Improving logarithmic derivative lookups using GKR

Shahar Papini, Ulrich Haböck
spapini@starkware.co, uhaboeck@polygon.technology

September 18,[*] 2023

Abstract

In this informal note, we instantiate the Goldwasser-Kalai-Rothblum (GKR) protocol to prove fractional sumchecks as present in lookup arguments based on logarithmic derivatives, with the following impact on the prover cost of [Hab22]: When looking up $M \geq 1$ columns in a (for the sake of simplicity) single column table, the prover has to commit only to a *single* extra column, i.e. the multiplicities of the table entries. In order to carry over the GKR fractional sumcheck to the univariate setting, we furthermore introduce a simple, yet (as far as we know) novel transformation for turning a univariate polynomial commitment scheme into a multilinear one. The transformation complements existing approaches [ZXZS20, BCHO22, CBBZ22, Ham22, KT23] and might be of independent interest for its elegant way to prove arbitrary powers of the lexicographic shift over the Boolean hypercube.

Contents

1 Notation **1**

2 logUp in a nutshell **1**

3 GKR for fractional sumchecks **3**
3.1 The circuit 3
3.2 The GKR protocol 4
3.3 Computational cost 5

4 Application to logUp **6**

5 How to tackle the univariate case **8**
5.1 A simple univariate IOP for multilinear evaluation 8
5.2 Univariate logUp with GKR 10

[*]This revision corrects several typos in the previous eprint.

A Appendix **12**
A.1 The univariate constraints for the Lagrange 12
A.2 Proving the lexicographical shift on the hypercube 14

1 Notation

Throughout this note we assume that F is a finite field of characteristic larger than 2, i.e. $\mathsf{char}(F) > 2$, F^* its multiplicative group, and H_n denotes the n-dimensional Boolean hypercube $\{\pm 1\}^n$ as a subgroup of $(F^*)^n$. A multilinear polynomial in n variables is a polynomial $p(X_1, \ldots, X_n)$ over F with individual degrees at most 1, i.e. $\deg_{X_i} p(X_1, \ldots, X_n) \leq 1$ for all $i = 1, \ldots, n$. Whenever convenient we shall use vector notation for the variables, for example $p((X_1, \ldots, X_n))$ or $p((X_1, \ldots, X_k), (X_{k+1}, \ldots, X_n))$ for $p(X_1, \ldots, X_n)$. The Lagrange kernel for H_n is the $(2 \cdot n)$-variate multilinear polynomial

$$L_n(\vec{X}, \vec{Y}) = \frac{1}{2^n} \cdot \prod_{i=1}^{n} (1 + X_i \cdot Y_i),$$

with $\vec{X} = (X_1, \ldots, X_n)$ and $\vec{Y} = (Y_1, \ldots, Y_n)$. Taking $\vec{Y} = \vec{y}$ from the Boolean hypercube gives the Lagrange polynomial $L_n(\vec{X}, \vec{y})$ which is the unique multilinear polynomial equal to 1 if $\vec{X} = \vec{y}$, and equal to zero elsewhere over H_n.

For our purpose all multilinear polynomials $p(\vec{X})$ in n variables will be given in Lagrange representation, i.e. by their values over H_n. To emphasize this fact, we describe our protocols as what is called a *Lagrange interactive oracle proof* in [Hab22], in which the oracles are loaded with a set of values over H_n (the Lagrange representations), and later accessed via tensor queries (in other words, evaluation queries for the Lagrange representation).

2 logUp in a nutshell

In *logUp*, which is the current unofficial name for the lookup argument from [Hab22], the prover is given the values over H_n (the "witness columns") of $M \geq 1$ multilinear polynomials $w_1(\vec{X}), \ldots, w_M(\vec{X})$, and wishes to convince the verifier that these are all members of a predefined table given by the values of another multilinear polynomial $t(\vec{X})$ over H_n. (Without loss in generality, we assume that the values of t over H_n are all distinct.) For this, the prover provides the (Lagrange representation of the) multilinear polynomial $m(\vec{X})$, the values of which correspond to the overall multiplicities $m(\vec{x})$ of the table values $t(\vec{x})$ as they occur in the witness columns, i.e.

$$m(\vec{x}) = \sum_{i=1}^{M} \left| \{\vec{y} \in H_n : w_i(\vec{y}) = t(\vec{x})\} \right|,$$

for every $\vec{x} \in H_n$. In order to convince verifier upon the virtual identity

$$\prod_{i=1}^{M} \prod_{\vec{x} \in H_n} (X - w_i(\vec{x})) = \prod_{\vec{x} \in H_n} (X - t(\vec{x}))^{m(\vec{x})}, \tag{1}$$

the prover shows instead[1]the equality of their (formal) logarithmic derivatives,

$$\sum_{i=1}^{M} \sum_{\vec{x} \in H_n} \frac{1}{X - w_i(\vec{x})} = \sum_{\vec{x} \in H_n} \frac{m(\vec{x})}{X - t(\vec{x})}, \tag{2}$$

which is reduced to a sumcheck of rational terms by evaluating at a random $\alpha \leftarrow\!\!\$\; F$ provided by the verifier,

$$\sum_{\vec{x} \in H_n} \left(\frac{m(\vec{x})}{\alpha - t(\vec{x})} - \sum_{i=1}^{M} \frac{1}{\alpha - w_i(\vec{x})} \right) = 0. \tag{3}$$

