22 Lookup Checks

22.1 Motivation

Suppose you need to develop the zero-knowledge circuit over some "unfriendly" construction such as AES-128, SHA-256, or pairing implemented on the non-native finite field. One way to resolve this issue is to use the proving scheme that is most "native" to the logic needed to implement such constructions (for example, you might want to switch to STARK instead of Groth16 for implementing hash functions). However, zero-knowledge world came up with one more way to resolve this issue without changing the whole system.

Consider the following problem: you have the witness $\mathbb{w}$ in which you want to check whether elements, say, $\vec{z} := \{z_i\}_{i\in[n]} \subseteq \mathbb{F}$ are contained in the lookup table $\vec{t} := \{t_j\}_{j\in[d]} \subseteq \mathbb{F}$ of size d. We write this check as $\{z_i\}_{i\in[n]} \subseteq \{t_j\}_{j\in[d]}$ or $\vec{z} \subseteq \vec{t}$ for short.

> **Example 22.1.** As a not very practical example, suppose $\vec{t} = \{0, 1\}$. Then the check $\vec{z} \subseteq \vec{t}$ is read as "check whether each of elements in $\vec{z}$ is binary". Another trivial example might be the following: set $\vec{t} := \{1, 6, 7, 10\}$. Then, for example, $\vec{z} = \{10, 6, 7, 1, 1, 6, 10, 7, 1\}$ should pass the lookup check since $\vec{z}$ consists of elements from $\vec{t}$ only. In turn, $\vec{z}^* = \{1, 6, 10, 5\}$ is not a valid witness corresponding to $\vec{t}$ since $5 \notin \vec{t}$.

Modern protocols have the following property: such check costs only $\mathcal{O}(n+d)$ constraints. Let us consider why this is significant for many applications in the wild.

Range Checks. The most obvious application is the range check. Assume that $t_j = j$ and $d = 2^w$. This way, the check $\{z_i\}_{i\in[n]} \subseteq \{t_j\}_{j\in[d]}$ simply corresponds to checking whether "is each z_i indeed a w-bit integer"? Needless to say, this has significant implications in various ZK applications: privacy-preserving payments, implementation of non-native arithmetic, or even zkML. In the case of zkML, for instance, *Bionetta* uses lookup checks to speed up the computation of the ReLU function, given by $\text{ReLU}(x) = \max\{0, x\}$, by the factor of up to $\times 20$.

Binary Arithmetic. Suppose you are implementing the SHA256, in which you need to frequently XOR binary strings. To simplify the problem, instead of XORing two, say, 256-bit strings, you XOR 8-bit strings: $\mathbf{c} = \mathbf{a} \oplus \mathbf{b}$ for $\mathbf{a}, \mathbf{b}, \mathbf{c} \in \{0, 1\}^8$. The naive approach is to bit decompose each of $\mathbf{a}, \mathbf{b}, \mathbf{c}$ and verify that the check holds, but that requires enormous number of constraints. The more clever lookup-based approach is as follows: compose the table $\vec{t} := \{(t_j^{\langle 1\rangle}, t_j^{\langle 2\rangle}, t_j^{\langle 3\rangle})\}_{j\in[d]} \subseteq \mathbb{F}^3$ which consists of all distinct valid 8-bit XORs: $t_j^{\langle 1\rangle} \oplus t_j^{\langle 2\rangle} = t_j^{\langle 3\rangle}$. Note that in such case $d = 2^{16}$ since there are 2^{16} distinct equalities of such form (which simply enumerate over all possibilities of two 8-bit inputs to the XOR gate). We thus want to show that $\vec{z} := \{(a_i, b_i, c_i)\}_{i\in[n]} \subseteq \vec{t}$ where a_i, b_i, c_i are the part of the witness.

Now what remains is to show how to reduce this check to the "one-dimensional" case: recall that lookup protocols can merely check for inclusion for the ordered set of *scalars* and not *tuples*. It is done fairly easy: sample some random $\alpha \leftarrow\$ \mathbb{F}$ and instead of viewing $\vec{z}$ as a tuple of three elements, view it as a list $\vec{z}_\alpha$ of n scalars of form $a_i + \alpha b_i + \alpha^2 c_i$. Similarly, each table elements of $\vec{t}_\alpha$ is viewed as $t_j^{\langle 1\rangle} + \alpha t_j^{\langle 2\rangle} + \alpha^2 t_j^{\langle 3\rangle}$. Then, the inclusion check $\vec{z} \subseteq \vec{t}$ due to Schwartz-Zippel lemma is equivalent to $\vec{z}_\alpha \subseteq \vec{t}_\alpha$ with overwhelming probability.

