Notes on Univariate Sumcheck

Malcom Mohamed
Ruhr-Universität Bochum
`malcom.mohamed@rub.de`

Draft, 02 May 2025

Abstract

These notes describe an adaptation of the multivariate sumcheck protocol to univariate polynomials interpolated over roots of unity.

Contents

1 Introduction

Modern proof systems like SNARKs first encode the to-be-proven computations using polynomials over finite fields and then use algebraic techniques to prove statements about such polynomials. In this context, these notes present information-theoretic interactive reduction protocols from generalized univariate dot products $\sum_i g(f_1(i), \ldots, f_q(i))$ to univariate evaluations $f_1(r), \ldots, f_q(r)$. For the most part, we recover results from Gemini [3] which contains a 2-phase protocol for the same task. Like Gemini's first phase, our first protocol reduces univariate claims to *multilinear* evaluations. However, we use the Lagrange basis throughout and avoid monomial-to-Lagrange basis conversions. Similarly to Gemini's second phase, our second protocol reduces multilinear evaluation to univariate evaluation. Interestingly, our second protocol only takes

1

a single round of interaction. Further, both protocols follow from the same generic template which uses an adaptation of the multivariate sumcheck protocol that "directly" applies to univariates.

It is intended for this article's full version to describe further implications and extensions of our approach.

2 Preliminaries

2.1 Notations

$[a]$ denotes the set of intergers $\{1,\ldots,a\}$. $[a,b]$ denotes the set of integers $\{a,\ldots,b\}$. $\mathbb{F}$ denotes a finite field. $H = \left\{ w^i \mid i \in [0, 2^m - 1] \right\}$ denotes the multiplicative subgroup of 2^mth roots of unity in $\mathbb{F}$, with w being the primitive element.

If p is a univariate polynomial in x, then $\deg(p)$ denotes the degree of p and $p[i]$ denotes the coefficient before x^i. If p is a multivariate polynomial in $x_1,\ldots,x_m$, then $\deg_j(p)$ denotes the degree of p in x_j and $p[i]$ denotes the coefficient before $x_1^{i_1}\ldots x_m^{i_m}$. Here, i is the unique integer with (mixed-radix) digit representation $(i_1,\ldots,i_m)$ for radices $\deg_1(p)+1,\ldots,\deg_m(p)+1$—the least significant digit being i_1. mlin denotes the (inverse) Kronecker substitution that re-interprets $f \in \mathbb{F}[x]$ of degree at most $2^m - 1$ as the multilinear polynomial with the same coefficient vector, that is, $\mathsf{mlin}(f) = \sum_{i_1,\ldots,i_m\in\{0,1\}} f[i]x_1^{i_1}\cdots x_m^{i_m}$. $f\%p$ denotes the remainder of f modulo p.

2.2 Polynomial Interactive Oracle Proofs and Reductions

In brief, *polynomial interactive oracle proofs (PIOPs)* are two-party protocols between a prover and a verifier which are defined for a given NP relation and a mathematical field with a polynomial ring. The prover's input is an instance-witness pair of the NP relation, while the verifier starts only with the instance and must output a bit. We think of the prover as making the *claim* that there exists a valid witness for the given instance. The messages sent between prover and verifier are field elements or so-called polynomial oracles, idealized objects which provide black-box query access to polynomial functions. The core proerties of a PIOP are completeness and soundness. By completeness, a prover with a valid instance-witness pair in its input causes the verifier to output 1. By soundness, no algorithm *without* a valid instance-witness pair in its input is able to play the prover's role in the protocol and cause the verifier to output 1 except with negligible probability.

Related to the notion of a PIOP is that of a *reduction*, by which we mean a protocol where the verifier outputs an instance of another NP relation and the prover outputs an instance-witness pair. Here, completeness means that a valid first instance-witness pair leads to a valid second instance-witness pair, while soundness means that if the first instance has no valid witness, neither will the second. Further background and formal definitions related to PIOPs and reductions may be found in [7].

The multivariate sumcheck protocol. We describe this article's main PIOP of concern, the multivariate sumcheck protocol. While its history dates back to [9], our fram-

ing as a (reduction-based) PIOP is influenced by recent works like [4, 7]. The protocol uses the following reduction, where P is an m-variate polynomial (for which the verifier already has an oracle), s is a field element and $H_1, \ldots, H_m$ are sets:

Claim: $\sum_{i_1 \in H_1, \ldots, i_m \in H_m} P(i_1, \ldots, i_m) = s$.

- The prover sends $p_1(y) = \sum_{i_2 \in H_2, \ldots, i_m \in H_m} P(y, i_2, \ldots, i_m)$, either explicitly or as an oracle.

