Basefold 笔记： MLE 求值证明

假设我们有一个 MLE 多项式 $\tilde{f}(\vec{X}) \in \mathbb{F}[\vec{X}]^{\leq 1}$，一个求值点 $\mathbf{u} \in \mathbb{F}^d$，以及多项式在求值点的运算结果 $v = \tilde{f}(\mathbf{u})$ 。我们想基于 Basefold-IOPP 协议来构造一个 Polynomial Evaluation Argument。

基于 FRI ，我们可以利用 DEEP Method 来构造一个 PCS 协议，即验证 Low Degree Polynomial $q(X) = (f(X) - f(u))/(X - u)$ 的存在性。但是 FRI 协议只能处理 Univariate Polynomial 的情况。而对于 MLE Polynomial，我们一般情况下需要利用下面的扩展的多项式余数定理来将其归约成商多项式存在性问题：

$$\tilde{f}(X_0, X_1, \ldots, X_{n-1}) - v = \sum_{i=0}^{n-1} (X_k - u_k) \cdot q_i(X_0, X_1, \ldots, X_{k-1}) \tag{1}$$

比如 Virgo 就是采用了这种方案的 MLE PCS 协议。然而 Basefold [ZCF23] 则给出了一个耳目一新的思路来实现 MLE 的 Evaluation Argument：结合 Basefold-IOPP 协议和 Sumcheck 协议。我们可以利用 Sumcheck 协议的求和证明来作为协议的主框架，然后利用 Basefold-IOPP 来确保 MLE 多项式的 Low degree，同时利用 Basefold-IOPP 协议在多轮折叠后所产生的 MLE 多项式在随机点的求值，来弥补 Sumcheck 协议最后一轮所需要的求值的正确性证明。

Basefold-IOPP 协议的最后输出

为了方便演示，我们采用 $d = 3$ 的 MLE 多项式 $\tilde{f}(X_0, X_1, X_2)$ 来演示，它的定义如下：

$$\tilde{f}(\vec{X}) = f_0 + f_1 \cdot X_0 + f_2 \cdot X_1 + f_3 \cdot X_0X_1 + f_4 \cdot X_2 + f_5 \cdot X_0X_2 + f_6 \cdot X_1X_2 + f_7 \cdot X_0X_1X_2 \tag{2}$$

其中 $\mathbf{f}$ 为 $\tilde{f}$ 的系数向量，上面定义为 MLE 多项式的 Coefficients Form。

我们回顾一下 Basefold-IOPP 协议的折叠过程，首先令 $f^{(3)}(X_0, X_1, X_2)$ 对应于 $\tilde{f}$ 经过 C_3 编码后的 codeword。对于每一轮折叠，Verifier 都会发送一个随机挑战数 α_i 给 Prover，然后 Prover 会根据 $f^{(i+1)}$ 折叠产生 $f^{(i)}$ ，然后将其编码 codeword（的 Oracle）发送给 Verifier。

当 Basefold-IOPP 协议运行一次折叠后，协议双方得到的 codeword 所对应的多项式 $f^{(2)}(X_0, X_1)$ 为：

$$\begin{aligned} f^{(2)}(X_0, X_1) &= f^{(3)}(X_0, X_1, \alpha_2) \\ &= (f_0 + \alpha_2 \cdot f_4) + (f_1 + \alpha_2 \cdot f_5) \cdot X_0 + (f_2 + \alpha_2 \cdot f_6) \cdot X_1 + (f_3 + \alpha_2 \cdot f_7) \cdot X_0X_1 \end{aligned} \tag{3}$$

再一轮折叠后，codeword 所对应的多项式 $f^{(1)}(X_0)$ 为：

$$\begin{aligned} f^{(1)}(X_0) &= f^{(2)}(X_0, \alpha_1) \\ &= (f_0 + \alpha_2 \cdot f_4 + \alpha_1 \cdot f_2 + \alpha_2\alpha_1 \cdot f_6) + (f_1 + \alpha_2 \cdot f_5 + \alpha_1 \cdot f_3 + \alpha_2\alpha_1 \cdot f_7) \cdot X_0 \end{aligned} \tag{4}$$

最后，codeword 所对应的「常数多项式」 $f^{(0)}$ 如下：

$$\begin{aligned} f^{(0)} &= f^{(1)}(\alpha_0) \\ &= (f_0 + \alpha_2 \cdot f_4 + \alpha_1 \cdot f_2 + \alpha_2\alpha_1 \cdot f_6) + (f_1 + \alpha_2 \cdot f_5 + \alpha_1 \cdot f_3 + \alpha_2\alpha_1 \cdot f_7) \cdot \alpha_0 \\ &= f_0 + f_1\alpha_0 + f_2\alpha_1 + f_3\alpha_0\alpha_1 + f_4\alpha_2 + f_5\alpha_0\alpha_2 + f_6\alpha_1\alpha_2 + f_7\alpha_0\alpha_1\alpha_2 \end{aligned} \tag{5}$$

仔细观察下，$f^{(0)}$ 恰好是 $\tilde{f}(X_0, X_1, X_2)$ 在 $(\alpha_0, \alpha_1, \alpha_2)$ 处的求值。

