Notes 0330

施宸昊

2023 年 3 月 29 日

1 数据结构的形式化定义

$RNodeGraph = (RNode, Edge)$
$Edge = \{< RNode_i, RNode_j > |\ p(Node_i, RNode_j) \bigwedge (RNode_i, RNode_j \in RNode)\}$

$RNode = ConstNode \mid VarNode$

$ConstNode = (id, name, op, const, child, father)$
$ConstNode.id = \{x \mid x \in \mathbb{N}^+\}$
$ConstNode.name = \sim ONE$
$ConstNode.op = NULL \mid MULL \mid ADD$
$ConstNode.const = \{x \mid x \in \mathbb{R}^+\}$
$ConstNode.child = \{node| < self, node > \in RNodeGraph.Edge\}$
$ConstNode.father = \{node| < node, self > \in RNodeGraph.Edge\}$

$VarNode = (id, name, op, child, father)$
$VarNode.id = \{x \mid x \in \mathbb{N}^+\}$
$VarNode.name = \_$
$VarNode.op = NULL \mid MULL \mid ADD$
$VarNode.child = \{node| < self, node > \in RNodeGraph.Edge\}$
$VarNode.father = \{node| < node, self > \in RNodeGraph.Edge\}$

在编程实现中, ConstNode 和 VarNode 由同一数据结构表示, 在算法中通过

name 这一属性来辨别当前 node 是哪一个类型的 node

$TileNode = (id, type, rnode, child, father)$
$TileNode.id = self.rnode.id$
$TileNode.type = (Quadratic \wedge self.rnode.op == MULL \wedge$
$(\forall f \in self.father, TYPE(f.rnode) == VarNode) \bigvee$
$(MullLinear \wedge self.rnode.op == MULL \wedge$
$(\forall f \in self.father, TYPE(f.rnode) == ConstNode \vee f.type == MullLinear) \bigvee$
$(AddLinear \wedge self.rnode.op == ADD \wedge$
$(\forall f \in self.father, TYPE(f.rnode) == ConstNode \vee f.type == AddLinear \vee$
$f.type == MullLinear))$
$TileNode.rnode = RNode$

$Tile = MullLinearTile \mid AddLinearTile \mid QuadraticTile$

$QuadraticTile = (TileTree, Type, Weight)$
$QuadraticTile.Type = Quadratic$
$QuadraticTile.TileTree = (D, R \mid (\forall d \in D, d.rnode \in RNodeGraph.RNode) \bigwedge$
$(\forall r \in R, < r.u.rnode, r.v.rnode > \in RNodeGraph.Edge) \bigwedge$
$(ROOT(self).type == Quadratic)$

$MullLinearTile = (TileTree, Type, Weight)$
$MullLinearTile.Type = MullLinear$
$MullLinearTile.TileTree = (D, R \mid (\forall d \in D, d.rnode \in RNodeGraph.RNode) \bigwedge$
$(\forall r \in R, < r.u.rnode, r.v.rnode > \in RNodeGraph.Edge) \bigwedge (ROOT(self).type ==$
$MullLinear)$

$AddLinearTile = (TileTree, type, weight)$
$AddLinearTile.Type = AddLinear$
$AddLinearTile.TileTree = (D, R \mid (\forall d \in D, d.rnode \in RNodeGraph.RNode) \bigwedge$
$(\forall r \in R, < r.u.rnode, r.v.rnode > \in RNodeGraph.Edge) \bigwedge (ROOT(self).type ==$
$AddLinear)$

GetTiles() 函数输入一个 RNodeGraph, 输出它分割后的瓦片集合
AbstractTileNode() 函数输入一个线性的瓦片, 输出代表该瓦片的抽象 TileNode
LinearTile() 函数输代表一个抽象 TileNode, 输出该抽象 TileNode 对应的线性瓦片