- The verifier checks that $\sum_{i_1 \in H_1} p_1(i_1) = s$.

- The verifier sends r_1, chosen at random.

- Both prover and verifier set $s' = p_1(r_1)$.

New claim: $\sum_{i_2 \in H_2, \ldots, i_m \in H_m} P(r_1, i_2, \ldots, i_m) = s'$.

The idea of the sumcheck protocol is to handle the *new claim* with a recursive invocation of the reduction. After m rounds, the claim will have been reduced to $P(r_1, \ldots, r_m) = s'$. This claim is then checked with a single query to the P oracle. In such a recursive execution, we will denote the verifier's randomness from the jth recursive call as r_j and we write the round polynomial as p_j. In full, this means

$$p_j(y) = \sum_{i_{j+1} \in H_{j+1}, \ldots, i_m \in H_m} P(r_1, \ldots, r_{j-1}, y, i_{j+1}, \ldots, i_m).$$

3 Univariate Sumcheck Protocols

This section first describes an inefficient generic univariate sumcheck reduction, followed by efficient specialized variants. One variant avoids intermediary oracles, however, turning the outcome of the reduction into a multilinear evaluation claim. Another variant restricts the protocol to multilinear evaluations and uses intermediary oracles.

3.1 Inefficient Generic Protocol

The starting idea for the new univariate protocol is to use the *multivariate* sumcheck protocol on suitably defined multivariate polynomials such that **(1)** the sum of the multivariates' values is the same as the univariates' and **(2)** the multivariates can be evaluated via the given univariate oracles. There is a straightforward way to guarantee these properties. We start with a polynomial function φ from $\mathbb{F}^m$ to $\mathbb{F}$ such that the restriction of φ to some domain H' is a bijective map to H. Any such φ naturally induces a ring homomorphism Φ between the quotient ring $\mathbb{F}[x]/I(H)$ and the quotient ring $\mathbb{F}[x_1, \ldots, x_m]/I(H')$ where $I(H)$ and $I(H')$ denote the ideals of H and H'. Specifically,

$$\Phi : \mathbb{F}[x] \to \mathbb{F}[x_1, \ldots, x_m], f \mapsto \Phi(f) = f(\varphi(x_1, \ldots, x_m))$$

3

where, firstly, $\Phi(f)$ is a polynomial in $\mathbb{F}[x_1,\ldots,x_m]$ because φ is a polynomial function. Secondly, due to φ being a bijection between H and H', the evaluations of any product polynomial $\Phi(f)\Phi(g)$ on H' are the same as the evaluations of fg on H. This is how Φ is a ring homomorphism between the quotient rings. As far as we can tell, the simplest φ and H' that fit this description are the product map $\varphi : (x_1,\ldots,x_m) \mapsto \prod_{j\in[m]} x_j$ and the domain $H' = \{1,w\} \times \{1,w^2\} \times \cdots \times \left\{1,w^{2^{m-1}}\right\}$. Clearly, φ is a polynomial, which also means that evaluating $\Phi(f)$ is easy given oracle access to f. Clearly, φ is bijective since every $w^i \in H$ can be uniquely mapped to a bit string $(i_1,\ldots,i_m)$, the bit decomposition of i, and every such bit string can be mapped to a unique element of H'. The resulting univariate sumcheck protocol is certainly sound (by construction as a special case of the multivariate sumcheck protocol) and the final claim can certainly be answered by queries to univariate oracles.

Still, the actual usefulness of φ and H' for constructing an *efficient* univariate sumcheck protocol is not obvious. To see the potential issues, let us define the univariate polynomial under the sum as $P(x) = g(f_1(x), \ldots, f_q(x))$ where each f_i has degree at most $2^m - 1$ and g is a polynomial function with total degree at most d. Consider the round polynomials $p_1,\ldots,p_m$ in an execution of the multivariate sumcheck protocol on $\Phi(P)$ over the domain H'. Clearly, $\Phi(P)$ having high degree in every variable means that every round polynomial will have high degree, too. But this, in turn, means that trying to send the round polynomials explicitly would blow up the protocol's communication complexity. Even when trying to send the round polynomials as oracles, the prover must still at least internally compute an explicit description of them. But because their degree is high, this inherently appears to require at least $\deg_j(P) = d \cdot 2^m - d$ operations in *every* round j—impossible for an efficient (*linear-time*) prover.

