WHIR: Reed-Solomon Proximity Testing with Super-Fast Verification

- Jade Xie jade@secbit.io
- Yu Guo yu.guo@secbit.io

本篇文章主要介绍 WHIR (Weights Help Improving Rate) 协议[ACFY24b]。和 FRI[BBHR18]、STIR [ACFY24a] 以及 BaseFold [ZCF24] 协议一样，WHIR 也是一个 IOPP 协议，但其有较小的查询复杂度以及比较快的验证时间。论文 [ACFY24b] 中提到 WHIR 的 verifier 通常运行几百微秒(1 微秒 = 10^{-6} 秒)，而其他协议的 verifier 则需要几毫秒 (1 毫秒 = 10^{-3} 秒)。另外 WHIR 是针对 *constrained Reed-Solomon codes* (CRS) 的 IOPP 协议，CRS 使得 WHIR 同时支持对多元线性多项式和对单变量多项式的查询，这也是为什么 WHIR 可以同时和 BaseFold、FRI 以及 STIR 进行比较[ACFY24b]。整体上来看，WHIR 结合了 BaseFold 和 STIR 的思想，使得 WHIR 协议能在不牺牲 Prover 效率以及 argument 大小的情况下，支持多元线性多项式，同时也有较小的查询复杂度。

从一元多项式到多元线性多项式

对于一个有限域 $\mathbb{F}$ ，evaluation domain $\mathcal{L} \subseteq \mathbb{F}$, 次数为 $d \in \mathbb{N}$ 的 Reed-Solomon 编码，其表示的是在 $\mathbb{F}$ 上所有的次数严格小于 d 的一元多项式在 $\mathcal{L}$ 上的求值的集合，记为 $\mathrm{RS}[\mathbb{F}, \mathcal{L}, d]$ 。假设 $\mathcal{L}$ 是 $\mathbb{F}^*$ 的一个乘法陪集，并且其阶为 2 的幂次 (称 $\mathcal{L}$ 是 "smooth" 的)，同时假设次数 $d = 2^m$ 也是 2 的幂次的形式，那么我们就可以将一元多项式看作是有 m 个变量的多元线性多项式。(来自 [ACFY24b, 1.1 Constrained Reed-Solomon codes])

先举一个 $d = 2^3$ 的简单例子，设

$$f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7 \tag{1}$$

令 $X_1 = x, X_2 = x^2, X_3 = x^4$ ，则 $f(x)$ 可以表示为：

$$\begin{aligned} f(x) &= a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7 \\ &= a_0 + a_1 X_1 + a_2 X_2 + a_3 X_1 X_2 + a_4 X_3 + a_5 X_1 X_2 + a_6 X_1 X_3 + a_7 X_1 X_2 X_3 \end{aligned} \tag{2}$$

记新的多元线性多项式为

$$\hat{f}(X_1, X_2, X_3) = a_0 + a_1 X_1 + a_2 X_2 + a_3 X_1 X_2 + a_4 X_3 + a_5 X_1 X_2 + a_6 X_1 X_3 + a_7 X_1 X_2 X_3 \tag{3}$$

这样 $f(x)$ 可以看作是一元多项式，也可以通过变量替换 $X_1 = x, X_2 = x^2, X_3 = x^4$ 之后看作是多元线性多项式。

对于 RS code $\mathrm{RS}[\mathbb{F}, \mathcal{L}, d]$ 中的一元多项式也是类似的，可以以多元线性多项式的视角来看待，即

$$\begin{aligned} \mathrm{RS}[\mathbb{F}, \mathcal{L}, d] &:= \{f : \mathcal{L} \to \mathbb{F} : \exists \hat{g} \in \mathbb{F}^{<2^m}[X] \text{ s.t. } \forall x \in \mathcal{L}, f(x) = \hat{g}(x)\} \\ &= \{f : \mathcal{L} \to \mathbb{F} : \exists \hat{f} \in \mathbb{F}^{<2}[X_1, \ldots, X_m] \text{ s.t. } \forall x \in \mathcal{L}, f(x) = \hat{f}(x^{2^0}, x^{2^1}, \ldots, x^{2^{m-1}})\} \end{aligned} \tag{4}$$

上式中的 $\hat{g}(x)$ 就是一元多项式，而 $\hat{f}(X_1, \ldots, X_m)$ 就是有 m 个变量的多元线性多项式。这里用到的思想就出现在 BaseFold 中。(来自 [ACFY24b, 1.1 Constrained Reed-Solomon codes])

另外，和在 FRI 协议中一致，用随机数 α_1 对一元多项式进行折叠，可以等价看作是在对多元线性多项式中的一个变量代入 α_1 。

例如对上述的 $f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7$ 先用 α_1 进行折叠，则

$$\begin{aligned} f(x) &= a_0 + a_2 x^2 + a_4 x^4 + a_6 x^6 + x(a_1 + a_3 x^2 + a_5 x^4 + a_7 x^6) \\ &:= f_1(x^2) + x f_2(x^2) \end{aligned} \tag{5}$$

折叠之后的多项式为

$$
\begin{aligned}
f^{(1)}(x) &= f_1(x) + \alpha_1 f_2(x) \\
&= a_0 + a_2 x + a_4 x^2 + a_6 x^3 + \alpha_1(a_1 + a_3 x + a_5 x^2 + a_7 x^3) \\
&= a_0 + a_2 X_1 + a_4 X_2 + a_6 X_1 X_2 + \alpha_1(a_1 + a_3 X_1 + a_5 X_2 + a_7 X_1 X_2)
\end{aligned} \tag{6}
$$

其等价于对原来的多元多项式 $\hat{f}(X_1, X_2, X_3)$ 直接进行代值和变量替换，具体过程是：

1. 先将 X_1 代为 α_1 ，可得到

$$
\begin{aligned}
\hat{f}(\alpha_1, X_2, X_3) &= a_0 + a_1 \cdot \alpha_1 + a_2 X_2 + a_3 \cdot \alpha_1 X_2 + a_4 X_3 + a_5 \cdot \alpha_1 X_2 + a_6 X_2 X_3 + a_7 \cdot \alpha_1 X_2 X_3 \\
&= a_0 + a_2 X_2 + a_4 X_3 + a_6 X_2 X_3 + \alpha_1(a_1 + a_3 X_2 + a_5 X_2 + a_7 X_2 X_3)
\end{aligned} \tag{7}
$$

2. 令新的变量 $X_1 = X_2$ ，以及 $X_2 = X_3$ 可得到折叠后的多项式为

$$
\begin{aligned}
\hat{f}^{(1)}(X_1, X_2) &= a_0 + a_2 X_1 + a_4 X_2 + a_6 X_1 X_2 + \alpha_1(a_1 + a_3 X_1 + a_5 X_2 + a_7 X_1 X_2) \\
&= f^{(1)}(x)
\end{aligned} \tag{8}
$$

可以看出两种折叠方式得到的多项式是等价的，只是 $f^{(1)}(x)$ 是一元多项式的形式，而 $\hat{f}^{(1)}(X_1, X_2)$ 是多元线性多项式的形式。