22.2 Plookup Protocol

One of the first lookup protocols that became practical in the zero-knowledge world is the *plookup protocol*. It is mostly used in Poly-IOPs but can presumably be compiled to other types of protocols as well, as long as *multiset equality* check can be implemented optimally.

Now, if you recall the $\mathcal{P}\text{lon}\mathcal{K}$ construction, we can check that the multiset $\{s'_i\}_{i\in[n]}$ is the permutation of the multiset $\{s_i\}_{i\in[n]}$. This was done by comparing the products $\prod_{i\in[n]}(\gamma + s'_i) = \prod_{i\in[n]}(\gamma + s_i)$ at a random point γ. Due to Schwartz-Zippel lemma, equality in such case implies $s_i = s'_{\sigma(i)}$ with overwhelming probability for aribtrary $i \in [n]$ and some permutation $\sigma \in S_n$.

The primary issue with the lookup check is that instead of multisets equality, we need to check whether one multiset is the subset of another, which is a much trickier problem: we need to show that polynomials $Z(X) := \prod_{i\in[n]}(X - z_i)$ and $T(X) := \prod_{j\in[d]}(X - t_j)$ have the same roots, ignoring multiplicities. Let us introduce the following tool which will help to reduce the problem to a simpler one.

Definition 22.1. The **difference set** of $\vec{s} = \{s_i\}_{i\in[n]}$, denoted by $\delta\vec{s}$, is $\delta\vec{s} := \{s_{i+1} - s_i\}_{i\in[n-1]}$. Additionally, by $\partial\vec{s}$ denote the non-zero difference set: the set $\delta\vec{s}$, but without zero elements.

Example 22.2. Suppose $\vec{s} = \{1, 2, 5, 10, 13\}$. Then, we have

$$\delta\vec{s} = \{2-1, 5-2, 10-5, 13-10\} = \{1, 3, 5, 3\} = \partial\vec{s}$$

A slightly more motivating example is $\vec{s} := \{1, 1, 4, 4, 8, 8, 8\}$. Then:

$$\delta\vec{s} = \{0, 3, 0, 4, 0, 0\}, \quad \partial\vec{s} = \{3, 4\}.$$

Now let us get to the initial goal of proving $\vec{z} = \{z_i\}_{i\in[n]} \subseteq \{t_j\}_{j\in[d]} =: \vec{t}$. Let us look at a sorted version $\vec{s}$ of the values of $\vec{z}$ and compare the non-zero difference sets $\partial\vec{s}$ and $\partial\vec{t}$. Notice the following: if $\vec{z} \subseteq \vec{t}$, then $\partial\vec{s} = \partial\vec{t}$. So the prover $\mathcal{P}$ might want to simply prove this set equality. Unfortunately, the converse (that is, $\partial\vec{s} = \partial\vec{t} \Rightarrow \vec{z} \subseteq \vec{t}$) is not generally true. Indeed, consider the

following example:

$$\vec{t} = \{1, 4, 8\}, \quad \vec{s} = \{1, 1, 4, 8, 8, 8\}, \quad \vec{s}^* = \{1, 5, 5, 5, 8, 8\}.$$

Note that $\partial\vec{t} = \partial\vec{s} = \partial\vec{s}^* = \{3, 4\}$, but obviously $\vec{s}^* \not\subseteq \vec{t}$. So we need to modify our checks to introduce more robust lookup check condition.

Now, consider the following version: let $\vec{s}$ be the *sorted* multiset of concatenation, which we denote by $(\vec{z}, \vec{t}) = \{z_0, \ldots, z_{n-1}, t_0, \ldots, t_{d-1}\}$. Now we impose two checks: (1) $\vec{s} = (\vec{z}, \vec{t})$, (2) $\partial\vec{s} = \partial\vec{t}$. Turns out this condition is necessary and sufficient.

Proposition 22.2. $\vec{z} \subseteq \vec{t} \Leftrightarrow \vec{s} = (\vec{z}, \vec{t}) \wedge \partial\vec{s} = \partial\vec{t}$.

Proof. $\Rightarrow$ **direction.** The first condition holds due to construction. Note that the second condition holds for the reason that $\partial\vec{s}$ contains only differences of form $z_i - t_j$, $z_i - z_j$, or $t_i - t_j$. If $\vec{z} \subseteq \vec{t}$ and given $\vec{s}$ is sorted, $\partial\vec{s}$ contains only elements of form $t_{j+1} - t_j$, which is exactly $\partial\vec{t}$.

$\Leftarrow$ **direction.** Given first condition, we know that $\vec{z} \subseteq \vec{s}$. Thus, if we show $\vec{s} \subseteq \vec{t}$, we are done. Suppose, on the contrary, that $\vec{s} \not\subseteq \vec{t}$. Without loss of generality, assume thus that there is some $h \in \vec{s}$ such that $t_i < h < t_{i+1}$ for some $i \in [d]$. Then, $h - t_i, t_{i+1} - h \in \partial\vec{s}$ but clearly $h - t_i, t_{i+1} - h \notin \partial\vec{t}$ since $h \notin \vec{t}$. Contradiction. Thus $\vec{z} \subseteq \vec{s} \subseteq \vec{t}$.

