理解 PLONK（七）：Lookup Gate

传统上我们通过编写算术电路来表达逻辑或者计算。而算术电路只有两种基本门：「加法门」与「乘法门」。当然通过组合，我们可以基于加法和乘法构造复杂一点的元件（Gadget）来复用，但是在电路处理过程中，这些 Gadget 还是会被展开成加法门和乘法门的组合。

自然我们想问：能否使用除加法和乘法之外的「新计算门」？

Plonk 相关的工作给出了一个令人兴奋的扩展：我们有能力构造出更复杂些的基本计算单元。如果一个计算的输入和输出满足一个预先设定的多项式的话，那么这个计算可以作为基本计算单元，这个改进被称为 「Custom Gate」，实际上你可以理解为这是一种多输入的「多项式门」。

故事还没有结束，论文 GW20 又给出了一个制造「Lookup Gate」的方法。这个门的输入输出没有必要局限于多项式关系，而是可以表达「任意的预定义关系」。What? 任意的关系？是的，你没听错，尽管这有点令人难以置信。

思路不难理解：如果我们在电路之外预设一个表格，表中每一行表示特定计算的输入输出关系，例如：

in1	in2	in3	out
1	2	3	4
5	6	7	8
1	1	5	9

这个表格就代表一个 Lookup 门的定义。如果你问我这个门究竟表达了什么计算，我无法回答（乱写的）。不过只要能给出这样一张表格，我们就可以在电路里面接入一个门，它的输入输出关系「存在于表中的某一行」。

这种门被称为 Lookup Gate，即查表门（或查表约束）。

如果当我们在 Plonk 电路中接入查表门，那么 Plonk 协议就要检查这个门的输入输出是否合法，然后就会去查我们实现预设的表格，看看其输入输出关系是否能在表中找到对应的一行。如果表中存在这样的条目，那么这个门就合法，否则被视为非法。

在现实应用中，最多采用查表方式的门是关于位运算。如一个 8-bit 异或运算，只需要 2^{16} 大小的表格即可。此外对于采用大量位运算的 SHA256算法，也可以通过建立一个 Spread Table 来大大加速各种位运算的效率。

基本思路

实现查表门的一个关键技术是 Lookup Argument 协议，即如何证明一条（或多条）记录是否存在于一个公开的表中。

可能有朋友会条件反射想到 Merkle Tree，如果我们把表格按行计算 hash，这些 hash 就可以产生一个 Merkle Root，然后通过 Merkle Path 就能证明一条记录是否存在表格中。但是这个方法（以及所有的 Vector Commitment 方案）不适合查表场景。原因有两个，一是这种方案会暴露记录在表格中的位置。假如 Prover 想隐藏记录的信息，即在查询证明不暴露位置，那么仅 Merkle Tree 就难以胜任了。理论点说，这里我们需要 Set-Membership Argument，而非 Vector-Membership Argument。第二个原因：如果有大量的记录条目（比如条目数量为 d ）需要查表，那么所产生的证明即 Merkle Path，可能会比较大，最坏情况是 $O(d\log n)$。

简而言之，我们需要一种新的，并且高效的查表协议。本文介绍两个常见的查表协议，为了简化表述，我们先只考虑单列表格的查询，然后再扩展到多列表格的情况。

Halo2-lookup 方案

基于 Permutation Argument，Halo2 给出了一个简洁易懂的 Lookup Argument 方案。

假如我们有一个表格向量 $\vec{t}=(t_0,t_1,t_2,\cdots,t_{n-1})$，表格中不存在相同元素。然后有一个查询向量 $\vec{f}=(f_0,f_1,f_2,\cdots,f_{m-1})$，我们接下来要证明 $\vec{f}\subseteq\vec{t}$，请注意 $\vec{f}$ 中会有重复元素。

我们引入一个关键的**辅助向量** $\vec{f}'$，它是 $\vec{f}$ 的一个重新排序（置换），使得 $\vec{f}$ 中的所有查询记录都按照 $\vec{t}$ 的顺序进行排序，比如 $\vec{t}=(1,2,3,4,5,6,7,8)$， $\vec{f}=(3,1,4,2,7,1,7,2)$，那么重排后， $\vec{f}'=(1,1,2,2,3,4,7,7)$ 。

可以看出， $\vec{f}'$ 中的重复元素被放在了一起，并且整体上按照 t 中元素出现的顺序。我们把 $\vec{f}'$ 中连续重复元素的第一个标记出来：

$$\vec{f}'=(\boxed{1},1,\boxed{2},2,\boxed{3},\boxed{4},\boxed{7},7)$$

我们再引入一个**辅助向量** $\vec{t}'$，它是对 $\vec{t}$ 的重新排序，使得 $\vec{f}'$ 中被标记元素可以正好对应到 $\vec{t}'$ 中相同位置上的元素：

$$\vec{t}'=(\boxed{1},5,\boxed{2},6,\boxed{3},\boxed{4},\boxed{7},8)$$

请注意看 $\vec{t'}$，其中被方框标记的元素和 $\vec{f'}$ 中相同位置的方框元素值完全相同，未被标记的元素则没有出现在 $\vec{f'}$ 中。