如果要对原来的多项式 $f(x)$ 进行 4 折，从一元多项式的角度来看，可以对 2 折 之后的多项式 $f^{(1)}(x)$ 再进行 2 折，即

$$
\begin{aligned}
f^{(1)}(x) &= a_0 + a_2 x + a_4 x^2 + a_6 x^3 + \alpha_1(a_1 + a_3 x + a_5 x^2 + a_7 x^3) \\
&= (a_0 + \alpha_1 a_1) + (a_2 + \alpha_1 a_3) \cdot x + (a_4 + \alpha_1 a_5) \cdot x^2 + (a_6 + \alpha_1 a_7) \cdot x^3 \\
&= ((a_0 + \alpha_1 a_1) + (a_4 + \alpha_1 a_5) \cdot x^2) + x \cdot ((a_2 + \alpha_1 a_3) + (a_6 + \alpha_1 a_7) \cdot x^2) \\
&:= f_1^{(1)}(x^2) + x f_2^{(1)}(x^2)
\end{aligned} \tag{9}
$$

用随机数 α_2 进行折叠，得到折叠后的多项式为

$$
\begin{aligned}
f^{(2)}(x) &= f_1^{(1)}(x) + \alpha_2 f_2^{(1)}(x) \\
&= ((a_0 + \alpha_1 a_1) + (a_4 + \alpha_1 a_5)x) + \alpha_2((a_2 + \alpha_1 a_3) + (a_6 + \alpha_1 a_7)x) \\
&= ((a_0 + \alpha_1 a_1) + (a_4 + \alpha_1 a_5)X_1) + \alpha_2((a_2 + \alpha_1 a_3) + (a_6 + \alpha_1 a_7)X_1)
\end{aligned} \tag{10}
$$

从多元线性多项式的角度来看，可以对 2 折之后的多元线性多项式 $\hat{f}^{(1)}(X_1, X_2)$ 再进行 2 折，即

1. 将 X_1 代为 α_2 ,可得到

$$
\begin{aligned}
\hat{f}^{(1)}(\alpha_2, X_2) &= a_0 + a_2 \alpha_2 + a_4 X_2 + a_6 \alpha_2 X_2 + \alpha_1(a_1 + a_3 \alpha_2 + a_5 X_2 + a_7 \alpha_2 X_2) \\
&= ((a_0 + \alpha_1 a_1) + (a_4 + \alpha_1 a_5)X_2) + \alpha_2((a_2 + \alpha_1 a_3) + (a_6 + \alpha_1 a_7)X_2)
\end{aligned} \tag{11}
$$

2. 令新的变量 $X_1 = X_2$ ，得到折叠后的多项式为

$$
\hat{f}^{(2)}(X_1) = ((a_0 + \alpha_1 a_1) + (a_4 + \alpha_1 a_5)X_1) + \alpha_2((a_2 + \alpha_1 a_3) + (a_6 + \alpha_1 a_7)X_1) \tag{12}
$$

可以发现对于多折，用一元多项式进行折叠和直接用多元线性多项式进行折叠是等价的。用随机数 (α_1, α_2) 对多元线性多项式进行折叠的过程，就是直接进行变量替换的过程，即得到 $\hat{f}^{(2)}(X_1) = \hat{f}(\alpha_1, \alpha_2, X_1)$ 。

下面引入论文[ACFY24b]给出的折叠函数的定义，其折叠方式与 FRI 协议中的折叠方式是一致的。

定义 1 [ACFY24b, Definition 4.14] 令 $f : \mathcal{L} \to \mathbb{F}$ 为一个函数，$\alpha \in \mathbb{F}$ 。定义 $\mathrm{Fold}(f, \alpha) : \mathcal{L}^2 \to \mathbb{F}$ 如下：

$$
\forall x \in \mathcal{L}^2,\ \mathrm{Fold}(f, \alpha)(x^2) = \frac{f(x) + f(-x)}{2} + \alpha \cdot \frac{f(x) - f(-x)}{2 \cdot x} \tag{13}
$$

为了计算 $\mathrm{Fold}(f, \alpha)(x^2)$ ，只需要查询 f 在 x 和 $-x$ 上的值就足够了。

对于 $k \le m$ 以及 $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_k) \in \mathbb{F}^k$ 时，定义 $\mathrm{Fold}(f, \boldsymbol{\alpha}) : \mathcal{L}^{(2^k)} \to \mathbb{F}$ ，记 $\mathrm{Fold}(f, \boldsymbol{\alpha}) := f_k$ ，递归定义：$f_0 := f$ 以及 $f_i := \mathrm{Fold}(f_{i-1}, \alpha_i)$ 。

下面的命题告诉我们，对一个 Reed-Solomon code 在任意点的集合进行折叠，其结果依然是一个 Reed-Solomon code。([ACFY24b])

命题 1 [ACFY24b, Claim 4.15] 令 $f : \mathcal{L} \to \mathbb{F}$ 为一个函数，$\boldsymbol{\alpha} \in \mathbb{F}^k$ 表示折叠随机数，令 $g := \mathrm{Fold}(f, \boldsymbol{\alpha})$ 。如果 $f \in \mathrm{RS}[\mathbb{F}, \mathcal{L}, m]$ 并且 $k \leq m$ ，那么 $g \in \mathrm{RS}[\mathbb{F}, \mathcal{L}^{(2^k)}, m - k]$ ，另外 g 的 multilinear extension 为 $\hat{g}(X_k, \ldots, X_m) := \hat{f}(\boldsymbol{\alpha}, X_k, \ldots, X_m)$ ，其中 $\hat{f}$ 是 f 的 multilinear extension。

命题中给出的 $\hat{g}(X_k, \ldots, X_m) := \hat{f}(\boldsymbol{\alpha}, X_k, \ldots, X_m)$ ，与上述提到的对一元多项式 f 的折叠与用随机数直接对多元线性多项式 $\hat{f}$ 进行折叠是一致的，从多元线性多项式的角度来看，就是直接用随机数 $\boldsymbol{\alpha}$ 进行变量替换，即 $\hat{f}(\boldsymbol{\alpha}, X_k, \ldots, X_m)$ 。

回顾 FRI 协议，就是不断用随机数 $(\alpha_1, \ldots, \alpha_m)$ 对一元多项式 f 进行折叠，直到最后得到一个常数多项式，如果用多元线性多项式的角度来看，最后就会得到 $\hat{f}(\alpha_1, \ldots, \alpha_m)$ 为一个常数。联系 Sumcheck 协议，最后一步也需要得到一个多元多项式在某个随机点的值，verifier 需要得到这个值来进行验证，这一步通常是用 oracle 来实现的，现在 FRI 协议最后也能提供 $\hat{f}(\alpha_1, \ldots, \alpha_m)$ 在一个随机点的值，如果 Sumcheck 协议和 FRI 协议选取同样的随机点 $(\alpha_1, \ldots, \alpha_m)$ ，那么 FRI 协议进行到最后就可以直接提供 Sumcheck 协议最后一步所需要的值。将 FRI 协议和 Sumcheck 协议这样结合起来，就是 BaseFold 协议 [ZCF24] 的思想。