The way logUp handles the rational sumcheck is by the common approach, which transforms it into a polynomial one: In the most elementary variant of the protocol, the prover provides additional *helper columns* for each of the fractional terms, corresponding to multilinear polynomials

$$h(\vec{X}) \text{ and } h_1(\vec{X}), \ldots, \vec{h}_M(\vec{X}),$$

which equal $\frac{1}{\alpha - t(\vec{x})}$ and $\frac{1}{\alpha - w_1(\vec{X})}, \ldots, \frac{1}{\alpha - w_M(\vec{X})}$ over H_n, respectively, and proves them correct using the corresponding domain identities. With these helper polynomials, the prover then shows that

$$\sum_{\vec{x} \in H_n} \left(m(\vec{x}) \cdot h(\vec{x}) - \sum_{i=1}^{M} h_i(\vec{x}) \right) = 0,$$

using the multivariate sumcheck protocol. For details, see [Hab22].

The main advantage of logarithmic derivatives (or, fractional decompositions) are their amenability for sumchecks. Sumchecks are more efficiently proven than grand products, and sumchecks can be efficiently combined with further sumchecks in the context of larger protocols. Even the above described elementary variant of logUp already improves over the multivariate variant of plookup [GW20] by essentially halving the number of additional columns the prover has to provide. Moreover, the performance can be optimized by a common trade-off between the number of helper functions and the algebraic degree of the sumcheck polynomial (c.f. [Hab22]). However, one can do significantly better: Inspired by Lasso [STW23] we take the Goldwassser-Kalai-Rothblum (GKR) protocol for directly proving the rational sumcheck (3) *without providing multilinear variants of the fractional expressions* (the helper functions).

[1]In order that the polynomial identity (1) and the fractional identity (2) are equivalent, one needs to demand that the total number of elements to be looked up does not exceed the characteristic of the field, $M \cdot |H_n| < \mathsf{char}(F)$. See [Hab22] for details.

3 GKR for fractional sumchecks

Given multilinear polynomials $p(\vec{X})$ and $q(\vec{X})$ in n variables over a finite field F, we show in this section how to use the Goldwasser-Kalai-Rothblum protocol [GKR08] to prove

$$\sum_{\vec{x} \in H_n} \frac{p(\vec{x})}{q(\vec{x})} = 0, \tag{4}$$

where H_n is the n-dimensional Boolean hypercube. We remark that everything in this section can be generalized in a straight-forward manner to higher-degree expressions of n-variate multilinear polynomials $w_1(\vec{X}), \ldots, w_M(\vec{X})$,

$$\sum_{\vec{x} \in H_n} \frac{P(w_1(\vec{x}), \ldots, w_M(\vec{x}))}{Q(w_1(\vec{x}), \ldots, w_M(\vec{x}))} = 0,$$

but for the sake of simplicity we keep with the more elementary variant given by Equation (4).

3.1 The circuit

We define a layered circuit for computing the cumulative sums of the fractions using *projective coordinates* for the additive group of the finite field F. The addition of two projective representations (a_0, b_0) and (a_1, b_1) from F^2 is

$$(a_0, b_0) +_F (a_1, b_1) = (a_0 \cdot b_1 + a_1 \cdot b_0, b_0 \cdot b_1),$$

where we write $+_F$ for not confusing it with an ordinary vector sum. In other words, we consider formal fractions over F, which are pairs $(a, b) \in F^2$ composed of a numerator a and a denominator b. For notational convenience, we consider our circuit built from gates with projective input and outputs, rather than a circuit with field-valued wire states. This leads to a somewhat non-standard presentation of the GKR protocol, but clarifies the recursively defined dependency between two adjacent layers of the circuit. The intermediate results of the circuit, which are throughout projective representations, are arranged along a binary tree of height n, reflecting the topology given by the hypercube. That is, the nodes at layer k of the tree, $1 \leq k \leq n$, correspond to the k-dimensional hypercube H_k, and the children of a $\vec{x} \in H_k$ correspond to $(\vec{x}, +1)$ and $(\vec{x}, -1)$ in H_{k+1} (using the simplified notation for appending ± 1 to $\vec{x}$). Layer $k = 0$ corresponds to the top node of the tree, having the children ± 1 from H_1. We consider H_0 as that single node set, with $\vec{x} \in H_0$ being the empty coordinate-vector.

The circuit is as follows. The inputs are supplied to the bottom layer, i.e. layer n, from which the values of the further layers, i.e. layer $n-1$, $n-2$, ..., are obtained step by step, until ending up with a projective representation of the overall sum (4) at layer 0.