于是， Prover 在 Basefold-IOPP 协议的最后一轮将折叠产生的常数多项式发送给 Verifier，然后 Verifier 通过 Query-phase 来证明一步步的折叠过程是正确无误的，与此同时，Verifier 还得到了 $\tilde{f}(\vec{X})$ 在 $\vec{X} = (\alpha_0, \alpha_1, \alpha_2)$ 处的求值结果 v 。换个角度来看，Basefold-IOPP 协议在完成 Proximity 证明的同时，还额外还提供了 $\tilde{f}(X_0, X_1, X_2)$ 在一个随机点的求值。或者严格点说，这是一个向量内积的证明：

$$\langle \mathbf{f}, (1, \alpha_0) \otimes (1, \alpha_1) \otimes (1, \alpha_2) \rangle = v \tag{6}$$

只不过，这个 MLE 求值点并不是提前协商好的公共输入，而是在 Basefold-IOPP 协议运行过程中产生的随机挑战数构成的求值点。

总结下，在 IOPP 协议的最后一轮，Prover 折叠产生了一个常数多项式，记为 $f^{(0)}$，它恰好是 $\tilde{f}$ 在随机点 $(\alpha_0, \alpha_1, \alpha_2)$ 处的求值，而我们需要的是证明 $\tilde{f}$ 在一个「公开点」的取值的正确性，比如说 (u_0, u_1, u_2) 点处的取值。

$$\tilde{f}(u_0, u_1, u_2) = f_0 + f_1 \cdot u_0 + f_2 \cdot u_1 + f_3 \cdot u_0u_1 + f_4 \cdot u_2 + f_5 \cdot u_0u_2 + f_6 \cdot u_1u_2 + f_7 \cdot u_0u_1u_2 \tag{7}$$

那么接下来的问题就转换成了：我们如何利用 $\tilde{f}$ 在一个「随机点」的求值，来证明 $\tilde{f}$ 在另一个「公开点」的求值的正确性?

借力 Sumcheck

我们可以利用 MLE 的性质来重新审视 $\tilde{f}(u_0, u_1, u_2)$ 。 根据 MLE 的定义，我们可以有下面的等式：

$$\mathsf{EQ}_1: \qquad \tilde{f}(X_0, X_1, X_2) = \sum_{\mathbf{b} \in \{0,1\}^3} \tilde{f}(b_0, b_1, b_2) \cdot \tilde{eq}_{(b_0, b_1, b_2)}(X_0, X_1, X_2) \tag{8}$$

这里的 $\tilde{eq}$ 是 MLE 的 Lagrange Polynomials，它的定义如下：

$$\tilde{eq}_{(b_0, b_1, \ldots, b_{n-1})}(X_0, X_1, \ldots, X_{n-1})) = \prod_{i=0}^{n-1} \big((1 - b_i)(1 - X_i) + b_i \cdot X_i\big) \tag{9}$$

容易验证，当 $(X_0, X_1, X_2) = \mathbf{b}', \mathbf{b}' \in \{0,1\}^3$ 时，上面 EQ_1 等式左边为 $\tilde{f}(\mathbf{b}')$，观察等式右边求和项中的每一个 $\tilde{eq}_{\mathbf{b}}(\mathbf{b}')$。只有当两个向量 $\mathbf{b} = \mathbf{b}'$ 完全相等时，这个 $\tilde{eq}_{\mathbf{b}}(\mathbf{b}')$ 为 1，其余的都为 0，因此等式的右边的求和项中只剩下了 $\tilde{f}(\mathbf{b}') \cdot \tilde{eq}_{\mathbf{b}}(\mathbf{b}') = \tilde{f}(\mathbf{b}')$，由此等式左右两边相等。

这意味着等式 EQ_1 的左右两边的两个 MLE 多项式在 3 维 Boolean HyperCube 上的取值都相同，根据 MLE 的唯一性性质，我们可以得知，EQ_1 对于任意的 $\mathbf{X} \in F^3$ 都成立。

虽然以上内容是 MLE 的基本常识，但请留意等式 EQ_1 带来了新的视角：我们可以把 $\tilde{f}(u_0, u_1, u_2)$ 的求值问题转化成一个 Sumcheck 问题：

$$\tilde{f}(u_0, u_1, u_2) = v = \sum_{\mathbf{b} \in \{0,1\}^3} \tilde{f}(\mathbf{b}) \cdot \tilde{eq}_{\mathbf{b}}(u_0, u_1, u_2) \tag{10}$$