CRS: Constrained Reed-Solomon codes

下面给出 WHIR论文 [ACFY24b] 给出的 constrained Reed-Solomon codes 的定义，它是 Reed-Solomon codes 的一个子集，但是新增了一个类似 Sumcheck 的约束。

定义 2 [ACFY24b, Definition 1] 对于域为 $\mathbb{F}$ ，smooth evaluation domain 为 $\mathcal{L} \subseteq \mathbb{F}$ ，变量的数量为 $m \in \mathbb{N}$ ，权重多项式 $\hat{w} \in \mathbb{F}[Z, X_1, \ldots, X_m]$ ，以及目标 $\sigma \in \mathbb{F}$ 的 **constrained Reed-Solomon code** ，定义为

$$\mathrm{CRS}[\mathbb{F}, \mathcal{L}, m, \hat{w}, \sigma] := \left\{ f \in \mathrm{RS}[\mathbb{F}, \mathcal{L}, m] : \sum_{\mathbf{b} \in \{0,1\}^m} \hat{w}(\hat{f}(\mathbf{b}), \mathbf{b}) = \sigma \right\}. \tag{14}$$

从定义可以看出，CRS(constrained Reed-Solomon code) 首先是 RS code，即定义中的 $f \in \mathrm{RS}[\mathbb{F}, \mathcal{L}, m]$ ，但在此之上需要满足一个类似 Sumcheck 的求和约束 $\sum_{\mathbf{b} \in \{0,1\}^m} \hat{w}(\hat{f}(\mathbf{b}), \mathbf{b}) = \sigma$ 。

[ACFY24b] 论文中提到定义中的权重多项式 $\hat{w}$ 是可以自己定义的，用途是比较广泛的。在论文中举了这样一个例子，例如一个求值约束 $\hat{f}(\mathbf{z}) = \sigma$ ，即约束多元多项式 $\hat{f}$ 在点 $\mathbf{z} \in \mathbb{F}^m$ 的值为目标值 σ 。首先对 $f \in \mathrm{RS}[\mathbb{F}, \mathcal{L}, m]$ 进行 multilinear extension，可以得到

$$\hat{f}(\mathbf{X}) = \sum_{\mathbf{b} \in \{0,1\}^m} f(\mathbf{b}) \cdot \mathrm{eq}(\mathbf{b}, \mathbf{X}) \tag{15}$$

其中 $\mathrm{eq}(\mathbf{b}, \mathbf{X}) = \prod_{i=1}^m (b_i X_i + (1 - b_i) \cdot (1 - X_i))$ 。因此当 $\mathbf{b}, \mathbf{X} \in \{0,1\}^m$ 时，如果 $\mathbf{b} = \mathbf{X}$ ，则 $\mathrm{eq}(\mathbf{b}, \mathbf{X}) = 1$ ，如果 $\mathbf{b} \neq \mathbf{X}$ ，则 $\mathrm{eq}(\mathbf{b}, \mathbf{X}) = 0$ 。因此

$$\hat{f}(\mathbf{z}) = \sum_{\mathbf{b} \in \{0,1\}^m} f(\mathbf{b}) \cdot \mathrm{eq}(\mathbf{b}, \mathbf{z}) = \sum_{\mathbf{b} \in \{0,1\}^m} \hat{w}(\hat{f}(\mathbf{b}), \mathbf{b}) \tag{16}$$

权重多项式 $\hat{w}(Z, \mathbf{X})$ 就可以定义为

$$\hat{w}(Z, \mathbf{X}) = Z \cdot \mathrm{eq}(\mathbf{X}, \mathbf{z}). \tag{17}$$

这样就可以用权重多项式来表示一个求值约束了。可以基于此来构造对应的 PCS(来自[ACFY24b, 1.1 Hash-based PCS from CRS codes])，分两种情况：

1. 约束多元线性多项式 $\hat{f}$ 在 $\mathbf{z} \in \mathbb{F}^m$ 的值为 σ ，令权重多项式为

$$\hat{w}(Z, \mathbf{X}) = Z \cdot \mathrm{eq}(\mathbf{X}, \mathbf{z}). \tag{18}$$

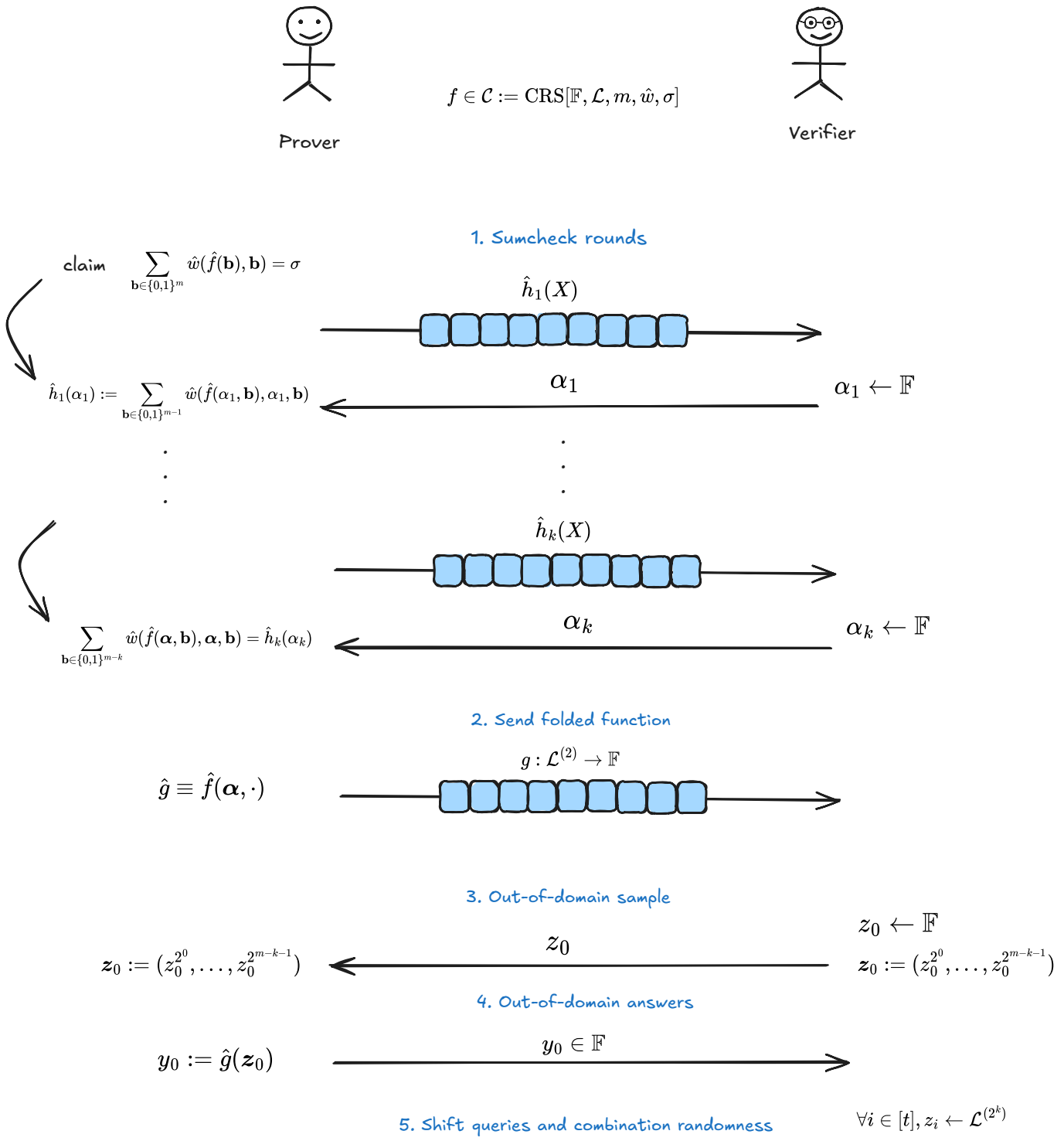
2. 约束一个单变量多项式 f 在 $z \in \mathbb{F}$ 的处值为 σ，将这种情况转换多元线性多项式的情况，考虑求值点为 $\mathbf{z} = (z^{2^0}, \ldots, z^{2^{m-1}})$，则权重多项式为