- On layer n, we start with the values of the fractions $\frac{p(\vec{x})}{q(\vec{x})}$ in projective representation

$$(p_n(x), q_n(x)) = (p(\vec{x}), q(\vec{x})), \quad \vec{x} \in H_n.$$

- On level k, $0 \leq k < n$, for each node $\vec{x}$ in H_k the projective representation $(p_k(\vec{x}), q_k(\vec{x}))$ is the sum of the projective representations carried by its children nodes $(\vec{x}, +1)$ and $(\vec{x}, -1)$ from H_{k+1}, i.e.

$$\begin{aligned} p_k(\vec{x}) &= (p_{k+1}(\vec{x}, +1) \cdot q_{k+1}(\vec{x}, -1) + p_{k+1}(\vec{x}, -1) \cdot q_{k+1}(\vec{x}, +1), \\ q_k(\vec{x}) &= q_{k+1}(\vec{x}, +1) \cdot q_{k+1}(\vec{x}, -1). \end{aligned}$$

In this manner, in each layer k the nodes $\vec{x} \in H_k$ carry a projective representation $(p_k(\vec{x}), q_k(\vec{x}))$ for the cumulative sum of the fractions "below", i.e.

$$\frac{p_k(\vec{x})}{q_k(\vec{x})} = \sum_{\vec{y} \in H_{n-k}} \frac{p(\vec{x}, \vec{y})}{q(\vec{x}, \vec{y})}.$$

(Using again the simplified notation $(\vec{x}, \vec{y})$ for the appended vector $\vec{x} \| \vec{y}$.) In particular, at top layer 0 of the tree we obtain the projective representation (p_0, q_0) of the overall sum

$$\frac{p_0}{q_0} = \sum_{\vec{y} \in H_n} \frac{p(\vec{y})}{q(\vec{y})}.$$

3.2 The GKR protocol

We do a minor variation of the GKR protocol to serve our design decision for the circuit, having gates with inputs and outputs being throughout projective representations.[2] In the first round $k = 0$, the claims for

$$p_1(+1), q_1(+1), p_1(-1), q_1(-1)$$

are reduced by a random $\mu_0 \leftarrow\!\!\$\; F$ from the verifier to a "single-point" claim on

$$p_1(r_0), q_1(r_0),$$

where $r_0 = 1 - \mu_0$, using linearity of p_1 and q_1.

In each further layer k, $1 \leq k \leq n-1$, the k-variate multilinear polynomials p_k and q_k depend on p_{k+1} and q_{k+1} from the next layer by the two linear relations

$$\begin{aligned} p_k(\vec{X}) &= \sum_{\vec{y} \in H_k} L_k(\vec{X}, \vec{y}) \cdot \Big(p_{k+1}(\vec{y}, +1) \cdot q_{k+1}(\vec{y}, -1) + p_{k+1}(\vec{y}, -1) \cdot q_{k+1}(\vec{y}, +1)\Big), \\ q_k(\vec{X}) &= \sum_{\vec{y} \in H_k} L_k(\vec{X}, \vec{y}) \cdot q_{k+1}(\vec{y}, +1) \cdot q_{k+1}(\vec{y}, -1). \end{aligned}$$

Taking the claims on p_k and q_k at the random point $\vec{r}_k$ from the previous layer, the two sumchecks for $p_k(\vec{r}_k)$ and $q_k(\vec{r}_k)$ are combined into a single sumcheck

[2]Again, we point out that this non-standard presentation of GKR, having two separate polynomials p_k and q_k at each layer, is for notational convenience only. The entire circuit can be written using a single multilinear polynomial v_k per layer, which combines the two polynomials as $v_k(\vec{x}, 1) = \frac{1+y}{2} \cdot p_k(\vec{x}) + \frac{1-y}{2} \cdot q_k(\vec{x})$, e.g.

for $p_k(\vec{r}_k) + \lambda_k \cdot q_k(\vec{r}_k)$, using a further randomness $\lambda_k \leftarrow\$ F$, which is then reduced by the sumcheck protocol to evaluation claims for

$$p_{k+1}(\vec{\rho}_k, +1), q_{k+1}(\vec{\rho}_k, +1), p_{k+1}(\vec{\rho}_k, -1), q_{k+1}(\vec{\rho}_k, -1),$$

with $\vec{\rho}_k \in F^k$ being the randomnesses sampled in the course of the protocol. Like in the first round, these two-point claims are combined by means of a further random $\mu_k \leftarrow\$ F$ into the single-point claim for

$$p_{k+1}(\vec{r}_{k+1}), q_{k+1}(\vec{r}_{k+1}),$$

with $\vec{r}_{k+1} = (\vec{\rho}_k, 1 - \mu_k)$.

After the last round $k = n - 1$, the initial evaluation claims for p_1 and q_1 at ± 1 are reduced to evaluation claims of $p_n(\vec{X}) = p(\vec{X})$ and $q_n(\vec{X}) = q(\vec{X})$ at the random point $\vec{r}_n$ obtained in that round. If the latter claims are proven true, and if

$$p_1(+1) \cdot q_1(-1) + p_1(-1) \cdot q_1(+1) = 0,$$

as well as

$$q_1(+1) \cdot q_1(-1) \neq 0,$$

then the verifier accepts that $\sum_{\vec{x} \in H_n} \frac{p(\vec{x})}{q(\vec{x})} = 0$.