Example 22.3. Again, consider the following lookup table and (sorted) witness elements:

$$\vec{t} = \{1, 4, 8\}, \quad \vec{z} = \{1, 1, 4, 8, 8, 8\}, \quad \vec{z}^* = \{1, 5, 5, 5, 8, 8\}.$$

Now consider the sorted multisets of concatenations:

$$\vec{s} = \{1, 1, 1, 4, 4, 8, 8, 8, 8\}, \quad \vec{s}^* = \{1, 1, 4, 5, 5, 5, 8, 8, 8\}$$

Let us check whether both conditions are satisfied. The first condition is satisfied for both $\vec{s}$ and $\vec{s}^*$ due to construction. The second condition, though, is satisfied only for $\vec{s}$. Indeed:

$$\partial\vec{t} = \{3, 4\} = \partial\vec{s} \neq \partial\vec{s}^* = \{1, 3\}$$

Reducing two checks to one. At this point, one can apply the $\mathcal{P}$lon$\mathcal{K}$-based protocol to check whether $\vec{s} = (\vec{z}, \vec{t})$ and $\partial\vec{s} = \partial\vec{t}$. However, *plookup* allows to use just a single one. Here how it works.

Definition 22.3. The **randomized difference set** with randomness $\beta \in \mathbb{F}^\times$

of multiset $\vec{s} = \{s_i\}_{i\in[n]}$, further denoted as $\partial_\beta \vec{s}$, is defined as follows:

$$\partial_\beta \vec{s} \triangleq \{s_i + \beta s_{i+1}\}_{i\in[n-1]}.$$

Remark. Note that $\partial_{-1}\vec{s}$ corresponds to the difference set of $\vec{s}$ (albeit with the negative sign).

Now, the verifier chooses a random $\beta \leftarrow\!\$\ \mathbb{F}^\times$ and it suffices to impose a single multiset check between $\partial_\beta \vec{s}$ and $((1+\beta)\vec{z}, \partial_\beta \vec{t})$. We provide the following proposition (without proof).

Proposition 22.4. $\vec{z} \subseteq \vec{t}$ if and only if $\partial_\beta \vec{s} = ((1+\beta)\vec{z}, \partial_\beta \vec{t})$ for $\beta \leftarrow\!\$\ \mathbb{F}^\times$.

Intuition. Such proposition came out of nowhere, so let us explain why (intuitively) it works. First, let us see how $\partial_\beta \vec{s}$ for $\vec{z} \subseteq \vec{t}$ looks like. Compared to $\partial\vec{s}$, two same consecutive elements s_i, s_{i+1} with $s_i = s_{i+1}$ don't get zeroed: instead, one gets $(1+\beta)s_i$. So at least we know where $1+\beta$ coefficient comes from! Other than same elements, in $\partial_\beta \vec{s}$, one gets coefficients of form $t_i + \beta t_{i+1}$, which is exactly $\partial_\beta \vec{t}$ by definition. This way $\partial_\beta \vec{s}$ consists of:

- Expressions of form $(1+\beta)z_i$, which together form $(1+\beta)\vec{z}$.
- Expressions of form $t_i + \beta t_{i+1}$, which together form $\partial_\beta \vec{t}$.

Thus we naturally require $\partial_\beta \vec{s} = ((1+\beta)\vec{z}, \partial_\beta \vec{t})$. Why this condition is sufficient is less obvious, so we leave it without proof.

Example 22.4. Once again, consider the following lookup table and (sorted) witness elements:

$$\vec{t} = \{1, 4, 8\}, \quad \vec{z} = \{1, 1, 4, 8, 8, 8\}.$$

As previously shown, the sorted concatenation is $\vec{s} = \{1, 1, 1, 4, 4, 8, 8, 8, 8\}$. Now, suppose we have sampled some random $\beta \leftarrow\!\$\ \mathbb{F}^\times$. Then we have:

$$\partial_\beta \vec{s} = \{1+\beta, 1+\beta, 1+4\beta, (1+\beta)4, 4+8\beta, (1+\beta)8, (1+\beta)8, (1+\beta)8\}$$

Now note that for $(1+\beta)\vec{z}$ and $\partial_\beta \vec{t}$ we have:

$$(1+\beta)\vec{z} = \{1+\beta, 1+\beta, (1+\beta)4, (1+\beta)8, (1+\beta)8, (1+\beta)8\}, \quad \partial_\beta \vec{t} = \{1+4\beta, 4+8\beta\}$$

Clearly, $\partial_\beta \vec{s} = ((1+\beta)\vec{z}, \partial_\beta \vec{t})$.

