

PLONK Course - Lecture 5

算术约束与拷贝约束

回顾置换证明

上一节，我们讨论了如何让 Prover 证明两个长度为 N 的向量 \vec{a} 与 \vec{b} 满足一个实现约定（公开）的置换关系 $\sigma(\cdot)$ ，即

$$a_i = b_{\sigma(i)}$$

基本思路是向 Verifier 要一个随机数 β ，把两个「原始向量」和他们的「位置向量」进行合体，产生出两个新的向量，记为 \vec{a}' 与 \vec{b}'

$$a'_i = a_i + \beta \cdot i, \quad b'_i = b_i + \beta \cdot \sigma(i)$$

第二步是再向 Verifier 要一个随机数 γ ，通过连乘的方法来编码 \vec{a}' 和 \vec{b}' 的 Multiset，记为 A 和 B ：

$$A = \prod (a'_i + \gamma), \quad B = \prod (b'_i + \gamma)$$

第三步是让 Prover 证明 $A/B = 1$ ，即

$$\prod_i \frac{(a'_i + \gamma)}{(b'_i + \gamma)} = 1$$

证明这个连乘，需要引入一个辅助向量 \vec{z} ，记录每次乘法运算的中间结果：

$$z_0 = 1, \quad z_{i+1} = z_i \cdot \frac{(a'_i + \gamma)}{(b'_i + \gamma)}$$

由于 $z_N = \prod \frac{a'_i + \gamma}{b'_i + \gamma} = 1$ ，而且 $\omega^N = 1$ ，因此我们可以用 $z(X)$ 来编码 \vec{z} ，从而把置换证明转换成关于 $z(X), a(X)$ 的关系证明。

最后 Verifier 发送挑战数 ζ ，得到 $z(\zeta), z(\omega \cdot \zeta), a(\zeta), b(\zeta)$ 然后检查它们之间的关系。

向量的拷贝约束

所谓拷贝约束 Copy Constraints，是说在一个向量中，我们希望能证明多个不同位置上的向量元素相等。我们先从一个简单例子开始：

$$\vec{a} = (a_0, a_1, a_2, a_3)$$

假设为了让 Prover 证明 $a_0 = a_2$ ，我们可以把 a_0 与 a_2 对调位置，这样形成一个「置换关系」，如果我们用 $(0, 1, 2, 3)$ 记录被置换向量的元素位置，那么我们把置换后的位置向量记为 σ ，而 \vec{a}_σ 为表示按照 σ 置换后的向量

$$\sigma = (2, 1, 0, 3), \quad \vec{a}_\sigma = (a_2, a_1, a_0, a_3)$$

显然，只要 Prover 可以证明置换前后的两个向量相等， $\vec{a} = \vec{a}_\sigma$ ，那么我们就可以得出结论： $a_0 = a_2$ 。

这个方法可以推广到证明一个向量中有多个元素相等。比如要证明 \vec{a} 中的前三个元素都相等，我们只需要构造一个置换，即针对这三个元素的循环右移：

$$\sigma = (2, 0, 1, 3), \quad \vec{a}_\sigma = (a_2, a_0, a_1, a_3)$$

那么根据 $\vec{a} = \vec{a}_\sigma$ 容易得出 $a_0 = a_1 = a_2$ 。

多个向量间的拷贝约束

对于 Plonk 协议，拷贝约束需要横跨 W 表格的所有列，而协议要求 Prover 要针对每一列向量进行多项式编码。我们需要对置换证明进行扩展，从而支持横跨多个向量的元素等价。

回忆比如针对上面电路的 W 表格：

i	w_a	w_b	w_c
0	0	0	<i>out</i>
1	<i>x</i> ₆	<i>x</i> ₅	<i>out</i>
2	<i>x</i> ₁	<i>x</i> ₂	<i>x</i> ₆
3	<i>x</i> ₃	<i>x</i> ₄	<i>x</i> ₅

看上面的表格，我们要求 $w_{a,1} = w_{c,2}$ ， $w_{b,1} = w_{c,3}$ 且 $w_{c,0} = w_{c,1}$ 。

支持跨向量置换的直接方案是引入多个对应的置换向量，比如上表的三列向量用三个置换向量统一进行位置编码：

i	$id_{a,i}$	$id_{b,i}$	$id_{c,i}$
0	0	4	8
1	1	5	9
2	2	6	10
3	3	7	11

多个向量间的拷贝约束

置换后的向量为 $\sigma_a, \sigma_b, \sigma_c$:

