Basefold 笔记：可折叠线性编码

Basefold 可以看成是对 FRI 的扩展，从而能支持 Multi-linear Polynomial 的 Proximity Proofs 和 Evaluation Arguments。与 Libra-PCS, Hyrax-PCS，Virgo-PCS 相比，Basefold 不依赖 MLE Quotients Equation 来证明一个 MLE 在 $\mathbf{u}$ 点处的取值：

$$\tilde{f}(X_0, X_1, X_2, \ldots, X_{n-1}) = \sum_{i_0=0}^{n-1}(X_i - u_i) \cdot q_i(X_0, X_1, X_2, \ldots, X_{i-1}) \tag{1}$$

Basefold 而是利用了 Sumcheck 协议，将 $\tilde{f}(\mathbf{X})$ 在一个事先给定的点的取值归约为 $\tilde{f}(\mathbf{X})$ 在一个随机点 $\vec{\alpha}$ 点的取值。而后者则可以利用类似 FRI 的思想，利用 Verifier 给出的挑战向量 $\vec{\alpha}$ 对 $\tilde{f}(\mathbf{X})$ 进行递归折叠，一方面这样 Prover 可以证明 $\tilde{f}(\mathbf{X})$ 的 Degree 的上界（Proof of Proximity），同时，根据 MLE 的特征，Prover 还同时证明了 $\tilde{f}(\vec{\alpha})$ 的取值。这样，结合 Sumcheck 协议与 FRI 风格的折叠协议，我们就可以优雅地得到一个 MLE 的 Evaluation Argument。

Basefold 的另一个重要洞见是如何在任意的有限域上使用 FRI 风格的 Proof of Proximity。我们知道， FRI 协议充分利用了有限域上的 Algebraic FFT 的结构并进行交互式的折叠，从而可以在保持编码的最小相对海明距离（Minimum Relative Hamming Distance）下界基本不变的情况下，把码字（Codeword）的长度以指数级的方式缩短，从而让 Verifier 可以轻松验证一个折叠后的短码字。但对于一般情况下难以构造 FFT 的有限域来说，我们无法直接使用 FRI 协议。

Basefold 引入了 Random Foldable Codes 的概念。这是一种采用递归方式进行编码的框架，递归编码可以看成是递归折叠的逆向过程。在 Basefold 协议的 Commit 阶段，Prover 首先利用一个基础编码方案 G_0 对消息的各个分段分别编码，然后对这些码字进行两两编码（这个过程类似 FFT 运算中的蝴蝶操作），最后得到一个单码字。在 Commit-phase 阶段，Prover 和 Verifier 以交互的方式对承诺的单个码字进行对半折叠，然后承诺折半后的码字。我们可以证明这个折半后的码字的 Relative Hamming Distance 仍然大于一个明确的下界。然后双方继续进行折叠，直到折叠到长度与 G_0 编码的长度为止。这样双方就得到了一系列码字的承诺。注意到由于编码过程是按照递归的方式来编码，因此码字就拥有了类似 RS Code 编码的折叠能力。这样做的优势很明显，采用递归折叠的 FRI 协议要相比 Tensor Codes 一类协议来说，证明尺寸要显著更优。同时，Basefold 的技术可以应用在任意有限域（$|\mathbb{F}| > 2^{10}$）上，包括扩张域 $\mathbb{F}_{p^k}$。当然，与 FRI 协议类似，Basefold 协议的承诺计算需要 $O(N\log(N))$ 的时间复杂度，而在随后的证明过程中，Prover 的计算量则保持在 $O(N)$，如果采用 Merkle Tree 作为 Codeword 的承诺方案，那么证明尺寸的复杂度为 $O(\log^2(N))$。

本文我们先了解下 Foldable Linear Codes 的概念。

什么是 Foldable linear codes

假设基于有限域 $\mathbb{F}_p$ 的一个线性编码 $C_0 : \mathbb{F}_p^{k_0} \to \mathbb{F}_p^{n_0}$，其中消息长度为 k_0，码字的长度为 n_0，假设编码后的长度相比消息长度，放大了 R 倍，即 $1/R$ 为传统概念的码率（Code Rate）。根据线性编码的定义，一定存在一个编码矩阵 $G_0 : \mathbb{F}_p^{k_0 \times n_0}$，使得

$$\mathbf{m}G_0 = \mathbf{c} \tag{2}$$

那么，我们可以基于编码矩阵 G_0 构造一个新的编码矩阵 G_1：

$$G_1 = \begin{bmatrix} G_0 & G_0 \\ G_0 \cdot T_0 & G_0 \cdot T_0' \end{bmatrix} \tag{3}$$