于是我们可以找出一个规律：$\vec{f'}$ 中的每一个未标记元素等于它左边的相邻元素，而每一个被标记元素等于 $\vec{t'}$ 同位置元素，即 $f'_i=f'_{i-1}$ 或者 $f'_i=t'_i$。

将两个向量 $\vec{f}$ 和 $\vec{t}$ 与重排向量 $\vec{f'}$ 和 $\vec{t'}$ 通过 Lagrange Basis 进行多项式编码，我们得到 $t(X)$，$f(X)$，$f(X)'$ 和 $t'(X)$，他们会满足下面的等式：

$$(f'(X)-f'(\omega^{-1}\cdot X))\cdot(f'(X)-t'(X))=0,\quad \forall x\in H$$

但上面这个等式不足以约束重排向量的可靠性。考虑如果 $\vec{f'}=(9,9,9,9,9,9,9,9)$，也会满足上面的等式，但是 $\vec{f'}$ 并不是合法的查询记录。因此，我们还要加入一条约束防止出现 $\vec{f'}$ 在 H 上 循环回卷导致的漏洞：要求 $\vec{f'}$ 和 $\vec{t'}$ 两个向量的第一个元素必须相同， 即 $f'_0=t'_0$，用多项式约束表达如下：

$$L_0(X)\cdot(f'(X)-t'(X))=0,\quad \forall x\in H$$

剩下的工作是证明 $(\vec{f},\vec{f'})$ 满足某一个「置换」关系，且 $(\vec{t},\vec{t'})$ 也满足某个「置换」关系。由于，这两个置换关系只不需要约束具体的置换向量，因此我们可以直接采用 Grand Product Argument 来约束这两个置换关系：

$$\frac{z(\omega\cdot X)}{z(X)}=\frac{(f(X)+\gamma_1)(t(X)+\gamma_2)}{(f'(X)+\gamma_1)(t'(X)+\gamma_2)}$$

$$L_0(X)\cdot(z(X)-1)=0,\quad \forall x\in H$$

下面重新整理下这个协议

协议框架

公共输入：表格向量 $\vec{t}$;

秘密输入：查询向量 $\vec{f}$；

预处理：Prover 和 Verifier 构造 $t(X)$,

第一步：Prover 构造多项式并发送承诺 $[f(X)]$，$[f'(X)]$，$[t'(X)]$

第二步：Verifier 发送挑战数 γ_1 与 γ_2,

第三步：Prover 构造多项式并发送承诺 $[z(X)]$

$$z(X)=L_0(X)+\sum_{i=1}^{N-1}\Big(L_i(X)\cdot\prod_{j=0}^{i-1}\frac{(f_i+\gamma_1)(t_i+\gamma_2)}{(f'_i+\gamma_1)(t'_i+\gamma_2)}\Big)$$

第四步：Verififer 发送挑战数 α

第五步：Prover 计算并发送商多项式 $[q(X)]$

$$q(X)=\frac{1}{v_H(X)}\left(\begin{array}{l}\big(f'(X)-f'(\omega^{-1}\cdot X))\cdot(f'(X)-t'(X)\big)\\ +\alpha\cdot\big(L_0(X)\cdot(f'(X)-t'(X))\big)\\ +\alpha^2\cdot\big(L_0(X)\cdot(z(X)-1)\big)\\ +\alpha^3\cdot\big(z(\omega\cdot X)\cdot(f'(X)+\gamma_1)(t'(X)+\gamma_2)\\ \qquad -z(X)\cdot(f(X)+\gamma_1)(t(X)+\gamma_2)\big)\end{array}\right)$$

第六步：Verifier 发送挑战数 ζ

第七步：Prover 发送 $f(\zeta),f'(\zeta),f'(\omega^{-1}\cdot\zeta),t'(\zeta),z(\zeta),z(\omega\cdot\zeta),q(\zeta)$，并附带上 evaluation proofs（略去）