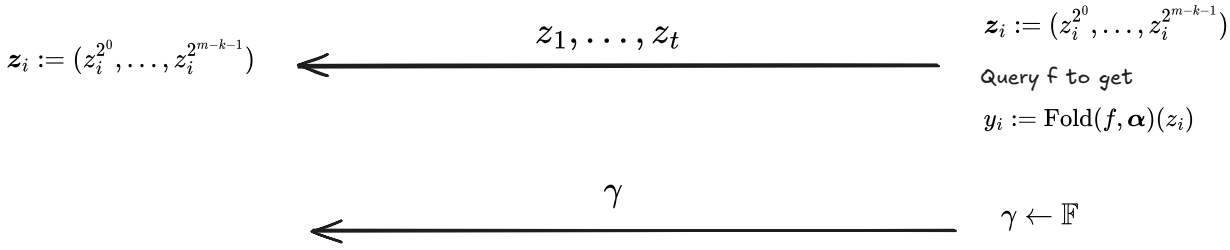
$$\hat{w}(Z, X) = Z \cdot \mathrm{eq}(\mathbf{X}, (z^{2^0}, \ldots, z^{2^{m-1}})). \tag{19}$$

WHIR 的一次迭代

前面提到 BaseFold 将 Sumcheck 和 FRI 协议结合了起来，而 WHIR 协议结合了 BaseFold 和 STIR 的思想，将 BaseFold 中的 FRI 协议改为了 STIR 协议，相比 FRI 协议，STIR 协议的查询复杂度更小。STIR 协议的核心思想是降低每一次迭代的码率，使得 Prover 发送的消息中的冗余增加，从而减少 Verifier 的查询复杂度。

下面深入 WHIR 协议的一次迭代(来自[ACFYb, 2.1.3 WHIR protocol])，看看 WHIR 是如何具体结合 BaseFold 与 STIR 协议的。经过一次迭代，将测试 $f \in \mathcal{C} := \mathrm{CRS}[\mathbb{F}, \mathcal{L}, m, \hat{w}, \sigma]$ 的 proximity 问题转换为测试 $f' \in \mathcal{C}' := \mathrm{CRS}[\mathbb{F}, \mathcal{L}^{(2)}, m-k, \hat{w}', \sigma']$。





6. Recursive claim

$$\hat{w}'(Z, \boldsymbol{X}) := \hat{w}(Z, \boldsymbol{\alpha}, \boldsymbol{X}) + Z \cdot \sum_{i=0}^{t} \gamma^{i+1} \cdot \mathrm{eq}(\boldsymbol{z}_i, \boldsymbol{X})$$

$$\sigma' := \hat{h}_k(\alpha_k) + \sum_{i=0}^{t} \gamma^{i+1} \cdot y_i$$

Test

$$g \in \mathrm{CRS}[\mathbb{F}, \mathcal{L}^{(2)}, m-k, \hat{w}', \sigma']$$

1. Sumcheck rounds. Prover 和 Verifier 针对 $\mathrm{CRS}[\mathbb{F}, \mathcal{L}, m, \hat{w}, \sigma]$ 中的约束

$$\sum_{\mathbf{b} \in \{0,1\}^m} \hat{w}(\hat{f}(\mathbf{b}), \mathbf{b}) = \sigma \tag{20}$$

进行 k 轮的 Sumcheck 交互，其中 $\hat{f}$ 是与 f 相对应的多元线性多项式。

a. Prover 向 Verifier 发送一个单变量多项式 $\hat{h}_1(X) := \sum_{\mathbf{b} \in \{0,1\}^{m-1}} \hat{w}(\hat{f}(X, \mathbf{b}), X, \mathbf{b})$，Verifier 检查 $\hat{h}_1(0) + \hat{h}_1(1) = \sigma$，选取随机数 $\alpha_1 \leftarrow \mathbb{F}$ 并发送，sumcheck 的 claim 就变为 $\hat{h}_1(\alpha_1) := \sum_{\mathbf{b} \in \{0,1\}^{m-1}} \hat{w}(\hat{f}(\alpha_1, \mathbf{b}), \alpha_1, \mathbf{b})$。 b. 对于第 i 轮，i 从 2 到 k，Prover 发送一个单变量多项式

$$\hat{h}_i(X) := \sum_{\mathbf{b} \in \{0,1\}^{m-i}} \hat{w}(\hat{f}(\alpha_1, \ldots, \alpha_{i-1}, X, \mathbf{b}), \alpha_1, \ldots, \alpha_{i-1}, X, \mathbf{b}) \tag{21}$$

Verifier 检查 $\hat{h}_i(0) + \hat{h}_i(1) = \hat{h}_{i-1}(\alpha_{i-1})$，选取随机数 $\alpha_i \leftarrow \mathbb{F}$，sumcheck 的 claim 就变为

$$\sum_{\mathbf{b} \in \{0,1\}^{m-i}} \hat{w}(\hat{f}(\alpha_1, \ldots, \alpha_{i-1}, \alpha_i, \mathbf{b}), \alpha_1, \ldots, \alpha_{i-1}, \alpha_i, \mathbf{b}) = \hat{h}_i(\alpha_i) \tag{22}$$

因此经过上述 k 轮的 sumcheck，prover 发送了多项式 $(\hat{h}_1, \ldots, \hat{h}_k)$，verifier 选取了随机数 $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_k) \in \mathbb{F}^k$。最初的 claim 变为下面这样的声明

$$\sum_{\mathbf{b} \in \{0,1\}^{m-k}} \hat{w}(\hat{f}(\boldsymbol{\alpha}, \mathbf{b}), \boldsymbol{\alpha}, \mathbf{b}) = \hat{h}_k(\alpha_k) \tag{23}$$