其中 T_0 与 T_0' 是两个对角矩阵，它们的对角元素为编码方案的参数，我们稍后会解释 T_0 与 T_0' 的参数选择。

如果把 G_1 看成是另一个线性编码 C_1 的编码矩阵，那么 C_1 的参数为 $[R, 2k_0, 2n_0]$。也就是与 C_0 相比，C_1 的码率保持不变，但消息和码字的长度扩大为两倍。

举个例子，如果假设 $\mathbf{m} = (m_0, m_1, m_2, m_3)$，基础编码方案 C_0 为一个简易的 Reptition Code，$k_0 = 2, n_0 = 4$。这样 $\mathbf{m}$ 的消息长度正好满足 G_1 的要求，即 $k_1 = 2 \cdot k_0 = 4$，编码后的长度为 $n_1 = 2 \cdot n_0 = 8$。

基础编码 C_0 的编码矩阵 G_0 定义为：

$$G_0 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \tag{4}$$

假设系统参数 T_0 的对角元素为 (t_0, t_1, t_2, t_3)，而 T_0' 的对角元素为 (t_0', t_1', t_2', t_3')，我们按照上面的公式来构造出 G_1：

$$G_1 = \left[\begin{array}{cccc|cccc} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ \hline t_0 & 0 & t_2 & 0 & t_0' & 0 & t_2' & 0 \\ 0 & t_1 & 0 & t_3 & 0 & t_1' & 0 & t_3' \end{array}\right] \tag{5}$$

直接代入 $\mathbf{m}$，我们可以得到:

$$\mathbf{m}G_1 = \begin{bmatrix} m_0 & m_1 & m_2 & m_3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ t_0 & 0 & t_2 & 0 & t_0' & 0 & t_2' & 0 \\ 0 & t_1 & 0 & t_3 & 0 & t_1' & 0 & t_3' \end{bmatrix} \tag{6}$$

编码后的码字向量我们单独列出：

$$\begin{matrix} m_0 + t_0 m_2 & m_1 + t_1 m_3 & m_0 + t_2 m_2 & m_1 + t_3 m_3 \\ m_0 + t'_0 m_2 & m_1 + t'_1 m_3 & m_0 + t'_2 m_2 & m_1 + t'_3 m_3 \end{matrix} \tag{7}$$

直接用矩阵运算，我们不太能看清楚这个编码的结构。我们换一个思路，消息 $\mathbf{m}$ 可以拆分为左右等长的两部分，左边 $\mathbf{m}_l = (m_0, m_1)$ 与 右边 $\mathbf{m}_r = (m_2, m_3)$，

$$\mathbf{m}G_1 = (\mathbf{m}_l \parallel \mathbf{m}_r)G_1 = [1 \quad 1] \begin{bmatrix} \mathbf{m}_l & 0 \\ 0 & \mathbf{m}_r \end{bmatrix} \begin{bmatrix} G_0 & G_0 \\ G_0 \cdot T_0 & G_0 \cdot T'_0 \end{bmatrix} = [1 \quad 1] \begin{bmatrix} \mathbf{m}_l G_0 & \mathbf{m}_l G_0 \\ \mathbf{m}_r G_0 T_0 & \mathbf{m}_r G_0 T'_0 \end{bmatrix} \tag{8}$$

对于等式右边的 2×2 矩阵，我们先计算其左上角与右上角的两个相等的子矩阵，即用 G_0 对 $\mathbf{m}_l$ 进行编码，得到：

$$\mathbf{m}_l G_0 = [m_0 \quad m_1] \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} = [m_0 \quad m_1 \quad m_0 \quad m_1] \tag{9}$$

还有左下角的子矩阵：

$$\mathbf{m}_r(G_0 T_0) = (\mathbf{m}_r G_0) T_0 = [m_2 \quad m_3] \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} T_0 = [m_2 \quad m_3 \quad m_2 \quad m_3] \begin{bmatrix} t_0 & 0 & 0 & 0 \\ 0 & t_1 & 0 & 0 \\ 0 & 0 & t_2 & 0 \\ 0 & 0 & 0 & t_3 \end{bmatrix} = (t_0 m_2, t_1 m_3, t_2 m_2, t_3 m_3) \quad ($$

