理解 PLONK（二）：多项式编码

在上篇文章里，我们可以把电路的计算的「合法性检查」转换成一组加法/乘法约束。假如总共有 N 个约束，那么Prover 可以通过多项式编码的方式把多个约束压缩成一个约束，让 Verifier 轻松检查。

多项式的概率检查

把多个约束验证合并的神奇能力来自于「多项式随机挑战」。如果有两个多项式 $f(X)$ 和 $g(X)$ 同为两个次数不超过 d 的多项式。那么 Verifier 只需要给出一个随机挑战值 $\zeta \in \mathbb{F}$，计算 $f(\zeta)$ 是否等于 $g(\zeta)$ 即可大概率得知 $f(X) = g(X)$，其中出错的概率 $\leq \frac{d}{|\mathbb{F}|}$。只要保证 $\mathbb{F}$ 足够大，那么检查出错的概率就可以忽略不计。

这个原理被称为 Schwartz-Zippel 定理。

假如要验证两个向量 $\vec{a} + \vec{b}$ 是否等于 $\vec{c}$，为了可以一步挑战验证，我们要先把三个向量编码成多项式。

一种最直接的方案是把向量当作多项式的「系数」进行编码

$$
\begin{aligned}
a(X) &= a_0 + a_1X + a_2X^2 + \cdots + a_{n-1}X^{n-1} \\
b(X) &= b_0 + b_1X + b_2X^2 + \cdots + b_{n-1}X^{n-1} \\
c(X) &= c_0 + c_1X + c_2X^2 + \cdots + c_{n-1}X^{n-1}
\end{aligned} \tag{1}
$$

显然，如果 $a_i + b_i = c_i$，那么 $a(X) + b(X) = c(X)$。然后我们可以通过挑战一个随机数 ζ 来检验三个多项式在 $X = \zeta$ 处的取值，验证：

$$
a(\zeta) + b(\zeta) \stackrel{?}{=} c(\zeta) \tag{2}
$$

如果上式成立，那么 $\vec{a} + \vec{b} = \vec{c}$。

Lagrange 插值 与 Evaluation Form

假如我们要验证 $\vec{a} \circ \vec{b} \stackrel{?}{=} \vec{c}$，用系数编码的方式就不容易处理了，因为 $a(X) \cdot b(X)$ 会产生很多的交叉项。并且 $a_i \cdot b_i$ 和 c_i 的项并不对应到 X^i 的系数，比如 $a_1 \cdot b_1$ 的系数出现在 X^2 上，但同时 X^2 项的系数组成还有 $a_0 \cdot b_2$ 和 $a_2 \cdot b_0$。而 c_1 是 X^1 的系数。

我们需要另一种多项式编码方案，利用 Lagrange Basis。如果我们要构造多项式 $a(X)$，使得它在定义域 $H = (w_0, w_1, \dots w_{N-1})$ 上的取值为 $\vec{a}$，即

$$
\begin{aligned}
a(w_0) &= a_0 \\
a(w_1) &= a_1 \\
&\vdots \\
a(w_{N-1}) &= a_{N-1}
\end{aligned} \tag{3}
$$

插值需要用到一组插值多项式：$\{L_i(X)\}_{i \in [0, N-1]}$，其中 $L_i(w_i) = 1$，并且 $L_i(w_j) = 0 (j \neq i)$。然后 $\vec{a}$ 可以按如下方式编码：

$$a(X) = a_0 \cdot L_0(X) + a_1 \cdot L_1(X) + a_2 \cdot L_2(X) + \cdots + a_{N-1} \cdot L_{N-1}(X) \tag{4}$$

可以简单心算一下，当 $X = w_0$ 时，等式右边除了第一项之外，其他项都等于零，于是 $a(w_0) = a_0$。看起来 $L_i(X)$ 像是一个选择器，这组多项式又被称为 Lagrange Polynomials。