2. Send folded function. Prover 发送函数 $g : \mathcal{L}^{(2)} \to \mathbb{F}$。在 Prover 诚实的情况下，$\hat{g} \equiv \hat{f}(\boldsymbol{\alpha}, \cdot)$，$g$ 的定义是 $\hat{g}$ 在 domian $\mathcal{L}^{(2)}$ 上的求值。

   这里的意思是先对 $\hat{f}$ 用随机数 $\boldsymbol{\alpha}$ 进行 2^k 折，得到 $\hat{g} = \hat{f}(\boldsymbol{\alpha}, \cdot)$，此时 $\hat{g} : \mathcal{L}^{(2^k)} \to \mathbb{F}$，其定义域的范围是 $\mathcal{L}^{(2^k)}$，由于 $\hat{g}$ 本质是一个多项式，那么我们可以改变其自变量的定义域，将其改为 $\mathcal{L}^{(2)}$，函数 g 与 $\hat{g}$ 在 $\mathcal{L}^{(2)}$ 上的求值是一致的。
3. Out-of-domain sample. Verifier 选取一个随机数 $z_0 \leftarrow \mathbb{F}$ 并发送给 Prover。设 $\boldsymbol{z}_0 := (z_0^{2^0}, \ldots, z_0^{2^{m-k-1}})$。
4. Out-of-domain answers. Prover 发送 $y_0 \in \mathbb{F}$。在诚实的情况下，$y_0 := \hat{g}(\boldsymbol{z}_0)$。

5. Shift quiers and combination randomness. 对于 Verifier，对于每一个 $i \in [t]$，选取随机数 $z_i \leftarrow \mathcal{L}^{(2^k)}$ 并发送，通过查询 f 可以得到 $y_i := \mathrm{Fold}(f, \boldsymbol{\alpha})(z_i)$。设 $\boldsymbol{z}_i := (z_i^{2^0}, \ldots, z_i^{2^{m-k-1}})$。Verifier 另外选取随机数 $\gamma \leftarrow \mathbb{F}$ 并发送。
6. Recursive claim. Prover 和 Verifier 定义新的权重多项式与目标值：

$$\hat{w}'(Z, \boldsymbol{X}) := \hat{w}(Z, \boldsymbol{\alpha}, \boldsymbol{X}) + Z \cdot \sum_{i=0}^{t} \gamma^{i+1} \cdot \mathrm{eq}(\boldsymbol{z}_i, \boldsymbol{X}) \tag{24}$$

$$\sigma' := \hat{h}_k(\alpha_k) + \sum_{i=0}^{t} \gamma^{i+1} \cdot y_i, \tag{25}$$

接着，递归地测试 $g \in \mathrm{CRS}[\mathbb{F}, \mathcal{L}^{(2)}, m-k, \hat{w}', \sigma']$。

下面先说明 $g \in \mathrm{CRS}[\mathbb{F}, \mathcal{L}^{(2)}, m-k, \hat{w}', \sigma']$ 中的约束是正确的，即证明

$$\sum_{\boldsymbol{b} \in \{0,1\}^{m-k}} \hat{w}'(g(\boldsymbol{b}), \boldsymbol{b}) = \sigma' \tag{26}$$

代入 $\hat{w}'$ 与 σ' 得

$$\sum_{\boldsymbol{b} \in \{0,1\}^{m-k}} \hat{w}(g(\boldsymbol{b}), \boldsymbol{\alpha}, \boldsymbol{b}) + \sum_{\boldsymbol{b} \in \{0,1\}^{m-k}} g(\boldsymbol{b}) \cdot \sum_{i=0}^{t} \gamma^{i+1} \cdot \mathrm{eq}(\boldsymbol{z}_i, \boldsymbol{b}) = \hat{h}_k(\alpha_k) + \sum_{i=0}^{t} \gamma^{i+1} \cdot y_i \tag{27}$$

分两部分证明：

1. 证明

$$\sum_{\boldsymbol{b} \in \{0,1\}^{m-k}} \hat{w}(g(\boldsymbol{b}), \boldsymbol{\alpha}, \boldsymbol{b}) = \hat{h}_k(\alpha_k) \tag{28}$$

由协议的第 2 步可知 $g(\boldsymbol{b}) = \hat{f}(\boldsymbol{\alpha}, \boldsymbol{b})$，因此

$$\begin{aligned} \sum_{\boldsymbol{b} \in \{0,1\}^{m-k}} \hat{w}(g(\boldsymbol{b}), \boldsymbol{\alpha}, \boldsymbol{b}) &= \sum_{\boldsymbol{b} \in \{0,1\}^{m-k}} \hat{w}(\hat{f}(\boldsymbol{\alpha}, \boldsymbol{b}), \boldsymbol{\alpha}, \boldsymbol{b}) \\ &= \hat{h}_k(\alpha_k) \end{aligned} \tag{29}$$

上式的最后一个等式是由协议的第 1 步 sumcheck 最后的声明得到的。

2. 证明

$$\sum_{\boldsymbol{b} \in \{0,1\}^{m-k}} g(\boldsymbol{b}) \cdot \sum_{i=0}^{t} \gamma^{i+1} \cdot \mathrm{eq}(\boldsymbol{z}_i, \boldsymbol{b}) = \sum_{i=0}^{t} \gamma^{i+1} \cdot y_i \tag{30}$$

证明：

$$\begin{aligned} \sum_{\boldsymbol{b} \in \{0,1\}^{m-k}} g(\boldsymbol{b}) \cdot \sum_{i=0}^{t} \gamma^{i+1} \cdot \mathrm{eq}(\boldsymbol{z}_i, \boldsymbol{b}) &= \sum_{i=0}^{t} \gamma^{i+1} \cdot \sum_{\boldsymbol{b} \in \{0,1\}^{m-k}} g(\boldsymbol{b}) \cdot \mathrm{eq}(\boldsymbol{z}_i, \boldsymbol{b}) \\ &= \sum_{i=0}^{t} \gamma^{i+1} \cdot g(\boldsymbol{z}_i) \\ &= \sum_{i=0}^{t} \gamma^{i+1} \cdot y_i \end{aligned} \tag{31}$$

其中 $\sum_{\boldsymbol{b}\in\{0,1\}^{m-k}} g(\boldsymbol{b})\cdot \mathrm{eq}(\boldsymbol{z}_i,\boldsymbol{b}) = g(\boldsymbol{z}_i)$ 正是前面提到的用权重多项式 $\hat{w}(Z,\mathbf{X}) = Z\cdot \mathrm{eq}(\mathbf{X},\mathbf{z})$ 可以来约束多元线性多项式在某个点的值。

至此也就说明 $g \in \mathrm{CRS}[\mathbb{F}, \mathcal{L}^{(2)}, m-k, \hat{w}', \sigma']$ 中的约束定义是正确的。