第八步：Verifier 验证（注意这里为了简化，去掉了KZG10的聚合优化和线性化优化）

$$q(\zeta)\cdot v_H(\zeta)\stackrel{?}{=}\left(\begin{array}{l}\big(f'(\zeta)-f'(\omega^{-1}\cdot\zeta))\cdot(f'(\zeta)-t'(\zeta)\big)\\ +\alpha\cdot\big(L_0(\zeta)\cdot(f'(\zeta)-t'(X))\big)\\ +\alpha^2\cdot\big(L_0(\zeta)\cdot(z(\zeta)-1)\big)\\ +\alpha^3\cdot\big(z(\omega\cdot\zeta)\cdot(f'(\zeta)+\gamma_1)(t'(\zeta)+\gamma_2)\\ \qquad -z(\zeta)\cdot(f(\zeta)+\gamma_1)(t(\zeta)+\gamma_2)\big)\end{array}\right)$$

Plookup 方案

然后我们再看看论文 GW20 给出的方案 —— Plookup。与 Halo2-lookup 相比，Plookup 可以省去 $\vec{t}'$ 向量。

重申一下 Plookup 证明的场景：Verifier 已知表格 $\vec{t}$ 向量，Prover 拥有一个秘密的查询向量 $\vec{f}$，Prover 要证明 $\vec{f}$ 中的每一个元素都在 $\vec{t}$ 中，即 $\{f_i\}\subseteq_{set}\{t_i\}$。

方案 Plookup 只需要引入一个辅助向量 $\vec{s}$ ，它被定义为 $\{f_i\}\cup\{t_i\}$ 上的重排，且向量元素的排列遵照 $\vec{t}$ 中各个元素出现的顺序。

举例说明，假设 $N=4$，如果 $\vec{t}=(1,2,3,4)$， $\vec{f}=(3,2,2,1)$，那么 $\vec{s}=(1,1,2,2,2,3,3,4)$。可以看到，和 Halo2-lookup 中的 $\vec{f}'$一样， $\vec{s}$ 中相等的元素被

排在了一起。

如果向量 $\vec{s}$ 满足 $\{s_i\} \subseteq_{set} \{t_i\}$，并且 $\{f_i\} \cup \{t_i\} =_{multiset} \{s_i\}$，那么就可以证明 $\{f_i\} \subseteq_{set} \{t_i\}$。

第一个关键点是因为 $\vec{f}$ 中的查询记录是任意的，查询顺序并没有遵守 $\vec{t}$ 中的元素顺序。而通过辅助向量 $\vec{s}$ ，我们就可以把 $\vec{f}$ 的查询记录进行重新排序，这有利于排查 $\vec{f}$ 中元素的合法性，确保每一个 f_i 都出现在 $\vec{t}$ 中。但如何保证由 Prover 构造的 $\vec{s}$ 是按照 $\vec{t}$ 的元素顺序进行排序的？Plookup 用了一个直接但巧妙的方法，考虑把 $\vec{s}$ 中的每一个元素和他相邻下一个元素绑在一起，然后可以构成一个新的 Multiset；同样，我们把 $\vec{t}$ 中的每一个元素与相邻下一个元素组成一个元组，并构成一个 Multiset；我们还要把 $\vec{f}$ 中的每一个元素和它自身构成一个二元组 Multiset。我们用 $S = \{(s_i, s_{i+1})\}$ ，$T = \{(t_i, t_{i+1})\}$ ， $F = \{(f_i, f_i)\}$ 来表示这三个新的 Multiset，并证明它们满足一定的关系，从而保证 $\vec{s}$ 排序的正确性。

$$\overset{S}{\{(s_i, s_{i+1})\}} \quad =_{multiset} \quad \overset{T}{\{(t_i, t_{i+1})\}} \quad \cup \quad \overset{F}{\{(f_i, f_i)\}}$$

这个方法与 Permutation Argument 的基本思想非常类似。回忆下，我们在 Permutation Argument 中，利用了 $\{(a_i, i)\}$ 绑定元素和其位置的「二元组」的 Multiset 来保证任一个 a_i 都会出现在位置 i 上；通过与另一个二元组 Multiset $\{(b_i, \sigma(i)\}$ 的相等，可以证明 $\vec{a}$ 与 $\vec{b}$ 满足置换函数 σ。比如下面这个置换函数为奇偶互换的例子：

$$\{(a_0, 0), (a_1, 1), (a_2, 2), (a_3, 3)\} =_{multiset} \{(b_0, 1), (b_1, 0), (b_2, 3), (b_3, 2)\}$$

假设两个向量 (a_0, a_1, a_2, a_3) 与 (b_0, b_1, b_2, b_3) ，如果它们满足上面的 Multiset 相等关系，我们可以知 $a_0 = b_1$， $a_1 = b_0$， $a_2 = b_3$， $a_3 = b_2$，满足奇偶互换的关系。