Taking the soundness errors of the batching steps involving the μ_k, $k = 0, \ldots, n-1$ and the λ_k, $k = 1, \ldots, n-1$, into account, the overall soundness error is as follows.

Proposition 1. *The soundness error ε_{GKR} of the above described GKR protocol for proving that $\sum_{\vec{x} \in H_n} \frac{p(\vec{x})}{q(\vec{x})} = 0$ is bounded by*

$$\varepsilon_{GKR} \leq \frac{2 \cdot (n-1) + 1}{|F|} + \sum_{k=1}^{n-1} \varepsilon_{Sumcheck}(|H_k|) \leq \frac{1}{2} \cdot \frac{n \cdot (3 \cdot n + 1)}{|F|},$$

using $\varepsilon_{Sumcheck}(|H_k|) \leq \frac{3 \cdot k}{|F|}$ for the soundness error of the sumcheck protocol for a degree $d = 3$ expression over the Boolean hypercube of size $|H_k|$.

We will provide a formal security analysis of the protocol in another document.

3.3 Computational cost

Let us count the number of field operations involved in the GKR fractional sumcheck. The sumcheck in round k, $1 \leq k \leq n-1$, is subject to the degree $d = 3$ expression

$$\begin{aligned} &p_k(\vec{r}_k) + \lambda_k \cdot q_k(\vec{r}_k) \\ &\quad = \sum_{\vec{y} \in H_k} Q\Big(L_k(r_k, \vec{y}), p_{k+1}(\vec{y}, +1), p_{k+1}(\vec{y}, -1) q_{k+1}(\vec{y}, +1), q_{k+1}(\vec{y}, -1)\Big), \end{aligned}$$

with

$$Q(L, p_{+1}, p_{-1}, q_{+1}, q_{-1}) = L \cdot \Big(p_{+1} \cdot q_{-1} + p_{-1} \cdot q_{+1} + \lambda_k \cdot q_{+1} \cdot q_{-1}\Big).$$

The value of Q as a polynomial in $\nu = 5$ variables, can be computed within $|Q|_{\mathsf{M}} = 5$ multiplications, and $|Q|_{\mathsf{A}} = 2$ additions. Using the approximate formula (12) from [Hab22], the prover cost of the sumcheck is about

$$|H_k| \cdot (d+1) \cdot \big((\nu + |Q|_{\mathsf{M}}) \cdot \mathsf{M} + (\nu + |Q|_{\mathsf{A}}) \cdot \mathsf{A}\big) = |H_k| \cdot (40 \cdot \mathsf{M} + 28 \cdot \mathsf{A}),$$

given the values of p_{k+1}, q_{k+1} over H_{k+1}, and of $L_k(\,.\,, \vec{r}_k)$ over H_k. (Here, and in the sequel, A and M denote field additions and multiplications.) Summing over $1 \leq k \leq n-1$ yields about

$$|H_n| \cdot (40 \cdot \mathsf{M} + 28 \cdot \mathsf{A}),$$

and together with cost for computing the values of all intermediate values of the circuit, which is about

$$|H_n| \cdot (3 \cdot \mathsf{M} + 1 \cdot \mathsf{A}),$$

we end up with the total cost of

$$|H_n| \cdot (43 \cdot \mathsf{M} + 29 \cdot \mathsf{A}) \tag{5}$$

for the entire GKR fractional sumcheck over H_n.

4 Application to logUp

For simplicity we assume that the number of columns subject to the lookup argument is equal to $M = 2^k - 1$, so that we have in total 2^k columns of length 2^n, including the table column. To prove the fractional sumcheck Equation (3), one simply applies the GKR protocol as described in Section 3 to the combined multilinear polynomial in $n + k$ variables,

$$\begin{aligned} p(\vec{X}, \vec{Y}) &= L_k(\vec{Y}, \vec{1}) \cdot m(\vec{X}) - \sum_{\vec{y} \in H_k \setminus \{\vec{1}\}} L_k(\vec{Y}, \vec{y}) \cdot 1, \\ q(\vec{X}, \vec{Y}) &= L_k(\vec{Y}, \vec{1}) \cdot (\alpha - t(\vec{X})) + \sum_{\vec{y} \in H_k \setminus \{\vec{1}\}} L_k(\vec{Y}, \vec{y}) \cdot (\alpha - w_{i(\vec{y})}(\vec{X})), \end{aligned}$$

where $\vec{1} = (1, \ldots, 1)$, and $i(\vec{y})$, $\vec{y} \in H_k \setminus \{\vec{1}\}$, is an enumeration of $\{1, \ldots, M\}$. This reduces the sumcheck claim to evaluation claims for $t(\vec{X})$, $m(\vec{X})$ and $w_1(\vec{X}), \ldots, w_M(\vec{X})$ at a random point $\vec{r} \in F^n$. The verifier queries the oracles for their values at $\vec{r}$, and accepts if the answers are consistent with the claims.