于是立即可以想到，我们可以用 Sumcheck 协议对上面的等式进行「求和证明」。而 Sumcheck 协议的一个重要功能是把一个「求和问题」继续转化为两个 MLE 多项式（分别是 $\tilde{f}$ 与 $\tilde{eq}$）在一个随机点的求值问题。不过一般情况下，这么做是没有意义的，这是因为我们会在 Sumcheck 的最后一轮，然后需要证明 $\tilde{f}$ 在一个新的点上的取值，这会出现循环证明的情况：我们本来证明一个多项式在一个点处的取值，然后通过上面的 Sumcheck 协议，把这个求值证明转化为了多项式在另一个点处的取值，这其中的 Sumcheck 协议过程显得多余且无用。通常情况下，协议需要再依赖一个 MLE Evaluation Argument 协议，Prover 发送最后的求值以及求值的正确性证明，从而最终结束 Sumcheck 协议。但在这里，我们并不能再依赖另一个 MLE PCS 协议，因为我们的目标本是构造一个 MLE Evaluation Argument 协议，而不是求助于另一个已有的 MLE PCS。

不过这个问题并不难解决，因为我们前面介绍的 Basefold-IOPP 协议可以提供一个副产品：$\tilde{f}$ 在「随机点」上的求值证明，如果 Sumcheck 协议和 Basefold-IOPP 协议共用随机数，那么 Basefold-IOPP 最后提供的值恰好可以弥补 Sumcheck 协议最后一轮所需要的「随机点」求值的正确性证明。然后还有一个剩下的问题是 $\tilde{eq}$ 的求值证明如何解决？这个问题更简单，因为 $\tilde{eq}$ 是一个公开的多项式，因此 Verifier 可以自行计算，无需让 Prover 发送求值证明，并且 $\tilde{eq}$ 的求值计算仅有 $O(\log N)$ 的复杂度，并不会增加 Verifier 的负担，也不影响 Verifier 的简洁性（Succinctness）。这时，Sumcheck 协议的角色不再是多余，而是非常关键：**它把多项式在一个公开点的求值证明问题转化为了多项式在一个随机点的求值证明。**

而我们接下来需要做的是，把 Basefold-IOPP 协议和 Sumcheck 协议一起同步执行，然后让这两个协议共用一组随机挑战数，这样就能在两个协议的最后一步，完成协同，从而最终证明 $\tilde{f}(u_0, u_1, u_2) = v$ 的正确性。

按这个思路，我们接下来尝试走一下 $s = 3$ 时协议流程。

公共输入

1. $\tilde{f}$ 的 Commitment $\pi_3 = [\tilde{f}] = \mathsf{Enc}_3(\mathbf{f})$
2. 求值点 $\mathbf{u} = (u_0, u_1, u_2)$
3. 运算值 v

Witness

- MLE $\tilde{f}$ 的系数向量 $\mathbf{f} = (f_0, f_1, f_2, f_3, f_4, f_5, f_6, f_7)$
- $\tilde{f}$ 的求值向量 $\mathbf{a} = (a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7)$

第一轮：Prover 发送 Univariate 多项式 $h^{(2)}(X)$

$$h^{(2)}(X) = \sum_{b_0, b_1 \in \{0,1\}^2} f(b_0, b_1, X) \cdot \tilde{eq}\big((b_0, b_1, X), (u_0, u_1, u_2)\big) \tag{11}$$

由于等式右边是一个次数为 2 的 Univariate 多项式，因此 Prover 可以计算 $h^{(2)}(X)$ 的系数：

$$\begin{aligned} h_0^{(2)} &= \sum_{i=0}^{3} a_i \cdot e_i \\ h_1^{(2)} &= \sum_{i=0}^{3} a_{i+4} \cdot e_i + a_i \cdot e_{i+4} \\ h_2^{(2)} &= \sum_{i=0}^{3} a_{i+4} \cdot e_{i+4} \end{aligned} \tag{12}$$

这里 $\mathbf{e}$ 为 $\tilde{eq}(\mathbf{X}, \mathbf{u})$ 的 Evaluation向量。容易验证 $h^{(2)}(X) = h_0^{(2)} + h_1^{(2)} \cdot X + h_2^{(2)} \cdot X^2$。

第二轮：Verifier 发送挑战数 $\alpha_2 \leftarrow \mathbb{F}_p$

第三轮：Prover 与 Verifier同时进行 Basefold-IOPP 协议和 Sumcheck 协议：

- Prover 计算 $\tilde{f}^{(2)}(X_1, X_2) = f(\alpha_0, X_1, X_2)$

$$\tilde{f}^{(2)}(X_0, X_1) = (f_0 + f_4\alpha_2) + (f_1 + f_5\alpha_2) \cdot X_0 + (f_2 + f_6\alpha_2) \cdot X_1 + (f_3 + f_7\alpha_2) \cdot X_0X_1 \tag{13}$$

- Prover 发送折叠后的向量编码：$\pi_2 = \mathrm{Enc}_2[\tilde{f}^{(2)}]$
- Prover 计算 $h^{(2)}(\alpha_2)$ 作为下一轮 Sumcheck 协议的求和值
- Prover 计算并发送 $h^{(1)}(X)$

$$h^{(1)}(X) = \sum_{b_0 \in \{0,1\}} f(b_0, X, \alpha_2) \cdot \tilde{eq}((b_0, X, \alpha_2), (u_0, u_1, u_2)) \tag{14}$$

等式右边同样是一个关于 X 次数为 2 的 Univariate 多项式，因此 Prover 可以计算出 $h^{(1)}(X)$ 的系数： $(h_0^{(1)}, h_1^{(1)}, h_2^{(1)})$ 。