22.2.1 plookup Precise Scheme

Fix integers n, d — witness size and lookup table size. Given witness $\boldsymbol{z} \in \mathbb{F}^n$ and lookup table $\boldsymbol{t} \in \mathbb{F}^d$, we want to ensure $\boldsymbol{z} \subseteq \boldsymbol{t}$. Let $\Omega = \{\omega^j\}_{j\in[n]} \leq \mathbb{F}^\times$ be a

multiplicative field subgroup. We say $\boldsymbol{z} \subseteq \boldsymbol{t}$ is *sorted by* $\boldsymbol{t}$ when values appear in the same order in $\boldsymbol{z}$ as they do in $\boldsymbol{t}$.

Now, given $\boldsymbol{t} \in \mathbb{F}^d$, $\boldsymbol{z} \in \mathbb{F}^n$, and $\boldsymbol{s} \in \mathbb{F}^{n+d}$, define bi-variate polynomials Z and T as follows:

$$Z(\beta, \gamma) \triangleq (1+\beta)^n \prod_{i \in [n]} (\gamma + z_i) \prod_{i \in [d-1]} (\gamma(1+\beta) + t_i + \beta t_{i+1})$$

$$T(\beta, \gamma) \triangleq \prod_{i \in [n+d-1]} (\gamma(1+\beta) + s_i + \beta s_{i+1})$$

Why on Earth do we need these two polynomials? Consider the following proposition:

Proposition 22.5. $Z \equiv T$ if and only if $\boldsymbol{z} \subseteq \boldsymbol{t}$ and $\boldsymbol{s}$ is $(\boldsymbol{z}, \boldsymbol{t})$ sorted by $\boldsymbol{t}$.

Now, this fact is completely unobvious, and you can see the proof in the original plookup paper. This motivates us to formulate the following protocol. Without loss of generality, assume $d = n + 1$ (if $d \leq n$, pad $\boldsymbol{t}$ with $n - d + 1$ repetitions of the last element).

Preprocessing. Compute the polynomial $t(X)$ that encodes the values of lookup table $\boldsymbol{t} \in \mathbb{F}^{n+1}$ over the domain Ω: that is, $t(\omega^j) = t_j$.

Inputs. Polynomial $z(X)$, encoding vector $\boldsymbol{z} \in \mathbb{F}^n$ over Ω: that is, $z(\omega^j) = z_j$.

Protocol.

1. $\mathcal{P}$ computes $\boldsymbol{s} = (\boldsymbol{z}, \boldsymbol{t}) \in \mathbb{F}^{2n+1}$ sorted by $\boldsymbol{t}$. Interpolate two polynomials $h_1, h_2 \in \mathbb{F}^{<(n+1)}[X]$ as follows: $h_1(\omega^j) = s_j$ for $j \in [n+1]$ and $h_2(\omega^j) = s_{n+j}$ for $j \in [n+1]$. That is, h_1 interpolates first $n+1$ elements of $\boldsymbol{s}$, while h_2 the remaining n.
2. $\mathcal{P}$ sends commitments of h_1 and h_2 to verifier $\mathcal{V}$ (namely, so that $\mathcal{V}$ will have an oracle access to both h_1 and h_2).
3. $\mathcal{V}$ picks random $\beta, \gamma \leftarrow\!\!\$\ \mathbb{F}$ and sends them to $\mathcal{P}$.
4. $\mathcal{P}$ computes a polynomial $F \in \mathbb{F}^{<(n+1)}[X]$ that aggregate the value $Z(\beta, \gamma)/T(\beta, \gamma)$ where Z, T are as described above. Specifically, let:
   (a) $F(\omega) = 1$.
   (b) For each $2 \leq j \leq n$, compute:
   $$F(\omega^i) = \frac{(1+\beta)^{i-1} \prod_{j<i} (\gamma + z_j) \prod_{1 \leq j < i} (\gamma(1+\beta) + t_j + \beta t_{j+1})}{\prod_{1 \leq j < i} (\gamma(1+\beta) + s_j + \beta s_{j+1})(\gamma(1+\beta) + s_{n+j} + \beta s_{n+j+1})}$$
   (c) $F(\omega^{n+1}) = 1$.
5. $\mathcal{P}$ sends commitment of F to $\mathcal{V}$.
6. $\mathcal{V}$ checks that F is of the form described above, and that $F(\omega^{n+1}) = 1$. More precisely, $\mathcal{V}$ checks the following identities for each $\omega^j \in \Omega$:
   (a) $L_1(\omega^j)(F(\omega^j) - 1) = 0$.