In regards of the soundness error, there is as good as no change. The soundness error of reducing the "virtual" fractional identity (2) to the fractional sumcheck (3) still dominates.

Proposition 2 (logUp-GKR Soundness)**.** *Let $M \geq 1$ be an integer, F a finite field with characteristic $\mathsf{char}(F) > M \cdot 2^n$. The soundness error ε for logUp on M witness columns and a single table column over H_n, using GKR for the fractional sumcheck, is bounded by*

$$\varepsilon \leq \frac{(M+1) \cdot 2^n - 1}{|F| - |H|} + \varepsilon_{GKR}(|H_{n+m}|),$$

where $\varepsilon_{GKR}(|H_{n+m}|)$ is the soundness error of the GKR protocol over a domain of size $|H_{n+m}|$, and $m = \lceil \log_2(M+1) \rceil$.

As pointed out in Section 2, the constraint on the number of elements to be looked up is essential for the uniqueness of fractional decompositions. The first term of the soundness error bound is due to the above mentioned reduction from fractional identity to sumcheck, whereas for completness of the protocol the random point needs to be taken from $F \backslash H$. We will give a formal description of the protocol, including its security analysis in another document.

Let us estimate the performance impact of using GKR for fractional sumchecks. In our proposed variant of logUp, the prover only needs to provide a single additional column, the one for the multiplicities as recorded by $m(\vec{X})$. According to Equation (5), the computational cost for the rational sumcheck is about

$$|H_{n+m}| \cdot (43 \cdot \mathsf{M} + 29 \cdot \mathsf{A}), \tag{6}$$

which amounts to $43 \cdot \mathsf{M} + 29 \cdot \mathsf{A}$ per element to be looked up (for large numbers of witness columns, so that we can neglect the table size). This is only about the double of the arithmetic costs for the elementary variant of logUp described in Section 2, which is about

$$|H_{n+m}| \cdot (19 \cdot \mathsf{M} + 16 \cdot \mathsf{A}), \tag{7}$$

the commitment effort for the $(M+1)$ helper functions left aside. (The arithmetic costs are about the half of (7), when combining the sumcheck with others.) Depending on the used commitment scheme, we expect the following impact on the prover costs:

- Elliptic curve based schemes consume an equivalent of several hundreds multiplications and additions per committed field element (see Table 1 in [Hab22]), yielding an at least 10-fold advantage,
- For Reed-Solomon code based schemes the difference from (7) (or the half of it) to (6) is already largely consumed by the encoding alone. Depending on the concrete hash function, we expect the advantage ranging from about 2-fold up to 10-fold.

We are curious of how well this operation count based forecast will be met in practice.

5 How to tackle the univariate case

In order to apply the fractional sumcheck GKR to univariate logUp, we introduce a simple univariate IOP for turning a univariate polynomial commitment scheme into a multilinear one. Contrary to the existing approaches [ZXZS20, BCHO22, CBBZ22, Ham22] and most recently [KT23], our transform works throughout with Lagrange representations (multilinear and univariate ones), i.e. we identify values on the hypercube with values over the univariate domain. This leads to a very elementary transformation which might be also of interest in other applications. (One application is complementary to [Ham22, KT23] and given in Appendix A.2.)

5.1 A simple univariate IOP for multilinear evaluation

As in the above mentioned track of work, we identify the univariate cyclic domain $H = \{x \in F : x^{2^n} = 1\}$ with the Boolean hypercube $H_n = \{\pm 1\}^n$, using the bit decomposition of the indices

$$i = i_0 + i_1 \cdot 2 + \ldots + i_{n-1} \cdot 2^{n-1},$$

which yields the bijective map $\iota : H \to H_n$ according to the following commutative diagram. (Here, and in the sequel, g is a generator of the cyclic domain.)

$$\begin{tikzcd}
\{0,1\}^n \arrow[r, "i_0+i_1\cdot 2+\ldots+i_{n-1}\cdot 2^{n-1}"] \arrow[d, "((-1)^{i_0},\ldots,(-1)^{i_{n-1}})"'] & {[0,2^n)} \arrow[d, "g^i"] \\
H_n = \{\pm 1\}^n & H \arrow[l, "\iota"']
\end{tikzcd}$$

We recall that the map ι is not a homomorphism between the two groups.

By means of ι we identify values over the Boolean hypercube as values over the univariate domain. In terms of Lagrange representations, given

$$f(\vec{X}) = \sum_{\vec{x} \in H_n} v(\vec{x}) \cdot L_n(\vec{X}, \vec{x}),$$

we commit to the univariate Lagrange representation over H given by $v \circ \iota$, which corresponds to the unique polynomial

$$u_f(X)$$

of degree $\deg u_f(X) \leq 2^n - 1$ such that

$$v_i = u_f(g^i) = v\left((-1)^{i_0}, \ldots, (-1)^{i_{n-1}}\right),$$

for every index $i \in [0, 2^n)$, where $(i_0, \ldots, i_{n-1})$ are the binary digits of it.