To build an intuition for the possibilities opened up by Φ and H', let us first rewrite the round polynomials p_j, for $j \in [m]$, as

$$
\begin{aligned}
p_j(y) &= \sum_{i_{j+1}\in\{1,w^{2^j}\},\ldots,i_m\in\{1,w^{2^{m-1}}\}} \Phi(P)(r_1,\ldots,r_{j-1},y,i_{j+1},\ldots,i_m) \\
&= \sum_{i_{j+1}\in\{1,w^{2^j}\},\ldots,i_m\in\{1,w^{2^{m-1}}\}} P(\varphi(r_1,\ldots,r_{j-1},y,i_{j+1},\ldots,i_m)) \\
&= \sum_{i_{j+1}\in\{1,w^{2^j}\},\ldots,i_m\in\{1,w^{2^{m-1}}\}} P(r_1 \cdots r_{j-1} y i_{j+1} \cdots i_m).
\end{aligned}
$$

Using the bit-wise association of elements of H' to elements of H, we may further rewrite p_j as

$$
\begin{aligned}
p_j(y) &= \sum_{i_{j+1}\in\{0,1\},\ldots,i_m\in\{0,1\}} P\left(r_1 \cdots r_{j-1} y w^{2^j(i_{j+1}+\cdots+2^{m-j-1}i_m)}\right) \\
&= \sum_{i\in[0,2^{m-j}-1]} P\left(r_1 \cdots r_{j-1} y w^{2^j i}\right).
\end{aligned}
$$

We next observe that *one* of the round polynomials *can* be efficiently represented.

Proposition 1. The first round polynomial can be represented as

$$p_1(y) = p_1'\left(y^{2^{m-1}}\right)$$

where p_1' has degree at most $2d - 1$.

Proof. We start by making a general statement about sums of polynomials over arbitrary cosets. A special case of the following fact has previously already appeared as an important lemma in [2, 8]. For any $a \in \mathbb{F}, p \in \mathbb{F}[x]$,

$$\sum_{i \in [0, 2^m - 1]} p(aw^i) = \sum_{i \in [0, 2^m - 1]} \sum_{k \in [0, \deg(p)]} p[k] a^k w^{ik}$$

$$= \sum_{k \in [0, \deg(p)]} p[k] a^k \sum_{i \in [0, 2^m - 1]} w^{ik}$$

$$= \sum_{k \in [0, \deg(p)] : k = 0 \pmod{2^m}} p[k] a^k \cdot 2^m$$

where the last equality used that, generally, $\sum_i w^{ib}$ is zero except if $w^{ib} = 1$ for all i in which case the sum is the size of the domain. Now consider replacing w with w^2 and m with $m - 1$, which is valid because w^2 generates the subgroup of 2^{m-1}th roots of unity. Then additionally substituting $p = P$ and replacing the fixed value a with a new formal indeterminate y yields the statement that the expression

$$\sum_{i \in [0, 2^{m-1} - 1]} P\left(yw^{2i}\right)$$

(which is the expression for $p_1(y)$ derived in the main text above) is, in fact, a sparse polynomial in y with non-zero coefficients only at the 2^{m-1}th powers of y. Since P has degree at most $d \cdot (2^m - 1) < 2d \cdot 2^{m-1}$, this is equivalent to what we needed to show. This completes the proof of Proposition 1. $\square$

It can also be seen from the proof of Proposition 1 that further round polynomials, unlike p_1 cannot be efficiently represented in the same way. They consist of denser and denser subsets of P's coefficients, scaled by the randomizers $r_1, \ldots, r_m$, to the point where

- p_{m-1} contains all *even* coefficients of P, scaled by powers of $r_1 \cdots r_{m-2}$, and

- $p_m(y)$ is the same as $P(r_1 \cdots r_{m-1} y)$.

It is known that the even part of P can be evaluated with just two queries to $f_1, \ldots, f_q$— and p_m can clearly be evaluated with just a single query to $f_1, \ldots, f_q$. However, there does not seem to be an easy way to evaluate the round polynomials in the middle rounds.

5

3.2 Univariate Sumcheck to Multilinear Evaluation

This section presents a straightforward modification of the previous section's approach that allows for efficient explicit representations of all round polynomials. The modification comes from observing that the only reason why Proposition 1 does not apply to every p_j is the ratio of P's degree to the jth summation domain. That is, the summation domains become smaller in each round while P's degree remains the same. But consider simply replacing the sum over P with a sum over a lower-degree polynomial *that agrees with P on that domain*. Clearly, the sums will be equal. Agreement over a certain domain can be expressed as taking a polynomial's remainder modulo the domain's vanishing polynomial. Specifically,

$$\sum_{i\in[0,2^{m-1}]} P\left(r_1 w^{2i}\right) = \sum_{i\in[0,2^{m-1}]} \left(P\%\left(x^{2^{m-1}} - r_1^{2^{m-1}}\right)\right)\left(r_1 w^{2i}\right).$$

More generally, any polynomial with the same remainder may safely be substituted for P. This is helpful when, like usual, P is given as $P = g(f_1, \ldots, f_q)$ because each f_i may be reduced individually. We immediately obtain the following protocol in which remainders are taken after each round in order to rewrite the reduced claim.