我们用同样的方法来编码 $b(X)$ 和 $c(X)$：

$$\begin{aligned} b(X) &= b_0 \cdot L_0(X) + b_1 \cdot L_1(X) + b_2 \cdot L_2(X) + \cdots + b_{N-1} \cdot L_{N-1}(X) \\ c(X) &= c_0 \cdot L_0(X) + c_1 \cdot L_1(X) + c_2 \cdot L_2(X) + \cdots + c_{N-1} \cdot L_{N-1}(X) \end{aligned} \tag{5}$$

如果 $a_i \cdot b_i = c_i$ 成立，那么 $a(w_i) \cdot b(w_i) = c(w_i)$。如果 $\vec{a} \circ \vec{b} = \vec{c}$，那么

$$a(X) \cdot b(X) = c(X), \quad \forall X \in H \tag{6}$$

我们现在已经把两个向量的按位乘积问题转换到了三个多项式之间的关系，接下来的问题是如何进行随机挑战验证。

我们发现：如果直接让 Verifier 发送随机数 ζ 挑战上面的等式，那么 ζ 只能属于 H。如果只存在一个 j 使得 $a_j \cdot b_j \neq c_j$，那么 Verifier 的一次挑战能发现这个错误的概率只有 $\frac{1}{|n|}$，这样 Verifier 需要挑战多次才能缩小检测出错的概率。不过这样不满足我们的要求，我们希望只通过一次挑战来检测出 Prover 的作弊行为。

我们可以把上面的等式的 X 取值范围去除，换成下面的等式：

$$a(X) \cdot b(X) - c(X) = q(X) \cdot z_H(X), \quad \forall X \in \mathbb{F} \tag{7}$$

这个等式在整个 $\mathbb{F}$ 定义域上都成立。这是为何?

首先我们看等式左边的多项式： $a(X) \cdot b(X) - c(X)$，不妨定义为 $f(X)$。我们可以看到 $f(X)$ 在 $X \in H$ 上等于零，那么意味着 H 恰好是 $f(X)$ 的「根集合」。于是 $f(X)$ 可以按照下面的方式进行因式分解：

$$f(X) = (X - w_0)(X - w_1)(X - w_2) \cdots (X - w_{N-1}) \cdot q(X) \tag{8}$$

换个说法， $f(X)$ 可以被多项式 $z_H(X) = (X - w_0)(X - w_1)(X - w_2) \cdots (X - w_{n-1})$ 整除，并得到一个商多项式 $q(X)$。零多项式 $z_H(X)$ 又被称为 Vanishing Polynomial。

如果我们让 Prover 计算出这个 $q(X)$，并且发送给 Verifier，又因为 H 是已知的系统参数，Verifier 可以自行计算 $z_H(X)$，那么 Verifier 只需要一次随机检测即可判断 $a(X) \cdot b(X) - c(X)$ 是否在 H 处等零。

$$a(\zeta) \cdot b(\zeta) - c(\zeta) \stackrel{?}{=} q(\zeta) \cdot z_H(\zeta) \tag{9}$$

进一步，如果我们使用多项式承诺（Polynomial Commitment），Verifier 可以让 Prover 来帮忙计算这些多项式在 $X = \zeta$ 处的取值，发送并证明这些值的正确性，这样能最大限度地减少 Verifier 的工作量。

但是， Verifier 计算 $z_H(\zeta)$ 需要 $O(n)$ 的计算量。

那能否让 Verifier 继续减少工作量？答案是可以的，只要我们选择特殊的 $H \subset \mathbb{F}$ 。

单位根 Roots of Unity

如果我们选择单位根作为 H，那么 $z_H(\zeta)$ 的计算量会降为 $O(\log n)$。

对于任何有限域 $\mathbb{F}_p = (0, 1, \ldots, p-1)$，其中阶数 p 为素数。那么去除零之后剩下的元素构成了乘法群 $\mathbb{F}_p^* = (1, \ldots, p-1)$，阶数为 $p-1$。由于 $p-1$ 一定为偶数，那么 $p-1$ 的乘法因子中一定包含若干个 2，假设记为 λ 个 2。那么 $\mathbb{F}_p^*$ 一定包含一个阶数为 2^λ 的乘法子群。不妨设 $n = 2^k, k \leq \lambda$，那么一定存在一个阶数为 n 的乘法子群，记为 H。该乘法子群必然含有一个生成元，记为 ω，并且 $\omega^N = 1$。这相当于把 1 开 N 次方根，因此被称为单位根。不过单位根不只有一个 ω，我们会发现 $\omega^2, \omega^3, \ldots, \omega^{N-1}$ 都满足单位根的特性，即 $(\omega^k)^N = 1, k \in (2, 3, \ldots, N-1)$。那么所有这些由 ω 产生的单位根就组成了乘法子群 H：