同理 $\mathbf{m}_r(G_0 T'_0)$ 的结果类似，只是 t_i 换成了 t'_i：

$$\mathbf{m}_r(G_0 T'_0) = (t'_0 m_2, t'_1 m_3, t'_2 m_2, t'_3 m_3) \tag{11}$$

不难验证，我们一样可以得到 $\mathbf{m}$ 在 G_1 编码后的结果。

> TODO: 补充验证的例子。

我们可以简化这个计算过程如下等式：

$$\mathbf{m}G_1 = \mathbf{m}_l G_0 + (t_0, t_1, t_2, t_3) \circ \mathbf{m}_r G_0 \parallel \mathbf{m}_l G_0 + (t'_0, t'_1, t'_2, t'_3) \circ \mathbf{m}_r G_0 \tag{12}$$

如果我们把 k_0 看成是系统参数（常数），那么上面等式的编码计算量为 $O(n_0)$，包括两次 G_0 编码与两次 n_0 长度的对位加法。另外这个等式看起来非常类似 （Multiplicative）FFT 算法中的蝴蝶操作：

$$a' = a + t \cdot b, \quad b' = a - t \cdot b \tag{13}$$

事实上如后文所示，这个 Foldable Code 编码过程正是 RS-Code 的一个泛化扩展（Generalization）。

而我们可以继续递归地构造 $G_2, G_3, \ldots, G_d$，最终得到线性编码 $C_d : \mathbb{F}_p^{k_0} \to \mathbb{F}_p^n$，其中消息长度 $k = k_0 \cdot 2^d$，编码长度 $n = c \cdot k_0 \cdot 2^d$。码率为 $\rho = \frac{1}{R}$， 这里 k_0 为基础编码的消息长度，n_0 为基础编码长度，而基础编码的选择则非常灵活。

我们沿用符号 G_d 表示 C_d 的编码矩阵（或称为生成矩阵） $G_d \in \mathbb{F}_p^{k \times n}$，那么对于 $i \in \{1, 2, \ldots, d\}$，我们有下面的递归关系：

$$G_i = \begin{bmatrix} G_{i-1} & G_{i-1} \\ G_{i-1} \cdot T_{i-1} & G_{i-1} \cdot T'_{i-1} \end{bmatrix} \tag{14}$$

这里 T_{i-1} 和 T'_{i-1} 是两个对角矩阵，

$$T_{i-1} = \begin{bmatrix} t_{i-1,0} & 0 & \cdots & 0 \\ 0 & t_{i-1,1} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & t_{i-1,n_i-1} \end{bmatrix}, \quad T'_{i-1} = \begin{bmatrix} t'_{i-1,0} & 0 & \cdots & 0 \\ 0 & t'_{i-1,1} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & t'_{i-1,n_i-1} \end{bmatrix}, \tag{15}$$

并且他们在相同位置上的对角元素不能相等：

$$\forall j \in [0, n_i - 1], t_{i,j} \neq t'_{i,j} \tag{16}$$

假如我们可以通过上面的递归等式求出 G_d，那么对于长度为 k 的消息 $\mathbf{m}_d$，其编码过程可以直接计算如下：

$$\mathbf{w}_d = \mathbf{m}_d G_d \tag{17}$$

不过这样直接用生成矩阵 G_d 去编码的方式比较低效，其计算时间复杂度为 $O(k \cdot n)$。而按照上面的推导，如果我们不直接利用 G_d 矩阵，而是采用递归的方式来对 m_d 进行编码。基本思路是，我们把 m 拆分成两部分 m_l 和 m_r，然后用 G_{d-1} 对 m_l 和 m_r 进行分别编码，然后再把编码后的 C_l 和 C_r 拼接起来。

$$\mathbf{w}_d = \left(m_l G_{d-1} + \mathsf{diag}(T_d) \circ m_r G_{d-1}\right) \parallel \left(m_l G_{d-1} + \mathsf{diag}(T'_d) \circ m_r G_{d-1}\right) \tag{18}$$

这样采用 G_d 的编码过程就被转化为了采用 G_{d-1} 的两次编码过程。我们可以继续递归地计算编码 $m_l G_{d-1}$ 和 $m_r G_{d-1}$，直到所拆分后的消息长度满足 k_0，然后我们只要使用 G_0 生成矩阵来计算基础编码即可，其编码时间复杂度为 $O(k_0 \cdot n_0)$。这样总共需要经过 d 轮递归，其总的计算量仅为 $O(n \log(n))$。