另一个关键点是如何保证 $\vec{f}$ 中的元素都在 $\vec{t}$ 中出现？这个问题被归结到一个新问题，即 $\vec{s}$ 中那些相邻的重复元素一定来自于 $\vec{f}$，假如 $\vec{f}$ 中有 l 个重复元素，那么我们可以要求其中第一个来自于 $\vec{t}$，剩下的 $l-1$ 个元素来自于 $\vec{f}$。如果 $\vec{f}$ 中一旦出现了一个不在 $\vec{t}$ 中的元素（假设为 f^*），那么因为 $\vec{s}$ 是 $(\vec{f}, \vec{t})$ 的重排，那么 $\vec{s}$ 中一定会出现 f^*（假设 $f^* = s_i$），这时在 S 中一定会出现 $(s_{i-1}, f^*), (f^*, s_{i+1})$ 这样两个元素，它们无法出现在 $\{(t_j, t_{j+1})\}$ 这个 Multiset中，也不会出现在 F 中。

举几个例子，假设 $\vec{f}$ 的长度为 0， 如果 $\{(s_i, s_{i+1})\} = \{(t_i, t_{i+1})\}$，那么 $\vec{s}$ 与 $\vec{t}$ 向量在各个位置上都相等。

假设增加一条查询记录，即 $\vec{f}=(1)$，那么 $T=\{(1,2),(2,3),(3,4),(4,1)\}$，$F=\{(1,1)\}$，这时候 S 只有唯一的表达，$S=\{(1,1),(1,2),(2,3),(3,4),(4,1)\}$，$S=_{multiset} T\cup F$。

假设 $\vec{f}=(9)$，9 为不出现在 $\vec{t}$ 中的元素，那么 $F=\{(9,9)\}$ 一定没有办法塞入到 S 中，因为在 $\vec{s}$ 中，和 9 相邻的元素 $s_{k-1},s_{k+1}\neq 9$。因此 $S\neq_{multiset} T\cup F$。

假设 $\vec{f}=(2,2)$，那么 S 也只有唯一的表达，$S=\{(1,2),(2,2),(2,2),(2,3),(3,4),(4,1)\}$，同样可以检验：$S=_{multiset} T\cup F$。

更形式化一些，我们可以用数学归纳法推导：先从 $\vec{f}$ 为空开始推理，$F_0=\emptyset$。这样我们只要检查 $S_0=\{(s_i,s_{i+1})\}$ 和 $T=\{(t_i,t_{i+1})\}$ 满足 Multiset 意义上的相等，就可以满足 $S_0=_{multiset} T\cup F_0$，且 $\{f_i\}\subseteq\{t_i\}$。

现在看归纳步，假设 $S_k=_{multiset} T\cup F_k$，如果我们在 $\vec{f}$ 中添加一个新元素 f_{k+1}，且 $f_{k+1}=t_l$，那么在 S_{k+1} 中会比 S_k 额外多一个元素 (f_{k+1},f_{k+1})。因为 $f_{k+1}\in\{t_i\}$，那么重排向量 $\vec{s}_{k+1}$ 中一定包含了相邻的两个 t_{k+1}，其中一个来自 t_l，另一个来自于 f_{k+1}。因此，我们可以得出结论：$S_{k+1}=T\cup F_k\cup\{(t_k,t_k)\}$。

另一种情况，假设 $S_k=_{multiset} T\cup F_k$，如果我们在 $\vec{f}$ 添加的新元素 $f_{k+1}\notin\vec{t}$，即是一条违法查询，假设为 u。那么 $\vec{s}$ 中存在与 u 相邻的两个元素，s_{l-1},s_{l+1}，即 $\vec{s}=(\ldots,s_{l-1},u,s_{l+1},\ldots)$。它们构成了 S 中的两个异类元素 $(s_{l-1},u),(u,s_{l+1})$，导致 $S_{k+1}\neq T\cup F_k\cup\{(u,u)\}$。

到此为止，我们已经可以确信，通过验证 $S=T\cup F$ 相等就可以判定 $\vec{s}$ 是正确的重排，并且 $\vec{f}$ 中的每一个元素都出现在 $\vec{t}$ 中。接下来我们把这个问题转换成多项式之间的约束关系。

首先 Prover 借助 Verifier 提供的挑战数 β，把 S,T,F 中的每一个二元组元素进行「折叠」，转换成单值。这样新约束等式为：

$$\{s_i+\beta s_{i+1}\}=\{t_i+\beta t_{i+1}\}\cup\{(1+\beta)f_i\}$$

然后 Prover 再借助 Verifier 提供的一个挑战数 γ，把上面的 Multiset Equality Argument 归结到 Grand Product Argument：

$$\begin{aligned}&\prod_i((1+\beta)f_i+\gamma)(t_i+\beta\cdot t_{i+1}+\gamma)\\=&\prod_i(s_i+\beta\cdot s_{i+1}+\gamma)\end{aligned}$$