The evaluation proof for f at a multivariate query point $\vec{t} = (t_0, \ldots, t_{n-1})$ from $(F \setminus \{-1\})^n$ is as follows.[3] As [ZXZS20] we represent f at $\vec{t}$ as inner product, which in our case is between the values given by v and the column c for the values of the multivariate Lagrangians $L_n(\vec{t}, \vec{x})$ over H_n, i.e.

$$f(t_0, \ldots, t_{n-1}) = \sum_{i=0}^{2^n - 1} v_i \cdot c_i, \tag{8}$$

where

$$c_i = \frac{1}{2^n} \cdot \prod_{k=0}^{n-1} (1 + (-1)^{i_k} \cdot t_k). \tag{9}$$

In the first round the prover provides a Lagrange oracle for the unique univariate polynomial

$$c(X)$$

of degree $\deg c(X) \leq 2^n - 1$, which extends the values of the column $c = (c_i)$ (i.e. $c(g^i) = c_i$). Correctness of the additional oracle $c(X)$ is directly enforced by the following periodic constraints over H,

$$c_0 = \frac{1}{2^n} \cdot (1 + t_0) \cdot \ldots \cdot (1 + t_{n-1}), \tag{10}$$

and

$$c_{i + 2^{k-1}} = c_i \cdot \frac{1 - t_{k-1}}{1 + t_{k-1}}, \quad i \in 2^k \cdot \mathbb{Z} \cap [0, 2^n), \tag{11}$$

for $k = 1, \ldots, n$. (These constraints yield quadratic identities over H, and their selector polynomials have a succinct rational representation. See Section A.1.) It then runs the univariate sumcheck protocol for

$$f(t_0, \ldots, t_{n-1}) = \sum_{x \in H} u_f(x) \cdot c(x), \tag{12}$$

leading to another quadratic identity over H. We assume that the reader is familiar with univariate sumcheck techniques (see [BSCR+19, CHM+20] or [HGdB21]), and skip an explicit description of the complete IOP.

Proposition 3. *Assume that F is a finite field with a two-adic multiplicative subgroup H of order 2^n. The above sketched protocol defines a univariate polynomial IOP for proving the value $f(\vec{t_0}, \ldots, \vec{t}_{n-1})$ of the unique multilinear polynomial $f(\vec{X})$, which extends the set of values over H_n given by $u_f(X)$ over H via $u_f \circ \iota^{-1}$. Its soundness error ε is bounded by*

$$\varepsilon \leq \frac{2 \cdot d_{max}}{|F|} + \varepsilon_{sumcheck},$$

where d_{max} is the maximum degree of oracle polynomials, and $\varepsilon_{sumcheck}$ is the soundness error of the univariate sumcheck argument.

[3]We remark that the assumption on the coordinates can be dropped by a more careful setup of the constraints. However, in most applications (as ours) the query point is, or can be made random, with a sampling set that avoids the case -1.

The protocol is zero-knowledge if the used sumcheck argument is, and is easily extended to a batch evaluation proof, proving the values of several multilinear polynomials $f_1(\vec{X}), \ldots, f_M(\vec{X})$ at given points $\vec{t}_1, \ldots, \vec{t}_M$, respectively. We will provide a formal security analysis in another document.

Remark 1. In a similar manner one can modify the transformation from [ZXZS20] into one with a succinct verifier. Given a multilinear polynomial with respect to the monomial basis, $f(\vec{X}) = \sum c_{i_0,\ldots,i_{n-1}} \cdot X_0^{i_0} \cdot \ldots \cdot X_{i_{n-1}}^{i_{n-1}}$, the prover simply provides the univariate oracle $w(X)$ for the column w of the monomial values at $\vec{t}$,

$$w_i = t_1^{i_0} \cdot \ldots \cdot t_{n-1}^{i_{n-1}},$$

for $i = 0, \ldots, 2^n - 1$, again with $(i_0, \ldots, i_{n-1})$ being the bit representation of the index i. Correctness of $w(X)$ is enforced by the periodic constraints $w_0 = 1$, and

$$w_{i+2^{k-1}} = w_i \cdot t_k, \quad i \in 2^k \cdot \mathbb{Z} \cap [0, 2^n),$$

for $k = 1, \ldots, n$. The approach can be generalized to multivariate polynomial of larger individual degree, but at the cost of having non-periodic constraints instead (unless the field has multiplicative subgroup of suitable order). These non-periodic constraints can be handled in the style of Plonk [GWC19] via precomputed selector polynomials.

5.2 Univariate logUp with GKR

For simplicity we again assume a single-colum table t, and that we have $M = 2^k - 1$, $k \geq 1$, witness columns $w_1, \ldots, w_M(X)$ subject to the lookup.