$$TileGraph = (Tiles, TileNodes, Edges)$$

$$TileGraph.Tiles = GetTiles(RNodeGraph)$$
$$TileGraph.TileNodes = \{n \mid (\exists t \in self.tiles, t.type == Quadratic \wedge n \in t.TileTree.D) \bigvee (\exists t \in self.tiles, t.type! = Quadratic \wedge n == AbstractTileNode(t))\}$$
$$TileGraph.edges = \{(u, v) \mid ((\exists t1 \in self.tiles, t1.type == Quadratic \wedge u \in t1.TileTree.D) \wedge (\exists t2 \in self.tiles, t2.type == Quadratic \wedge v \in t1.TileTree.D) \wedge ((u.rnode, v.rnode \in RNodeGraph.Edge))) \bigvee ((\exists t1 \in self.tiles, t1.type == Quadratic \wedge u \in t1.TileTree.D) \wedge (\exists t2 \in self.tiles, t2.type! = Quadratic \wedge v == AbstractTileNode(t2))) \bigvee ((\exists t1 \in self.tiles, t1.type! = Quadratic \wedge u == AbstractTileNode(t1)) \wedge (\exists t2 \in self.tiles, t2.type! = Quadratic \wedge v == AbstractTileNode(t2)) \wedge (\exists n \in t1.TileTree.D, n \in t2.TileTree.D))\}$$

2 代码实现下的流程

以两个等价的约束组在目前所实现的流程中的变化来介绍目前的研究成果
约束组 A:

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

约束组 B:

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

2.1 建立 Rnode Graph

将每一个约束先按照 $a \cdot b = c$ 转换为算式, 比如

$$\begin{pmatrix} 0 & 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 1 \end{pmatrix} \Rightarrow x_2 * x_2 = x_4$$

其中 x 的下标表示在原 R1CS 中变量所处的列数

将原 R1CS 中每一个约束, 都化成这样的算式, 再将其结合在一起, 就会得到一个以 DAG 形式存储的含有公共子式的算式树 A 与 B 两个约束组在 DAG 完成后生成的 RNode Graph 的结构如图 1所示.

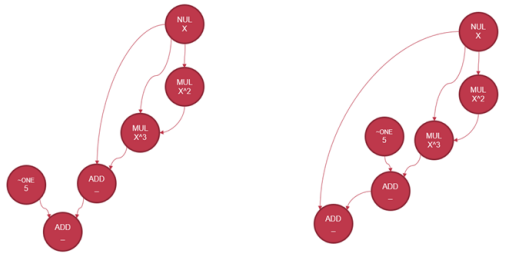

图 1: A 与 B 两个约束组生成的 RNode Graph

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:

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

$$\downarrow$$

$$\begin{pmatrix} 1 & 2 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\downarrow$$

$$\begin{pmatrix} 1 & 2 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}$$

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 $x_3 + x + 5 = out$ 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, 在生成 RNodeGraph 时必然先将其中的两个相加. 约束组 A 中为 $(x_3 + x) + 5 = out$, 而约束组 B 中为 $(x_3 + 5) + x = out$. 虽然形如 pagerank 之类的算法对图的微小变化不太反应明显, 同时也统计了一些等价约束组形成的 RNodeGraph 中的结点的权重, 发现对于同一节点, 权重的波动值大概是权重本身的 10%, 在大部分情况下不会对节点的权重排序产生影响, 但是一旦数据流 RNodeGraph 中权重的序列发生变化, 范式的生成将难以进行, 因此对于这种情况, 我们需要对 RNodeGraph 进行进一步的抽象.

2.2 瓦片选择

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 $x_3 + 5 + x = z$

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 Linear 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 瓦片选择时, 我们将上一个步骤中的 RNodeGraph, 分割成上述三种类型, 这样选取有几个考量:

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在进行
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也相对简单

根据上一步骤中的基本单元对树进行瓦片选取获得瓦片如图 3和图 4所示. 可以发现连加链的顺序不同不会影响到瓦片的选取, 也就是说, 对于等价的 R1CS 约束组, 我们可以选取出相同的瓦片.

