



上海交通大学学位论文

# 基于数据流分析的 R1CS 语言等价性验证 及范式生成

姓 名：施宸昊

学 号：51021910434

导 师：李国强

学 院：电子信息与电气工程学院

学科/专业名称：软件工程

申请学位层次：学士

2023 年 05 月

**A Dissertation Submitted to**  
**Shanghai Jiao Tong University for Bachelor Degree**

**DATA FLOW BASED NORMALIZATION  
GENERATION ALGORITHM OF R1CS FOR  
ZERO-KNOWLEDGE PROOF**

**Author:** Chenhao Shi  
**Supervisor:** Guoqiang Li

School of Electronic Information and Electrical Engineering  
Shanghai Jiao Tong University  
Shanghai, P.R.China  
May 17<sup>th</sup>, 2023

# 上海交通大学

## 学位论文原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全知晓本声明的法律后果由本人承担。

学位论文作者签名：施宸昊

日期：2023 年 5 月 17 日

# 上海交通大学

## 学位论文使用授权书

本人同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。

本学位论文属于：

☒ 公开论文

☐ 内部论文，保密 ☐ 1 年 / ☐ 2 年 / ☐ 3 年，过保密期后适用本授权书。

☐ 秘密论文，保密\_\_\_\_年（不超过 10 年），过保密期后适用本授权书。

☐ 机密论文，保密\_\_\_\_年（不超过 20 年），过保密期后适用本授权书。

（请在以上方框内选择打“√”）

学位论文作者签名：施宸昊

指导教师签名：李国强

日期：2023 年 5 月 17 日

日期：2023 年 5 月 17 日

## 摘 要

近年来,随着区块链技术的发展和应用,零知识证明在这个领域中变得越来越重要。同时,随着对信息隐私安全性的重视程度的提高,零知识证明在隐私方面的优势也越来越明显。在生成一个零知识证明的过程中,用户可以使用 Circom 轻松定义约束,然后使用编译器将其转换为 R1CS 格式,最后将其转换为可验证的形式。

但是,由 Circom 到 R1CS 约束这一步转换存在着诸多局限,首要问题就是 R1CS 的可合并性较差,同时由相同语义的 Circom 可以生成很多等价但形式上不相同的 R1CS,这就对生成的正确性验证带来了很大的挑战。

本论文提出了一种基于数据流分析的 R1CS 范式生成算法。它将 R1CS 转换为数据流图,并在一系列抽象和排序的操作后将等效的 R1CS 转换为相同的唯一范式。同时,本论文还根据主要 Circom 编译器底层的逻辑,总结了等价 R1CS 的生成规律,并以此整理了首个等价 R1CS 的测试集。

通过 R1CS 范式的提出,我们可以较容易地判断等价 R1CS 之间的等价性和正确性。这对我们验证程序的等价性以及正确性,包括后续更加深入研究 R1CS 的可合并性方面都具有潜在的益处。

**关键词:** 零知识证明, R1CS, 数据流图, Circom, 范式生成

## ABSTRACT

In recent years, with the development of blockchain technology, zero-knowledge proofs have become increasingly important. Simultaneously, as the importance of information privacy and security has grown, the advantages of zero-knowledge proofs in terms of privacy have become increasingly evident. During the process of generating a zero-knowledge proof, users can easily define constraints using Circom and then use a compiler to convert them into R1CS, finally transforming them into a verifiable form.

However, there are several limitations in the conversion from Circom to R1CS. The primary issue is the poor composability of R1CS, and there can be many equivalently but structurally different R1CS generated from the same semantic Circom, posing significant challenges for correctness verification of the generated proofs.

This paper proposes a data flow analysis-based R1CS paradigm generation algorithm. It transforms R1CS into a data flow graph and, through a series of abstract and sorting operations, converts equivalent R1CS into the same unique paradigm. Additionally, based on the underlying logic of the main Circom compiler, this paper summarizes the generation patterns of equivalent R1CS and compiles the first test set of equivalent R1CS.

With the introduction of the R1CS paradigm, it becomes easier to determine the equivalence and correctness between equivalent R1CS. This will greatly benefit the verification of program equivalence and correctness, including further research on the composability of R1CS.

**Key words:** Zero-knowledge proof, R1CS, Data flow graph, Circom, Normalization

## 目 录

摘 要 .....	I
ABSTRACT .....	II
第一章 绪论 .....	1
1.1 研究背景和研究意义 .....	1
1.2 国内外研究现状 .....	2
1.3 研究内容 .....	3
1.4 章节结构安排 .....	4
第二章 背景知识 .....	5
2.1 零知识证明 .....	5
2.2 算术电路描述语言 .....	7
2.3 一阶约束系统 (R1CS) .....	8
2.3.1 R1CS 的基本定义 .....	8
2.3.2 R1CS 的生成过程 .....	10
2.4 PAGERANK .....	11
2.4.1 传统 PageRank 算法 .....	11
2.4.2 加权网络排名算法 .....	13
2.5 本章小结 .....	14
第三章 问题描述 .....	15
3.1 等价 R1CS 约束组生成原因 .....	15
3.2 范式生成算法目标 .....	17
3.3 算法总体流程 .....	17
3.4 本章小结 .....	20
第四章 算法设计 .....	21

4.1 基本定义 .....	21
4.1.1 RICS 约束组 .....	21
4.1.2 RNode .....	21
4.1.3 瓦片 .....	22
4.1.4 RICS 范式 .....	22
4.2 RNode GRAPH 的生成 .....	23
4.2.1 方案一：操作符和变量使用不同节点类型存储 .....	24
4.2.2 方案二：操作符和变量使用相同节点类型存储 .....	25
4.2.3 方案对比与分析 .....	25
4.3 瓦片选取 .....	26
4.4 抽象数据流图的生成 .....	27
4.5 瓦片权重计算 .....	29
4.6 约束生成 .....	30
4.7 对线性约束的调整 .....	31
4.8 本章小结 .....	33
<b>第五章 实验 .....</b>	<b>34</b>
5.1 测试集设计 .....	34
5.2 测试结果展示与分析 .....	35
5.3 本章小结 .....	36
<b>第六章 总结与展望 .....</b>	<b>37</b>
6.1 全文总结 .....	37
6.2 研究展望 .....	39
<b>参 考 文 献 .....</b>	<b>41</b>
<b>符号与标记（附录 1） .....</b>	<b>43</b>
<b>数据流图生成伪代码（附录 2） .....</b>	<b>44</b>

瓦片分割伪代码（附录 3） .....	46
致 谢 .....	47



## 第一章 绪论

### 1.1 研究背景和研究意义

零知识证明在现代社会日益被认可其重要性,越来越多的密码学社区试图解决一些区块链最重要的挑战:隐私和可扩展性。它也是 Zcash 的基本技术之一<sup>[1]</sup>。从用户和开发者的角度来看,对信息隐私和安全性的高度重视已经引起了人们对零知识证明提供的隐私优势的更大赞赏。在这个数字化时代,个人信息已经成为了一种可被商业化和利用的商品,而随着技术的进步,这种信息泄露的风险也随之增加。因此,保护个人隐私和数据安全已经成为了当今社会亟需解决的重要问题。

尤其是在区块链领域,隐私安全和扩展性问题是大家关注的焦点。区块链是一种去中心化的技术,在这个系统中,每个参与者都能够自主地验证交易,并且所有的交易记录都是公开可见的。但是,这种公开透明的交易方式却也暴露了用户的身份和交易信息。为了保护用户的隐私,零知识证明应运而生。

零知识证明是一种可以证明某个事实或声明的方法,而不必透露与该事实或声明相关的任何其他信息。它的基本原理是让证明者通过一系列计算,向验证者证明自己拥有某种知识或权限,而不需要透露具体的信息。这种技术可以实现信息的匿名传输和保护,同时又不会泄露任何敏感信息,因此被广泛应用于密码学、金融、医疗保健、政府和监管机构等领域。

在现代社会中,对于信息隐私安全性的加剧使得零知识证明在隐私方面的优势越来越得到重视。随着去中心化金融(Decentralized Finance, DeFi)<sup>[2]</sup>使用量的增长,具有可扩展性和隐私安全性优势的零知识应用将有更多的机会提高行业的广泛采用率。

但是零知识证明并不能直接应用于任何计算问题。当我们想要使用零知识证明解决某个实际问题时,我们需要将这个问题转换成一种特定的形式,也就是二次算数程序(Quadratic Assignment Problem, QAP)<sup>[3]</sup>。这个过程涉及到将原始的复杂代码转化为简单的表达式,并将这些表达式转化为算术电路。算术电路是由“门”和“线”组成的结构,其中“门”表示运算符,而“线”表示运算数值。

一旦我们得到了算术电路,我们就需要将其转换成一阶约束系统(Rank-1 Constraint System, R1CS)的形式。这个过程涉及到将算术电路的运算结果表示为

R1CS 约束的形式，这些约束通常是由线性方程或不等式组成的。

在这个阶段之后，我们需要将 R1CS 约束进一步转化为 QAP 形式。QAP 是一种多项式间的整除性问题，可以用来表示一组约束。这个过程涉及到将 R1CS 约束表示为 QAP 的多项式形式，并将其进一步分解成更小的多项式。

最后，我们需要为这个 QAP 创建实际的零知识证明。这个过程涉及到将算术电路的可满足性问题规约为多项式间的整除性问题，并使用随机数生成器生成证据。这个证据可以被发送给验证者进行验证，而验证者只需要检查证据是否正确，而不需要知道证据本身的内容。

需要注意的是，整个过程都非常复杂，需要进行大量的计算和处理。在实践中，人们通常使用特定的软件和工具来完成这个过程，以确保零知识证明的正确性和安全性。

在零知识证明的底层工具链中由电路语言到 R1CS 约束这一步转换存在着很多局限，R1CS 描述了使用高级编程语言编写的语句的执行方式，并被许多零知识证明应用程序所使用，但目前没有标准的表示方法。这就导致 R1CS 的可合并性较差。由代码段 A 与代码段 B 合并后所生成的 R1CS 与代码段 A 和代码段 B 独立生成的两个 R1CS 从形式上看可能毫无关联，这也为我们验证所生成 R1CS 的正确性带来了很多困难。而这与 R1CS 本身表达能力局限性有关外，根本原因便是相同语义的程序本身可以生成多个等价的 R1CS 约束，R1CS 约束组中约束之间的合并与拆分也会导致 R1CS 在形式上发生变化。所以我们需要在等价的 R1CS 约束组中提出 R1CS 约束组的范式构造方式，使得对于不同的 R1CS 约束，我们可以较容易地判断其等价性和正确性。这对我们验证程序的等价性以及正确性，包括后续更加深入研究 R1CS 的可合并性方面都将大有裨益。

## 1.2 国内外研究现状

目前，国内外关于零知识证明所发表的论文，大部分集中在对算术电路可满足性的验证方面，所以导致关于 R1CS 范式生成的相关资料较少。但是，如果将电路语言与 R1CS 看作是编译前后的两种语言，将等价的 R1CS 归纳到统一的范式与程序间语义相似性的研究方向较为接近，而在这方面，国内外研究较为丰富。因此本节主要从 R1CS 可满足性研究以及编译一致性两个方面来展开介绍。

在零知识证明程序中，需要解决的问题首先要被转化成多项式，再进一步转化成

电路。这一类零知识证明体系在现今非常常见,其代表方案有 PGHR13<sup>[4]</sup>、Groth16<sup>[5]</sup>、GKMMM18<sup>[6]</sup>等。这些方案的逻辑基本上遵循以下范式:首先将计算函数转化成算术电路;然后利用 QAP。将算数电路可满足性问题规约成多项式间的整除性问题;最后设计方案,使得方案满足完备性、可靠性和零知识性。

Eli 等人为 R1CS 设计、实现并评估了一个零知识简洁非交互式论证<sup>[7]</sup>。它使用轻量级密码学,对于一个由  $n$  个约束条件组成的约束系统,它生成的可满足性的证明的大小为  $O(\log 2n)$ ,并且该证明可以在  $O(n \log n)$  的操作次数下生成,并在  $O(n)$  的时间内进行验证。相比于之前类似的零知识非交互式论证,他的效率高达十倍以上。

Jonathon 等人提出了 Cerberus,这是一个高效的交互式证明系统,将证明 R1CS 实例的可满足性的证明规模和验证者的时间减少到多棵树时间,从而得到了目前最快的且后量子安全的验证器<sup>[8]</sup>。

目前国内外的专利申请和学术论文提出了在编译过程中衡量程序相似程度的想法和解决方案,主要探索了数据流、语法树或语义映射等方面。这些研究为编译过程中语义一致程序所涉及的基本信息提供了关键见解。