新的权重多项式的定义 $\hat{w}'$ 为

$$\hat{w}'(Z,\boldsymbol{X}) := \hat{w}(Z,\boldsymbol{\alpha},\boldsymbol{X}) + Z\cdot\sum_{i=0}^{t}\gamma^{i+1}\cdot\mathrm{eq}(\boldsymbol{z}_i,\boldsymbol{X}) \tag{32}$$

分为两部分：

1. 第一部分 $\hat{w}(Z,\boldsymbol{\alpha},\boldsymbol{X})$ 约束了协议第 1 步中 k 轮 sumcheck 的正确性。
2. 第二部分 $Z\cdot\sum_{i=0}^{t}\gamma^{i+1}\cdot\mathrm{eq}(\boldsymbol{z}_i,\boldsymbol{X})$ 约束了 g 在 $\boldsymbol{z}_i$ 的值是正确的，并用随机数 γ 对这 $t+1$ 个约束进行了线性组合。 a. 对 $g(\boldsymbol{z}_0)=y_0$ 的约束，实际是在验证 out-of-domain answers 的正确性。 b. 对 $i\in[t]$，约束 $g(\boldsymbol{z}_i)=y_i$，是要求 shift queries 的正确性。

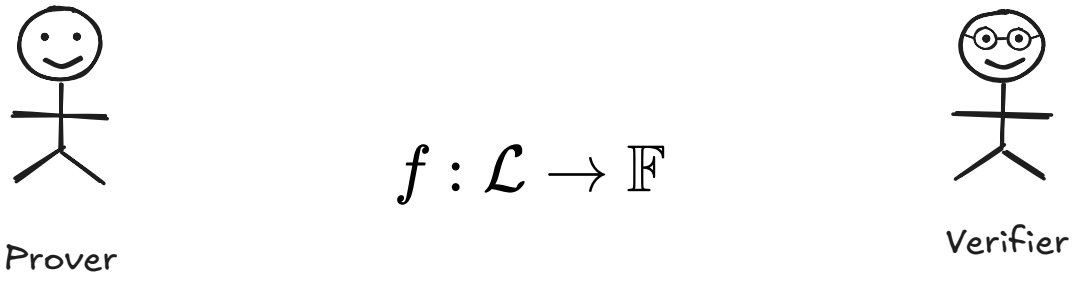
由此也可见权重多项式定义的灵活性，能一次实现多个约束。

WHIR 与 BaseFold 的联系

WHIR 采用了 BaseFold 的思想，本身 CRS 的定义就引入了类似 sumcheck 的约束。在协议的第 1 步，先做 k 轮的 sumcheck，这里 sumcheck 选取的随机数 $\boldsymbol{\alpha}=(\alpha_1,\ldots,\alpha_k)$ 与后续对 $\hat{f}$ 折叠所用的随机数是完全一致的，即协议的第 2 步 $\hat{g}=\hat{f}(\boldsymbol{\alpha},\cdot)$，这里对 $\hat{f}$ 进行了 2^k 折。

WHIR 与 STIR 的联系

在协议第 1 步使用 sumcheck 之后，后续的第 2-5 步与 STIR 协议类似。下图是 STIR 协议的一次迭代流程。



1. Sample folding randomness

r^{fold} ← $r^{\text{fold}} \leftarrow \mathbb{F}$

2. Send folded function

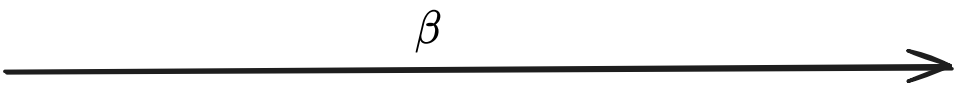
$g : \mathcal{L}' \to \mathbb{F}$

3. Out-of-domain sample

r^{out} ← $r^{\text{out}} \leftarrow \mathbb{F} \backslash \mathcal{L}'$

4. Out-of-domain reply

β



5. Shift queries

$r_1^{\text{shift}}, \ldots, r_t^{\text{shift}}$ ← $\forall i \in [t], r_i^{\text{shift}} \leftarrow \langle \omega^k \rangle$

$\mathcal{G} := \{r^{\text{out}}, r_1^{\text{shift}}, \ldots, r_t^{\text{shift}}\}$

Query f to get $y_i := f_{\text{fold}}(r_i^{\text{shift}})$

$p : \mathcal{G} \to \mathbb{F} \quad p(r^{\text{out}}) = \beta, p(r_i^{\text{shift}}) = y_i$

$f' := \text{Quotient}(f, \mathcal{G}, p)$

Test f'

STIR 协议的核心思想是在每一次迭代中降低码率，具体做法是在下一次迭代中，得到的折叠多项式 $\hat{g}$ 不在 $\mathcal{L}^{(2^k)}$ 上去求值，而是选择在一个只有原来 domain $\mathcal{L}$ 一半大小的 domain $\mathcal{L}^{(2)}$ 上进行求值，这里对应 WHIR 协议的第 2 步，这样做的好处是大大增加了发送消息的冗余，减少了 verifier 的查询复杂度。

对于 $f \in \mathcal{C} := \mathrm{CRS}[\mathbb{F}, \mathcal{L}, m, \hat{w}, \sigma]$，其码率为 $\rho = \frac{2^m}{|\mathcal{L}|}$，而经过一次 WHIR 迭代之后 $f' \in \mathcal{C}' := \mathrm{CRS}[\mathbb{F}, \mathcal{L}^{(2)}, m-k, \hat{w}', \sigma']$，其码率为

$$\rho' = \frac{2^{m-k}}{|\mathcal{L}^{(2)}|} = \frac{2^{m-k}}{\frac{|\mathcal{L}|}{2}} = \frac{2^{m-k+1}}{|\mathcal{L}|} = 2^{1-k} \cdot \rho = \left(\frac{1}{2}\right)^{k-1} \cdot \rho \tag{33}$$