Claim: $\sum_{i\in[0,2^m-1]} (g(f_1, \ldots, f_q))\left(w^i\right) = s.$

- The prover sends $p_1(y) = \sum_{i\in[0,2^{m-1}]} (g(f_1, \ldots, f_q))\left(yw^{2i}\right)$ *explicitly* (Proposition 1).

- The verifier checks that $p_1(1) + p_1(w) = s.$

- The verifier sends r_1, chosen at random.

- Both prover and verifier set $s' = p_1(r_1)$.

New claim:

$$\sum_{i\in[0,2^{m-1}-1]} \left(g\left(f_1\%\left(x^{2^{m-1}} - r_1^{2^{m-1}}\right), \ldots, f_q\%\left(x^{2^{m-1}} - r_1^{2^{m-1}}\right)\right)\right)\left(r_1 w^{2i}\right) = s'.$$

The above protocol is an *exact* analogue of the multivariate sumcheck protocol in the sense that the multivariate protocol's core step of replacing $P(x_1, \ldots, x_m)$ with $P(r_1, x_2, \ldots, x_m)$ can also be seen as passing to the equivalence class modulo $x_1 - r_1$. Under the appropriate Kronecker substitution, this would indeed be the same as taking remainders modulo $x^{2^{m-1}} - r_1^{2^{m-1}}$.

Next, note that in the subsequent recursive round, after the verifier sends randomness, say r_2, we arrive at a sum over the set of all $r_1 r_2 w^{4i}$. This means that repeating the modulo reduction approach leads to a claim about remainder polynomials

$$\left(f_i\%\left(x^{2^{m-1}} - r_1^{2^{m-1}}\right)\right)\%\left(x^{2^{m-2}} - (r_1 r_2)^{2^{m-2}}\right).$$

Continuing this way, the final claim eventually contains

$$\left(\cdots f_i \% \left(x^{2^{m-1}} - r_1^{2^{m-1}} \right) \cdots \right) \% \left(x - r_1 \cdots r_m \right).$$

But taking m such remainders of a polynomial of degree $2^m - 1$ is, in fact, a kind of *multilinear* evaluation. Namely, when denoting as $\mathsf{mlin}(f_i)$ the multilinear polynomial with the same coefficient vector as f_i, the final claim is exactly

$$g \left(\mathsf{mlin}(f_1) \left(r_1 \cdots r_m, \ldots, r_1^{2^{m-1}} \right), \ldots, \mathsf{mlin}(f_q) \left(r_1 \cdots r_m, \ldots, r_1^{2^{m-1}} \right) \right) = s'.$$

This completes the description of our reduction from univariate sumcheck to multilinear evaluation.

Remark (Prover algorithm). It is intended for later versions of this article to describe in detail a linear-time prover algorithm that computes explicit representations of the round polynomials p_j, or rather of the low-degree polynomials p'_j. We sketch the main idea. The prover would represent each p'_j by its evaluations on a fixed set of points $a_1^{2^{m-j}}, \ldots, a_{2d}^{2^{m-j}}$. The values on these points could, by Proposition 1, be computed as sums over the a_k-cosets of the 2^{m-j}th roots of unity $w^{2^j i}$. It is not hard to see that such sums over the a_k-cosets would be the same as sums over the roots of unity themselves, that is, over the 1-coset, for the polynomials $f_i \% \left(x^{2^{m-j}} - a_k^{2^{m-j}} \right)$. This fact, and the need to anyway obtain the values of remainders modulo $x^{2^{m-j-1}} - (r_1 \cdots r_{j+1})^{2^{m-j-1}}$ before starting the next round, means the prover algorithm really only requires a method to compute values of remainders. But here, we could re-use the linear-time multivariate sumcheck prover algorithm almost as-is. As hinted at above, when viewed under an appropriate Kronecker substitution, that algorithm already does nothing but compute values of remainders.