(b) $(\omega^j - \omega^{n+1})F(\omega^j)(1+\beta)(\gamma+z(\omega^j))(\gamma(1+\beta)+t(\omega^j)+\beta t(\omega\cdot\omega^j)) = (\omega^j - \omega^{n+1})F(\omega\cdot\omega^j)(\gamma(1+\beta)+h_1(\omega^j)+\beta h_1(\omega\cdot\omega^j))(\gamma(1+\beta)+h_2(\omega^j)+\beta h_2(\omega\cdot\omega^j))$.

(c) $L_{n+1}(\omega^j)(h_1(\omega^j) - h_2(\omega\cdot\omega^j)) = 0$.

(d) $L_{n+1}(\omega^j)(F(\omega^j) - 1) = 0$.

Recall that $L_j(X)$ is the Lagrange basis polynomial: that is, $L_j(\omega^i) = \delta_{i,j}$.

Intuition. Four checks that $\mathcal{V}$ imposes are not very obvious (especially the second one). So here is the brief explanation of their meaning:

1. First equation checks whether $F(\omega) = 1$: note that since $L_1(\omega^j)$ is non-zero only for $j = 1$, the equation essentially reduces to $F(\omega) - 1 = 0$.
2. This is the main verification that is needed to check whether $Z(\beta,\gamma) \equiv T(\beta,\gamma)$. However, since both Z and T are high-degree polynomials, we cannot directly check their equivalence. For that reason, we introduce the "accumulator" polynomial F that should satisfy $F(\omega) = F(\omega^{n+1}) = 1$ and recursive condition

$$\frac{F(\omega^{j+1})}{F(\omega^j)} = \frac{(1+\beta)(\gamma+z_j)(\gamma(1+\beta)+t_j+\beta t_{j+1})}{(\gamma(1+\beta)+s_j+\beta s_{j+1})(\gamma(1+\beta)+s_{n+j}+\beta s_{n+j+1})}$$

   Note that given $F(\omega) = 1$ and this recursive relation, by taking the product of both sides from 1 to n one obtains $F(\omega^{n+1}) = Z(\beta,\gamma)/T(\beta,\gamma)$ which is one, according to the previously specified lemma.
3. Since $L_{n+1}(\omega^j)$ is non-zero only for $j = n+1$, this check verifies that $h_1(\omega^{n+1}) = h_2(\omega^{n+2})$. This is the consistency check for h_1 and h_2 that checks whether h_1 and h_2 are properly "glued" together to encode $\boldsymbol{s}$.
4. Finally, the last check verifies whether $F(\omega^{n+1}) = 1$.

Lemma 22.6. Soundness of the above protocol is at least $1 - (5n+4)/|\mathbb{F}|$.

22.3 Logup

While *plookup* is the Poly-IOP-based protocol, *logup* allows to prove the lookup inclusion using SumCheck and rather more algebraic (compared to plookup) approach. Let's see how it works.

22.3.1 Preliminaries

Compared to previously discussed Sum-Check protocol, to stick with the original logup notation, we consider a slighly different hypercube: instead of boolean version $\{0,1\}^n$, we consider the hypercube $\{-1,1\}^n = \{\pm 1\}^n$, which we denote by $\mathcal{Q}_n$ for short. Note that the SumCheck protocol is exactly the same as for the boolean case, but we need to use "-1" instead of "0". Besides, the

multilinear lagrange basis polynomials have a much nicer form:

$$\mathsf{eq}(\mathbf{x};\mathbf{y}) = \frac{1}{2^n}\prod_{j=1}^{n}(1+x_j y_j)$$

22.3.2 Logarithmic Derivative

One of the ideas of the logup protocol is to reduce the polynomial check to the fractional one. To show this transition, we need a couple of formalities before we proceed.

Definition 22.7. Given a polynomial $q(X) := \sum_{j=0}^{d} q_j X^j \in \mathbb{F}[X]$, its **formal derivative**, in a similar fashion to calculus, is given by $q'(X) \triangleq \sum_{j=1}^{d} j q_j X^{j-1}$.

Example 22.5. For instance, given $q(X) = 1 + 2X + 3X^2$, the formal derivative is $q'(X) = 2 + 6X$.

It can be shown that the formal derivative obeys all the usual rules taught in calculus. Now, one more definition.