Regarding values over the univariate domain H as values over the Boolean hypercube $\{\pm 1\}^n$, as in Section 5.1, we run the GKR protocol for the fractional sumcheck as described in Section 4. The GKR protocol eventually ends up with evaluation claims for the multilinear extensions $t(\vec{X})$ and $w_1(\vec{X}), \ldots, w_M(\vec{X})$, and $m(\vec{X})$ at a random point $\vec{r} = (r_0, \ldots, r_{n-1})$,

$$\begin{gathered} t(\vec{r}) = v_t, \quad m(\vec{r}) = v_m, \\ w_i(\vec{r}) = v_i, \quad i = 1, \ldots, M, \end{gathered}$$

which are then reduced to a single evaluation claim on a random linear combination,

$$t(\vec{X}) + \lambda \cdot m(\vec{X}) + \sum_{i=1}^{M} \lambda^{i+1} \cdot w_i(\vec{X}),$$

at $\vec{X} = \vec{r}$, using $\lambda \leftarrow\!\!\$\ F$. This claim is proven by the univariate IOP from Section 5.1.

The soundness error ε for this protocol variant, which we call univariate logUp/GKR, increases at most by the soundness error of the PCS transformation,

$$\varepsilon \leq \varepsilon_{MVlogUp} + \varepsilon_{trafo},$$

where $\varepsilon_{MVlogUp}$ is the soundness error of the multivariate logUp/GKR protocol (Proposition 1), and ε_{trafo} is the soundness error of our univariate IOP for multilinear evaluation (Proposition 3). Again, we postpone a formal security analysis to another document.

Acknowledgements

We would like to thank Ariel Gabizon, Adrian Hamelink, Justin Thaler, and Jeroen Zuiddam for their feedback. In addition to that, thanks to Sai Deng for pointing out several typos.

References

[BCHO22] Jonathan Bootle, Alessandro Chiesa, Yuncong Hu, and Michele Orr'u. Gemini: Elastic SNARKs for diverse environments. In *EUROCRYPT 2022*, 2022. Full paper: `https://eprint.iacr.org/2022/420`.

[BSBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. In *IACR ePrint Archive 2018/046*, 2018. `https://eprint.iacr.org/2018/046`.

[BSCR+19] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In Y. Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019*, volume 11476 of *LNCS*. Springer, 2019.

[BSGKS20] Eli Ben-Sasson, Lior Goldberg, Swastik Kopparty, and Shubhangi Saraf. DEEP-FRI: Sampling outside the box improves soundness. In *ITCS 2020*, 2020. Full paper: `https://eprint.iacr.org/2019/336`.

[CBBZ22] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. Hyperplonk: PLONK with a linear-time prover and high-degree costum gates. In *IACR ePrint Archive 2020/1355*, 2022. `https://eprint.iacr.org/2022/1355`.

[CHM+20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In *EUROCRYPT 2020*, volume 12105 of *LNCS*, 2020. Full paper: `https://eprint.iacr.org/2019/1047.pdf`.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for muggles. In *STOC'08*, 2008. https://eprint.iacr.org/2022/1056.

[GW20] Ariel Gabizon and Zachary J. Williamson. Plookup: A simplified polynomial protocol for lookup tables. In *IACR ePrint Archive 2020/315*, 2020. https://eprint.iacr.org/2020/315.

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over Lagrange-bases for oecumenical non-interactive arguments of knowledge. In *IACR ePrint Archive 2019/953*, 2019. https://eprint.iacr.org/2019/953.

[Hab22] Ulrich Haböck. Multivariate lookups based on logarithmic derivatives. In *IACR ePrint Archive 2022/1530*, 2022. https://eprint.iacr.org/2022/1530.

[Ham22] Adrian Hamelink. Gemini. hackmd.io, 2022. https://hackmd.io/@adrian-aztec/BJxoyeCqj.

[HGdB21] Ulrich Haböck, Alberto Garoffolo, and Daniele di Benedetto. Darlin: Recursive proofs based on Marlin. In *IACR preprint archive 2021/930*, 2021. https://eprint.iacr.org/2021/930.

[KT23] Tohru Kohrita and Patrick Towa. Zeromorph: Zero-knowledge multilinear-evaluation proofs from homomorphic univariate commitments. In *IACR preprint archive 2023/917*, 2023. https://eprint.iacr.org/2023/917.

[Sta21] StarkWare Team. ethSTARK documentation – version 1.1. In *IACR preprint archive 2021/582*, 2021. https://eprint.iacr.org/2021/582.

[STW23] Srinath Setty, Justin Thaler, and Riad Wahby. Unlocking the lookup singularity with Lasso. In *IACR ePrint Archive 2023/1216*, 2023. https://eprint.iacr.org/2023/1216.

[ZXZS20] Jiaheng Zhang, Tianchen Xie, Yupeng Zhang, and Dawn Song. Transparent polynomial delegation and its applications to zero knowledge proof. In *IEEE S&P 2020*, 2020. Full paper: https://eprint.iacr.org/2019/1482.