第四轮：Verifier 发送挑战数 $\alpha_1 \leftarrow \mathbb{F}_p$

第五轮：Prover 继续运行 Basefold-IOPP 协议和 Sumcheck 协议：

- Prover 计算 $\tilde{f}^{(1)}(X_2)$

$$\tilde{f}^{(1)}(X_2) = f(X_0, \alpha_1, \alpha_2) = (f_0 + f_4\alpha_2 + f_2\alpha_1 + f_6\alpha_1\alpha_2) + (f_1 + f_5\alpha_2 + f_3\alpha_1 + f_7\alpha_1\alpha_2) \cdot X_0 \tag{15}$$

- Prover 发送折叠后的向量编码 $\pi_1 = \mathrm{Enc}_1[\tilde{f}^{(1)}]$
- Prover 计算下一轮 Sumcheck 协议的求和值： $h^{(1)}(\alpha_1)$
- Prover 计算并发送 $h^{(0)}(X)$

$$h^{(0)}(X) = \sum_{b_0 \in \{0,1\}} f(X, \alpha_1, \alpha_2) \cdot \tilde{eq}((b_0, \alpha_1, \alpha_2), (u_0, u_1, u_2)) \tag{16}$$

假设 $h^{(0)}(X)$ 的系数表示为 $(h_0^{(0)}, h_1^{(0)}, h_2^{(0)})$

第六轮：Verifier 发送挑战数 $\alpha_0 \leftarrow \mathbb{F}_p$

第七轮：Prover 继续进行 Basefold-IOPP 协议：

- Prover 计算 $\tilde{f}^{(0)}$，这是一个 **常数多项式**

$$\tilde{f}^{(0)} = f(\alpha_0, \alpha_1, \alpha_2) = f_0 + f_1\alpha_0 + f_2\alpha_1 + f_3\alpha_0\alpha_1 + f_4\alpha_2 + f_5\alpha_0\alpha_2 + f_6\alpha_1\alpha_2 + f_7\alpha_0\alpha_1\alpha_2 \tag{17}$$

- Prover 发送折叠后的向量编码 $\pi_0 = \mathsf{Enc}_0[\tilde{f}^{(0)}]$

第八轮：Verifier 验证下面的等式：

1. Verifier 验证若干轮的 Basefold-Query, $Q = \{q_i\}$
2. Verifier 检验 Sumcheck 的每一步折叠的正确性：

$$\begin{aligned} h^{(2)}(0) + h^{(2)}(1) &\overset{?}{=} v \\ h^{(1)}(0) + h^{(1)}(1) &\overset{?}{=} h^{(2)}(\alpha_2) \\ h^{(0)}(0) + h^{(0)}(1) &\overset{?}{=} h^{(1)}(\alpha_1) \end{aligned} \tag{18}$$

3. Verifier 验证最后的编码 π_0 是否正确

$$\pi_0 \overset{?}{=} \mathsf{Enc}_0\left(\frac{h^{(0)}(\alpha_0)}{\tilde{eq}((\alpha_0, \alpha_1, \alpha_2), (u_0, u_1, u_2))}\right) \tag{19}$$

另一种折叠方式

Basefold 论文中的 PCS 协议是要求 MLE 多项式 $\tilde{f}(X_0, X_1, \ldots, X_{d-1})$ 事先被转换到 Coefficients Form，然后按照上一节的协议进行证明。不过，在很多基于 Sumcheck 协议的 SNARK 系统，Sumcheck 协议在最后一步产生的是 MLE 多项式的 Evaluation Form，即 MLE 多项式在 Boolean HyperCube 上的求值。因此，如果直接用 SNARK 协议对接 Basefold-PCS，我们还需要将 MLE 多项式转换到 Coefficients Form，这个转换算法为 $O(N\log(N))$ ，其中 $N = 2^d$ 。这里 d 为 MLE 多项式中未知数的个数。下面就是 $\tilde{f}(X_0, X_1, \ldots, X_{d-1})$ 的 Evaluation Form 的定义：

$$\tilde{f}(X_0, X_1, \ldots, X_{d-1}) = \sum_{\mathbf{b} \in \{0,1\}^d} a_{\mathbf{b}} \cdot eq_{\mathbf{b}}(X_0, X_1, \ldots, X_{d-1}) \tag{20}$$

正如 FRI-Binius 论文指出，其实我们并不需要将 MLE 多项式从 Evaluation Form 转换到 Coefficients Form，而是可以直接在 Evaluation Form 下完成 PCS 协议的构造。前面一节，之所以我们需要 $\tilde{f}(X_0, X_1, \ldots, X_{d-1})$ 的 Coefficients Form，是因为我们按照 Basefold-IOPP 协议的 Commit 方式，需要用到 $\tilde{f}(X_0, X_1, \ldots, X_{d-1})$ 的系数。而如果按照我们下面即将介绍的 Basefold-PCS 协议的折叠方式，我们就可以直接在 Evaluation Form 下完成 PCS 协议的构造。