袁子牧等人发明了一种基于源代码与二进制代码间的语义比对方法和装置<sup>[9]</sup>,通过提取出源代码与二进制代码间的关键变量的数据流分析生成抽象语法树,判定源代码与二进制代码之间的语义相似性。

赵长海等人提出了基于编译优化和反汇编的程序相似性检测方法<sup>[10]</sup>,通过对汇编指令集合的分析,先筛选出对程序影响不大的变量,并且从程序集合中聚类出相似的程序子集,从代码段这一更宽泛的角度上进行测评对比,对程序之间的相似性进行检测。

Aravind 等人通过捕获程序的逻辑控制流,将程序片段转换成为控制流图<sup>[11]</sup>。他们应用图神经网络来测量程序流图之间的相似性,并以此来估计程序的相似性。该方案实现了较低的错误率,具有更快的速度和更好的扩展性。

### 1.3 研究内容

本文以数据流为基础,设计了 R1CS 约束组到数据流图的生成算法。同时根据等价 R1CS 约束组生成的特点和规律,对原始的数据流图进行了分割与抽象。对 R1CS 约束组中的约束以及变量的排序提出了一系列权重计算准则,以确定 R1CS 范式中约束和变量的具体排列顺序。

通过阅读当前主流编译器的源码以及相关文档,总结出目前主流编译器的约束生成逻辑以及 R1CS 的约束表达特性,并且对等价 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测试集。目前,本文所提出的算法能通过测试集中所有的用例。也就是说,在不同情形下,等价的 R1CS 均能被转换至唯一并且相同的范式。

## 1.4 章节结构安排

本文共有六个章节,其主要内容如下所示:

第一章为绪论部分。绪论中阐述了本文的研究背景与意义,目前国内外对于 R1CS 和范式生成的研究现状,以及本文的研究内容。

第二章是对本文所用到的背景知识以及跟本研究密切相关的工作介绍。具体包括零知识证明的相关知识,其底层工具链中所应用的电路语言以及 R1CS,还有本文所提出的算法中所应用及参考的相关算法。

第三章是问题描述。介绍了等价 R1CS 约束组产生的原因,以及本文所提出的算法所要达成的目标,以及介绍输入的 R1CS 约束组在算法中各阶段具体的中间结果。

第四章是算法的基本流程。介绍了算法中各阶段的转化步骤。算法的主要步骤包括:数据流图的建立、瓦片的划分、数据流图的抽象、瓦片权重的计算以及范式的最终生成等步骤。

第五章是实验部分。在研究过程中,通过阅读主流由电路语言到 R1CS 的编译器的源码与相关文档,总结了等价 R1CS 的生成原因,并且根据原因不同整理出了五个方面的测试集。通过分析算法在这几个测试集上的表现,来评估范式生成算法的最终效果。

第六章是总结与展望,主要是总结了本文研究的一些贡献和局限性,为未来更加全面的范式生成算法提出展望。

## 第二章 背景知识

本章是对本论文的背景知识的阐述。在本章节中,首先介绍了本研究所属的领域:零知识证明;其次介绍了零知识证明中将证明者的需求转换成可计算问题的过程中与本研究密切相关的两个工具:电路语言与一阶约束系统(Rank-1 Constraint System, R1CS),并简单介绍了零知识证明中计算问题的转化过程;最后,介绍了本论文所实现的范式生成算法中处理数据流图时所借鉴的加权 Pagerank 算法。

### 2.1 零知识证明

当敏感信息需要保护时,隐私和安全性变得至关重要。在 20 世纪 80 年代初, S.Goldwasser、S.Micali 以及 C.Rackoff 提出了一种名为零知识证明 (Zero-Knowledge Proof) 的密码学协议<sup>[12]</sup>,使得证明者可以向验证者证明某个论断是正确的,而不需要泄漏任何关于被证明消息的信息。这种协议实质上是一种涉及两方或多方的协议,由一系列步骤组成,以完成某项任务。

在一个典型的零知识证明中,证明者必须证明自己拥有特定的信息,但不能将该信息直接传递给验证者。相反,证明者必须找到一种方式,通过执行一些操作,向验证者证明他们拥有所需的信息。验证者可以检查操作结果,并得出结论,而无需了解证明者拥有的任何详细信息。

零知识证明在密码学领域非常有用,并已被广泛应用于各种场景,例如数字货币交易、身份验证、电子选举等。通过将其应用于验证过程中,我们可以有效地解决许多问题,例如如何在不暴露敏感信息的情况下进行验证,以及如何保护隐私并增强安全性。

Zk-SNARKs 在 2012 年被首次提出<sup>[13]</sup>,它是现今零知识证明中应用最广泛的技术之一,它表示“Zero-Knowledge Succinct Non-Interactive Argument of Knowledge”,也就是零知识简明非交互式知识。它可以让一个人证明自己拥有某些信息,而不需要透露这些信息。相反,证明者只需向验证者提供一个非常短的证明即可。这种证明具有高度的压缩性,可以在几百个字节内完成。

Zk-SNARKs 工作的原理是将要证明的语句转换为一个计算机程序,并通过这个

程序来生成一个证明。这个证明由证明者提供给验证者，验证者可以使用这个证明来验证声明的真实性，从而达到零知识证明的目的。整个过程是非交互式的，因此可以保护隐私和安全性。尽管在安全性和隐私性方面具有显著优势，但其生成证明的计算成本较高，具体过程如图 2-1 所示。

总之，zk-SNARKs 工作原理可以分为以下步骤：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。

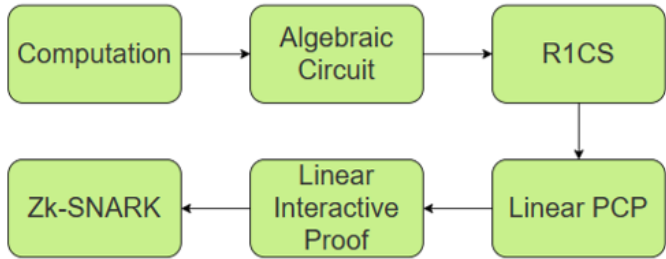


图 2-1 ZK-SNARKs 工作原理具体流程<sup>[14]</sup>

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的形式，这个公式系统表达了算术电路的约束条件，将在后续章节进行详细介绍。可验证的算术电路是 zk-SNARKs 算法的输入之一。

接下来是 QAP 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。

## 2.2 算术电路描述语言

当今数字世界中的许多重要问题,例如如何在不泄露私人信息的情况下验证身份、如何保护隐私数据不被非法利用等,都可以通过零知识证明技术来解决。其中,算术电路作为描述和计算各种复杂运算的重要工具,在零知识证明中扮演着不可或缺的角色。

在计算机科学中,电路被描述为一个三元组 $(M, L, G)$ ,其中 $M$ 是电路中变量的值, $L$ 是一组门标签,每个门标签都代表一个由 $M^i$ 到 $M$ 的一个函数(此处 $i$ 表示门的输入数量), $G$ 是一个带标签的有向无环图,图中的结点都代表 $L$ 中的一个变量<sup>[15,16]</sup>。

具体来说,算术电路由一系列逻辑门组成,可以用于描述和计算各种类型的算术运算,例如加法、乘法、除法等。通过将这些基本运算进行组合,可以构建出复杂的算术电路,用于实现各种计算任务。在零知识证明中,算术电路通常用于描述交易验证和状态转换等计算任务,例如验证一笔交易的签名是否正确、验证账户余额是否足够等。

此外,算术电路还可以用于实现零知识证明系统中的密码学哈希函数。哈希函数是一种将任意长度的消息映射到固定长度的摘要的算法,具有不可逆、唯一性和抗碰撞等特性。密码学哈希函数可以用于实现数字签名、密钥派生和消息认证码等重要的密码学协议。在零知识证明中,密码学哈希函数通常用于计算证明的哈希值,以保证证明的安全性和不可伪造性。

在零知识证明体系中,存在许多常见的算术电路描述语言,包括 `Arithmetica`、`libsark DSL`、`Circom` 等,通常用于构建和验证零知识证明系统。算术电路描述语言可以描述各种类型的算术电路,例如线性约束系统(`Linear Constraint Systems, LCS`)、双线性配对(`Bilinear Pairings`)和量子电路等。在本章节中,主要介绍 `zk-SNARKs` 底层中用于描述算术电路的 `Circom` 语言。

`Circom` 是一种用于描述算术电路的领域特定语言(DSL),旨在为零知识证明系统的开发和部署提供方便和高效的工具<sup>[17]</sup>。它基于 `JavaScript` 语言,可以轻松地与其他区块链和密码学工具集成,提供了一种简洁而强大的方式来描述和实现各种算术电路。

`Circom` 具有许多优点,其中之一是它具有高度模块化的结构,使得电路的设计和实现变得非常简单和灵活<sup>[18]</sup>。用户可以通过 `Circom` 内置的语法和函数来描述各种

基本运算，例如加法、乘法、除法等，也可以通过组合这些基本运算来构建出复杂的算术电路，以实现各种计算任务。此外，Circom 还支持用户自定义函数和类型，以满足不同的需求。

另一个重要的优点是 Circom 提供了一个自动优化工具，可以对电路进行优化和简化，以提高性能和效率。优化工具会自动检测和消除无用的计算和冗余的逻辑门，从而减少电路的大小和运行时间，并提高证明的速度和可靠性。这使得开发人员可以专注于电路的功能和逻辑，而无需担心性能和优化问题。

此外，Circom 还支持零知识证明系统中的一些关键功能，例如范围证明、批量验证、证明可组合性等。这些功能可以使零知识证明系统更加安全、高效和灵活，满足不同的应用场景和需求。

最后，Circom 还提供了丰富的文档和示例，帮助用户快速了解和使用该语言。用户可以通过阅读文档和参考示例来学习 Circom 的基本语法和使用方法，也可以通过讨论和交流社区来获取更多的支持和帮助。

总之，Circom 是一种强大而灵活的算术电路描述语言，它为零知识证明系统的开发和部署提供了方便和高效的工具。通过 Circom，开发人员可以轻松地实现各种复杂的算术运算和逻辑，并将其集成到零知识证明系统中，以实现各种实际应用场景。

## 2.3 一阶约束系统 (R1CS)

### 2.3.1 R1CS 的基本定义

Rank-1 Constraint Systems (R1CS) 是一种用于描述约束系统的数学模型。在零知识证明系统中，R1CS 是最常用的约束系统模型之一，因为它可以转换为 QAP 和零知识证明系统的 SNARKs (Succinct Non-Interactive Arguments of Knowledge) 和 STARKs (Scalable Transparent ARguments of Knowledge)。

R1CS 是由一系列三元向量组  $(\vec{a}, \vec{b}, \vec{c})$ ，而对于每一个 R1CS 约束组，都存在一个解向量  $\vec{s}$ ，其中  $\vec{s}$  必须满足等式：

$$\langle \vec{s}, \vec{a} \rangle * \langle \vec{s}, \vec{b} \rangle = \langle \vec{s}, \vec{c} \rangle \quad (2.1)$$

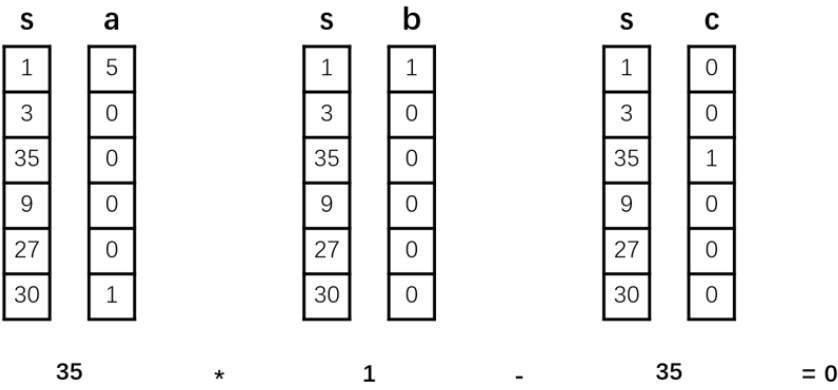
此处

$$\langle \vec{s}, \vec{a} \rangle = s_1 \cdot a_1 + s_2 \cdot a_2 + \dots + s_n \cdot a_n \quad (2.2)$$



一个约束被满足的 R1CS 如图 2-2 所示。

在 R1CS 中，每一个三元变量组都对应一个在 Circom 构建的一个约束条件，而，



s

c

1

3

35

9

27

30

0

0

1

0

0

0

35

\*

1

-

35

= 0

图 2-2 一个约束被满足的 R1CS

解向量 $\vec{s}$ 可以看作是 R1CS 中所用到的变量的取值，而向量 $\vec{a}$ ， $\vec{b}$ ， $\vec{c}$ 中，对应下标的值则是对应在解向量 $\vec{s}$ 该下标所表示的变量在这一约束中的系数。而 R1CS 中的每一个阶约束，都对应一个电路中的乘法门。将所有约束组合起来，就得到了一个一阶约束系统。