Definition 22.8. For a function $q(X)/r(X)$ from the rational function field $\mathbb{F}(X)$, the formal derivative is defined as:

$$\left(\frac{q(X)}{r(X)}\right)' \triangleq \frac{q'(X)r(X) - q(X)r'(X)}{r(X)^2}$$

For the further derivation, we will be interested in particular cases when derivatives turn out to be zero: that is $q'(X) = 0$ for the polynomial case or $\left(\frac{q(X)}{r(X)}\right)' = 0$ for a rational one. If polynomials are given in the real field, then these conditions immediately imply $q(X) \equiv \text{const}$ for polynomial case and $q(X) = cr(X)$ with $c = \text{const}$ for the rational case. Surprisingly, when considering the field $\mathbb{F}$ of finite characteristic p, these results are not generally true. Fortunately for us, they turn out to be false only for extreme case when degrees of polynomials are more than p, which, as you can imagine, never happens in practice. So we give the following lemma.

Lemma 22.9. Assume $q(X)$ and $r(X)$ are polynomials of degree less than p. Then:

- If $q'(X) = 0$, then q is constant.
- If $(q(X)/r(X))' = 0$, then $q(X)/r(X) = c$ for some $c \in \mathbb{F}$.

Finally, the logarithmic derivative.

Definition 22.10. The **logarithmic derivative** of $q(X) \in \mathbb{F}[X]$ is given by the rational function $q'(X)/q(X)$.

Remark. The name "logarithmic" comes from the fact that in calculus, for a function $f : \mathbb{R} \to \mathbb{R}$, the derivative of $\log f(x)$ is given by $f'(x)/f(x)$ by the chain rule.

One reason why logarithmic derivatives are attractive for our application is the following: the logarithmic derivative of $q(X)r(X)$ is the sum of logarithmic derivatives of $q(X)$ and $r(X)$. Indeed, note that the logarithmic derivative of $q(X)r(X)$ is nothing but:

$$\frac{(q(X)r(X))'}{q(X)r(X)} = \frac{q'(X)r(X) + q(X)r'(X)}{q(X)r(X)} = \frac{q'(X)}{q(X)} + \frac{r'(X)}{r(X)}$$

In particular, logarithmic derivative of $\prod_{j=1}^{n}(X + z_j)$ is given by $\sum_{j=1}^{n} \frac{1}{X+z_j}$. From all previously mentioned facts we can finally formulate some useful facts about lookup checks. Let us start from the simple observation.

Lemma 22.11. Let $\vec{a} = \{a_i\}_{i\in[n]}$ and $\vec{b} = \{b_i\}_{i\in[n]}$ be two sequences of elements from $\mathbb{F}$. To verify with overwhelming probability whether two multisets are equal, it suffices to check

$$\sum_{j=1}^{n} \frac{1}{\gamma + a_j} = \sum_{j=1}^{n} \frac{1}{\gamma + b_j}$$

for randomly chosen $\gamma \leftarrow\!\!\$\ \mathbb{F}$.

Proof. Recall that two multisets are equal with overwhelming probability if for randomly selected $\gamma \leftarrow\!\!\$\ \mathbb{F}$, we have $\prod_{j=1}^{n}(a_j + \gamma) = \prod_{j=1}^{n}(b_j + \gamma)$. Thus it suffices to show that imposing this product equality is equivalent to the statement in the lemma. Notice that if products are indeed the same, so are their logarithmic derivatives (with respect to γ), so we immediately get $\sum_{j=1}^{n} \frac{1}{\gamma+a_j} = \sum_{j=1}^{n} \frac{1}{\gamma+b_j}$. To show the opposite direction, assume that logarithmic derivatives of $q_{\vec{a}}(X) = \prod_{j=1}^{n}(X + a_j)$ and $q_{\vec{b}}(X) = \prod_{j=1}^{n}(X + b_j)$ are the same, so $\frac{q'_{\vec{a}}(X)}{q_{\vec{a}}(X)} = \frac{q'_{\vec{b}}(X)}{q_{\vec{b}}(X)}$. Then:

$$\left(\frac{q_{\vec{a}}(X)}{q_{\vec{b}}(X)}\right)' = \frac{q'_{\vec{a}}(X)q_{\vec{b}}(X) - q_{\vec{a}}(X)q'_{\vec{b}}(X)}{q_{\vec{b}}(X)^2} = 0$$

Thus $\frac{q_{\vec{a}}(X)}{q_{\vec{b}}(X)} = c$ for constant $c \in \mathbb{F}$. Since the leading coefficients of both $q_{\vec{a}}(X)$ and $q_{\vec{b}}(X)$ are 1, we conclude that $c = 1$, which immediately implies

$\prod_{j=1}^{n}(a_j+\gamma) = \prod_{j=1}^{n}(b_j+\gamma)$.

The consequence of this fact is the following.