不过这里请注意的是，在 Plookup 论文方案中，并没有采用上面的证明转换形式。而是调换了 β 和 γ 的使用顺序：

$$\{(s_i+\gamma)+\beta(s_{i+1}+\gamma)\} = \{(t_i+\gamma)+\beta(t_{i+1}+\gamma)\} \cup \{(f_i+\gamma)+\beta(f_i+\gamma$$

归结后的 Grand Product 约束等式为：

$$\begin{aligned}&\prod_i (1+\beta)(f_i+\gamma)(t_i+\beta\cdot t_{i+1}+(1+\beta)\gamma)\\ =&\prod_i (s_i+\beta\cdot s_{i+1}+\gamma)\end{aligned}$$

注：个人认为，上述两种证明转换形式没有本质上的区别。为了方便理解论文，我们后文遵从 Plookup 原论文的方式。

接下来，我们要对向量进行多项式编码，但是这里会遇到一个新问题。即 $\vec{s}$ 多项式的次数会超出 $\vec{f}$ 的次数或 $\vec{t}$ 的次数，特别当 $\vec{f}$ 或 $\vec{t}$ 的长度接近或者等于 H 的大小， $\vec{s}$ 的次数可能超出 H 的大小。Plookup 的解决方式是将 $\vec{s}$ 拆成两半， $\vec{s}^{lo}$ 与 $\vec{s}^{hi}$，但是 $\vec{s}^{lo}$ 的最后一个元素要等于 $\vec{s}^{hi}$ 的第一个元素：

$$\vec{s}^{lo}_{N-1} = \vec{s}^{hi}_0$$

这样做的目的是，确保能在两个向量中描述 $\vec{s}$ 中相邻两个元素的绑定关系。比如 $\vec{s} = (1,2,2,3,4,4,4)$，那么 $\vec{s}^{lo} = (1,2,2,\underline{3})$ ，而 $\vec{s}^{hi} = (\underline{3},4,4,4)$ ，可以看出他们头尾相接。

这样一来， $\vec{s}$ 的长度最长也只能是 $2N-1$，但如果 $\vec{f}$ 与 $\vec{t}$ 要按照 2^k 对齐，那么 $\vec{s}$ 的长度就不够了（无法在长度为 N 的乘法子群上编码成多项式）。为了解决这个问题，Plookup 选择把 $\vec{f}$ 的有效长度限制在 $N-1$，所谓有效长度是指， $\vec{f}$ 的实际长度为 N，但是其最后一条查询记录并不考虑其合法性。

于是 $\vec{s}$ 向量可以拆成两个长度为 N 的向量，其中一半 $\vec{s}^{lo} = (s_0, s_1, \ldots, s_{N-1})$，另一半 $\vec{s}^{hi} = (s_{N-1}, s_N, \ldots, s_{2N-2})$

接下来 Prover 要引入 Accumulator 辅助向量 $\vec{z}$ 来证明 Grand Product：

$$z_0 = 1,\quad z_{i+1} = z_i\cdot\frac{(1+\beta)(f_i+\gamma)(t_i+\beta\cdot t_{i+1}+\gamma(1+\beta))}{(s^{lo}_i+\beta\cdot s^{lo}_{i+1}+\gamma(1+\beta))(s^{hi}_i+\beta\cdot s^{hi}_{i+1}+\gamma(1+\beta))},$$

我们仍然看下这样一个例子： $\vec{t} = (1,2,3,4)$， $\vec{f} = (2,4,4)$，于是 $\vec{s} = (1,2,2,3,4,4,4)$，拆成两个头尾相接的向量： $\vec{s}^{lo} = (1,2,2,3)$， $\vec{s}^{hi} = (3,4,4,4)$ 。那么，我们可以把相邻元素构成的二元组向量写出来：

$\mathrm{F} = (\mathrm{f_i}, \mathrm{f_i}) = (2,2),(4,4),(4,4)\ \mathrm{T} = (\mathrm{t_i}, \mathrm{t_i}) = (1,2),(2,3),(3,4)\ \mathrm{S^{lo}} = (\mathrm{s_i^{lo}}, \mathrm{s_i^{lo}}) = (1,2),$

容易检验，他们满足下面的关系：

$$S^{lo} \cup S^{hi} =_{multiset} F \cup T$$

于是，利用一个辅助函数 $G(a,b) = a + \beta \cdot b + \gamma \cdot (1+\beta)$，我们定义 $\vec{z}$：

$$\begin{aligned}
z_0 &= 1 \\
z_1 &= \frac{G(2,2)\cdot G(1,2)}{G(1,2)\cdot G(3,4)} \\
z_2 &= \frac{G(2,2)\cdot G(1,2)\cdot G(4,4)\cdot G(2,3)}{G(1,2)\cdot G(3,4)\cdot G(2,2)\cdot G(4,4)} \\
z_3 &= \frac{G(2,2)\cdot G(1,2)\cdot G(4,4)\cdot G(2,3)\cdot G(4,4)\cdot G(3,4)}{G(1,2)\cdot G(3,4)\cdot G(2,2)\cdot G(4,4)\cdot G(2,3)\cdot G(4,4)} = 1
\end{aligned}$$