基本上，R1CS 约束系统采用广义线性规划技术来解决问题，最终目标是找到一组变量值，使得满足所有约束并且计算结果与预期输出相匹配。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他们声称的正确性。

基本上，R1CS 约束系统采用广义线性规划技术来解决问题，最终目标是找到一组变量值，使得满足所有约束并且计算结果与预期输出相匹配。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他们声称的正确性。

与传统的零知识证明系统相比，R1CS 具有更高的灵活性和可扩展性。它能够适用于各种不同类型的问题，例如数字货币交易、区块链智能合约、隐私保护等。同时，R1CS 还可以通过零知识证明技术实现数据共享和验证，大大提高了数据的安全性和可靠性。

在实际应用中，R1CS 系统已经被广泛采用。例如，在数字货币交易中，用户需要证明自己拥有一定数量的数字货币，而不需要将具体的交易信息公开。这时，R1CS 可以作为一种有效的证明方式，保护用户的隐私。此外，R1CS 还可以用于智能合约的验证和执行、密码协议的设计等方面。

### 2.3.2 R1CS 的生成过程

在本章节中，将通过实例介绍如何从实际的代码中生成 R1CS 约束组。我们选用的例子如图 2-3 中的代码片段所示。

```
def qeval(x):
    y = x**3
    return x + y + 5
```

图 2-3 R1CS 生成实例代码片段

第一步叫做“拍平”。在这个步骤中，我们将原代码片段中可能包含的复杂运算符和表达式拆分并转化成更加简单的表达式，这些表达式由两种形式。

1.  $x = y$ ，此处  $y$  可以是变量或者常数
2.  $x = y (op) z$ ，此处  $op$  可以是  $+$ ,  $-$ ,  $\times$ ,  $\div$ ， $y$  和  $z$  可以是变量或者常数

这些化简后的表达式每一个都可以被看作是数字电路中的一个逻辑门。对于实例代码片段，这阶段的拍平操作的结果为

$$\begin{aligned} sym_1 &= x * x \\ y &= sym_1 * x \\ sym_2 &= y + x \\ \sim out &= sym_2 + 5 \end{aligned}$$

可以看出化简后的表达式与源代码片段是等价的，只是加入了一系列的新变量，来储存幂函数和连加式子中的中间结果。

然后，我们将其转换为称为一阶约束系统 (R1CS) 的内容。将逻辑门转换为三元组有一种标准的方法，具体取决于操作符的种类，以及参数是变量还是数字。每个向量的长度等于系统中所有变量的总数，包括一个虚拟变量  $\sim one$  在第一个索引位置表示数字 1、输入变量、一个虚拟变量  $\sim out$  表示输出，然后所有中间变量（如上述的  $sym_1$  和  $sym_2$ ）。向量通常会非常稀疏，因为非零元素只会填入与某个特定逻辑门相关的变量对应的槽位。

首先，需要设定好将要使用的变量映射，也就是整个 R1CS 中所用到的变量与向量中的元素的对应关系。此处的变量映射为：

$$(\sim one, x, \sim out, sym_1, y, sym_2)$$

然后，根据对应拍平步骤中产生的表达式中变量的系数，依次生成约束。为各个表达式生成的三元向量组如表 2-1 所示。

表 2-1 表达式和 R1CS 三元向量组

表达式	R1CS 三元向量组
$sym_1 = x * x$	$\vec{a} = (0, 1, 0, 0, 0, 0)$
	$\vec{b} = (0, 1, 0, 0, 0, 0)$
	$\vec{c} = (0, 0, 0, 1, 0, 0)$
$y = sym_1 * x$	$\vec{a} = (0, 0, 0, 1, 0, 0)$
	$\vec{b} = (0, 1, 0, 0, 0, 0)$
	$\vec{c} = (0, 0, 0, 0, 1, 0)$
$sym_2 = y + x$	$\vec{a} = (0, 1, 0, 0, 1, 0)$
	$\vec{b} = (1, 0, 0, 0, 0, 0)$
	$\vec{c} = (0, 0, 0, 0, 0, 1)$
$\sim out = sym_2 + 5$	$\vec{a} = (5, 0, 0, 0, 0, 1)$
	$\vec{b} = (1, 0, 0, 0, 0, 0)$
	$\vec{c} = (0, 1, 0, 0, 0, 0)$

在更加正式的定义中，R1CS 是三个矩阵(A, B, C)的集合，矩阵的每一行都由对应的拍平后的表达式组成。在这样的定义下，R1CS 所要满足的等式为

$$(A * \vec{s}^T) \cdot (B * \vec{s}^T) - (C * \vec{s}^T) = 0 \quad (2.3)$$

此处 \* 表示矩阵乘法， $\vec{s}^T$  表示解向量的转置。

## 2.4 PageRank

### 2.4.1 传统 PageRank 算法

随着网络的迅速发展，根据用户的查询向他们提供最高质量的网页变得越来越困难。其原因在于有些网页与其对自身的描述相去甚远，且有些网络链接仅仅是超链接，

仅仅为了导航而存在。因此，仅仅通过一个依靠网页的搜索引擎来寻找合适的页面内容或者利用超链接信息都是非常困难的。

为了解决上述问题，谷歌开发了 PageRank 算法<sup>[19]</sup>，根据网络结构来对页面进行排名。PageRank 算法的核心思想是，一个网页的权重取决于链接到它的其他网页的数量和质量。它通过标题标签和关键词等因素，检索与给定查询相关的页面列表。然后，它使用 PageRank 来调整结果，以便在页面列表的顶部提供更多更重要的页面。

该算法指出，如果一个页面有重要的链接，那么它与其他页面的链接也会变得重要。因此 PageRank 将反向链接考虑在内，并通过链接传递排名：如果一个页面的反向链接的排名之和得到，那么该页面就有很高的排名。图 2-4 显示了以恶搞反向链接的例子：A 页是 B 页和 C 页反向链接，而 B 页和 C 页是 D 页的反向链接。

PageRank 算法的主要步骤如下：

1. 构建图结构：首先，需要将互联网上的网页和链接转换成图结构。在该结构中，每个网页对应一个节点，每个链接对应一个指向连接的网页的有向边。
2. 计算每个页面的初始得分：在 PageRank 中，每个页面的初始得分都设置为

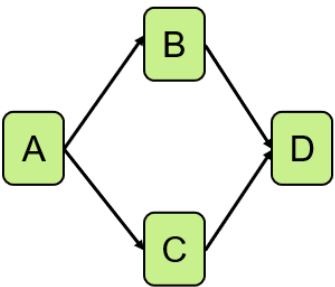


图 2-4 一个反向链接的例子

1. 这意味着，最初，每个节点具有相等的得分。
3. 迭代计算每个页面的得分：每个节点的得分是根据其传入链接进行迭代计算

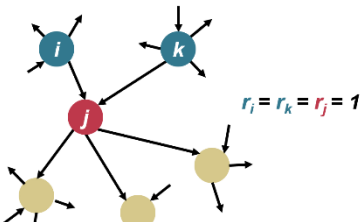


图 2-5 PageRank 算法的初始阶段

的，并在每次迭代中平均到其传出链接上。迭代公式为：

$$PR(u) = (1 - d) + d * \sum_{v \in B(u)} \frac{PR(v)}{N_v} \quad (2.4)$$

其中  $u$  表示一个网页， $B(u)$ 是指向  $u$  的页面集合。 $PR(u)$ 和 $PR(v)$ 分别是页面  $u$  和  $v$  的排名分数。 $N_v$ 表示  $v$  的外向链接数。 $d$  是一个阻尼系数，通常被设为 0.85，也可以把  $d$  看作是用户跟踪链接的频率，可以把  $1 - d$  看作是来非直接链接的页面的贡献。图 2-6 展示了 PageRank 迭代的过程。

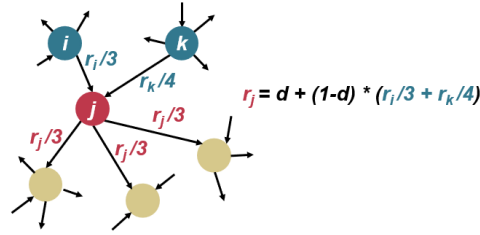


图 2-6 PageRank 算法的迭代过程

4. 考虑链接的数量和质量：除了节点之间的关系，PageRank 还考虑指向网页的链接的数量和质量。来自高质量网站的链接可能比低质量网站的链接更有价值。因此，在计算得分时，算法根据链接的数量和质量对链接进行加权。
5. 迭代直到收敛：当节点的得分稳定时，算法停止迭代。这表示所有节点的最终得分已经确定，可以用于排名搜索结果。

#### 2.4.2 加权网络排名算法

在实际的网络中，一个网络中的一些链接可能比其他的更重要。针对这个特性，Ali 等人提出了加权 PageRank 算法<sup>[20]</sup>。该算法没有将一个页面的排名值平均分配给它的外链界面，而是分配给更重要的页面。每个外链页面得到的数值和它的受欢迎程度，也就是它的内链和外链数量，成正比。从内链和外链来看，受欢迎程度分别被记作  $W_{(u,v)}^{in}$  和  $W_{(u,v)}^{out}$ ，计算公式分别为

$$W_{(u,v)}^{in} = \sum_{p \in R(v)} \frac{I_u}{I_p} \quad (2.5)$$

$$W_{(u,v)}^{out} = \sum_{p \in R(v)} \frac{O_u}{O_p} \quad (2.6)$$

其中  $I_u$  和  $I_p$  分别代表页面  $u$  和页面  $p$  的内链数量。 $O_u$  和  $O_p$  分别代表页面  $u$  和页面  $p$  的外链数量。 $R(v)$  表示页面  $v$  的参考页面，也就是边  $(u, v)$  中页面  $u$  所有外链的目标页面的集合。

考虑到页面的重要性，最初的 PageRank 公式被修改为

$$PR(u) = (1 - d) + d * \sum_{v \in B(u)} PR(v) W_{(u,v)}^{in} W_{(u,v)}^{out} \quad (2.7)$$

经过测试，与传统的 PageRank 算法相比，加权 PageRank 算法能够识别更多的与给定查询相关的网页。

在本论文所提出的算法中，采用了调整后的加权 PageRank 算法，来计算数据流图中的各个节点的权重，以便对生成的范式中的约束和变量进行排序。

## 2.5 本章小结

本章主要介绍了本论文的一些背景知识。首先有本文与零知识证明有着非常紧密的关系，介绍了本文设计的零知识证明体系 zK-SNARKs 以及其它几种常见的零知识证明体系。其次，由于零知识证明底层的工作原理非常的复杂，所以又详细介绍了与本文研究内容密切相关的两个工具：数字电路和一阶约束系统。最后，介绍了本文所提出算法中在分析数据流图时进行参考的 PageRank 算法以及改良后的加权 PageRank 算法。

## 第三章 问题描述

本章节主要介绍在将电路语言编译成为 R1CS 约束组的过程中存在的限制与问题,介绍了等价 R1CS 约束组及其产生的原因,指出了本文所提出的算法所要解决的问题,同时还列举了一个 R1CS 约束组在算法中的处理过程,以便更直观地理解算法中每个步骤的输入、输出以及对各种情况的处理方式。

### 3.1 等价 R1CS 约束组生成原因

在零知识证明底层的实现中,往往首先要将所要解决的问题转化成数字电路,在 zk-SNARKs 中,就需要将具体的计算问题转化到数字电路,再由编译器将数字电路转化成 R1CS 约束组。

在数字电路中,变量的值是用电路门之间的连线表示的。但是变量的值会随着时间变化,电路中连线的值却是固定不能改变的。因此在将计算问题转化到数字电路的过程中,需要引入大量的中间变量表述随循环而不断变化的各阶段的变量值。图 3-1 展示了一个程序片段添加中间变量前后的变化以及相对应的数字电路结构。

而编译器将数字电路转化至 R1CS 这一步骤,实际上就是以编译器内部的逻辑,对数据电路的结构进行分割,并且将分割下来的每一个子图转化为一阶约束,最后将这些子图所转化出的一阶约束合并在一起,得到完整的 R1CS。

显然,在分割这一步,存在着很多逻辑上不确定的问题,R1CS 并没有一个明确的表达形式<sup>[21]</sup>,所以对数字电路不同的分割方式,自然能产生不同但等价的 R1CS。对数字电路最简单且直接的分割方法显而易见,那就是将每个乘法门和加法门都单独分割开来,但是这样一来,最后生成的 R1CS 约束组中会含有大量的约束与中间变量,导致系数矩阵非常的系数。因此主流的电路语言编译器会在门与门之间进行合并,以生成更加紧凑的 R1CS 约束组。根据 R1CS 所要满足的等式的特性,乘法门可以与加法门合并,加法门可以与相邻的加法门的合并。但是由于不同编译器之间对于该合并的逻辑实现不同,中间变量的选择与合并的结果也往往不同,最终导致相同的数字电路,所分割出的约束表达也不同。同时等价 R1CS 约束组这种在形式上的多样性也使对 R1CS 正确性的判别难度大大提高。