RS code (foldable)

协议 FRI 中采用的 RS code 满足上面的条件，因此 RS code 是可折叠的。

我们先看看 RS code 中的生成矩阵：

$$\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{k-1} & \omega^{2(k-1)} & \cdots & \omega^{(n-1)(k-1)} \end{bmatrix} \tag{19}$$

其中 ω 是 n 次 Root of Unity，满足 $\omega^n = 1$，其中 n 是码字长度，k 是消息长度。我们可以看到，上述矩阵是一个范德蒙矩阵。我们接下来解释，上面这个 RS code 的生成矩阵满足 Foldable Code 的定义。为了方便演示，我们假设 $k_0 = 1, n_0 = 1$，那么 $d = 3, k_3 = 8, n_3 = 8$，$\omega^8 = 1$，

$$G^{(RS)} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \omega^4 & \omega^5 & \omega^6 & \omega^7 \\ 1 & \omega^2 & \omega^4 & \omega^6 & 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^3 & \omega^6 & \omega^1 & \omega^4 & \omega^7 & \omega^2 & \omega^5 \\ 1 & \omega^4 & 1 & \omega^4 & 1 & \omega^4 & 1 & \omega^4 \\ 1 & \omega^5 & \omega^2 & \omega^7 & \omega^4 & \omega & \omega^6 & \omega^3 \\ 1 & \omega^6 & \omega^4 & \omega^2 & 1 & \omega^6 & \omega^4 & \omega^2 \\ 1 & \omega^7 & \omega^6 & \omega^5 & \omega^4 & \omega^3 & \omega^2 & \omega \end{bmatrix} \tag{20}$$

首先，我们需要按照 Reversed Bit Order (RBO) 序对 G^{RS} 的各行进行重排。这里为何要进行 RBO 重排，这与 Multiplicative FFT 的结构有关，具体解释请参考另一篇文章。所谓的 RBO 序是指把索引的二进制表示倒序，然后把这个倒序的二进制数当作新的索引。那么 RBO 序列是 $(0, 4, 2, 6, 1, 5, 3, 7)$。这样我们可以把上面的范德蒙矩阵的各行按照 RBO 序重排，可以得到下面的矩阵，记为 G_2：

$$G_2 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \omega^4 & 1 & \omega^4 & 1 & \omega^4 & 1 & \omega^4 \\ 1 & \omega^2 & \omega^4 & \omega^6 & 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^6 & \omega^4 & \omega^2 & 1 & \omega^6 & \omega^4 & \omega^2 \\ 1 & \omega & \omega^2 & \omega^3 & \omega^4 & \omega^5 & \omega^6 & \omega^7 \\ 1 & \omega^5 & \omega^2 & \omega^7 & \omega^4 & \omega & \omega^6 & \omega^3 \\ 1 & \omega^3 & \omega^6 & \omega^1 & \omega^4 & \omega^7 & \omega^2 & \omega^5 \\ 1 & \omega^7 & \omega^6 & \omega^5 & \omega^4 & \omega^3 & \omega^2 & \omega \end{bmatrix} \tag{21}$$

观察重排后的矩阵，我们可以看出这个矩阵可以分解为 G_1 和 T_1, T_1' 的子矩阵运算：

$$G_2 = \begin{bmatrix} G_1 & G_1 \\ G_1 T_1 & G_1 T_1' \end{bmatrix} = \left[\begin{array}{cccc|cccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \omega^4 & 1 & \omega^4 & 1 & \omega^4 & 1 & \omega^4 \\ 1 & \omega^2 & \omega^4 & \omega^6 & 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^6 & \omega^4 & \omega^2 & 1 & \omega^6 & \omega^4 & \omega^2 \\ \hline 1 & \omega & \omega^2 & \omega^3 & \omega^4 & \omega^5 & \omega^6 & \omega^7 \\ 1 & \omega^5 & \omega^2 & \omega^7 & \omega^4 & \omega & \omega^6 & \omega^3 \\ 1 & \omega^3 & \omega^6 & \omega^1 & \omega^4 & \omega^7 & \omega^2 & \omega^5 \\ 1 & \omega^7 & \omega^6 & \omega^5 & \omega^4 & \omega^3 & \omega^2 & \omega \end{array}\right] \tag{22}$$

其中 G_1 为：

$$G_1 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega^4 & 1 & \omega^4 \\ 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^6 & \omega^4 & \omega^2 \end{bmatrix} \tag{23}$$