对 $\vec{z}$ 进行编码，我们可以得到 $z(X)$ 多项式，它应该满足下面三条约束：

$$\begin{aligned}
L_0(X)\cdot(z(X)-1) &= 0 \\
L_{N-1}(X)\cdot(s^{lo}(X) - s^{hi}(X)) &= 0 \\
L_{N-1}(X)\cdot(z(X)-1) &= 0
\end{aligned}$$

此外，根据 $\vec{z}$ 的递推关系， $z(X)$ 还要满足下面的约束：

$$\begin{aligned}
&(X-\omega^{N-1})\cdot z(X)\cdot\Big((1+\beta)(f(X)+\beta)\Big)\cdot\Big(t(X)+\beta\cdot t(\omega \\
-&(X-\omega^{N-1})\cdot z(\omega\cdot X)\cdot\Big(s^{lo}(X)+\beta\cdot s^{lo}(\omega\cdot X)+\gamma(1+\beta)\Big)\cdot\Big(s^{hi}(X)+\beta\cdot s^{hi}(\omega
\end{aligned}$$

总共有四条多项式约束，这里略去完整的协议。

Plonkup 的优化

在论文 Plonkup 论文中给出了一个简化方法，可以去除一个多项式约束。在 Plookup 方案中， $\vec{s}$ 向量被拆分成两个向量， $\vec{s}^{lo}$ 与 $\vec{s}^{hi}$ ，但要要求这两个向量头尾相接。

Plonkup 给出了一种新的拆分方案，即按照 $\vec{s}$ 的奇偶项进行拆分，拆成 $\vec{s}^{even}$ 与 $\vec{s}^{odd}$
：

$$
\begin{aligned}
\vec{s}^{even} &= (s_0, s_2, s_4, \ldots, s_{2n-2}) \\
\vec{s}^{odd} &= (s_1, s_3, s_5, \ldots, s_{2n-1})
\end{aligned}
$$

注意，这里不再需要限制 $\vec{f}$ 的长度为 $N-1$，而是可以到 N，这样 $\vec{s}$ 的长度可以到 $2N$，拆分成两个长度为 N 的向量，之所以可以去除这个限制，是因为 $(\vec{f}, \vec{t}, \vec{s}^{even}, \vec{s}^{odd})$ 之间的关系可以在 H 回卷到起始位置，这样只需要要求 $z_0 = 1$ 即可。 $\vec{z}$ 向量可以重新定义为：

$$
z_0 = 1, \quad z_{i+1} = z_i \cdot \frac{(1+\beta)(f_i+\gamma)(t_i + \beta \cdot t_{i+1} + \gamma(1+\beta))}{(s_i^{even} + \beta \cdot s_i^{odd} + \gamma(1+\beta))(s_i^{odd} + \beta \cdot s_{i+1}^{even} + \gamma(1+\beta))}
$$

我们可以举一个简单的例子：假设 $N = 4$， $\vec{t} = (1,2,3,4)$， $\vec{f} = (2,4,4,1)$，于是 $\vec{s} = (1,1,2,2,3,4,4,4)$

$$
\vec{s}^{even} = (1,2,3,4), \qquad \vec{s}^{odd} = (1,2,4,4)
$$

$$
\begin{aligned}
F &= (f_i, f_i) = \{(2,2),(4,4),(4,4),(1,1)\} \\
T &= (t_i, t_{i+1}) = \{(1,2),(2,3),(3,4),(4,1)\} \\
S^{\text{even}} &= \left(s_i^{\text{even}}, s_i^{\text{odd}}\right) = \{(1,1),(2,2),(3,4),(4,4)\} \\
S^{\text{odd}} &= \left(s_i^{\text{odd}}, s_{i+1}^{\text{even}}\right) = \{(1,2),(2,3),(4,4),(4,1)\}
\end{aligned}
$$