当 $k \geq 2$ 时，可以看到 ρ' 会比 ρ 小，码率减小。

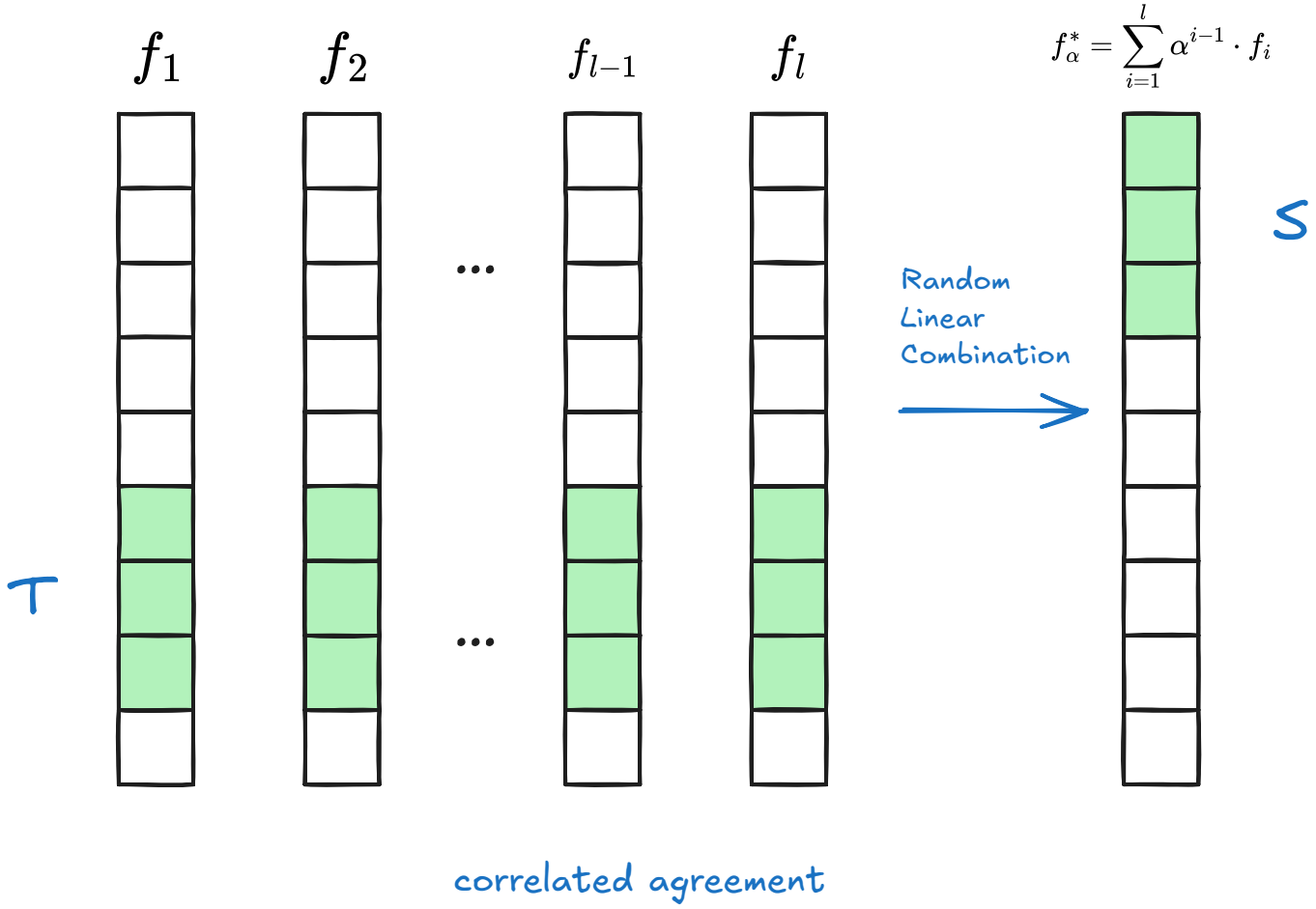
Mutual correlated agreement

[BCIKS20] 论文中给出的 correlated agreement 定理是证明 FRI 和 STIR 协议安全性的一个关键定理，其能确保在 FRI 协议或 STIR 协议中用随机数对原来的函数进行折叠的过程是安全的。在 WHIR 的安全性分析中，引入了一个新的概念 mutual correlated agreement，其结论比 correlated agreement 更强。

[ACFYb, 1.2 Mutual correlated agreement] 给出了correlated agreement 与 mutual correlated agreement 相关定义。码 $\mathcal{C} := \mathrm{RS}[\mathbb{F}, \mathcal{L}, m]$ 具有 (δ, ε) -correlated agreement 是说：如果对于每一个 $f_1, \ldots, f_\ell$，在 $\alpha \leftarrow \mathbb{F}$ 的均匀选择下，以概率 $1-\varepsilon$ 满足：如果存在一个集合 $S \subseteq \mathcal{L}$，其中 $|S| \geq (1-\delta) \cdot |\mathcal{L}|$，在 S 上 $f^*_\alpha := \sum_{i=1}^{\ell} \alpha^{i-1} \cdot f_i$ 与 $\mathcal{C}$ 一致，那么存在一个集合 $T \subseteq \mathcal{L}$，其中 $|T| \geq (1-\delta) \cdot |\mathcal{L}|$，在 T 上每个 f_i 与 $\mathcal{C}$ 一致。

上述定义中描述一个函数 f 与码 $\mathcal{C}$ 在一个集合 S 上“一致”的意思是，在编码空间 $\mathcal{C}$ 中存在一个码字 $u \in \mathcal{C}$ 使得对任意的 $x \in S$，都有 $f(x) = u(x)$。

correlated agreement 的定义如下图所示(参照视频 ZK12: WHIR: Reed-Solomon Proximity Testing with Super-Fast Verification)。



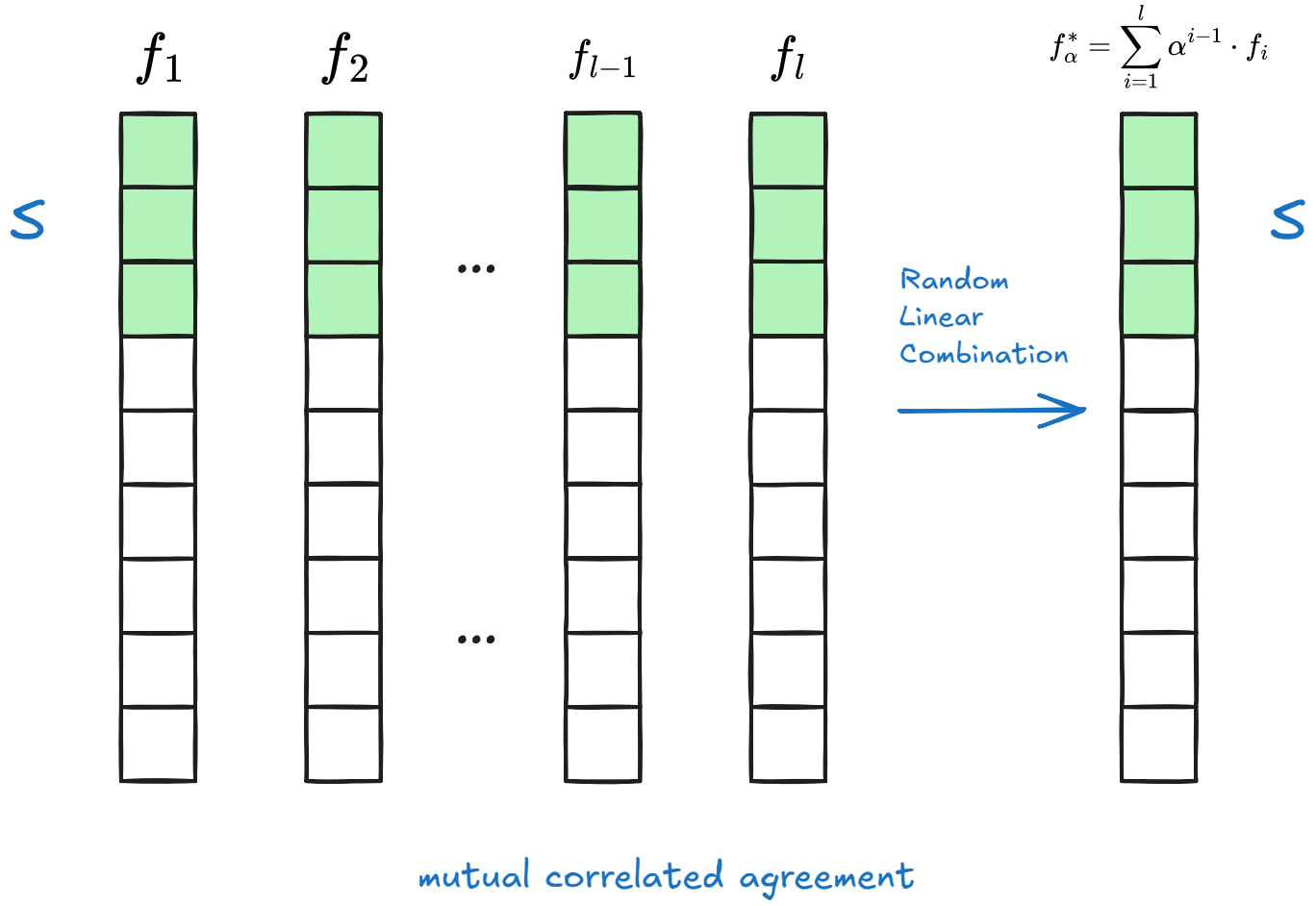
[BCIKS20] 论文给出码率为 ρ 的 Reed-Solomon 码具有 (δ, ε) -correlated agreement，其中 $\delta \in (0, 1-\sqrt{\rho})$，$\varepsilon := \frac{\mathrm{poly}(2^m, 1/\rho)}{|\mathbb{F}|}$。换句话说，如果 $\delta \in (0, 1-\sqrt{\rho})$ 并且