Lemma 22.12. Given two sequences of elements $\{t_i\}_{i\in[d]}$ and $\{z_i\}_{i\in[n]}$, the inclusion check $\{z_i\}_{i\in[n]} \subseteq \{t_i\}_{i\in[d]}$ is satisfied if and only if there exist the set of multiplicities $\{\mu_i\}_{i\in[d]}$ where $\mu_i = \#\{j \in [n] : z_j = t_i\}$ such that:

$$\sum_{i\in[n]} \frac{1}{X+z_i} = \sum_{i\in[d]} \frac{\mu_i}{X+t_i}$$

In particular, checking such equality at random point from $\mathbb{F}$ results in the soundness error of up to $(n+d)/|\mathbb{F}|$, which becomes negligible for fairly large $|\mathbb{F}|$.

22.3.3 Applying SumCheck

Now we need to build the SumCheck equation. Instead of considering the inclusion check for scalars, we consider the inclusion check for *multivariate polynomials*. Notice that the lookup table $\vec{t} = \{t_j\}_{j\in[d]}$ for $d = 2^v$ can be viewed as a function $t : \mathcal{Q}_v \to \mathbb{F}$. However, here is the issue: our witness vector $\vec{z} = \{z_i\}_{i\in[n]}$ typically has a significantly larger size than $\vec{t}$, so we cannot view $\vec{z}$ as a function $\mathcal{Q}_v \to \mathbb{F}$ since the input domain size is too small (also quite obvious remark — we cannot have two different hypercubes for the sumcheck protocol). For that reason, we consider a list of $m := \lceil n/d \rceil$ functions $z_1, \ldots, z_m : \mathcal{Q}_v \to \mathbb{F}$ that encode all n values of $\{z_i\}_{i\in[n]}$.

This way, we want to check whether $\bigcup_{i\in[m]}\{z_i(\mathbf{x})\}_{\mathbf{x}\in\mathcal{Q}_v} \subseteq \{t(\mathbf{x})\}_{\mathbf{x}\in\mathcal{Q}_v}$. This way, we can rewrite our condition $\sum_{i\in[n]} \frac{1}{X+z_i} = \sum_{i\in[d]} \frac{\mu_i}{X+t_i}$ in the "sumcheck" world as follows:

$$\sum_{\mathbf{x}\in\mathcal{Q}_v} \sum_{i\in[m]} \frac{1}{\gamma + z_i(\mathbf{x})} = \sum_{\mathbf{x}\in\mathcal{Q}_v} \frac{\mu(\mathbf{x})}{\gamma + t(\mathbf{x})}$$

Here, $\mu(\mathbf{x})$ (if it is injective, which is typically the case) gives the number of occurancies of $t(\mathbf{x})$ in $z_1, \ldots, z_m$ altogether, i.e. $\mu(\mathbf{x}) = \sum_{i\in[m]} \#\{\mathbf{y} \in \mathcal{Q}_v : z_i(\mathbf{y}) = t(\mathbf{x})\}$.

Now, the idea of logup is to sample random $\gamma \leftarrow\!\!\$\ \mathbb{F}$ and apply the SumCheck on the function:

$$\zeta(\mathbf{x}) = \sum_{i\in[m]} \frac{1}{\gamma + \widetilde{z}_i(\mathbf{x})} - \frac{\widetilde{\mu}(\mathbf{x})}{\gamma + \widetilde{t}(\mathbf{x})},$$

where by $\widetilde{t}_i$ we denoted the multilinear extension, that is:

$$\widetilde{t}(\mathbf{x}) = \sum_{\mathbf{y}\in\mathcal{Q}_v} t(\mathbf{y})\mathsf{eq}(\mathbf{x};\mathbf{y}) = \frac{1}{2^v} \sum_{\mathbf{y}\in\mathcal{Q}_v} t(\mathbf{y}) \prod_{j=1}^{v}(1 + x_j y_j)$$

The only issue left is that sumcheck protocol cannot work with rational functions, which is the case here for $\zeta(\mathbf{x})$. For that reason, roughly, the prover $\mathcal{P}$ will divide the sum of m terms into ℓ chunks and provide multilinear helper functions for each such sum. Note that ℓ is chosen depending on the chosen polynomial commitment scheme.

Namely, suppose we split $[m] = \bigcup_{j\in[k]} \mathcal{I}_j$ into $k = \lceil m/\ell \rceil$ subintervals. Suppose

$$\zeta_j(\mathbf{x}) = \sum_{i\in\mathcal{I}_j} \frac{\mu_i(\mathbf{x})}{\phi_i(\mathbf{x})}, \quad j \in [k],$$

is the respective partial sum of consecutive terms in the overall expression $\frac{\mu(\mathbf{x})}{\gamma+\widetilde{t}(\mathbf{x})} - \sum_{i\in[m]} \frac{1}{\gamma+\widetilde{z}_i(\mathbf{x})}$. That is, we used the notation $\mu_0(\mathbf{x}) = \mu(\mathbf{x})$ and $\mu_i(\mathbf{x}) \equiv -1$ for $i > 0$. Similarly, $\phi_0(\mathbf{x}) = \gamma + \widetilde{t}(\mathbf{x})$ and $\phi_i(\mathbf{x}) = \gamma + \widetilde{z}_{i-1}(\mathbf{x})$ for $i > 0$.