现在，我们考虑下 $\tilde{f}(X_0, X_1, X_2)$ 的 Evaluation Form 的折叠方式。先展开 $\tilde{f}(X_0, X_1, X_2)$ 的 Evaluation Form 定义：

$$\begin{aligned} \tilde{f}^{(3)}(X_0, X_1, X_2) &= \tilde{f}(X_0, X_1, X_2) \\ &= a_0 \cdot eq_{(0,0,0)}(X_0, X_1, X_2) + a_1 \cdot eq_{(1,0,0)}(X_0, X_1, X_2) + a_2 \cdot eq_{(0,1,0)}(X_0, X_1, X_2) + a_3 \cdot eq_{(1,1,0)}(X_0, X_1, X_2) \\ &+ a_4 \cdot eq_{(0,0,1)}(X_0, X_1, X_2) + a_5 \cdot eq_{(1,0,1)}(X_0, X_1, X_2) + a_6 \cdot eq_{(0,1,1)}(X_0, X_1, X_2) + a_7 \cdot eq_{(1,1,1)}(X_0, X_1, X_2) \end{aligned} \tag{21}$$

其实我们可以对上面的 Evaluation Form 进行折叠，最终也可以得到 $\tilde{f}^{(3)}(\alpha_0, \alpha_1, \alpha_2)$ 的求值。比如，我们可以用 α_2 对上面的 Evaluation Form 进行折叠，得到：

$$\begin{aligned} \tilde{f}^{(2)}(X_0, X_1) &= \tilde{f}^{(3)}(X_0, X_1, \alpha_2) \\ &= a_0 \cdot eq_{(0,0,0)}(X_0, X_1, \alpha_2) + a_1 \cdot eq_{(1,0,0)}(X_0, X_1, \alpha_2) + a_2 \cdot eq_{(0,1,0)}(X_0, X_1, \alpha_2) + a_3 \cdot eq_{(1,1,0)}(X_0, X_1, \alpha_2) \\ &+ a_4 \cdot eq_{(0,0,1)}(X_0, X_1, \alpha_2) + a_5 \cdot eq_{(1,0,1)}(X_0, X_1, \alpha_2) + a_6 \cdot eq_{(0,1,1)}(X_0, X_1, \alpha_2) + a_7 \cdot eq_{(1,1,1)}(X_0, X_1, \alpha_2) \end{aligned} \tag{22}$$

这时，我们需要分解下 $eq_{\mathbf{b}}(X_0, X_1\alpha_2)$：

$$eq_{(b_0,b_1,b_2)}(X_0, X_1, \alpha_2) = eq_{(b_0,b_1)}(X_0, X_1) \cdot \Big((1-b_2)\cdot(1-\alpha_2) + b_2 \cdot \alpha_2\Big) \tag{23}$$

当 $b_2 = 0$ 时，上面等式右边化简为 $(1-\alpha_2)\cdot eq_{(b_0,b_1)}(X_0, X_1)$；当 $b_2 = 1$ 时，上面等式右边化简为 $\alpha_2 \cdot eq_{(b_0,b_1)}(X_0, X_1)$。然后继续化简 $\tilde{f}^{(2)}(X_0, X_1)$：

$$\begin{aligned}\tilde{f}^{(2)}(X_0, X_1) = &((1-\alpha_2)\cdot a_0 + \alpha_2 \cdot a_4)\cdot eq_{(0,0)}(X_0, X_1) + ((1-\alpha_2)\cdot a_1 + \alpha_2 \cdot a_5)\cdot eq_{(0,1)}(X_0, X_1)\\ &+ ((1-\alpha_2)\cdot a_2 + \alpha_2 \cdot a_6)\cdot eq_{(1,0)}(X_0, X_1) + ((1-\alpha_2)\cdot a_3 + \alpha_2 \cdot a_7)\cdot eq_{(1,1)}(X_0, X_1)\end{aligned} \tag{24}$$

这相当于我们用 $(1-\alpha_2, \alpha_2)$ 对向量 (a_0, a_1, a_2, a_3) 和 (a_4, a_5, a_6, a_7) 进行折叠（线性组合）。而相比较 Basefold-IOPP 协议中的折叠方式，它是用 $(1, \alpha_2)$ 对向量 (f_0, f_1, f_2, f_3) 和 (f_4, f_5, f_6, f_7) 进行折叠（线性组合）。如果继续按照新的方式折叠下去，我们最终可以得到和 Basefold-IOPP 协议一样的折叠结果。