除了约束间不同的合并策略之外,在电路语言中,一些等价的程序变化也会引起

```

1 func example(a, b int) (aOUT, bOUT int) {
2   for i := 0; i < 5; i++ {
3     a += b
4     b *= a
5   }
6   aOUT = a
7   bOUT = b
8   return
9 }

```

```

1 func example(a, b int) (aOUT, bOUT int) {
2   var tmp1, tmp2, tmp3, tmp4, tmp5, tmp6, tmp7, tmp8 int
3   tmp1 = a + b
4   tmp2 = b * tmp1
5   tmp3 = tmp1 + tmp2
6   tmp4 = tmp2 * tmp3
7   tmp5 = tmp3 + tmp4
8   tmp6 = tmp4 * tmp5
9   tmp7 = tmp5 + tmp6
10  tmp8 = tmp6 * tmp7
11  aOUT = tmp7 + tmp8
12  bOUT = tmp8 * aOUT
13  return
14 }

```

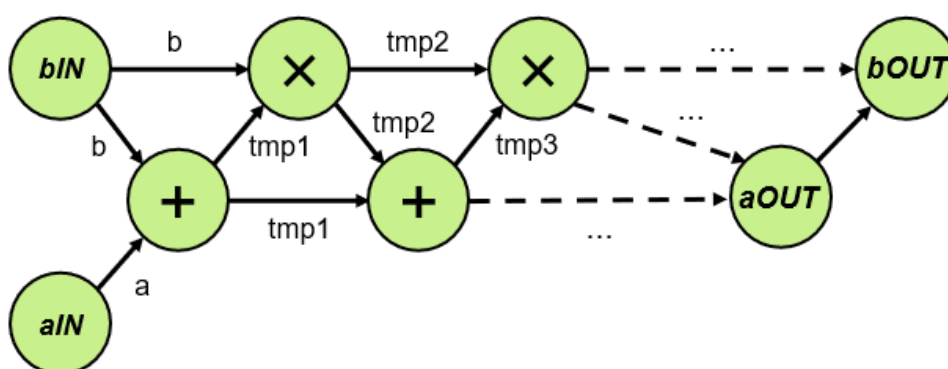


图 3-1 添加中间变量前后的程序片段以及相对应的数字电路

所编译出的 R1CS 约束组有所不同，而由于编译器直接与数字电路描述语言为输入，



并不能很好的识别语义上的等价性。比如，改变电路语言中信号量的定义顺序，对整个程序的语义来说显然是没有影响的，但这一调整往往会改变该变量在整个程序中权重，进而影响变量与所生成的 R1CS 约束组解向量中的映射关系，进一步改变约束之间的合并选择以及约束在 R1CS 约束组中的排列顺序。

### 3.2 范式生成算法目标

本文旨在提出基于数据流分析的 R1CS 约束组的范式生成算法，使得对输入的等价 R1CS 约束组，经过一系列的抽象与转换，能最终输出相同且唯一的范式。根据当前主流编译器中所存在的局限性，本算法需要解决以下几个问题：

1. 根据输入的 R1CS 约束组生成数据流图，以重现变量之间的逻辑关系。
2. 对数据流图的统一化划分，以确定 R1CS 范式中约束之间的合并与分割情况。
3. 对约束的顺序以及 R1CS 解向量中变量的映射顺序提出排序标准，以解决 R1CS 约束组三元矩阵组中的三个矩阵同时进行相同行列变化时所产生的等价 R1CS 约束组的凡是生成问题。

### 3.3 算法总体流程

在本节中，我们将通过 Vitalik 在他的博客<sup>[14]</sup>中介绍的 R1CS,介绍任意输入的 R1CS 在算法流程中被转化成范式的过程。约束组如下所示：

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 5 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$B = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$C = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

首先,算术树生成过程包括为输入的 R1CS 约束组中的每个约束创建一个算术树,然后将它们合并在一起。得到的算术树中包含公共算式树,以有向无环图的结构进行存储。输入的约束组所构建的数据流图的结构如图 3-2 所示,它说明了如何将算术树组合成数据流图。

随后,基于数据流图实现了一个瓦片选择算法,它将图形划分为若干个合法瓦片。瓦片选择过程的具体内容,包括瓦片的合法形式和选择逻辑,将在后续章节中详细阐

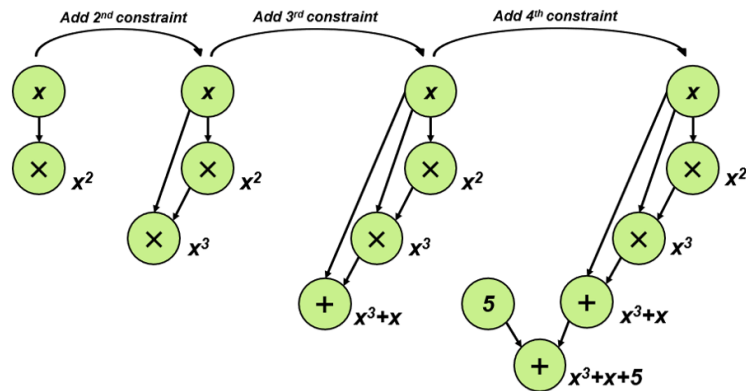


图 3-2 R1CS 转化成数据流图的过程

述。将整个数据流图划分为瓷砖瓦片的过程如图 3-3 所示,展示了瓦片选择算法的总体过程。

接下来,随着对瓷砖的选择作为参考,数据流图进一步抽象化。由瓷砖表示的线性约束被一个新的抽象节点替换在数据流图中。抽象节点可以表示为一个映射,它保留了原本瓦片中所含的变量之间的线性关系,使得在范式生成过程中中间结果的计算更快。这种抽象化过程简化了 R1CS 范式生成过程,并降低了生成 R1CS 范式所需要的计算成本。图 3-4 展示了数据流图抽象的过程。

接着,我们使用改进的加权 PageRank 算法计算选定的单个瓷砖的权重。我们使用约束的系数来计算每个节点的权重。图 3-5 展示了抽象图中四个节点的 PageRank 值的收敛过程。

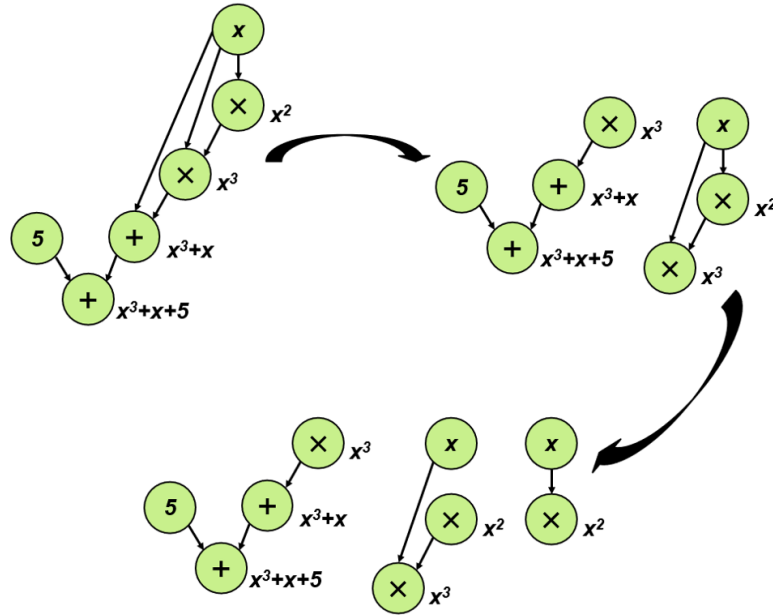


图 3-3 数据流图被划分为瓦片集合的过程

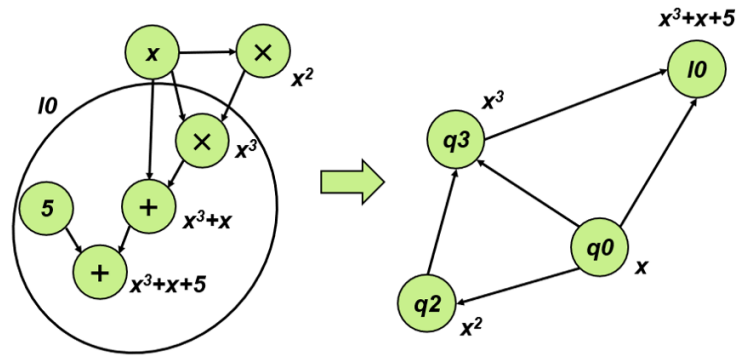


图 3-4 数据流图的抽象过程

最后，对于每个瓦片，都会单独生成符合 R1CS 范式的约束。根据之前计算出的节点权重，约束和变量会被排序。

现在，我们将输入的 R1CS 转换为其范式：

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 5 & 1 & 0 & 1 & -1 \end{pmatrix}$$

$$B = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$C = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

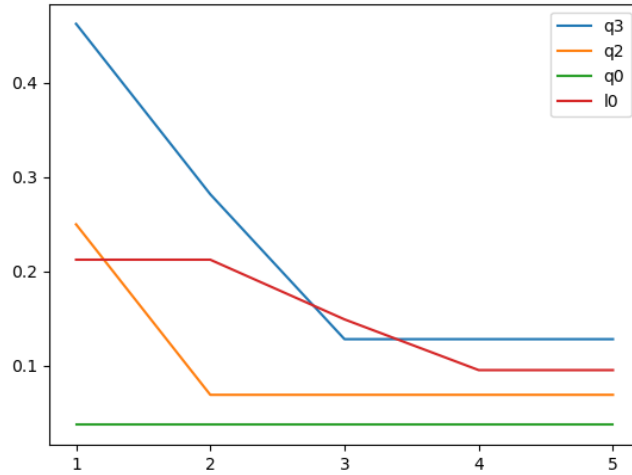


图 3-5 抽象后的数据流图中各节点的加权 PageRank 得分收敛过程

### 3.4 本章小结

本章主要介绍了不同的数字电路编译器所产生的等价 R1CS 在形式上存在很大不同以及等价 R1CS 约束组产生在编译器底层逻辑层面上的原因，并通过举例说明了即使是相同语义的电路语言程序片段，也能编译产生不同的 R1CS 约束组。同时根据编译器底层逻辑的不足之处，提出了本文算法所需要达成的目标。最后通过一个简单的例子初步介绍了算法的整体处理过程以及各个中间阶段的转换结果。

## 第四章 算法设计

本章主要介绍了本论文中所提出的 R1CS 范式生成算法。首先介绍了本算法中使用的数据结构。然后详细介绍了算法各个步骤的具体输入与输出，以及该算法是如何在抽象过程中消除等价 R1CS 约束组之间的不同之处的。

### 4.1 基本定义

本章节主要介绍本论文所提出算法的过程中，使用到的一些特殊的数据结构和概念，以便对后续算法具体步骤进行理解。同时也对本论文所提出的 R1CS 范式的具体要求进行了介绍。

#### 4.1.1 R1CS 约束组

R1CS 约束组是三个矩阵( $A$ ,  $B$ ,  $C$ )的集合，矩阵的每一行表示一个单独的约束，同时矩阵中的值表示对应变量的系数。每个 R1CS 都含有一个解向量 $\vec{s}$ ，其中存储了 R1CS 约束组中所涉及的变量和矩阵列数下标的映射关系。在这样的定义下，R1CS 所要满足的等式为

$$(A * \vec{s}^T) \cdot (B * \vec{s}^T) - (C * \vec{s}^T) = 0 \quad (4.1)$$

此处  $*$  表示矩阵乘法， $\vec{s}^T$  表示解向量的转置。

在本论文中，将 R1CS 约束组中的约束分为了两种类型：

1. 线性约束：只需使用常数和变量之间的乘法和加法即可表示的约束。比如  $a + b = c$  和  $5 \times a + b = d$ 。
2. 二次约束：必须使用变量和变量之间的乘法才能表示的约束。比如  $a \times b = c$  和  $a \times b + c + d = f$

#### 4.1.2 RNode

在算法中，设计了 RNode 数据结构，用于存储 R1CS 约束组中的基本信息。在生成的数据流图中，存在着两种类型的 RNode 类型，一种存储 R1CS 约束组解向量 $\vec{s}$