$$H = (1, \omega, \omega^2, \omega^3, \ldots, \omega^{N-1}) \tag{10}$$

这些元素满足一定的对称性：比如 $\omega^{\frac{N}{2}} = -1$，$\omega = -\omega^{\frac{N}{2}+1}$，$\omega^i = -\omega^{\frac{N}{2}+i}$。又比如把所有的单位根求和，我们会得到零：

$$\sum_{i=0}^{N-1} \omega^i = 0 \tag{11}$$

举一个简单的例子，我们可以在 $\mathbb{F}_{13}$ 中找到一个阶数为 4 的 H。

$$\mathbb{F}_{13} = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12) \tag{12}$$

其中乘法群的生成元为 $g = 2$。由于 $13 - 1 = 3 * 2 * 2$，所以存在一个阶数为 4 的乘法子群，其生成元为 $\omega = 5$：

$$H = (\omega^0 = 1, \omega^1 = 5, \omega^2 = 12, \omega^3 = 8) \tag{13}$$

而 $\omega^4 = 1 = \omega^0$。

在实际应用中，我们会选择一个较大的有限域，它能有一个较大的 Powers-of-2 乘法子群。比如椭圆曲线 `BN254` 的 Scalar Field，含有一个阶数为 2^{28} 的乘法子群，`BLS-12-381` 的Scalar Field 含有一个阶数为 2^{32} 的乘法子群。

在乘法子群 H 上，具有下面的性质：

$$z_H(X) = \prod_{i=0}^{N-1}(X - \omega^i) = X^N - 1 \tag{14}$$

我们可以进行简单的推导，假设 $N = 4$，由于 ω^i 的对称性，这个计算过程可以不断化简：

$$
\begin{aligned}
&(X-\omega^0)(X-\omega^1)(X-\omega^2)(X-\omega^3)\\
=&(X-1)(X-\omega)(X+1)(X-\omega^3)\\
=&(X^2-1)(X-\omega)(X+\omega)\\
=&(X^2-1)(X^2-\omega^2)\\
=&(X^2-1)(X^2+1)\\
=&(X^4-1)
\end{aligned}
\tag{15}
$$

Lagrange Basis

对于 Lagrange 多项式，$L_i(w_i)=1$，并且 $L_i(w_j)=0,(j\neq i)$。接下来，我们给出 $L_i(X)$ 的构造。

为了构造 $L_i(X)$，先构造不等于零的多项式部分。由于 $L_i(\omega_j)=1, j=i$，因此他一定包含 $\prod_{j,j\neq i}(X-\omega_j)$ 这个多项式因子。但该因子显然在 $X=\omega_i$ 处可能不等于 1，即可能 $\prod_{j,j\neq i}(\omega_i-\omega_j)\neq 1$。然后，我们只要让该因子除以这个可能不等于 1 的值即可，于是 $L_i(X)$ 定义如下：

$$
L_i(X)=\frac{\prod_{j\in H\setminus\{i\}}(X-\omega_j)}{\prod_{j\in H\setminus\{i\}}(\omega_i-\omega_j)}=\prod_{j\in H\setminus\{i\}}\frac{X-\omega_j}{\omega_i-\omega_j}
\tag{16}
$$

不难发现，$L_i(X)$ 在 $X=\omega_i$ 处等于 1，其它位置 $X=\omega_j, j\neq i$ 处等于 0。

对于任意次数小于 N 的多项式 $f(X)$，那么它都可以唯一地表示为：

$$
f(X)=a_0\cdot L_0(X)+a_1\cdot L_1(X)+a_2\cdot L_2(X)+\cdots+a_{N-1}\cdot L_{N-1}(X)
\tag{17}
$$