i	$\sigma_{a,i}$	$\sigma_{b,i}$	$\sigma_{c,i}$
0	0	4	9
1	10	11	8
2	2	6	1
3	3	7	5

Prover 用一个随机数 β (Verifier 提供) 来合并 (\vec{w}_a, \vec{id}_a) , (\vec{w}_b, \vec{id}_b) , (\vec{w}_c, \vec{id}_c) , 还有置换后的向量: (\vec{w}'_a, σ_a) , (\vec{w}'_b, σ_b) , (\vec{w}'_c, σ_c) 。然后再通过一个随机数 γ (Verifier 提供) 和连乘来得到 W 和 W' 的 Multisets, $\{f_i\}$ 与 $\{g_i\}$

$$f_i = (w_{a,i} + \beta \cdot id_{a,i} + \gamma)(w_{b,i} + \beta \cdot id_{b,i} + \gamma)(w_{c,i} + \beta \cdot id_{c,i} + \gamma)$$
$$g_i = (w'_{a,i} + \beta \cdot \sigma_{a,i} + \gamma)(w'_{b,i} + \beta \cdot \sigma_{b,i} + \gamma)(w'_{c,i} + \beta \cdot \sigma_{c,i} + \gamma)$$

又因为拷贝约束要求置换后的向量与原始向量相等, 因此 $w_a = w'_a$, $w_b = w'_b$, $w_c = w'_c$ 。

如果我们用多项式对 $\vec{w}_a, \vec{w}_b, \vec{w}_c, \vec{id}_a, \vec{id}_b, \vec{id}_c, \sigma_a, \sigma_b, \sigma_c$ 编码, 得到 $w_a(X), w_b(X), w_c(X), id_a(X), id_b(X), id_c(X), \sigma_a(X), \sigma_b(X), \sigma_c(X)$, 于是 $f(X)$, $g(X)$ 满足下面的约束关系:

$$f(X) = \left(w_a(X) + \beta \cdot id_a(X) + \gamma\right) \left(w_b(X) + \beta \cdot id_b(X) + \gamma\right) \left(w_c(X) + \beta \cdot id_c(X) + \gamma\right)$$
$$g(X) = \left(w_a(X) + \beta \cdot \sigma_a(X) + \gamma\right) \left(w_b(X) + \beta \cdot \sigma_b(X) + \gamma\right) \left(w_c(X) + \beta \cdot \sigma_c(X) + \gamma\right)$$

多个向量间的拷贝约束

如果两个 Multiset 相等 $\{f_i\} = \{g_i\}$ ，那么下面的等式成立：

$$\prod_{X \in H} f(X) = \prod_{X \in H} g(X)$$

上面的等式稍加变形，可得

$$\prod_{X \in H} \frac{f(X)}{g(X)} = 1$$

我们进一步构造一个辅助的累加器向量 z ，表示连乘计算的一系列中间过程

$$z_0 = 1, \quad z_{i+1} = z_i \cdot \frac{f_i}{g_i}$$

多个向量间的拷贝约束

其中 z_0 的初始值为 1，Prover 按照下表计算出 \vec{z} :

i	H_i	z_i
0	$\omega^0 = 1$	1
1	ω^1	$1 \cdot \frac{f_0}{g_0}$
2	ω^2	$\frac{f_0}{g_0} \cdot \frac{f_1}{g_1}$
3	ω^3	$\frac{f_0 f_1}{g_0 g_1} \cdot \frac{f_2}{g_2}$
\vdots		\vdots
$N - 1$	ω^{N-1}	$\frac{f_0 f_1 \cdots f_{N-3}}{g_0 g_1 \cdots g_{N-3}} \cdot \frac{f_{N-2}}{g_{N-2}}$
N	$\omega^N = 1$	$\frac{f_0 f_1 \cdots f_{N-1}}{g_0 g_1 \cdots g_{N-1}} = 1$

如果 \vec{f} 能与 \vec{g} 连乘等价的话，那么最后一行 z_N 正好等于 1，即

$$z_N = z_0 = 1$$

而又因为 $\omega^N = 1$ 。这恰好使我们可以把 $(z_0, z_1, z_2, \dots, z_{N-1})$ 完整地编码在乘法子群 H 上。因此如果它满足下面两个多项式约束，我们就能根据数学归纳法得出 $z_N = 1$ ，这是我们最终想要的「拷贝约束」：