中所存储的变量映射中的变量,以及在创建数据流图的过程中创建的,用于存储中间变量的结点;另一种则是存储构建数据流图的过程中所要用到的常数的节点。

表达变量的 RNode 可以被表达成一个三元组:

$$VarNode = \{Operation, Child, Father\}$$

表达常数的 RNode 可以被表达成一个四元组:

$$ConstNode = \{ConstValue, Operation, Child, Father\}$$

在数据流图的构造过程中,常数并不与 R1CS 约束组中的某一行对应,而是在构建过程中需要存储常数项或者系数是创建,因此在这一定义下,代表常数的 RNode 不可能同时拥有父结点与子结点。

RNode 是本论文所创建的数据流图的顶点的数据结构,因此本论文中数据流图又叫 RNode Graph。

### 4.1.3 瓦片

瓦片这一概念借鉴了编译过程中指令选择中的概念。

在编译过程中,首先将编译语言转换成中间表示语言树(以下简称 IR 树)后,可以将一条机器指令表示成 IR 树的一段树枝。使用基于树的中间表示来实现指令选择的基本思想是,用一些瓦片来覆盖 IR 树,其中瓦片是与合法机器指令对应的树型。

与指令选取步骤中的概念类似,瓦片也是整个由 R1CS 所构造出的数据流图的一个树状子图,它与对数据流图分割完毕后的瓦片集合中的一个元素相对应。在本论文所提出的算法中,也设计了一些合法的瓦片类型,并在瓦片分割阶段使用这些类型的瓦片来覆盖语法树。

在算法的后半部分,将以瓦片为单位进行数据流图的抽象、权重的计算以及 R1CS 约束的生成与调整。对于瓦片的合法类型将在后面的章节进行更加详细的介绍。

### 4.1.4 R1CS 范式

通过对约束的形式,变量约束的排序方式进行限制,我们在本论文中提出了 R1CS 的范式。总的来说, R1CS 约束组的范式需要满足以下几个要求:

1. 如果 R1CS 的范式约束组的某约束中含有变量与变量之间的乘法,那么该约束中不能含有其他运算符。

2. 如果 R1CS 的范式约束组中的某约束中不含变量与变量之间的乘法, 那么该约束中不得含有由其他线性约束所生成的中间变量。
3. R1CS 的范式约束组中的约束以及变量的排序需要与本论文中所提出的方式的排序结果一致。

下面将通过具体约束的实例, 解释为了从普通的约束组转化成 R1CS 范式, 需要根据每个要求做出什么调整。

要求 1 表明, 对于 R1CS 约束组中过于复杂的二次约束, 需要进行拆分。比如

$$\begin{aligned} a \times b + c + d = f & \implies a \times b = r, r + c + d = f \\ 5 \times a \times b = c & \implies 5 \times a = r, r \times b = c \end{aligned}$$

要求 2 表明, 对于 R1CS 约束组中的线性约束, 需要消去其中线性约束所定义的中间变量, 并消除定义中间变量的线性约束。比如

$$a + b = c, c + d = e \implies a + b + d = e$$

要求 3 中排序的具体方式将在后续对算法的具体步骤介绍的章节中进行具体的介绍。

## 4.2 RNode Graph 的生成

本章节主要介绍从 R1CS 约束组生成 RNode Graph, 也就是数据流图的过程。在算法的过程中所建立的算式树相对传统算式树比较特殊。RNode Graph 生成主要分为三个步骤:

1. 将每一个约束按照 R1CS 约束需要满足的等式  $(A * \vec{s}^T) \cdot (B * \vec{s}^T) = (C * \vec{s}^T)$  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。

在本章节中通过两种方案的对比来体现最终所选择方案的优越性与便利性。

4.2.1 方案一：操作符和变量使用不同节点类型存储

将操作符和变量使用不同的顶点存储，看起来比较符合逻辑，但是在生成 R1CS 约束组对应的数据流图时，却存在着比较明显的问题，那就是在对于因为约束拆分或者合并的原因而产生的等价 R1CS 约束组时，数据流图的结构会产生比较明显的变化。

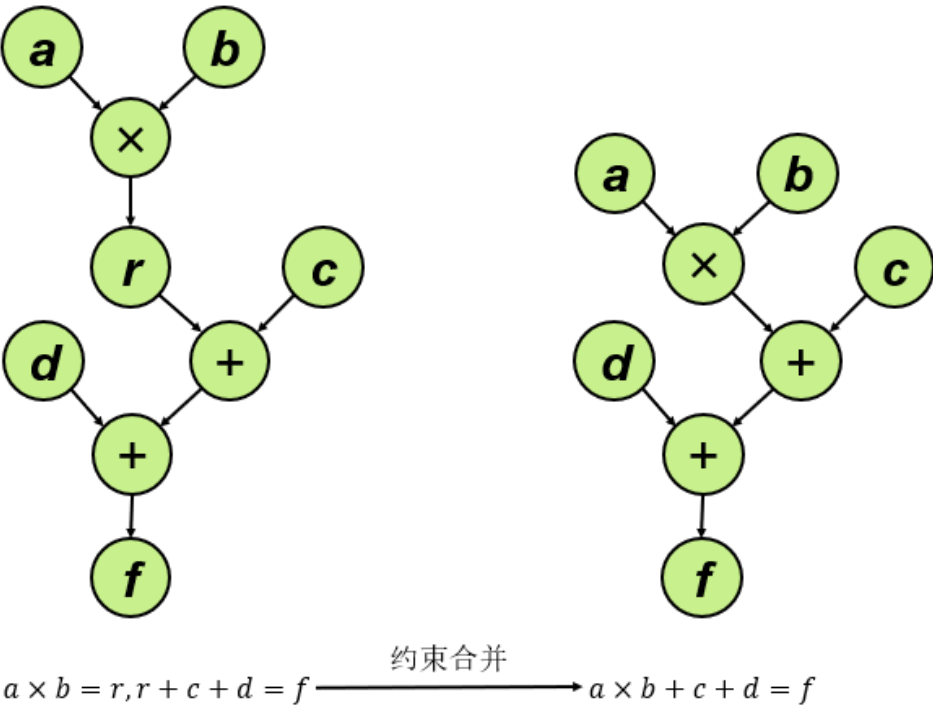


图 4-1 操作符和变量使用两种结点表示时约束合并与拆分对数据流图造成的影响

图 4-1 展示了一个约束经过合并前后，在该方案下数据流图产生的变化。将操作符和变量使用不同的定点类型存储，生成数据流图的过程比较直观，因为对于任何操作符的顶点，他在数据流图中的前继和后续顶点确定起来比较方便，但是由于约束拆分时会引入中间变量，而这一变化将进而导致数据流图的结构发生变化，进而导致后续瓦片分割以及抽象时的算法变得更加复杂，实现难度更加大。



#### 4.2.2 方案二：操作符和变量使用相同节点类型存储

将操作符和变量使用相同顶点类型储存，在这一方案下，一个节点既是一个操作符，同时也代表着以它为根的算式子树的结果。虽然这样会使单一结点所需要存储的信息量增加，但是，这样的设计能显著减少约束间的合并与拆分对数据流图带来的结构上的变化。

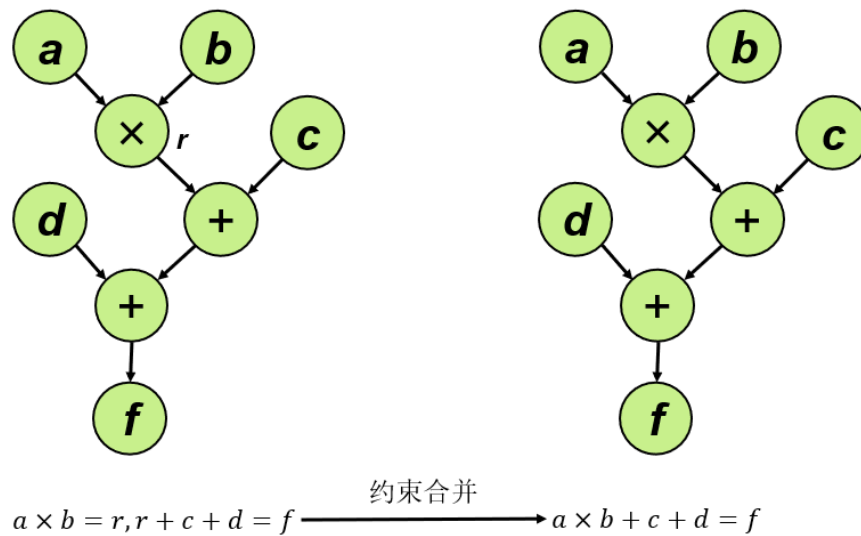


图 4-2 操作符和变量使用一种结点表示时约束合并与拆分对数据流图造成的影响

图 4-2 展示了一个约束经过合并前后，在该方案下数据流图产生的变化。月数组合并前数据流图中的  $r$  字母表示边上的乘法节点代表  $r$  这一中间变量。将操作符和变量使用相同的顶点类型存储，生成数据流图的过程较为复杂，在生成过程中需要通过相对复杂的几种顶点类型，但是这个方案相对于前述方案，能显著减少约束拆分合并时带来的中间变量选择的问题对数据流图结构上的影响，同时后续瓦片分割以及抽象时的算法也变得更加简单。

#### 4.2.3 方案对比与分析

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。

在两个方案中,我们举了同样的约束合并过程为例,展示了此类情况堆所产生数据流图在结构上的影响。而在实际情况中这种约束之间的合并与拆分正是因为 R1CS 约束组等价性难以判别的原因之一,但是在我们的算法中,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这个删去的过程如下所示

$$\begin{aligned}
 A &= \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix} B = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} C = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \\
 &\downarrow \\
 A &= (1 \quad 2 \quad 0 \quad 0) \quad B = (1 \quad 0 \quad 0 \quad 0) \quad C = (0 \quad 0 \quad 0 \quad 1) \\
 &\downarrow \\
 A &= (1 \quad 2 \quad 0) \quad B = (1 \quad 0 \quad 0) \quad C = (0 \quad 0 \quad 1)
 \end{aligned}$$

在这个过程中,合并前的约束组中的第二个约束中含有第一个约束产生的中间变量,因此可以将中间变量代入第二个约束中,并且删去第一个约束以及约束中中间变量相对应的系数,从而得到了一个等价的约束组。但是删去的这个变量会在建立 RNode Graph 时,作为连加式中的中间节点被加入到 RNode Graph 中。反之亦然。

然而等价的 R1CS 约束组生成的 RNode Graph 并不总是相同,其主要的不同在于,在构造类似连加等式的线性约束时,变量之间相加的前后顺序不同,进而导致在数据流图中该部分结构不同,这是由于约束组中变量与下标的映射关系不同而导致在约束中的排序不同,而在这一阶段我们没有足够的信息去判别加法执行的顺序,算法只能随机的选取约束中的两个变量进行相加。因此对于这种情况,我们需要对 RNode Graph 进行进一步的抽象,以消除这一不同之处。数据流图生成算法的伪代码详见附录 2。

### 4.3 瓦片选取

本章节主要介绍从将上一个步骤生成的数据流图分割为瓦片的过程。

这里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. **MulLinear**: 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  $x^3 + 5 + x = z$ 。

三种瓦片的示意图如图 4-3 所示。**Quadratic** 类型的瓦片被限制在最简单那的形式, 只允许两个变量之间的相乘, 并且不允许乘数与常数相乘。而线性约束则是尽可能的扩大范围, 以消除约束中中间变量的存在。具体到算法中, 线性的瓦片只有在遇见没有前继结点或者由 **Quadratic** 瓦片中表示结果的节点才会停止。论文中定义了两种线性瓦片 **AddLinear** 和 **MulLinear**。这两种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对于两个瓦片的逻辑完全不同, 所以在此将其分为两个种类讨论。

在瓦片选择时, 我们将输入到算法中的 R1CS 所构造出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。

瓦片生成的伪代码详见附录 3。

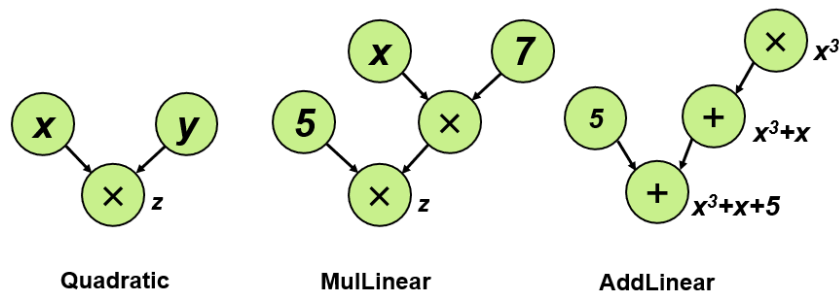


图 4-3 瓦片结构示意图

#### 4.4 抽象数据流图的生成

本章节主要介绍以瓦片为基础, 对数据流图进行抽象的步骤。

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, 但是如果在瓦片分割步骤完成后, 在一个线

性瓦片中所包含的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将各节点加入线性瓦片的顺序的不同,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。