容易检验，他们满足下面的关系：

$$
S^{even} \cup S^{odd} =_{multiset} F \cup T
$$

我们也可以通过定义 $\vec{z}$ ，并仔细检查每一项，确认只需要约束 $z_0 = 1$ 就可以约束 $\vec{f}$ 与 $\vec{s}$ 的正确性。

$$
\begin{aligned}
z_0 &= 1 \\
z_1 &= \frac{G(2,2) \cdot G(1,2)}{G(1,1) \cdot G(1,2)} \\
z_2 &= \frac{G(2,2) \cdot G(1,2) \cdot G(4,4) \cdot G(2,3)}{G(1,1) \cdot G(1,2) \cdot G(2,2) \cdot G(2,3)} \\
z_3 &= \frac{G(2,2) \cdot G(1,2) \cdot G(4,4) \cdot G(2,3) \cdot G(4,4) \cdot G(3,4)}{G(1,1) \cdot G(1,2) \cdot G(2,2) \cdot G(2,3) \cdot G(3,4) \cdot G(4,4)} \\
z_4 &= \frac{G(2,2) \cdot G(1,2) \cdot G(4,4) \cdot G(2,3) \cdot G(4,4) \cdot G(3,4) \cdot G(1,1) \cdot G(4,1)}{G(1,1) \cdot G(1,2) \cdot G(2,2) \cdot G(2,3) \cdot G(3,4) \cdot G(4,4) \cdot G(4,4) \cdot G(4,1)}
\end{aligned}
$$

这里辅助函数 $G(a,b) = a + \beta \cdot b + \gamma \cdot (1+\beta)$。

于是多项式 $z(X)$ 只需要满足如下两条约束：

$$L_0(X)(z(X)-1)=0$$

还有

$$\frac{z(\omega\cdot X)}{z(X)}=\frac{(1+\beta)(f(X)+\gamma)(t(X)+\beta\cdot t(\omega\cdot X)+\gamma(1+\beta))}{(s^{even}(X)+\beta\cdot s^{odd}(X)+\gamma(1+\beta))(s^{odd}(X)+\beta\cdot s^{even}(\omega\cdot X)}$$

多列表格与多表格扩展

通常查询表是一个多列的表，比如一个 8bit-XOR 计算表是一个三列的表。对于 Plookup 方案与 Halo2-lookup 方案，我们直接可以通过随机挑战数来把一个多列表格折叠成一个单列表格。

假如计算表格为 $(\vec{t}_1,\vec{t}_2,\vec{t}_3)$，那么相应的查询记录也应该是个三列的表格，记为 $(\vec{f}_1,\vec{f}_2,\vec{f}_3)$。如果 $(f_{1,i},f_{2,i},f_{3,i})=(t_{1,j},t_{2,j},f_{3,j})$，对所有的 $i\in[0,N)$都成立，那么 $(\vec{f}_1,\vec{f}_2,\vec{f}_3)$ 是一个合法的查询记录。 通过向 Verifier 要一个随机挑战数 η，我们可以把计算表格横向折叠起来：

$$\vec{t}=\vec{t}_1+\eta\cdot\vec{t}_2+\eta^2\cdot\vec{t}_2$$

同样，Prover 在证明过程中，也将查询记录横向折叠起来：

$$\vec{f}=\vec{f}_1+\eta\cdot\vec{f}_2+\eta^2\cdot\vec{f}_2$$

接下来，Prover 和 Verifier 可以利用单列表格查询协议（ Plookup 协议或 Halo2-lookup 协议）完成证明过程。

如果存在多张不同的表格，那么可以给这些表格增加公开的一列，用来标记表格编号，这样可以把多表格视为增加一列的多列的单一表格。

与 Plonk 协议的整合

由于计算表格 $\vec{t}$ 是一个预定义的多列表格，因此它可以在 Preprocessing 阶段进行承诺计算，并把这些表格的承诺作为后续协议交互的公开输入。

在 Plonk 协议中，因为我们把表格的查询视为一种特殊的门，因此查询记录 $\vec{f}$ 本质上正是 $(\vec{w}_a, \vec{w}_b, \vec{w}_c)$ 的折叠。为了区分「查询门」和「算术门」，我们还需要增加一个选择向量 q_K，标记 Witness table 中的某一行是算术门，还是查询门。

下面我们按照 Plonkup 论文中的协议，大概描述下如何将 Lookup Argument 整合进 Plonk 协议。

预处理：Prover 和 Verifier 构造 $[q_L(X)]$，$[q_R(X)]$，$[q_O(X)]$，$[q_M(X)]$，$[q_C(X)]$，$[q_K(X)]$，$[\sigma_a(X)]$，$[\sigma_b(X)]$，$[\sigma_c(X)]$，$[t_1(X)]$，$[t_2(X)]$，$[t_3(X)]$