$$\Pr_{\alpha\in\mathbb{F}}\left[\Delta(f_\alpha^*,\mathcal{C})\le\delta\right]>\varepsilon=\frac{\mathrm{poly}(2^m,1/\rho)}{|\mathbb{F}|}\tag{34}$$

那么，存在集合 $T\subseteq\mathcal{L}$，以及码 $c_0,\dots,c_l\in\mathcal{C}$ 使得

1. $|T|\ge(1-\delta)\cdot|\mathcal{L}|$
2. 在 T 上每个 f_i 与 c_i 一致

可以发现 (δ,ε) -correlated agreement 定义中并未要求 S 集合和 T 集合是同一个集合，而在 WHIR 中引入了一个比 correlated agreement 更强的概念，叫做 *mutual correlated agreement* ，其要求 S 集合和 T 集合是同一个集合。如下图所示(参照视频 ZK12: WHIR: Reed-Solomon Proximity Testing with Super-Fast Verification):



在 WHIR 论文中给出了关于 mutual correlated agreement 如下的一个猜想。

猜想 1 [ACFY24b, Conjecture 1] (informal). 对于每一个 Reed-Solomon 码 $\mathcal{C}=\mathrm{RS}[\mathbb{F},\mathcal{L},m]$ ，如果其有 (δ,ε) -correlated agreement，其中 $\varepsilon=\frac{\mathrm{poly}(2^m,1/\rho)}{|\mathbb{F}|}$ ，那么其有 (δ,ε') -mutual correlated agreement，其中 $\varepsilon'=\frac{\mathrm{poly}(2^m,1/\rho)}{|\mathbb{F}|}$ 。

WHIR 论文中证明了在唯一解码情况下，即 $\delta\in(0,\frac{1-\rho}{2})$ 时，猜想 1 成立，有 $\varepsilon'=\varepsilon$ 。这样也就将 correlated agreement 与 mutual correlated agreement 联系了起来。

总结

WHIR 协议结合了 BaseFold 与 STIR 的思想。首先对于 RS 编码中的一元多项式，可以通过变量替换的方式，看做等价的多元线性多项式，对于一元多项式的折叠也等价于对对应的多元线性多项式进行折叠。这样就使得 WHIR 既支持一元多项式也支持多元线性多项式。

其次，给出了新的 CRS 编码定义，在 RS 编码的基础上增加了一个类似 sumcheck 的约束，也就是对权重多项式 $\hat{w}$ 进行类似 sumcheck 的约束。这里权重多项式定义的灵活性使得在协议中可以一次要求满足多个约束，包括约束 sumcheck 的正确性，out-of-domain answers 的正确性以及 shift queries 的正确性。

随后，通过深入 WHIR 的一次迭代协议，可以看出其与 BaseFold 和 STIR 协议的联系。这里的关键还是先搭建起一元多项式与多元线性多项式的桥梁，单变量函数 f 和多元线性多项式 $\hat{f}$ 之间可以自由切换。通过 CRS 的引入，协议的目标增加了验证一个类似 sumcheck 的约束的正确性，

$$\sum_{\mathbf{b}\in\{0,1\}^m} \hat{w}(\hat{f}(\mathbf{b}), \mathbf{b}) = \sigma \tag{35}$$

因此结合 BaseFold 的思想，先进行 k 轮的 sumcheck，在多元线性多项式 $\hat{f}$ 中用 sumcheck 协议中的随机数 $\boldsymbol{\alpha}$ 替换掉 k 个变量。对 $\hat{f}$ 的折叠依然用同样的随机数 $\boldsymbol{\alpha}$ 。结合 STIR 的思想，为了降低码率，折叠之后的函数在一个更大的域 $\mathcal{L}^{(2)}$ 上进行求值。后续 WHIR 协议中的 out-of-domain sample 以及 shift queries 等步骤与 STIR 协议类似。

最后，介绍了 WHIR 协议安全性证明中用到的一个比 correlated agreement 结论更强的 mutual correlated agreement 结论。

References

- [ACFY24a] Gal Arnon, Alessandro Chiesa, Giacomo Fenzi, and Eylon Yogev. "STIR: Reed-Solomon proximity testing with fewer queries." In *Annual International Cryptology Conference*, pp. 380-413. Cham: Springer Nature Switzerland, 2024.
- [ACFY24b] Gal Arnon, Alessandro Chiesa, Giacomo Fenzi, and Eylon Yogev. "WHIR: Reed–Solomon Proximity Testing with Super-Fast Verification." *Cryptology ePrint Archive* (2024).
- [BBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. “Fast Reed–Solomon Interactive Oracle Proofs of Proximity”. In: *Proceedings of the 45th International Colloquium on Automata, Languages and Programming (ICALP)*, 2018.
- [BCIKS20] Eli Ben-Sasson, Dan Carmon, Yuval Ishai, Swastik Kopparty, and Shubhangi Saraf. Proximity Gaps for Reed–Solomon Codes. In *Proceedings of the 61st Annual IEEE Symposium on Foundations of Computer Science*, pages 900–909, 2020.
- [ZCF24] 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.
- Blog: WHIR: Reed–Solomon Proximity Testing with Super-Fast Verification
- video: ZK12: WHIR: Reed-Solomon Proximity Testing with Super-Fast Verification