因此我们要以选出的瓦片为基础,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。具体的步骤为对线性瓦片进行进一步的抽象,将整个线性瓦片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,我们对 RNode Graph 中线性瓦片外部的节点屏蔽了线性瓦片内部具体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。这样一来,之前数据流图中因为相加顺序不同导致的线性瓦片结构对数据流图的影响,以及外部节点到线性瓦片节点中具体的节点的边对数据流图的影响,都在线性瓦片抽象节点的建立后得到了统一。

抽象之后的数据流图与抽象前相比,结构更加简单。在抽象后的图中存在着两种类型的节点:

1. 在 Quadratic 瓦片中出现过的节点,将不用被抽象,保留在抽象后的数据流图中,因为它们不会收到线性瓦片内部具体结构的影响。
2. 线性瓦片所抽象出的节点,代表整个线性约束,但是向外界屏蔽了线性瓦片内部的具体结构。

而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。

由于在抽象过程中,我们将外部节点到线性瓦片的内部结点的有向边,抽象成了到与线性瓦片相对应的抽象节点的边,在数据流图中消去了线性瓦片具体结构导致的图结构的不同,消除了等价 R1CS 约束组所产生的数据流图之间的不同,从而在计算各个约束权重的过程中得到一致的结果。

## 4.5 瓦片权重计算

在本步骤中,使用加权 PageRank 算法计算出抽象后的数据流图中各顶点的得分。

先前的步骤消除了等价 R1CS 所生成的数据流图的不同。通过对线性瓦片的抽象,我们得到了相同的抽象后的数据流图。

在本步骤中,将以瓦片为单位生成约束,还需要提出一个瓦片排序的准则,以便为所生成的约束进行排序。在本论文所提出的算法中,我们使用加权 PageRank 算法计算出抽象数据流图中每个节点的权重,进而以结点的权重为基础计算瓦片相对应的约束的权重。相对于传统的 PageRank 算法,该算法为图中的每一条边都赋予了权重,并调整了节点权重的迭代公式。在加权 PageRank 算法中,节点得分的计算公式为

$$PR(u) = (1 - d) + d * \sum_{v \in B(u)} PR(v) W_{(u,v)}^{in} W_{(u,v)}^{out} \quad (4.2)$$

其中,  $W_{(u,v)}^{in}$  和  $W_{(u,v)}^{out}$  分别是按照出度和入度计算的节点权重。

本算法使用加权 PageRank 的主要目的,是为了降低抽象后的数据流图的对称性。由于前一步骤中,对线性瓦片进行了简化,导致数据流图的结构得到了大幅度的简化,在图中的部分结构上存在对称的情况。如果用一般的算法计算图中节点的权重,在图结构中对称存在的节点,容易计算出相同的权重,对后续对约束以及排序的问题造成困扰,所以利用加权 PageRank 算法为不同的节点计算出权重,增加图的不对称性,从而尽可能避免相同得分的出现。

在本算法中,对加权 PageRank 中得分迭代公式进行了进一步的调整。将原数据流图中保留的节点权重设为 1,而对于由线性约束抽象而来的节点,我们通过一系列步骤计算其权重。

首先,对于线性约束

$$\sum_{i=1}^n a_i b_i = c \quad (4.3)$$

将其转化为

$$\sum_{i=1}^n a_i b_i - c = 0 \quad (4.4)$$

最后使用归一化后的系数的方差来充当该线性约束的抽象节点的权重

$$W = \frac{\sum_{i=1}^n \left( a_i - \frac{\sum_{i=1}^n a_i - 1}{n+1} \right)^2 + \left( -1 - \frac{\sum_{i=1}^n a_i - 1}{n+1} \right)^2}{\left( \frac{\sum_{i=1}^n a_i - 1}{n+1} \right)^2} \quad (4.5)$$

在本算法中加权 PageRank 算法的节点得分迭代公式为

$$PR(u) = (1 - d) + d * \sum_{v \in B(u)} PR(v) W_u W_v \quad (4.6)$$

计算出各节点的得分后，以得分的高低确定 R1CS 范式中各约束的排序情况。

## 4.6 约束生成

在这一步骤中，将以前一个步骤中计算出的数据流图中顶点的权重为基础，按次序对瓦片生成的约束进行排序。在抽象后的数据流图中，存在两种节点，一种是抽象前即存在的、代表变量的节点，另一种是线性瓦片的抽象节点。

对于线性瓦片，可以直接使用其对应抽象节点在 PageRank 算法所计算出的得分作为线性约束的权重。但是对于二次瓦片，在数据流图中并没有直接对应的节点。但是由于我们对二次瓦片的分割进行了严格的控制，所以二次瓦片的形式一定是由三个代表变量的顶点组成：两个代表乘数、一个代表乘积。同时在二次瓦片中出现过的顶点也均在抽象过程之后得到了保留。于是可以通过

$$W(tile) = \frac{\sum_{node \in tile} W(node)}{3} \quad (4.7)$$

来计算出二次瓦片的权重。

计算出约束的权重之后，我们以权重降序生成约束。在 R1CS 范式中，先由二次瓦片生成约束，并将约束按瓦片的权重降序排列在 R1CS 范式的三元矩阵的上部。待二次瓦片全部生成完毕后，再由线性瓦片生成约束。

在这一阶段，通过二次瓦片确定了一部分的变量的排序。在 R1CS 范式的三元矩阵组中，与二次约束相对应的三个行向量均各自只有一个非零系数，但是在矩阵 A 和矩阵 B 中，对应的两个行向量在约束表示中是等价的，这也与乘法的交换率相符。这就导致对同一二次约束，有两种等价的表达。图 4-4 展示了一个二次约束等价表达的例子。

$$\begin{aligned}
 & a \times b = c \\
 & \text{variable mapping} = (\sim one, a, b, c) \\
 & A = (0 \quad 1 \quad 0 \quad 0) B = (0 \quad 0 \quad 1 \quad 0) C = (0 \quad 0 \quad 0 \quad 1) \\
 & A = (0 \quad 0 \quad 1 \quad 0) B = (0 \quad 1 \quad 0 \quad 0) C = (0 \quad 0 \quad 0 \quad 1)
 \end{aligned}$$

图 4-4 二次约束的两种等价表达

由于在抽象的数据流图中保留了所有的代表在二次约束中出现过的变量的顶点，所以可以依靠各个变量的权重来确定 R1CS 范式的三元矩阵组中矩阵 A 和矩阵 B 中对应的两个行向量中非零系数的选择。具体到算法中，权重更高的变量被分配至矩阵 A 中，并且会在变量映射中被分配更小的下标。

对二次约束中出现过的变量在变量映射中的排序规则可总结如下：

1. 以变量出现过的所有二次约束的权重中的最高值为基准进行排序，最高值更高的变量在变量映射中会拥有更小的下标。
2. 在同一约束中出现的且权重最高值相同的变量，以数据流图中与变量相对应的节点在加权 PageRank 算法中的得分为基准进行排序，节点得分更高的变量在变量映射中会拥有更小的下标，并会被分配至矩阵三元组中的矩阵 A 的对应行向量中。

## 4.7 对线性约束的调整

本步骤中对只在线性约束中出现过的变量进行排序。

在瓦片划分步骤中，确定了 R1CS 范式中约束的划分；在瓦片权重计算的步骤中，确定了 R1CS 范式中约束的排序方式；在前一步骤中，确定了在二次约束中出现过的

变量在变量映射中的位置。因此，距离最终的 R1CS 范式生成，只剩下那些只在线性约束中出现过的变量在变量映射中的位置还没有确定。这也是正常的现象，因为在对数据流图的抽象步骤中，消去了线性变量的内部具体结构对整体数据流图结构的影响。所以在之前的步骤中，没有足够的信息对其中的变量进行排序。

这种信息的缺失也会导致当在一个线性约束中引入多个变量时，新变量在变量映射中的顺序可能会出现混乱。图 4-9 展现了该情况下的一个例子。

因此对于这类变量，本文提出了新的权重排序方式，公式为：

$$\begin{aligned} A &= \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 5 & 1 & 0 & 1 & 2 & 1 \end{pmatrix} B = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} C = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \\ A &= \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 5 & 1 & 0 & 1 & 1 & 2 \end{pmatrix} B = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} C = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{aligned}$$

图 4-5 单线性约束中引入多个新变量时引起的变量顺序混乱

$$W(u, t') = \sum_{t \in T(u) - \{t'\}} |field(u, t) \times W(t)| \quad (4.8)$$

其中， $u$  表示需要计算权重的新节点， $t'$  表示当前引入  $u$  这一新节点的约束对应的瓦片。 $W(u)$  表示节点  $u$  的权重， $T(u)$  表示包含节点  $u$  的瓦片， $field(u, t)$  表示在瓦片  $t$  中节点  $u$  的系数。

也就是说在每一个新加入的线性瓦片中，对于每一个新引入的变量，其权重是除去本身线性瓦片以外的其他线性瓦片中该变量的系数和该瓦片权重的乘积的绝对值之和。

对于只在线性约束中使用过的节点在变量映射中的排序规则可总结如下：

1. 在该线性约束中，权重更高的新节点在变量映射中对应的下标更加小。
2. 对于权重相同的节点，进一步比较其在线性约束中的系数，系数更大的新节点在变量映射中对应的下标更小。

在线性约束中引入的新变量，在其它线性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。同时如果出现多个变量的权重相同的情况，那么说明很大可能下这些变量出现的线性约束相同，并且在这些线性约束中的系数也是相同的，那么这种情况下，这些变量在映射中的顺序对整个 R1CS 范式其实是没有影响的。图 4-6 展示了这种情况的一个例子。



$$\begin{aligned}
& \text{variable mapping} = (\sim one, x, x^2, x^3, a, b) \\
& A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 5 & 1 & 0 & 1 & 2 & 2 \\ 4 & 2 & 0 & 0 & 3 & 3 \end{pmatrix} B = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} C = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \\
& \text{variable mapping} = (\sim one, x, x^2, x^3, b, a) \\
& A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 5 & 1 & 0 & 1 & 2 & 2 \\ 4 & 2 & 0 & 0 & 3 & 3 \end{pmatrix} B = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} C = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}
\end{aligned}$$

图 4-6 不同变量映射下，单线性约束中引入多个权重相同的新变量

## 4.8 本章小结

本章主要介绍了 R1CS 范式生成算法的具体流程，并且通过两个等价 R1CS 约束组为例展示了等价约束组之间的不同之处是如何被消除的，同时对 R1CS 约束组中变量和约束的排列方式提出了一系列标准

在本章节介绍的算法中，R1CS 约束组首先被转化成一个类似于含有共有子式的算式树的数据流图；随后规定了瓦片的合法形式，并设计了图分割算法以将前一步骤中生成的数据流图分割为若干个合法形式的瓦片，确定了 R1CS 范式中约束的划分；然后以瓦片为单位，数据流图被进一步的抽象，消除了等价的 R1CS 约束组生成的数据流图中结构的不同；随后在抽象的数据流图上应用了加权 PageRank 算法，计算出了各个瓦片与其对应约束的权重；最后根据约束的类型以及权重确定了 R1CS 范式中约束的排列情况；并且对于不同约束中出现的变量提出了不同的权重计算规则，并确定了 R1CS 范式中变量映射的顺序。

## 第五章 实验

本章节主要介绍了本论文中所设计的测试集，并且展示了所实现算法在数据上的测试效果。

### 5.1 测试集设计

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结果。进行的模拟包括五个主要活动:

1. 从任意 R1CS 约束组生成 RNode Graph。
2. 从 RNode Graph 中选取瓦片的集合。
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测试集。于是通过主流数字电路编译器生成 R1CS 约束组的底层逻辑代码,以及在算法测试过程中发现的一些问题,对等价 R1CS 约束组生成的规律进行了更加细致的归纳与划分,并根据得出的规律设计出了一个较为完备的约束组。

根据所反映的情形不同,测试集中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。

测试组中不同的类别对应的总结出的是等价 R1CS 约束组生成的不同原因。测试组中的每个类别包含 2 到 3 个基础的 R1CS 约束组。为了更全面地测试算法的鲁棒性和正确性,根据类别所对应的原因对每个约束组都生成了 5 到 6 个与之等价的 R1CS 约束组。每个约束组的等价约束组都被成对地输入到算法中进行测试,以验证算法在处理不同的等价约束组时是否能够生成一致且符合前述 R1CS 范式定义的输出结果

经过测试,等价 R1CS 约束组的差别在对 RNode Graph 进行抽象后均会被消除。

同时后续的加权 PageRank 算法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到了唯一且符合定义的范式。

