抽象节点到线性瓦片抽象节点: 当且仅当抽象节点所代表的线性瓦片中存在非抽象节点时存在。

3. 线性瓦片抽象节点到线性瓦片抽象节点: 当且仅当两个抽象节点所代表的线性瓦片存在公有的非抽象节点时存在。

Here, we categorize tiles into three types:

1. Quadratic: Tiles with the form x \* y = z, where x, y, and z are variables.

2. MulLinear: Tiles whose root is obtained by multiplying its two parents, with at least one of the parents being a constant, such as (5 \* x) \* 7 = z.

3. AddLinear: Tiles whose root is obtained by adding its two parents, such as x3 + 5 + x = z.

The schematic diagrams for these three types of tiles are shown in Figure 2. While AddLinear and MulLinear tiles are both generated by linear constraints and are essentially linear tiles, their logical processing differs significantly. Hence, we discuss them as two separate types. During tile selection, we divide the data flow graph from the previous step into these three types for various considerations:

1. We temporarily put aside the constraint merging step until we obtain more information about the tree in subsequent steps.

2. If there is a need to generate merged formulas later, it can be achieved simply by applying a fixed algorithm to the unmerged formulas.

3. The implementation of the tile selection algorithm is relatively simple.

As we mentioned earlier, the difference between data flow graphs generated by equivalent R1CS constraint systems lies in the order of node additions when processing linear tiles. However, if we consider the nodes added within a linear constraint as a set, they are actually equivalent. That is to say, the different addition order only means that the traversal order of the nodes is different. Therefore, if we regard the selected linear tiles as the products of the summand nodes and their respective coefficients, then the selected sets of linear tiles from equivalent R1CS constraint systems are the same. In other words, there is no difference between the selected sets of tiles for equivalent R1CS constraint systems.

Based on the selected tiles, we further abstract the data flow graph to eliminate the differences between various equivalent R1CS constraint systems. Specifically, we abstract linear tiles using a new node in the data flow graph. By doing so, we mask the difference in addition order of summands within linear tiles in the RNodeGraph and transform the relationship between external nodes and specific nodes within the linear tile into a relationship between external nodes and the tile to which the specific node belongs. In this abstracted data flow graph, the types of edges are as follows:

1. Non-linear tile abstract node to non-linear tile abstract node: The two vertices already existed in the pre-abstracted data flow graph, and this edge type remains consistent with the pre-abstracted data flow graph.

2. Non-linear tile abstract node to linear tile abstract node: This edge type exists only if there are non-abstract nodes in the linear tile represented by the abstract node.

3. Linear tile abstract node to linear tile abstract node: This edge type exists only if the two abstract nodes represent linear tiles that share common non-abstract nodes.

### Tile Weight Calculation

计算出所选出的各个瓦片的权重。在对数据流图进行进一步的抽象之后,整个图看起来简单多了，节点和边的数量也变少了很多。在具体的约束组中比较容易出现一些对称的情形，进而会让一些节点在算法中得到相同的权重。让后续的约束的排序变得比较困难。所以参考了一些文献中的做法, 使用了Weighted Pagerank算法。 使用线性瓦片中系数归一化后的方差来作为数据流图中边的权重。

After the further abstraction of the data flow graph, the entire graph appears to be much simpler, with a significantly reduced number of nodes and edges. In specific constraint groups, symmetric scenarios may arise, causing certain nodes to receive identical weights in the algorithm, thereby making subsequent constraint sorting challenging. To address this issue, we employed the Weighted Pagerank algorithm, drawing inspiration from previous literature. The edge weights in the data flow graph were determined using the coefficient-normalized variance of linear tiles.

### Adjustment of Linear Constraints

至此，约束组中约束的划分与约束的排序已经确定，在二次约束中出现过的变量的排序也已经排序完毕。但是在这个步骤中还需要对线性瓦片中新出现的变量的排序进行调整。这也是正常的,因为在前面的步骤中,为了消除RNodeGraph中的不同,我们将线性约束的瓦片的具体结构抽象化了. 因此在这个步骤我们需要一个新的方式来替在线性瓦片中新出现的变量进行排序.替每一个在线性瓦片中新加入的变量计算权重，其权重的计算方式为除去本身线性瓦片以外的其他所有线性瓦片中，自己的系数和线性瓦片权重的乘积的绝对值之和。然后便可以对新引入的变量进行排序，首先比较变量的权重, 如果一致, 再对本身在线性瓦片中的系数进行排序。在线性瓦片中引入的新变量, 其在其他 Linear 瓦片中出现的情况某种程度上反映了其在整个约束组中的重要程度。 同时如果某些新变量只在本身的约束中出现，他们的权重都将为0。 并且他们的排序只会对自身的线性瓦片所产生的约束产生影响，并不会改变其他约束的顺序，所以只需将他们按照系数降序排序即可。

At this stage, the partition and ordering of constraints within a constraint group have been established, and the ordering of variables that appeared in quadratic constraints has also been determined. However, it is necessary to adjust the ordering of newly introduced variables in linear tiles in this step. This is because, in the previous steps, the specific structures of linear tile constraints were abstracted to eliminate differences in the RNodeGraph. Therefore, a new method is required to order the newly introduced variables in linear tiles. For each new variable introduced in a linear tile, its weight is calculated as the sum of the absolute values of the products of its coefficient and the weights of other linear tiles, excluding itself. Then, the new variables can be sorted based on their weights. If the weights are the same, the coefficients of the variable in its own linear tile are considered for comparison. The appearance of new variables in other linear tiles to some extent reflects their importance in the entire constraint group. Additionally, if certain new variables only appear in their own constraints, their weights will be zero, and their ordering will only affect the constraints generated by their own linear tile, without changing the ordering of other constraints. Therefore, they can be sorted in descending order based on their coefficients alone.