$$\begin{aligned}\tilde{f}^{(1)}(X_0) &= \tilde{f}^{(2)}(X_0, \alpha_1)\\ &= ((1-\alpha_2)\cdot a_0 + \alpha_2 \cdot a_4)\cdot eq_{(0,0)}(X_0, \alpha_1) + ((1-\alpha_2)\cdot a_1 + \alpha_2 \cdot a_5)\cdot eq_{(1,0)}(X_0, \alpha_1)\\ &+ ((1-\alpha_2)\cdot a_2 + \alpha_2 \cdot a_6)\cdot eq_{(0,1)}(X_0, \alpha_1) + ((1-\alpha_2)\cdot a_3 + \alpha_2 \cdot a_7)\cdot eq_{(1,1)}(X_0, \alpha_1)\\ &= ((1-\alpha_2)\cdot a_0 + \alpha_2 \cdot a_4)\cdot (1-\alpha_1)\cdot eq_{(0)}(X_0) + ((1-\alpha_2)\cdot a_1 + \alpha_2 \cdot a_5)\cdot (1-\alpha_1)\cdot eq_{(1)}(X_0)\\ &+ ((1-\alpha_2)\cdot a_2 + \alpha_2 \cdot a_6)\cdot \alpha_1 \cdot eq_{(0)}(X_0) + ((1-\alpha_2)\cdot a_3 + \alpha_2 \cdot a_7)\cdot \alpha_1 \cdot eq_{(1)}(X_0)\\ &= \Big((1-\alpha_1)\cdot((1-\alpha_2)\cdot a_0 + \alpha_2 \cdot a_4) + \alpha_1 \cdot ((1-\alpha_2)\cdot a_2 + \alpha_2 \cdot a_6)\Big)\cdot eq_{(0)}(X_0)\\ &+ \Big((1-\alpha_1)\cdot((1-\alpha_2)\cdot a_1 + \alpha_2 \cdot a_5) + \alpha_1 \cdot ((1-\alpha_2)\cdot a_3 + \alpha_2 \cdot a_7)\Big)\cdot eq_{(1)}(X_0)\end{aligned} \tag{25}$$

继续折叠下去，最终我们可以得到：

$$\begin{aligned}\tilde{f}^{(0)} &= \tilde{f}^{(1)}(\alpha_0)\\ &= \Big((1-\alpha_0)\cdot((1-\alpha_1)\cdot((1-\alpha_2)\cdot a_0 + \alpha_2 \cdot a_4) + \alpha_1 \cdot ((1-\alpha_2)\cdot a_2 + \alpha_2 \cdot a_6))\\ &\quad + \alpha_0 \cdot ((1-\alpha_1)\cdot((1-\alpha_2)\cdot a_1 + \alpha_2 \cdot a_5) + \alpha_1 \cdot ((1-\alpha_2)\cdot a_3 + \alpha_2 \cdot a_7))\Big)\\ &= \tilde{f}^{(3)}(\alpha_0, \alpha_1, \alpha_2)\end{aligned} \tag{26}$$

我们能否改进 Basefold-IOPP 协议的折叠方式，使之直接可以折叠 Evaluation Form 的 MLE 多项式?

基于 Evaluation Form 的 Basefold-IOPP 协议

我们接下来尝试写一下 $d = 3$ 的情况，即 $\tilde{f}(X_0, X_1, X_2)$ 的 Evaluation Form 的 Basefold-IOPP 协议。首先，我们改写下前一篇文章中的折叠函数

$$\mathsf{fold}^*_\alpha(y_0, y_1) = (1-\alpha)\cdot\frac{t_j \cdot y_1 - t'_j \cdot y_0}{t_j - t'_j} + \alpha \cdot \frac{y_0 - y_1}{t_j - t'_j} \tag{27}$$

通过前文，我们知道折叠函数具有这样的同态性质：折叠之后的 codeword 等价于原始消息经过折叠后再编码产生的 codeword，用公式表示如下，

$$\mathsf{fold}_\alpha(\mathsf{Enc}_i(\mathbf{m})) = \mathsf{Enc}_i(\mathsf{fold}_\alpha(\mathbf{m})) \tag{28}$$

同样我们可以证明新的折叠函数一样满足上面的性质。我们可以尝试用新的折叠函数来折叠 codeword π_i 中的一个元素：

$$\begin{aligned}\mathsf{fold}^*_\alpha(\pi_i[j], \pi_i[n_{i-1}+j]) &= \frac{1-\alpha}{t_j - t'_j}\cdot\Big(t_j \cdot (\mathbf{m}_l G_{i-1}[j] + t'_j \cdot \mathbf{m}_r G_{i-1}[j]) - t'_j \cdot (\mathbf{m}_l G_{i-1}[j] + t_j \cdot \mathbf{m}_r G_{i-1}[j])\Big)\\ &+ \frac{\alpha}{t_j - t'_j}\cdot\Big(\mathbf{m}_l G_{i-1}[j] + t_j \cdot \mathbf{m}_r G_{i-1}[j] - \mathbf{m}_l G_{i-1}[j] - t'_j \cdot \mathbf{m}_r G_{i-1}[j]\Big)\\ &= (1-\alpha)\cdot \mathbf{m}_l G_{i-1}[j] + \alpha \cdot \mathbf{m}_r G_{i-1}[j]\end{aligned} \tag{29}$$

上面的推导已经说明，新的折叠函数一样满足关于编码函数的同态性质：

$$\mathsf{fold}^*_\alpha(\mathsf{Enc}_i(\mathbf{m})) = \mathsf{Enc}_i(\mathsf{fold}^*_\alpha(\mathbf{m})) \tag{30}$$

并且，我们同样可以证明新的折叠过程不影响 Basefold-IOPP 协议的可靠性，即折叠后 codeword 的 Minimum Hamming Weight 仍然大于一个下界。