5.2 测试结果展示与分析

表 5-1 展示了实验结果的数据。

表 5-1 R1CS 等价约束组转化实验结果

等价 R1CS 约束生成原因	实验组数	成功生成组数	通过率
R1CS 中变量顺序的替换。	55	55	100%
R1CS 中约束顺序的变换。	21	21	100%
R1CS 中单个线性约束中多个新变量的引入。	15	15	100%
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。	15	15	100%
R1CS 中约束的合并与拆分。	6	6	100%

从实验结果上看，本论文设计的基于数据流的范式生成算法能检测到多种情况产生的等价 R1CS 约束组，并且能将它们转化至统一的范式。经过检查，所生成的范式均符合本文所定义的要求，并且语义与转化前的 R1CS 约束组均相同。

通过对转化过程各阶段的中间输出进行分析。发现对于由交换约束顺序生成的等价 R1CS 约束组，在生成的数据流图中唯一的区别在于创建表示中间变量的节点的顺序不同。这是由于遍历约束组的过程中，对 R1CS 约束组中每个约束的处理顺序不同，会导致引入各个约束中的中间变量的顺序不同。

对于通过交换变量映射关系生成的等价 R1CS 约束组，差异在于在 R1CS 中表示初始变量的 RNode 的创建顺序以及由于线性约束中变量顺序不同而导致的加法链结构中添加顺序的不同。但是，这些变化不会影响所选的 tile 集，在抽象后得到相同的数据流图。

在线性约束中引入多个新变量产生的等价 R1CS 约束组，均在对线性约束组进行抽象操作后得到了相同的数据流图，并且在对新引入变量进行排序时也得到了与定义

相符合的变量映射序列。

由约束的合并与拆分产生的等价 RICS 约束组，其产生的数据流图的不同之处在于代表中间变量的顶点所代表的到底是初始存在的变量，还是在创建数据流图过程中引入的中间变量。但是这并未对数据流图的结构造成影响。对线性约束的拆分与合并引起了数据流图中连加链的相加顺序的变化，但是也在对数据流图进行抽象后得到解决。

### 5.3 本章小结

本章节主要介绍了等价 RICS 的测试集的设计策略以及算法的实验结果，并且对算法各阶段的中间输出进行了进一步的分析与说明，均得到了与设计阶段相符合的结果。

## 第六章 总结与展望

### 6.1 全文总结

本论文中设计一个基于数据流的 R1CS 范式生成算法为主要研究线路。算法的具体流程如图 6-1 所示。算法的具体流程如下所述。

首先根据输入的 R1CS 约束组，将约束组中的每一个约束生成算式树，再将这些算式树合并。最终生成了一个带有公共子式的以 DAG 形式存储的算式树。

其次，在数据流图的基础上实现了瓦片选取算法。实现了 R1CS 范式中各个约束的选取方式。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. MulLinear: 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  $x^3 + 5 + x = z$ 。

然后，在数据流图的基础上，以瓦片的选取为参考，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。也就是对自身以外的节点，屏蔽线性瓦片内部相加的顺序，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。

接下来的步骤是参考相关论文，利用改进后的加权 PageRank 算法，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，使用了加权 PageRank 算法。使用线性瓦片中系数归一化后的方差作为数据流图中边的权重，极大地降低了抽象后的数据流图的对称性。

最后，将各个瓦片分别生成 RICS 的范式，并按照前部所计算出的节点权重对约束和变量进行排序。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

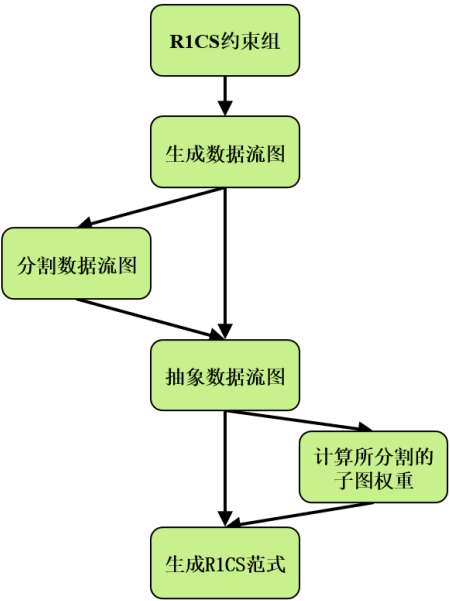


图 6-1 范式生成算法具体流程

束中出现过的变量的排序也已经排序完毕。但是由于在对数据流图的抽象过程中，线性约束的具体内部结构被忽略。因此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。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，其与其他 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。

以往对 RICS 的研究往往都是直接对原始约束组进行可满足性研究，但本文首次

创新性地提出了以数据流为基础的 R1CS 范式生成算法。从数据流的角度出发,研究等价 R1CS 约束组之间的内在联系,并提出了多条对 R1CS 范式中约束以及变量各方面性质的要求与规则,定义了 R1CS 范式的形式。

除此之外,本文还根据主流电路语言编译器的内在逻辑,总结并归纳了等价 R1CS 约束组生成的几种不同的类型。最终根据产生的规则整理出了首个等价 R1CS 相关的测试集。并且在测试集上对本论文提出的算法进行测试,在各种情形下,本算法均能将 R1CS 约束组转化至唯一且符合定义的 R1CS 范式。

## 6.2 研究展望

本文虽然提出了 R1CS 的范式并实现了转化算法,且通过测试验证了算法的实用性,但仍然存在着诸多限制。

首先,在本文所提出的算法中瓦片分割的步骤,所规定的合法瓦片形式中将二次约束的形式限定在了最简单的形式,同时,保留了线性约束中由二次约束产生的中间变量,而没有进行最后的约束合并操作。这就导致 R1CS 范式中仍然存在着许多可以消去的约束以及中间变量。未来研究中应该进一步对约束之间的合并提出新的规则,使得最终生成的 R1CS 范式的矩阵三元组更加紧凑,所使用的空间更加地高效;或者提出更加复杂的瓦片范例,在分割阶段就将可以合并的约束划分到一个瓦片中。

同时,测试集仍然不够完备。由于相关领域研究的缺失,并不存在一个公认的较完备的测试集。本论文所使用的测试集是根据主流编译器底层的约束生成逻辑中与电路语言具体实现相关的部分所整理。由于时间与精力有限,并没有对所有电路语言编译器的底层逻辑进行详细分析与总结,整理而出的测试集也难免存在纰漏。希望未来能在进一步的总结的基础上,提出规模更大、形式更加完整的等价 R1CS 约束集。

其次,算法的效率不高。本论文所实现的算法步骤较为繁琐,且存在可以并行的部分,比如数据流图的生成与瓦片划分。并且在算法的某一阶段,内存中存在着两幅数据流图,一幅是最初阶段生成的,另一幅是瓦片分割阶段所存储的瓦片,即数据流图子图的集合。这样的冗余必然带来内存上的浪费。希望未来能在保证算法正确性的基础上,对可并行步骤的执行进行进一步的优化,提高算法执行的效率,并且加强内存管理,减少算法运行时所占用的资源。

最后,本算法是在数据流图阶段对数据流图进行进一步的抽象,以消除等价 R1CS 约束组产生的数据流图之间的不同。但如果可以对所创建的数据流图提出范式,将等

价的 R1CS 约束组所产生的数据流图归约到相同的形式，那么便可以对数据流图直接进行分割，并生成 R1CS 范式，大大简化算法的流程，提高算法的效率。

本论文所提出的 R1CS 的范式，旨在解决 R1CS 可合并性较差的问题，并提高对 R1CS 约束组正确性检验的效率。未来将在通过 R1CS 范式限制 R1CS 约束组的表达形式的基础上，限制原本高自由度的 R1CS 月数组的表达。依照 R1CS 范式特殊的形式与特性，提出一系列合并的规则，实现电路语言片段所生成的 R1CS 约束组之间的合并与拆分。



## 参 考 文 献

- [1] Sasson E B, Chiesa A, Garman C, et al. Zerocash: Decentralized anonymous payments from bitcoin[C]//2014 IEEE symposium on security and privacy. IEEE, 2014: 459-474.
- [2] Jensen J R, von Wachter V, Ross O. An introduction to decentralized finance (defi)[J]. Complex Systems Informatics and Modeling Quarterly, 2021 (26): 46-54.
- [3] Simpson W. QAP: The quadratic assignment procedure[C]//North American STATA users' group meeting. 2001: 12-13.
- [4] Parno B, Howell J, Gentry C, et al. Pinocchio: Nearly practical verifiable computation[J]. Communications of the ACM, 2016, 59(2): 103-112.
- [5] Groth J. On the size of pairing-based non-interactive arguments[C]//Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35. Springer Berlin Heidelberg, 2016: 305-326.
- [6] Groth J, Kohlweiss M, Maller M, et al. Updatable and universal common reference strings with applications to zk-SNARKs[C]//Advances in Cryptology–CRYPTO 2018: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part III. Cham: Springer International Publishing, 2018: 698-728.
- [7] Ben-Sasson E, Chiesa A, Riabzev M, et al. Aurora: Transparent succinct arguments for R1CS[C]//Advances in Cryptology–EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part I 38. Springer International Publishing, 2019: 103-128.
- [8] Lee J, Setty S, Thaler J, et al. Linear-time and post-quantum zero-knowledge SNARKs for R1CS[J]. Cryptology ePrint Archive, 2021.
- [9] 袁子牧, 冯牧玥, 班固等. semantic comparison method and device between a kind of source code and binary code [P].中国专利: CN110147235A, 2019 -08-20.
- [10] 赵长海, 晏海华, 金茂忠. 基于编译优化和反汇编的程序相似性检测方法[D]. 北京航空航天大学学报, 2008.
- [11] Nair A, Roy A, Meinke K. funcgnn: A graph neural network approach to program similarity[C]//Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software

- Engineering and Measurement (ESEM). 2020: 1-11.
- [12] Goldwasser S, Micali S, Rackoff C. The knowledge complexity of interactive proof-systems[M]//Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali. 2019: 203-225.
- [13] Bitansky N, Canetti R, Chiesa A, et al. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again[C]//Proceedings of the 3rd Innovations in Theoretical Computer Science Conference. 2012: 326-349.
- [14] Vitalik Buterin. Quadratic Arithmetic Programs: from Zero to Hero[EB/OL].(2016-12-12)[2023-05-08].<https://medium.com/@VitalikButerin/quadratic-arithmetic-programs-from-zero-to-hero-f6d558cea649>
- [15] Vollmer H. Introduction to circuit complexity: a uniform approach[M]. Springer Science & Business Media, 1999.
- [16] Yang K. Integer circuit evaluation is PSPACE-complete[J]. Journal of Computer and System Sciences, 2001, 63(2): 288-303.
- [17] Albert E, Bellés-Muñoz M, Isabel M, et al. Distilling constraints in zero-knowledge protocols[C]//Computer Aided Verification: 34th International Conference, CAV 2022, Haifa, Israel, August 7–10, 2022, Proceedings, Part I. Cham: Springer International Publishing, 2022: 430-443.
- [18] Bellés-Muñoz M, Isabel M, Muñoz-Tapia J L, et al. Circom: A Circuit Description Language for Building Zero-knowledge Applications[J]. IEEE Transactions on Dependable and Secure Computing, 2022.
- [19] Page L, Brin S, Motwani R, et al. The PageRank citation ranking: Bringing order to the web: Stanford InfoLab[J]. Navigation, findability and the usage of cultural heritage on the web: an exploratory study, 1999.
- [20] Xing W, Ghorbani A. Weighted pagerank algorithm[C]//Proceedings. Second Annual Conference on Communication Networks and Services Research, 2004. IEEE, 2004: 305-314.
- [21] Guillaume Drevon. J-R1CS – a JSON Lines format for R1CS[EB/OL]. [2023-05-15]. <https://docs.zkproof.org/pages/standards/accepted-workshop2/proposal--zk-interop-jr1cs.pdf>.