## Experiment

为了评估本论文中提出的算法,我们使用python实现了范式生成的整个过程来验证他的结果。我们进行的模拟包括五个主要活动：

1. 从任意R1CS生成RNode Graph

2. 从RNode Graph中选取出瓦片的集合

3. 以瓦片为基础对RNode Graph进行抽象化处理

4. 计算瓦片的权重，以便后续对R1CS范式中的变量进行排序

5. 由瓦片生成R1CS范式

由于相关研究的缺失，目前该领域并没有一个非常完备的benchmark。于是我们根据主流Circom编译器生成R1CS的逻辑，总结了一些等价R1CS约束组生成的规律，并根据得出的规律设计出了一个较为完备的benchmark。根据所反映的情形不同，benchmark包含以下几个主要类别：

1. R1CS中变量顺序的替换

2. R1CS中约束顺序的变换

3. R1CS中单个线性约束中多个新变量的引入

4. R1CS中多个线性约束中多个新变量的引入且存在新变量共用的情况

5. R1CS中约束的合并与拆分

Benchmark中不同的类别对应的是等价R1CS生成的不同原因。经过测试，等价R1CS的差别在对RNode Graph进行抽象后均会被消除。同时后续的Weighted Pagerank Algorithm也能正确地计算出约束与变量的权重序列，进而确定R1CS范式中约束和变量的排列顺序。在实验中，所有实例均被正确转化至了唯一的范式。

## Related Work

Historically, research investigating the factors associated with X has focused on satisfiability of R1CS. Eli et al. design, implement, and evaluate a zero knowledge succinct non-interactive argument (SNARG) for Rank-1 Constraint Satisfaction (R1CS). Jonathan et al. studies zero-knowledge SNARKs for NP, where the prover incurs finite field operations to prove the satisfiability of a n -sized R1CS instance. Alexander et al. introduce Brakedown, the first built system that provides linear-time SNARKs for NP. Collectively, these studies outline a critical role for simpler and more direct provement proof. The proposal of the R1CS paradigm clearly accelerates research in this area.

Considering Circom and R1CS as two languages before and after compilation, research on the generation of R1CS paradigm is actually more akin to research on semantic consistency in compilation. Currently, both domestic and foreign patent applications and research papers propose ideas and solutions for generating compilation paradigms in other languages, mainly exploring data flow, syntax tree, or semantic mapping aspects. these studies offer crucial insights into the fundamental information that semantically identical programs entail in the process of compilation. However, due to the inherent constraints embedded within the R1CS form itself, this paper ultimately elects to use data flow as a starting point for research.

## Conclusion

R1CS是零知识证明底层工具链中不可缺少的一个部分,但是由于约束构造方式的多样性和灵活性, 对R1CS约束组的正确性和等价性的研究一直困难重重。在本论文中，一种基于数据流分析的算法被提出来构造R1CS的范式，在一系列的抽象过程后，生成的有向无环图成功消除了等价R1CS约束组之间的不同。同时，本文还提出了一系列对R1CS内部变量和约束的排序方式，为最终的范式生成提供了参考信息。实验结果表明，该算法能识别出因为约束合并、中间变量选取、变量和约束顺序调换等原因产生的等价R1CS并将它们转化至唯一的范式。

The current study was limited by 产生的范式矩阵较为稀疏，存在空间存储上的浪费。这是因为目前我们没有将约束的合并步骤放入范式生成的过程中而导致的。同时由于相关研究的缺失，目前该领域并没有一个非常完备的benchmark，文章中所整理出的benchmark难免存在一些纰漏。

Further work needs to be done to establish 约束之间合并的规则和更加完备的benchmark。这需要我们对R1CS生成规律更加深入的研究和探讨，以便持续性地改进本文中所提出的算法。

R1CS is an indispensable component in the underlying toolchain for zero-knowledge proofs. However, the correctness and equivalence of R1CS constraint systems have long been difficult to study due to the diversity and flexibility of constraint construction methods. In this paper, we propose an algorithm based on data flow analysis to construct a paradigm for R1CS, which successfully eliminates the differences between equivalent R1CS constraint systems through a series of abstraction processes. Additionally, we propose a series of ordering methods for internal variables and constraints in R1CS, providing reference information for the final paradigm generation. Experimental results demonstrate that our algorithm can identify equivalent R1CS resulting from constraint merging, intermediate variable selection, and variable and constraint reordering, and transform them into a unique paradigm.

However, our current study is limited by the sparsity of the generated paradigm matrix, resulting in storage waste. This is because we have not yet incorporated the step of constraint merging into the paradigm generation process. Furthermore, due to the lack of relevant research, there is currently no comprehensive benchmark in this field, and the benchmarks compiled in this paper may contain some omissions.

Further research is needed to establish rules for merging constraints and a more comprehensive benchmark. This requires us to conduct further research and exploration into the generation rules of R1CS in order to continuously improve the algorithms presented in this paper.