Then, the prover provides oracles for $\{\zeta_i\}_{i\in[k]}$ subject to $\sum_{\mathbf{x}\in\mathcal{Q}_v} \sum_{i\in[k]} \zeta_i(\mathbf{x}) = 0$ and the domain identities:

$$\zeta_j(\mathbf{x}) \prod_{i\in\mathcal{I}_j} \phi_i(\mathbf{x}) = \sum_{i\in\mathcal{I}_j} \mu_i(\mathbf{x}) \prod_{k\in\mathcal{I}_j\setminus\{i\}} \phi_j(\mathbf{x})$$

These identities are finally checked using SumCheck protocol and then combined into a single one using random scalars $\lambda_0, \ldots, \lambda_{k-1} \leftarrow\!\!\$\ \mathbb{F}$.

22.3.4 Precise Scheme

Suppose the prover $\mathcal{P}$ wants to convince the verifier $\mathcal{V}$ that $\{z_i\}_{i\in[n]} \subseteq \{t_j\}_{j\in[d]}$. Assume $d = 2^v$.

Preprocessing stage. Compute multilinear extension $\widetilde{t} : \mathbb{F}^v \to \mathbb{F}$ that encodes the lookup table $\{t_j\}_{j\in[d]}$ and, if needed, multilinear Lagrange basis polynomials $\{\mathsf{eq}(\mathbf{x};\mathbf{y})\}_{\mathbf{y}\in\mathcal{Q}_v}$.

Protocol. Interaction between $\mathcal{P}$ and $\mathcal{V}$ proceeds as follows:

1. $\mathcal{P}$ divides witness into $k = \lceil n/d \rceil$ pieces and perceives values $\{z_i\}_{i\in[n]}$ as a set of m functions $z_j : \{\pm 1\}^v \to \mathbb{F}$. Additionally, $\mathcal{P}$ computes the multilinear extensions $\{\widetilde{z}_j(\mathbf{x})\}_{j\in[m]}$.
2. $\mathcal{P}$ computes the multiplicity function multilinear extension $\widetilde{\mu}$ and sends the oracle access to it $\mathcal{O}^{\mu}(\cdot)$ to the verifier $\mathcal{V}$.
3. $\mathcal{V}$ samples the challenge $\gamma \leftarrow\!\!\$\ \mathbb{F}$ and sends to $\mathcal{P}$.
4. $\mathcal{P}$ computes $k = \lceil m/\ell \rceil$ functions $\{\zeta_j(\mathbf{x})\}_{j\in[k]}$ according to constraints above and sends oracle access $\mathcal{O}^{\zeta_0}(\cdot), \mathcal{O}^{\zeta_1}(\cdot), \ldots, \mathcal{O}^{\zeta_{k-1}}(\cdot)$ to the verifier $\mathcal{V}$.
5. $\mathcal{V}$ responds with a random vector $\boldsymbol{\alpha} \leftarrow\!\!\$\ \mathbb{F}^n$ and random scalars $\lambda_0, \ldots, \lambda_{k-1} \leftarrow\!\!\$\ \mathbb{F}$. Now, both $\mathcal{P}$ and $\mathcal{V}$ engage in the sumcheck protocol for:

$$\sum_{\mathbf{x}\in\mathcal{Q}_v} G(\mathbf{x}, \mathsf{eq}(\mathbf{x};\boldsymbol{\alpha}), \mu(\mathbf{x}), \phi_0(\mathbf{x}), \ldots, \phi_{m-1}(\mathbf{x}), \zeta_0(\mathbf{x}), \ldots, \zeta_{k-1}(\mathbf{x})) = 0,$$

where the function G is defined as follows:

$$G(\mathbf{x}, \star) = \sum_{r \in [k]} \zeta_r(\mathbf{x}) + \mathsf{eq}(\mathbf{x}; \boldsymbol{\alpha}) \lambda_r \left(\zeta_r \prod_{i \in \mathcal{I}_r} \phi_i(\mathbf{x}) - \sum_{i \in \mathcal{I}_r} \mu_i(\mathbf{x}) \prod_{j \in \mathcal{I}_r \setminus \{i\}} \phi_j(\mathbf{x}) \right)$$

During sumcheck, $\mathcal{V}$ uses oracles provided by $\mathcal{P}$.

Lemma 22.13. The cost for running the protocol is $\mathcal{O}(n\ell)$ field multiplications, while the communication size is $\mathcal{O}(m/\ell)$ oracles of size $\mathcal{O}(d)$.