至此，我们得到一个重要的结论，不管采用 fold_α 还是 fold^*_α，它们都可以用来构造 Proximity-Proof 协议，并且不存在什么显著的区别。

下面我们走下 Commit-phase 协议的流程，方便大家理解：

公共输入

- MLE 多项式 $\tilde{f}$ 的 codeword， $\pi_3 = \mathsf{Enc}_3(\mathbf{a}) = \mathbf{a}G_3$

Witness

- MLE 多项式 $\tilde{f}$ 的取值向量 $\mathbf{a} = (a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7)$

第一轮：Verifier 发送随机数 α_2

第二轮：Prover 计算 $\pi_2 = \mathsf{fold}^*_\alpha(\pi_3)$，然后发送给 Verifier

计算 π_2 的过程如下：

$$\pi_2[j] = \mathsf{fold}^*_\alpha(\pi_3[j], \pi_3[j+4]), \quad j \in \{0,1,2,3\} \tag{31}$$

计算出的 $\pi_2 = \mathsf{Enc}_2(f^{(2)})$，即 π_2 是 $f^{(2)}$ 的 codeword，而 $f^{(2)}$ 是 f 的关于 α_2 的折叠结果：

$$\begin{aligned} f^{(2)}(X_0, X_1) &= ((1-\alpha_2)a_0 + \alpha_2 a_4) \cdot eq_{(0,0)}(X_0, X_1) + ((1-\alpha_2)a_1 + \alpha_2 a_5) \cdot eq_{(1,0)}(X_0, X_1) \\ &+ ((1-\alpha_2)a_2 + \alpha_2 a_6) \cdot eq_{(0,1)}(X_0, X_1) + ((1-\alpha_2)a_3 + \alpha_2 a_7) \cdot eq_{(1,1)}(X_0, X_1) \\ &= f(X_0, X_1, \alpha_2) \end{aligned} \tag{32}$$

第三轮：Verifier 发送随机数 α_1

第四轮：Prover 计算 $\pi_1 = \mathsf{fold}^*_\alpha(\pi_2)$，然后发送给 Verifier

计算 π_1 的过程如下：

$$\pi_1[j] = \mathsf{fold}^*_\alpha(\pi_2[j], \pi_2[j+2]), \quad j \in \{0,1\} \tag{33}$$

计算出的 $\pi_1 = \mathsf{Enc}_1(f^{(1)})$，这里 $f^{(1)}$ 是 $f^{(2)}$ 的关于 α_1 的折叠结果：

$$\begin{aligned} f^{(1)}(X_0) &= \Big((1-\alpha_1)((1-\alpha_2)a_0 + \alpha_2 a_4) + \alpha_1((1-\alpha_2)a_1 + \alpha_2 a_5)\Big) \cdot eq_{(0)}(X_0) \\ &+ \Big((1-\alpha_1)((1-\alpha_2)a_2 + \alpha_2 a_6) + \alpha_1((1-\alpha_2)a_3 + \alpha_2 a_7)\Big) \cdot eq_{(1)}(X_0) \\ &= f(X_0, \alpha_1, \alpha_2) \end{aligned} \tag{34}$$

第五轮：Verifier 发送随机数 α_0

第六轮：Prover 计算 $\pi_0 = \mathsf{fold}_\alpha(\pi_1)$，然后发送给 Verifier

计算 π_0 的过程如下：

$$\pi_0[j] = \mathsf{fold}^*_\alpha(\pi_1[j], \pi_1[j+1]), \quad j = 0 \tag{35}$$

同样 $\pi_0 = \mathsf{Enc}_0(f^{(0)})$，即 π_0 是 $f^{(0)}$ 的 codeword，而 $f^{(0)}$ 是 f 的关于 $(\alpha_0, \alpha_1, \alpha_2)$ 的折叠结果：

$$f^{(0)} = f(\alpha_0, \alpha_1, \alpha_2) \tag{36}$$

然后我们就可以改进下 Evaluation Argument 协议了。

Evaluation Form 的 Basefold Evaluation Argument 协议

公共输入

1. $\tilde{f}$ 的 Commitment $\pi_3 = \mathsf{Enc}_3(\mathbf{a})$
2. 求值点 $\mathbf{u}$
3. 运算值 $v = \tilde{f}(\mathbf{u})$

Witness

- MLE $\tilde{f}$ 的 Evaluation Form 向量 $\mathbf{a} = (a_0, a_1, \ldots, a_7)$

满足

$$\tilde{f}(X_0, X_1, X_2) = \sum_{\mathbf{b} \in \{0,1\}^3} a_{\mathbf{b}} \cdot eq_{\mathbf{b}}(X_0, X_1, X_2) \tag{37}$$

第一轮：Prover 发送 $h^{(2)}(X)$ 的取值，$(h^{(2)}(0), h^{(2)}(1), h^{(2)}(2))$

$$h^{(2)}(X) = \sum_{b_1, b_2 \in \{0,1\}^2} f(X, b_1, b_2) \cdot \tilde{eq}((X, b_1, b_2), \mathbf{u}) \tag{38}$$