可以验证，左下角子矩阵可以表示为 $G_1 T_1$：

$$G_1 T_1 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega^4 & 1 & \omega^4 \\ 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^6 & \omega^4 & \omega^2 \end{bmatrix} \begin{bmatrix} 1 & & & \\ & \omega & & \\ & & \omega^2 & \\ & & & \omega^3 \end{bmatrix} = \begin{bmatrix} 1 & \omega & \omega^2 & \omega^3 \\ 1 & \omega^5 & \omega^2 & \omega^7 \\ 1 & \omega^3 & \omega^6 & \omega^1 \\ 1 & \omega^7 & \omega^6 & \omega^5 \end{bmatrix} \tag{24}$$

其中 T_1 矩阵正是一个对角矩阵，满足其对角线元素互不相等的要求。而右下角的子矩阵可以按如下分解：

$$\begin{bmatrix} \omega^4 & \omega^5 & \omega^6 & \omega^7 \\ \omega^4 & \omega & \omega^6 & \omega^3 \\ \omega^4 & \omega^7 & \omega^2 & \omega^5 \\ \omega^4 & \omega^3 & \omega^2 & \omega^1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega^4 & 1 & \omega^4 \\ 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^6 & \omega^4 & \omega^2 \end{bmatrix} \begin{bmatrix} \omega^4 & & & \\ & \omega^5 & & \\ & & \omega^6 & \\ & & & \omega^7 \end{bmatrix} = G_1 T_1' \tag{25}$$

而 T_1' 正好等于 $(-1)T_1$，确保 T_1' 的元素与 T_1 的元素互不相等，

$$T_2' = \begin{bmatrix} \omega^4 & & & \\ & \omega^5 & & \\ & & \omega^6 & \\ & & & \omega^7 \end{bmatrix} = \begin{bmatrix} -1 & & & \\ & -\omega & & \\ & & -\omega^2 & \\ & & & -\omega^3 \end{bmatrix} = -T_2 \tag{26}$$

接下来，我们可以继续递归分解 G_1：

$$G_1 = \left[\begin{array}{cc|cc} 1 & 1 & 1 & 1 \\ 1 & \omega^4 & 1 & \omega^4 \\ \hline 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^6 & \omega^4 & \omega^2 \end{array}\right] = \begin{bmatrix} G_0 & G_0 \\ G_0 \cdot T_0 & G_1 \cdot T_0' \end{bmatrix} \tag{27}$$

可以看出，这里 G_0, T_0, T_0' 为：

$$G_0 = \begin{bmatrix} 1 & 1 \\ 1 & \omega^4 \end{bmatrix}, \quad T_0 = \begin{bmatrix} 1 & \\ & \omega^2 \end{bmatrix}, \quad T_0' = \begin{bmatrix} \omega^4 & \\ & \omega^6 \end{bmatrix} = -T_0 \tag{28}$$

递归分解到最后，我们得到基础编码 G_0，其仍然是一个 RS-Code，记为 $RS[k_0 = 1, n_0 = 1]$ 。这样我们就证明了 RS-Code 是一个可折叠编码，不过显然 RS-Code 对 $\mathbb{F}_p$ 有要求，需要其包含一个 Order 为 2^s 的乘法子群，其中 s 为正整数，并且保证 s 足够大，能容纳 $k_0 \cdot 2^{n_d}$ 的码字长度。

References

- [ZCF23] Hadas Zeilberger, Binyi Chen, and Ben Fisch. "BaseFold: efficient field-agnostic polynomial commitment schemes from foldable codes." In *Annual International Cryptology Conference*, pp. 138-169. Cham: Springer Nature Switzerland, 2024.
- Riad S. Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. "Doubly-efficient zkSNARKs without trusted setup." In *2018 IEEE Symposium on Security and Privacy (SP)*, pp. 926-943. IEEE, 2018.
- Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. "Libra: Succinct zero-knowledge proofs with optimal prover computation." In *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part III 39*, pp. 733-764. Springer International Publishing, 2019.
- Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. "Transparent polynomial delegation and its applications to zero knowledge proof." In *2020 IEEE Symposium on Security and Privacy (SP)*, pp. 859-876. IEEE, 2020.
- Charalampos Papamanthou, Elaine Shi, and Roberto Tamassia. "Signatures of correct computation." In *Theory of Cryptography Conference*, pp. 222-242. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.