A Appendix

A.1 The univariate constraints for the Lagrange

Let $c(X)$ be the univariate oracle for the Lagrange column $c_i = L_n(\vec{t}, \iota(g^i))$, $i = 0, \ldots, 2^n - 1$, as provided by the prover in Section 5.1. The complete set of constraints are as follows. We take the non-normalized quotients

$$s_k(X) = \frac{v_{H_n}(X)}{v_{H_k}(X)} = \frac{X^{2^n} - 1}{X^{2^k} - 1},$$

as selector polynomials for the periodic constraints (10) and (11). The selector $s_k(X)$ is zero over H, except for points in the subgroup H_k of the (2^k)-th roots of unity, generated by $g^{2^{n-k}}$. The polynomial form of (10) is

$$s_0(X) \cdot \left(c(X) - \frac{1}{2^n} \cdot (1 + t_0) \cdot \ldots \cdot (1 + t_{n-1}) \right) = 0, \tag{13}$$

and for (11), $k = 1, \ldots, n$, we have

$$s_{k-1}(X) \cdot \Big((1 - t_{n-k}) \cdot c(X) + (1 + t_{n-k}) \cdot c(g^{2^{n-k}} \cdot X) \Big) = 0, \tag{14}$$

all identities over H, i.e. modulo $v_H(X)$. Recall that $\vec{t} \in (F \setminus \{-1\})^n$, so all coordinates of $\vec{t}$ are different from -1, so that the coefficient $(1 + t_{n-k})$ in (14) is non-zero for every k.

Let us dwell on the necessity and sufficiency of the constraints. First of all, notice that the domain points are subject to a varying number of transitional constraints, depending in which of the subgroups H_k, $k = 0, \ldots n-1$, they are contained. Since these form an increasing chain

$$\{1\} = H_0 \subset H_1 \subset \ldots \subset H_{n-1} \subset H_n,$$

a point $x = g^i$ that is in $H_{k'}$ but not in $H_{k'-1}$, corresponding to an index i which is an odd multiple of $2^{n-k'}$, has the constraints (14) active only for $k > k'$. This means that x is constrained to index transitions of smaller step size only, i.e. those with offset 2^{n-k}, $k > k'$, each of which leads into a larger (or "finer") subgroup (for offset 2^{n-k} we end up in H_k). The constraint (13) enforces the correct Lagrange value at index $i = 0$. Since $\vec{t} \in (F \setminus \{-1\})^n$, each of the active constraints from (14) enforces the proper replacement quotient in the Lagrange product, when flipping the corresponding bit from zero to one. This shows that the constraints are necessary. To see that this they are sufficient, consider the following path of subsequent transitions: We start at the trivial index with bit representation $(0, \ldots, 0)$, and successively add to the index the non-zero bits of our target index i, starting with the most significant one, and going down to the least significant. The value at $(0, \ldots, 0)$ is enforced correct by the constraint (10). In each transition step, corresponding to a non-zero bit $i_{n-k} \neq 0$, we leave the subgroup from the last step, corresponding to some $H_{k'}$ with $k' < k$, and enter the subgroup H_k. By the preceeding discussion, the constraint (11) is active and enforces the proper quotient for the transition triggered by i_{n-k}. Since the target index i was arbitrary, the proof is complete.

A.2 Proving the lexicographical shift on the hypercube

The univariate IOP for multilinear evaluation from Section 5.1 can be easily extended to prove openings of functions under the lexicographical shift on the Boolean hypercube more elegantly as in [Ham22] and [KT23].

The lexicographical shift T on the Boolean hypercube $H_n = \{\pm 1\}^n$ is the circular rotation induced by the lexicographical order on H_n. It sends each element $\vec{x}$ to its upper neighbor with respect to the lexicographical order, except the largest element $-\vec{1} = (-1, \ldots, -1)$, which is mapped to the smallest element $\vec{1} = (+1, \ldots, +1)$. Assuming that F admits a multiplicative subgroup H of order 2^n, and using the identification $\iota : H \mapsto H_n$ from Section 5.1, where g is a generator of H, the lexicographical shift translates to the multiplication by g over the univariate domain H,

$$T = \iota \circ T_g \circ \iota^{-1},$$

where $T_g : x \mapsto g \cdot x$. More generally the k-th power T^k translates to T_g^k, which corresponds to the multiplication by g^k. Thus, for proving the multilinear opening of the shift of a function f over H_n at a point $\vec{t} \in F^n$,

$$\langle f \circ T, L_n(\,.\,, \vec{t}) \rangle = v,$$

the protocol simply uses $u_f(g \cdot X)$ in the univarate sumcheck for the inner product with the Lagrange column c for $\vec{t}$, i.e.

$$(f \circ T)(t_0, \ldots, t_{n-1}) = \sum_{x \in H} u_f(g \cdot x) \cdot c(x).$$

This protocol is easily extended to opening higher order shifts at the same point,

$$(f \circ T^k)(t_0, \ldots, t_{n-1}) = \sum_{x \in H} u_f(g^k \cdot x) \cdot c(x),$$

and across multiple polynomials, which is the typical situation that arises in multivariate variants of AIR [BSBHR18, BSGKS20, Sta21], and Plonk [GWC19].

We note that unlike [Ham22] and [KT23], in which values over the hypercube are identified as coefficients of the univariate polynomial, no degree proofs are needed, and no elements over the trace domain need to be kept free. To the best of our knowledge the above described evaluation proof is the first solution that does not have any restriction on the power of the shift. We will provide a multivariate protocol for efficiently proving arbitrary powers of the lexicographical shift in another document.