第一步：Prover 针对 W 表格的每一列，构造 $[w_a(X)]$，$[w_b(X)]$，$[w_c(X)]$，$\phi(X)$ 使得

$$q_L(X)w_a(X) + q_R(X)w_b(X) + q_M(X)w_a(X)w_b(X) - q_O(X)w_c(X) + q_C(X$$

第二步：Verifier 发送随机数 η，用以折叠表格

第三步：Prover 构造并发送 $[f(X)]$ 与 $[t(X)]$，分别编码 $\vec{f}$ 与 $\vec{t} = \vec{t}_1 + \eta \cdot \vec{t}_2 + \eta^2 \cdot \vec{t}_3$，其中 $\vec{f}$ 计算如下

$$f_i = \begin{cases} w_{a,i} + \eta \cdot w_{b,i} + \eta^2 \cdot w_{c,i}, & \text{if } q_K(i) = 1 \\ t_{1,N-1} + \eta \cdot t_{2,N-1} + \eta^2 \cdot t_{3,N-1}, & \text{if } q_K(i) = 0 \end{cases}$$

这里请注意，当 $q_K(\omega_i) = 0$ 时，表示这一行约束不是查询门，因此需要填充上一个存在 $\vec{t}$ 中的值，这里我们取表格的最后一个元素作为查询记录填充。

Prover 计算 $\vec{s}$，并拆分为 $\vec{s}^{even}$ 与 $\vec{s}^{odd}$，构造并发送 $[s^{even}(X)]$ 与 $[s^{odd}(X)]$

第四步： Verifier 发送随机数 (β_1, γ_1) 与 (β_2, γ_2)

第五步：Prover 构造（并发送）拷贝约束累乘多项式 $[z(X)]$，使得

$$L_0(X)(z(X) - 1) = 0$$
$$z(\omega \cdot X)g_2(X) - z(X)g_1(X) = 0$$

其中

$$g_1(X) = \Big(w_a(X) + \beta_1 \cdot id_a(X) + \gamma_1\Big)\Big(w_b(X) + \beta_1 \cdot id_b(X) + \gamma_1\Big)\Big(w_c(X) +$$

$$g_2(X) = \Big(w_a(X) + \beta_1 \cdot \sigma_a(X) + \gamma_1\Big)\Big(w_b(X) + \beta_1 \cdot \sigma_b(X) + \gamma_1\Big)\Big(w_c(X) +$$

Prover 构造（并发送）查询累乘多项式 $[z'(X)]$，使得：

$$L_0(X)(z'(X)-1)=0$$
$$z'(\omega\cdot X)g_4(X)-z'(X)g_3(X)=0$$

其中

$$g_3(X)=\Big((1+\beta_2)(f(X)+\gamma_2)\Big)\cdot\Big(t(X)+\beta_2\cdot t(\omega\cdot X)+\gamma_2(1+\beta_2)\Big)$$
$$g_4(X)=\Big(s^{even}(X)+\beta_2\cdot s^{odd}(X)+\gamma_2(1+\beta_2)\Big)\cdot\Big(s^{odd}(X)+\beta_2\cdot s^{even}(\omega\cdot$$

第六步：Verifier 发送随机挑战数 α

第七步：Prover 计算 $h(X)$，并构造商多项式 $[t(X)]$

$$\begin{aligned}t(X)\cdot z_H(X)=&\ q_L(X)w_a(X)+q_R(X)w_b(X)+q_M(X)w_a(X)w_b(X)-q_O(\\&+\alpha(z(\omega X)\cdot g_2(X)-z(X)\cdot g_1(X))\\&+\alpha^2(L_0(X)\cdot(z(X)-1))\\&+\alpha^3(q_K(X)\cdot(w_a(X)+\eta w_b(X)+\eta^2w_c(X)-f(X)))\\&+\alpha^4(z'(\omega X)\cdot g_4(X)-z'(X)\cdot g_3(X))\\&+\alpha^5(L_0(X)\cdot(z'(X)-1))\end{aligned}$$

后续步：Verifier 发送随机挑战数 ζ，Prover 打开各个多项式，Verifier 自行计算 $z_H(\zeta)$ 与 $L_0(\zeta)$，并验证各个多项式在 ζ 与 $\omega\cdot\zeta$ 处的计算证明，并验证这些打开点满足上面等式。

完整的协议请参考Plonkup论文 [2]。

Reference

- [1] Ariel Gabizo, Dmitry Khovratovich. flookup: Fractional decomposition-based lookups in quasi-linear time independent of table size. https://eprint.iacr.org/2022/1447.

- [2] Luke Pearson, Joshua Fitzgerald, Héctor Masip, Marta Bellés-Muñoz, and Jose Luis Muñoz-Tapia. PlonKup: Reconciling PlonK with plookup. https://eprint.iacr.org/2022/086.

- [3] https://zcash.github.io/halo2/design/proving-system/lookup.html
- [4] Ariel Gabizon. Multiset checks in PLONK and Plookup. https://hackmd.io/@arielg/ByFgSDA7D
- [5] Modified Lookup Argument (improved). https://hackmd.io/_Q8YR_JLTvefW3kK92KOFgv

---