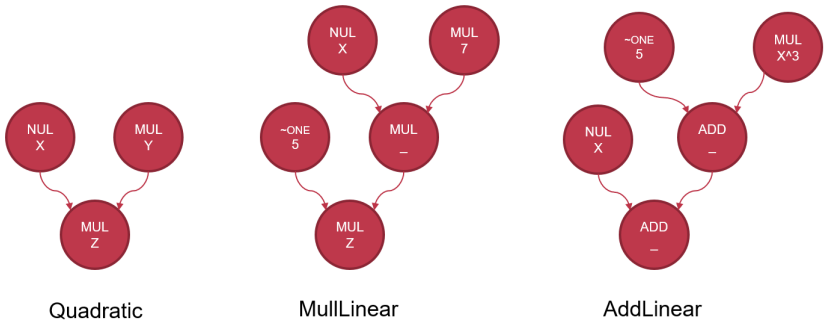

图 2: 瓦片示意图

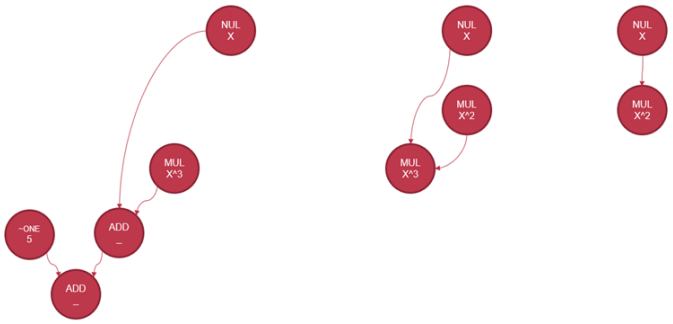

图 3: a 瓦片示意图

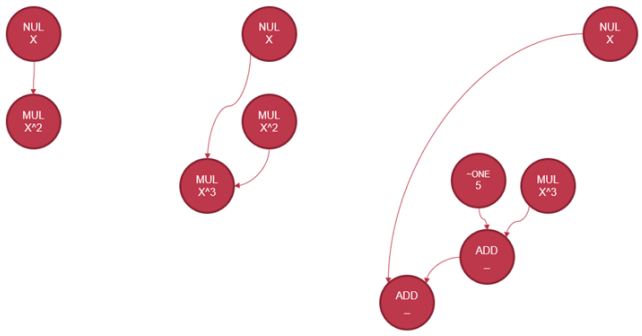

图 4: b 瓦片示意图

然后我们再将 ADDlinear 和 MullLinear 瓦片抽象成一个大的节点, 而剩下的 Quadratic 瓦片中的 RNode 保持不变, 得到了 Linear 瓦片抽象后的 RNodeGraph, 在抽象后的 RNodeGraph 中, 节点包括两种类型, 一种是 Quadratic 瓦片中包含的 RNode 节点, 在抽象前后保持不变, 另一种是由整个 Linear 瓦片抽象出的节点. 边的类型有以下几种:

1. RNode 节点到 RNode 节点: 与抽象前的 RNodeGraph 保持一致
2. RNode 节点到 Linear 瓦片抽象节点: 如果抽象节点所代表的 Linear 瓦片中存在 RNode 节点, 则存在
3. Linear 瓦片抽象节点节点到 Linear 瓦片抽象节点: 如果两个抽象节点所代表的 Linear 瓦片存在公有的 RNode 节点. 则存在边

由于在抽象前的 RNodeGraph 中, 等价的 R1CS 约束组产生的 RNodeGraph 的不同之处在于连加链相加的先后顺序不同, 所以将其抽象成一整个节点后, 对于等价的 R1CS 约束组产生的 RNodeGraph, 节点之间的依赖关系, 即是否存在边, 并不会产生影响.
约束组 A 和约束组 B 抽象后得到的 DAG 图像如图 5所示.

2.3 瓦片权重计算

首先对上一个步骤所得到的图应用 PageRank 算法, 得到各节点的权重, 然后以此计算瓦片的权重, 其中 quadratic 瓦片的权重为各节点的算术平均值,

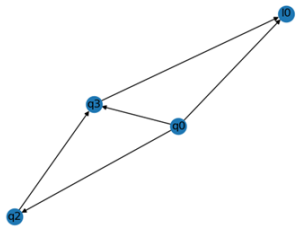

图 5: 抽象后的 RNodeGraph

linear 瓦片的权重即为其抽象节点的权重

2.4 约束生成

以瓦片为单位, 以权重的降序生成约束, 先生成 quadratic 约束, 再生成 linear 约束. 约束组 A 和约束组 B 被转化为:

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

2.5 约束产生后的调整

在之前产生的范式中存在一个问题. 那就是如果在单一 Linear 瓦片中引入超过一个新变量, 那么缺少这些变量的排序信息, 例如

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

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

如果在第三个 AddLinear 约束中引入两个之前从未使用过的变量, 那么根据输入的初始等价约束组会产生如上两个约束组 0, 因此要确定单一 AddLinear 变量中引入的新变量进行排序.

排序的方式目前比较简单, 为每一个新引入的变量计算权重, 公式如下:

$$weight = \sum_{other\ linear\ tiles} |field * weight\ of\ linear\ tile|$$

也就是说在每一个新加入的 Linear 瓦片中, 每一个新引入的变量, 其权重是除去本身 Linear 瓦片以外的其他 Linear 瓦片中自己的系数和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, 再对本身的系数进行排序. 在 Linear 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瓦片产生影响, 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.