## 符号与标记（附录 1）

$R(u)$	PageRank 算法中结点 $u$ 的得分
$d$	PageRank 算法中的阻尼系数
$B(u)$	结点 $u$ 的邻居节点
$N_v$	表示 $v$ 的外向链接数
$W_{(u,v)}^{in}$	在加权 PageRank 算法中，边 $(u,v)$ 根据 $u$ 的入度计算出的权重
$W_{(u,v)}^{out}$	在加权 PageRank 算法中，边 $(u,v)$ 根据 $u$ 的出度计算出的权重
$I_u$	结点 $u$ 的入度
$O_u$	结点 $u$ 的出度
$RNode$	本文所创建的数据流图中的结点的数据结构
$RNode\ Graph$	本文所创建的数据流图
$ConstNode$	本文所创建的数据流图中代表常数的结点
$VarNode$	本文所创建的数据流图中代表变量的结点
$Tile$	本文分割数据流图以及 R1CS 范式中约束生成的基本单位
$W(node)$	本文范式生成算法中使用加权 PageRank 算法计算出的结点得分
$W(tile)$	根据所含结点计算出的 $Tile$ 的权重
$W(u, t')$	R1CS 范式的线性约束中新引入的变量 $u$ 在线性瓦片 $t'$ 中的权重

## 数据流图生成伪代码（附录 2）

**Output:** A RICS  $r$  consisting of  $w$  three vectors, and the length of each vector is  $l$

**Input:** the list  $res$  of RNode in the created graph

```

    res ← ∅
    for i ← 1 to w do
        res.append(new rightnode)
    end for
    for z ← 1 to l do
        nodea ← None
        cons ← zth constraint in r
        for i ← 0 to l do
            nodea ← None
            if cons.a[i] ≠ 0 then
                if i == 0 then
                    nodea ← CreateConstNode(cons.a[i])
                    res.append(nodea)
                else
                    tmp ← CreateConstNode(cons.a[i])
                    nodea ← Multiple(tmp, res[i])
                    res.append(nodea)
                    res.append(tmp)
                end if
            end if
            for j ← 0 to l do
                nodeb ← None
                if cons.b[j] ≠ 0 then
                    if j == 0 then
                        nodeb ← CreateConstNode(cons.b[j])
                        res.append(nodeb)
                    else
                        tmp ← CreateConstNode(cons.b[j])
                        nodea ← Multiple(tmp, res[j])
                        res.append(nodeb)
                        res.append(tmp)
                    end if
                end if
                if i and j are the indices corresponding to the last non-zero elements in the constraint. then
                    if only one element in the cons.c is not zero then
                        if leftnode == None then
                            result node ← correspondingnodeofnone – zeroelementinc
                            result node.operation ← Multiple
                            result node.father ← {nodea, nodeb}
                            nodea.child.append{result node}
                            nodeb.child.append{result node}
                        else
                            rightnode ← Multiple(nodea, nodeb)
                            res.append(rightnode)
                            result node ← correspondingnodeofnone – zeroelementinc
                            result node.operation ← Add
                            result node.father ← {leftnode, rightnode}
                            leftnode.child.append{result node}
                            rightnode.child.append{result node}
                        end if
                    end if
                else
                    for k ← 0 to l do
                        if cons.c[k] ≠ 0 then
                            if i == 0 then
                                nodec ← CreateConstNode(cons.c[k])
                                res.append(nodec)
                            else
                                tmp ← CreateConstNode(cons.c[k])
                                nodec ← Multiple(tmp, res[k])
                                res.append(nodec)
                                res.append(tmp)
                            end if
                        end if
                        if rightnode == None then
                            rightnode ← nodec
                        else

```

附图 2-1 数据流图生成伪代码 1

---

```

        rightnode  $\leftarrow$  Add(rightnode, nodec);
        res.append(rightnode)
    end if
end if
end for
if leftnode == None then
    leftnode  $\leftarrow$  Multiple(nodea, nodeb)
    res.append(leftnode)
else
    tmp  $\leftarrow$  Multiple(nodea, nodeb)
    leftnode  $\leftarrow$  Add(leftnode, tmp)
    res.append(leftnode)
    res.append(tmp)
end if
end if
    gotonextconstraint
else
    if leftnode == None then
        leftnode  $\leftarrow$  Multiple(nodea, nodeb)
        res.append(leftnode)
    else
        tmp  $\leftarrow$  Multiple(nodea, nodeb)
        leftnode  $\leftarrow$  Add(leftnode, tmp)
        res.append(leftnode)
        res.append(tmp)
    end if
end if
end if
end for
end if
end for
end for
end for

```

---

附图 2-2 数据流图生成伪代码片段 2

## 瓦片分割伪代码（附录 3）

---

**Output:** the root RNode of the tile to be chosen and a flag

**Input:** set of the edges of the chosen tile, s

```

function GETTILE(r, flag, s)
    if r has no predecessor nodes then
        return
    end if
    if flag == False & r.opretion == Multiple & both father nodes of r represents constant value then
        return
    end if
    if flag == True & r.opretion == Multiple & both father nodes of r represent variables then
        s.add(< r, fatherleft >)
        s.add(< r, fatherright >)
        return
    end if
    if r.operation == Multiple then
        s.add(< r, fatherleft >)
        s.add(< r, fatherright >)
        if fatherleft represents constant value then
            GetTile(r, false, fatherleft)
        end if
        if fatherright represents constant value then
            GetTile(r, false, fatherright)
        end if
    else
        s.add(< r, fatherleft >)
        s.add(< r, fatherright >)
        GetTile(r, false, fatherleft)
        GetTile(r, false, fatherright)
    end if
end function

```

---

附图 2-3 瓦片分割伪代码

## 致 谢

在本论文完成之际，我要感谢所有在这段时间里给予我帮助和支持的人们。

首先要感谢我的指导老师李国强老师。感谢李老师在论文的选题与撰写过程中给予我充分的指导与帮助。本科期间因为些许的不自信，总是不敢与老师做太多的交流，但是李老师爽朗的笑声却让我感到格外平易近人，使我可以沉浸在做研究的快乐之中。同时还要感谢李老师给予我充分的信任，指导我在学术期刊上发表论文，在临近毕业时完成了我本科期间的一个遗憾。

其次要感谢我的父母，为我在外求学提供物质和精神上的帮助。我的父母自幼家贫，在升上高中之际均因此选择了能更早工作补贴家用的师范专科院校。家境有了起色后，他们对我的求学之路更是不留余力。从高中的选课，到本科和硕士的专业选择，从义乌到上海再到香港，我对未来的选择从来是带一些随性的。这样的自由自然是来自父母对我的任性的包容，以及他们在我迷茫沮丧时给予我的支持。随着年岁增长，我离故乡越来越远，我的生活也在逐渐远离他们熟悉的范畴，我的母亲常因为担心我整夜睡不着觉，但他们仍在每次接起我打回家的电话时，强压住心里对我的担忧，反过来安慰我。我的父母为了家庭操劳了大半辈子，现已近退休之年，他们常说我是他们最大的骄傲，希望我以后能有相应的成就，报答他们的付出与奉献。

还要感谢在大学认识的朋友们，尤其是时常在软院一起自习、同甘共苦的王廷宇、李煌政、陈佳雯、刘茁、江雨泽等几位同学。在软院的四年是繁忙的，常常为完成作业在电脑前学习到深夜。学习的苦总是难以下咽的，但是想到朋友们的鼓励与陪伴，总是会坚持着在睡前再多敲下一行代码。

最后这一段，我要单独留给我的外婆。父母在外工作不常回家，我的童年是更多是由外婆带着在田埂上度过的。外婆不识字，却教会了我劳动人民最质朴的美德，勤奋与节俭。

写完论文，更觉毕业时限将至。又一次面临我生活新阶段的开场，心中有不舍，有伤感，也有对新生活到来的期待与兴奋。四年本科，不知道因为项目和作业度过了多少难熬的日夜，但是站在终场，回想起来却只剩下珍惜和怀念。

最后的最后，感谢自己四年来的坚持，祝我毕业快乐。

# DATA FLOW BASED NORMALIZATION GENERATION ALGORITHM OF R1CS FOR ZERO-KNOWLEDGE PROOF

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 most significant challenges: privacy and scalability. From both user and developer perspectives, the heightened emphasis on information privacy and security 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.

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. The flow of the algorithm is described below.

First, an arithmetic tree is generated for each constraint in the constraint group based



on the input R1CS constraint group, and then these arithmetic trees are merged. The result is an arithmetic tree with common subformulas stored as a DAG.

Next, a tile selection algorithm is implemented on the basis of the data flow graph. The way in which each constraint in the R1CS paradigm is selected is implemented. Here I have divided the tiles into three main types:

1. Quadratic: tiles with  $x * y = z$ , where  $x, y, z$  are all variables.
2. MulLinear: tiles whose root node is obtained by multiplying its two fathers, where at least one of the two fathers is a constant, e.g.  $(5 * x) * 7 = z$ .
3. AddLinear: the tile obtained by adding the root node by its two fathers, e.g.  $x^3 + 5 + x = z$ .

The dataflow graph is then further abstracted with the selection of tiles as a reference, further eliminating the differences between the various equivalent R1CS constraint groups at the dataflow graph level. A further abstraction of the linear tile replaces him in the dataflow graph with an abstracted new node. In other words, the node outside itself is blocked from the order of summation within the linear tile, so that the connection from an external node to a specific node in the linear tile becomes a connection to the specific tile to which the node belongs. In this abstracted dataflow graph, the types of edges are as follows:

1. Non-linear tile abstraction node to non-linear tile abstraction node: the two vertices already exist in the pre-abstraction dataflow graph. Consistent with the dataflow graph before abstraction.
2. Non-linear tile abstraction node to linear tile abstraction node: exists if and only if there are non-abstraction nodes in the linear tile represented by the abstraction node.
3. Linear tile abstraction node to linear tile abstraction node: exists when and only if there is a common non-abstraction node in the linear tile represented by the two abstraction nodes.

The next step is to calculate the weights of the selected individual tiles using the improved Weighted PageRank algorithm, referring to the relevant papers. After further abstraction of the dataflow graph, the whole graph looks much simpler and the number of nodes and edges becomes much smaller. It is easier to have some symmetry in the specific set of constraints, which in turn will allow some nodes to receive the same weight in the algorithm. This makes the subsequent ordering of the constraints more difficult. Therefore,

the Weighted PageRank algorithm is used, taking into account some practices in the literature. Using the normalised variance of the coefficients in a linear tile as the weights of the edges in the dataflow graph greatly reduces the symmetry of the abstracted dataflow graph.

Finally, the paradigms of R1CS are generated separately for each tile and the constraints and variables are ranked according to the node weights computed in the previous section. At this point, the division of constraints and the ordering of constraints in the constraint group has been determined, and the ordering of variables that have appeared in the secondary constraints has also been ordered. However, the specific internal structure of the linear constraints has been ignored in the abstraction of the data flow graph. Therefore the ordering of the newly appearing variables in the linear tile also needs to be adjusted in this step. The weights are calculated for each newly introduced variable in the linear tile as the sum of the absolute values of the products of their own coefficients and the linear tile weights in all linear tiles except their own. The newly introduced variables can then be ranked by first comparing the weights of the variables, and if they are consistent, then ranking their own coefficients in the linear tile. The presence of a new variable in a Linear tile in other Linear tiles is somewhat reflective of its importance in the overall constraint set. Also if some of the new variables only appear in their own constraints, they will all have a weight of 0. And their ordering will only have an effect on the constraints generated by their own Linear tiles, and will not change the order of the other constraints, so it is sufficient to order them in descending order of coefficients.

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. The reasons includes the following main categories depending on the reflected situation:

1. Replacement of variable order in R1CS.
2. Transformation of constraint order in R1CS.
3. Introduction of multiple new variables in a single linear constraint in R1CS.
4. Introduction of multiple new variables in multiple linear constraints in R1CS, with shared new variables.
5. Merging and splitting of constraints in R1CS.

Moreover, based on the identified reasons for producing equivalent R1CS, we create a

relatively complete benchmark.

The experimental results show that the data flow-based paradigm generation algorithm designed in this thesis is able to detect equivalent R1CS constraint sets generated for multiple cases and transform them to a unified paradigm. After inspection, the generated paradigms all conform to the requirements defined in this paper, and the semantics are all identical to the R1CS constraint sets before transformation.

The intermediate outputs from each stage of the transformation process were analysed. It is found that for the equivalent R1CS constraint sets generated by the order of swapping constraints, the difference in the generated data flow graphs is that the nodes representing intermediate variables in the graphs are created in a different order, which is due to the different order of processing each constraint when traversing the constraint sets, which leads to a different order of intermediate variables introduced into each constraint.

In contrast, the equivalent R1CS constraint groups resulting from the substitution of variable order differ in the order in which the nodes representing the initial variables in the graph are created. As well as the different order of variables in the linear constraint, resulting in a different order of summation in the concatenated additive chain structure, none of these variations resulted in a different set of selected tiles and, after abstraction, resulted in the same data flow graph.

The equivalent R1CS constraint sets resulting from the introduction of multiple new variables into the linear constraints all yielded the same dataflow graph after the abstraction operation on the linear constraint sets and also yielded a sequence of variable mappings that matched the definition when the newly introduced variables were sorted.

The difference between the equivalent R1CS constraint groups resulting from the merging and splitting of constraints is whether the vertices representing the intermediate variables represent variables that were initially present or intermediate variables introduced during the creation of the data flow graph. However, this has no impact on the structure of the data flow diagram. The splitting and merging of linear constraints caused a change in the summation order of the concatenated chains in the data flow diagram, but this was also resolved by abstraction of the data flow diagram.

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.

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