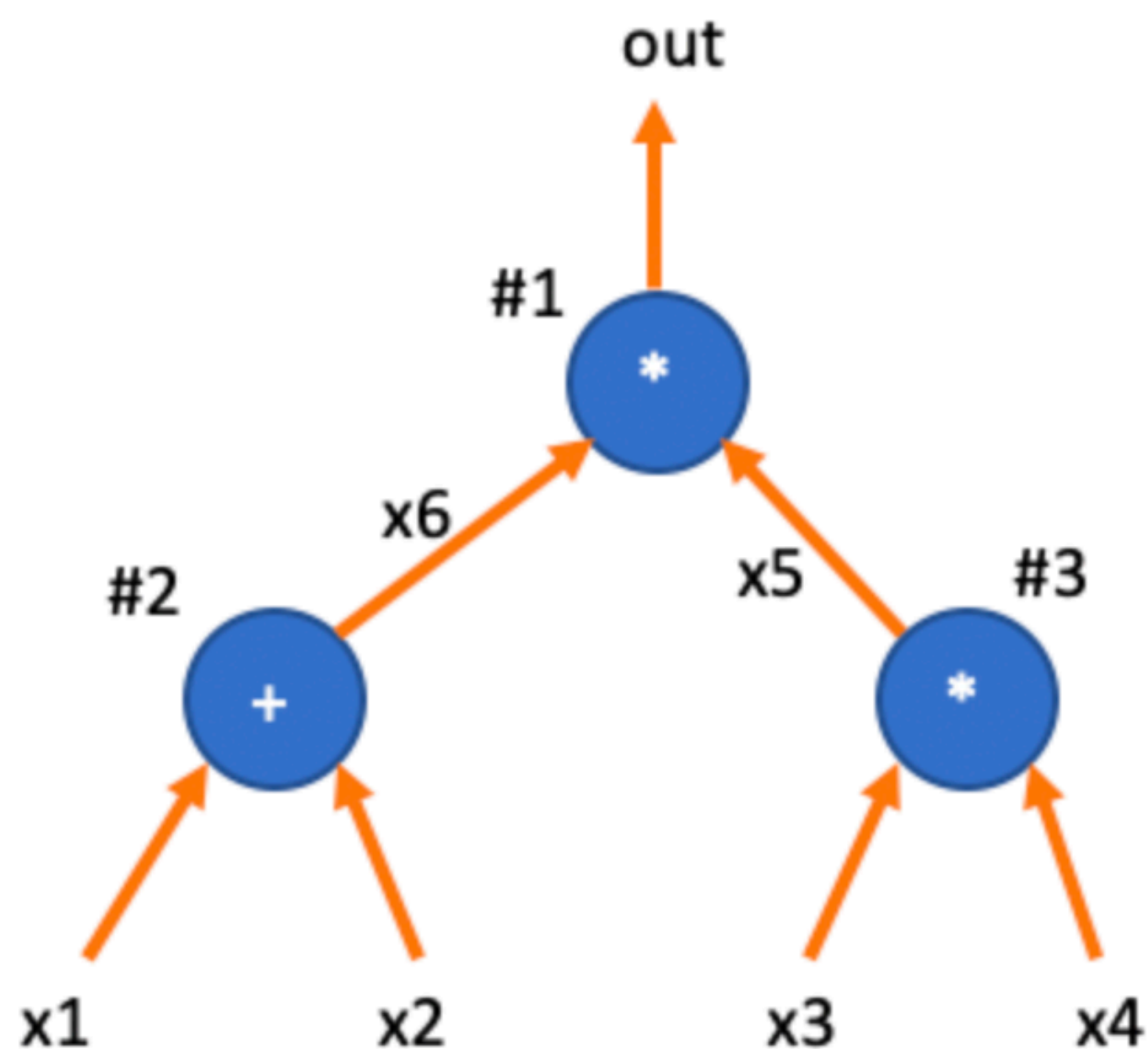
$$z(\omega^0) = 1$$

$$z(\omega \cdot X)g(X) = z(X)f(X)$$

置换关系 σ

在构造拷贝约束前，置换关系 σ 需要提前公开共识。表格 W 含有所有算术门的输入输出，但是并没有描述门和门之间是否通过引线相连，而置换关系 σ 实际上正是补充描述了哪些算术门之间的连接关系。

因此，对于一个处于「空白态」的电路，通过 (Q, σ) 两个表格描述，其中 Q 由选择子向量构成，而 σ 则由「置换向量」构成。



置换关系 σ

下面是 Q 表格

i	q_L	q_R	q_M	q_C	q_O
0	0	0	0	99	1
1	0	0	1	0	1
2	1	1	0	0	1
3	0	0	1	0	1

下面是 S 表格，描述了哪些位置做了置换

i	$\sigma_{a,i}$	$\sigma_{b,i}$	$\sigma_{c,i}$
0	0	4	[9]
1	10	<u>11</u>	[8]
2	2	6	1
3	3	7	<u>5</u>