我们可以用多项式在 H 上的值 $(a_0,a_1,a_2,\ldots,a_{N-1})$ 来表示 $f(X)$。这被称为 多项式的求值形式（Evaluation Form），区别于系数形式（Coefficient Form）。

两种形式可以在 H 上可以通过 (Inverse) Fast Fourier Transform 算法来回转换，计算复杂度为 $O(N\log N)$。

多项式的约束

利用 Lagrange Basis 我们可以方便地对各种向量计算进行约束。

比如我们想约束 $\vec{a}=(h,a_1,a_2,\ldots,a_{N-1})$ 向量的第一个元素为 h。那么我们可以这个向量进行编码，得到 $a(X)$，并且进行如下约束：

$$
L_0(X)(a(X)-h)=0,\quad \forall X\in H
\tag{18}
$$

Verifier 可以挑战验证下面的多项式等式：

$$
L_0(X)(a(X)-h)=q(X)\cdot z_H(X)
\tag{19}
$$

再比如，我们想约束 $\vec{a}=(h_1,a_1,a_2,\ldots,a_{N-2},h_2)$ 向量的第一个元素为 h_1，最后一个元素为 h_2，其它元素任意。那么 $a(X)$ 应该满足下面两个约束。

$$
\begin{aligned}
L_0(X)\cdot(a(X)-h_1)=0,\quad \forall X\in H\\
L_{N-1}(X)\cdot(a(X)-h_2)=0,\quad \forall X\in H
\end{aligned}
\tag{20}
$$

那么通过 Verifier 给一个随机挑战数（α），上面两个约束可以合并为一个多项式约束：

$$L_0(X) \cdot (a(X) - h_1) + \alpha \cdot L_{n-1}(X) \cdot (a(X) - h_2) = 0, \quad \forall X \in H \tag{21}$$

接下来，Verifier 只要挑战下面的多项式等式即可：

$$L_0(X) \cdot (a(X) - h_1) + \alpha \cdot L_{n-1}(X) \cdot (a(X) - h_2) = q(X) \cdot z_H(X) \tag{22}$$

如果想验证 $\vec{a}$ 和 $\vec{b}$ 两个等长向量除第一个元素之外，其它元素都相等，那要如何约束呢？假设 $a(X)$ 和 $b(X)$ 为两个向量的多项式编码，那么它们应该满足：

$$(X - \omega^0)(a(X) - b(X)) = 0 \tag{23}$$

当 $X = \omega^0$ 时，左边多项式的第一个因子等于零，而 $X \in H \backslash \{\omega^0\}$ 时，则左边第二因子等于零，即表达了除第一项可以不等之外，其它点取值都必须相等。

可以看出，采用 Lagrange 多项式，我们可以灵活地约束多个向量之间的关系，并且可以把多个约束合并在一起，让 Verifier 仅通过很少的随机挑战就可验证多个向量约束。

Coset

在素数有限域的乘法群中，对于每一个乘法子群 H，都有多个等长的陪集（Coset），这些 Coset 具有和 H 类似的性质，在 Plonk 中也会用到 Coset 的概念，这里只做部分性质的介绍。

还拿 $\mathbb{F}_{13}$ 为例，我们取 $H = (1, 5, 12, 8)$，并且乘法群的生成元 $g = 2$。于是我们可以得到下面两个 Coset：

$$\begin{aligned} H_1 &= g \cdot H = (g, g\omega, g\omega^2, g\omega^3) &&= (2, 10, 11, 3) \\ H_2 &= g^2 \cdot H = (g^2, g^2\omega, g^2\omega^2, g^2\omega^3) &&= (4, 7, 9, 6) \end{aligned} \tag{24}$$

可以看到 $\mathbb{F}_{13}^* = H \cup H_1 \cup H_2$，并且它们交集为空，没有任何重叠。并且它们的 Vanishing Polynomial 也可以快速计算：

$$z_{H_1}(X) = X^N - g^N, \quad z_{H_2}(X) = X^N - g^{2N} \tag{25}$$

References

- Schwartz–Zippel lemma. https://en.wikipedia.org/wiki/Schwartz%E2%80%93Zippel_lemma