由于等式右边是一个次数为 2 的 Univariate Polynomial，因此 Prover 可以计算 $h^{(2)}(X)$ 在 $X = 0, 1, 2$ 三个点的取值：

$$\begin{aligned} h^{(2)}(0) &= a_0 \cdot e_0 + a_1 \cdot e_1 + a_2 \cdot e_2 + a_3 \cdot e_3 \\ h^{(2)}(1) &= a_4 \cdot e_4 + a_5 \cdot e_5 + a_6 \cdot e_6 + a_7 \cdot e_7 \\ h^{(2)}(2) &= \sum_{i=0}^{3} (2 \cdot a_{i+4} - a_i) \cdot (2 \cdot e_{i+4} - e_i) \end{aligned} \tag{39}$$

这里 $\mathbf{e}$ 为 $\tilde{eq}(\mathbf{X}, \mathbf{u})$ MLE 多项式在 Boolean HyperCube 上的取值向量。

第二轮：Verifier 发送挑战数 $\alpha_2 \leftarrow \mathbb{F}_p$

第三轮：Prover 同时进行 Basefold-IOPP 协议和 Sumcheck 协议：

- Prover 发送折叠后的向量编码： $\pi_2 = \mathsf{fold}^*_{\alpha_2}(\pi_3)$
- Prover 计算 $h^{(2)}(\alpha_2)$ 作为下一轮 Sumcheck 协议的求和值
- Prover 计算 $f^{(2)}(X_0, X_1)$ 的 Evaluations 为 $\mathbf{a}^{(2)} = \mathsf{fold}^*_{\alpha_2}(\mathbf{a})$
- Prover 计算并发送 $h^{(1)}(X)$

$$h^{(1)}(X) = \sum_{b_0 \in \{0,1\}} f(b_0, X, \alpha_2) \cdot \tilde{eq}((b_0, X, \alpha_2), (u_2, u_1, u_0)) \tag{40}$$

等式右边同样是一个关于 X 次数为 2 的 Univariate Polynomial，因此 Prover 可以根据 $\mathbf{a}^{(2)}$ 计算出 $h^{(1)}(X)$ 在 $X = 0, 1, 2$ 处的取值：$(h^{(1)}(0), h^{(1)}(1), h^{(1)}(2))$ 。

第四轮：Verifier 发送挑战数 $\alpha_1 \leftarrow \mathbb{F}_p$

第五轮：Prover 继续进行 Basefold-IOPP 协议和 Sumcheck 协议：

- Prover 发送折叠后的向量编码 $\pi_1 = \mathsf{fold}^*_{\alpha_1}(\pi_2)$
- Prover 计算下一轮 Sumcheck 协议的求和值： $h^{(1)}(\alpha_1)$
- Prover 计算 $\tilde{f}^{(1)}(X_0)$ 的 Evaluations 向量： $\mathbf{a}^{(1)} = \mathsf{fold}^*_{\alpha_1}(\mathbf{a}^{(2)})$
- Prover 计算并发送 $h^{(0)}(X)$ 在 $X = 0, 1, 2$ 处取值：$(h^{(0)}(0), h^{(0)}(1), h^{(0)}(2))$

$$h^{(0)}(X) = f(X_0, \alpha_1, \alpha_2) \cdot \tilde{eq}((X_0, \alpha_1, \alpha_2), (u_0, u_1, u_2)) \tag{41}$$

第六轮：Verifier 发送挑战数 $\alpha_0 \leftarrow F$

第七轮：Prover 继续进行 Basefold-IOPP 协议：

- Prover 发送折叠后的向量编码 $\pi_0 = \mathsf{fold}^*_{\alpha_0}(\pi_1)$

第八轮：Verifier 验证下面的等式：

1. Verifier 发送若干轮的 Query, $Q = \{q_i\}$
2. Verifier 检验 Sumcheck 的每一步折叠的正确性：

$$\begin{aligned} h^{(2)}(0) + h^{(2)}(1) &\overset{?}{=} v \\ h^{(1)}(0) + h^{(1)}(1) &\overset{?}{=} h^{(2)}(\alpha_2) \\ h^{(0)}(0) + h^{(0)}(1) &\overset{?}{=} h^{(1)}(\alpha_1) \end{aligned} \tag{42}$$

3. Verifier 验证最后的编码 π_0 是否正确

$$\pi_0 \overset{?}{=} \mathsf{enc}_0 \left(\frac{h^{(0)}(\alpha_0)}{\tilde{eq}((\alpha_0, \alpha_1, \alpha_2), \mathbf{u})} \right) \tag{43}$$

到此为止，我们得到了一个基于 Evaluation Form 的 Basefold Evaluation Argument 协议。可以直接对接 Sumcheck-based zkSNARKs，或者类似 Jolt 风格的 zkVMs。

References

- [ZCF23] Hadas Zeilberger, Binyi Chen, and Ben Fisch. "BaseFold: efficient field-agnostic polynomial commitment schemes from foldable codes." Annual International Cryptology Conference. Cham: Springer Nature Switzerland, 2024.
- 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.