置换关系 σ

假如在上面给出的小电路中，要证明存在一个 Assignment，使得 out 的输入为一个特定的公开值，比如 $out = 99$ 。最简单的办法是使用 Q 表中的 q_C 列，并增加一行约束，使得 $q_L = q_R = q_M = 0$ ，因此满足下面等式

$$q_C(X) - q_O(X)w_c(X) = 0$$

但这个方案的问题是：这些公开值输入输出值被固定成了常数，如果公开值变化，那么 $q_C(X)$ 多项式需要重新计算。如果整体上 W 表格的行数比较大，那么这个重新计算过程会带来很多的性能损失。

能否在表格中引入参数，以区分电路中的常数列？并且要求参数的变化并不影响其它电路的部分？这就需要再引入一个新的列，专门存放公开参数，记为 ϕ ，因此，算术约束会变为：

$$q_L(X)w_a(X) + q_R(X)w_b(X) + q_M(X)w_a(X)w_b(X) - q_O(X)w_c(X) + q_C(X) + \phi(X) = 0$$

我们还可以通过修改拷贝约束的方式引入公开参数。

位置向量的优化

我们上面在构造三个 σ 向量时，直接采用的自然数 $(0, 1, 2, \dots)$ ，这样在协议开始前，Verifier 需要构造 3 个多项式 $id_a(X), id_b(X), id_c(X)$ ，并且在协议最后一步查询 Oracle，获得三个多项式在挑战点 $X = \zeta$ 处的取值 $(id_a(\zeta), id_b(\zeta), id_c(\zeta))$ 。

思考一下， σ 向量只需要用一些互不相等的值来标记置换即可，不一定要采用递增的自然数。如果我们采用 $H = (1, \omega, \omega^2, \dots)$ 的话，那么多项式 $id_a(X)$ 会被大大简化：

$$\begin{aligned}\vec{id}_a &= (1, \omega, \omega^2, \omega^3) \\ \vec{id}_b &= (k_1, k_1\omega, k_1\omega^2, k_1\omega^3) \\ \vec{id}_c &= (k_2, k_2\omega, k_2\omega^2, k_2\omega^3)\end{aligned}$$

其中 k_i 为互相不等的二次非剩余。

$$id_a(X) = X, \quad id_b(X) = k_1 \cdot X, \quad id_c(X) = k_2 \cdot X$$

这样一来，这三个多项式被大大简化，它们在 $X = \zeta$ 处的计算轻而易举，可以直接由 Verifier 完成。

陪集

定义1(陪集) 设 G 为(乘法)群, $H < G$ 为子群, $a \in G$ 为任一元素, 则

$$aH = \{ah \mid h \in H\}$$

称为 H 在 G 的一个(左)陪集(coset), a 称为此陪集的一个代表元(representative).
类似地, Ha 称为右陪集, 以下, 陪集均指左陪集.

例1 在有限域 $G = F_{101}$ 中存在一个乘法子群 $H = \{1, 10, 100, 91\}$, 则可计算一些左陪集。

- $a = 6$, 则 $aH = \{6, 60, 95, 41\}$
- $b = 33$, 则 $bH = \{33, 27, 68, 74\}$

陪集

引理1 子群 H 的陪集有如下性质：

1. 若 $h \in H$, 则 $hH = H$. 特别有 $HH = H$.
2. 若 $a' \in aH$, 则 $a'H = aH$. (陪集内元素皆可作为代表元)
3. 若 $b \notin aH$ 或 $aH \neq bH$, 则 $aH \cap bH = \emptyset$. (不同的陪集无交)
4. $\#(H) = \#(aH)$. (各陪集的元素个数相等, 陪集等长)

群的陪集分解

$$G = H \cup a_1H \cup a_2H \cup a_3H \cup \cdots$$

陪集

例 2 乘法群 $W_4 = \{1, i, -1, -i\}$, 子群 $H = \{1, -1\} = \langle -1 \rangle$. W_4 分为两个陪集:

$$H = \{-1, 1\}, iH = \{i, -i\}.$$




对 W_4 作陪集分解

$$W_4 = H \cup iH = \{1, -1\} \cup \{i, -i\}$$

协议框架

- 🧑 理解 PLONK（四）：算术约束与拷贝约束：<https://github.com/wenjin1997/awesome-zkp-learning/blob/main/courses/Plonk-GuoYu/lecture04/notes-plonk-lecture4-constraints.ipynb>

Reference

-  理解 PLONK（四）：算术约束与拷贝约束：<https://github.com/sec-bit/learning-zkp/blob/master/plonk-intro-cn/4-plonk-constraints.md>
-  Plonk intro notebook：<https://github.com/Antalpha-Labs/plonk-intro-notebook/blob/main/4-plonk-constraints.ipynb>
-  抽象代数：<https://book.douban.com/subject/36007684/>