3.3 Multilinear Evaluation to Univariate Evaluation

This section shows that, when applying the generic protocol from Section 3.1 to multilinear evaluation claims, a variant of Proposition 1 allows for efficiently computable oracle representations of all round polynomials. First, let us write the multilinear evaluation statement of concern, say, $\mathsf{mlin}(f)(z_1, \ldots, z_m) = s$, as a univariate sumcheck. This is possible because there exists a polynomial kernel function $K(x, x_1, \ldots, x_m)$ of degree $2^m - 1$ in x and degree 1 in $x_1, \ldots, x_m$ such that

$$\mathsf{mlin}(f)(z_1, \ldots, z_m) = \sum_{i \in [0, 2^m - 1]} f\left(w^i \right) K\left(w^i, z_1, \ldots, z_m \right). \tag{1}$$

To derive an expression for K, recall that $\mathsf{mlin}(f)$ has the same monomial coefficients as f. This implies that $\mathsf{mlin}(f)$ takes the value $f\left(w^j \right)$ at $\left(w^j, w^{2j}, \ldots, w^{2^{m-1}j} \right)$. But this, in turn, means that K must be such that, for all $i, j \in [0, 2^m - 1]$, $K\left(w^i, x_1, \ldots, x_m \right)$ is multilinear in $(x_1, \ldots, x_m)$ and that $K\left(w^i, w^j, \ldots, w^{2^{m-1}j} \right)$ is 1 if $i = j$ and 0 if $i \neq j$.

These properties are uniquely met by

$$K\left(w^i, x_1, \ldots, x_m\right) = 2^{-m} \prod_{\ell \in [m]} \left(w^{-2^{\ell-1}i} x_\ell + 1\right)$$

$$= 2^{-m} \sum_{\ell_1, \ldots, \ell_m \in \{0,1\}} \left(w^{-i} x_1\right)^{\ell_1} \cdots \left(w^{-2^{m-1}i} x_m\right)^{\ell_m}$$

$$= 2^{-m} \sum_{\ell \in [0, 2^m - 1]} w^{-i\ell} x_1^{\ell_1} \cdots x_m^{\ell_m}$$

where, in each term of the last sum, $\ell_1, \ldots, \ell_m$ are the bits of ℓ, ℓ_1 being the least significant bit. The just derived coefficient representation of $K\left(w^i, \cdots\right)$ makes it possible to write down the general formula:

$$K(x, x_1, \ldots, x_m) = 2^{-m} \left(1 + \sum_{\ell \in [2^m - 1]} x^{2^m - \ell} x_1^{\ell_1} \cdots x_m^{\ell_m}\right)$$

$$= 2^{-m} \left(1 - x^{2^m} + \sum_{\ell \in [0, 2^m - 1]} x^{2^m - \ell} x_1^{\ell_1} \cdots x_m^{\ell_m}\right)$$

$$= 2^{-m} \left(1 - x^{2^m} + x^{2^m} \sum_{\ell \in [0, 2^m - 1]} \left(x^{-1} x_1\right)^{\ell_1} \cdots \left(x^{-2^{m-1}} x_m\right)^{\ell_m}\right)$$

$$= 2^{-m} \left(1 - x^{2^m} + x^{2^m} \prod_{\ell \in [m]} \left(x^{-2^{\ell-1}} x_\ell + 1\right)\right).$$

Here, we have already rewritten K in a convenient product form. This product structure is precisely what the univariate sumcheck protocol can take advantage of as per the following proposition.

Proposition 2. When applying the generic univariate sumcheck protocol from Section 3.1 to the right-hand side of Equation (1), each round polynomial can be represented as

$$p_j(y) = 2^{-m} \left(\left(1 - (r_1 \cdots r_{j-1} y)^{2^m}\right) p'_j \left((r_1 \cdots r_{j-1} y)^{2^{m-j}}\right) \right.$$

$$\left. + (r_1 \cdots r_{j-1} y)^{2^m} \left(\prod_{\ell \in [m-j+1, m]} \left((r_1 \cdots r_{j-1} y)^{-2^{\ell-1}} z_\ell + 1\right) \right) p''_j \left((r_1 \cdots r_{j-1} y)^{2^{m-j}}\right) \right)$$

where $\deg(p'_j), \deg(p''_j) \leq 2^j - 1$.

Proof. We start by substituting the product form of K in the expression for the round

8

polynomials and rearranging:

$$
p_j(y) = \sum_{i \in [0, 2^{m-j}-1]} f\left(r_1 \cdots r_{j-1} y w^{2^j i}\right) K\left(r_1 \cdots r_{j-1} y w^{2^j i}, z_1, \ldots, z_m\right)
$$

$$
= 2^{-m} \left(\left(1 - (r_1 \cdots r_{j-1} y)^{2^m}\right) \sum_{i \in [0, 2^{m-j}-1]} f\left(r_1 \cdots r_{j-1} y w^{2^j i}\right) \right.
$$

$$
\left. + (r_1 \cdots r_{j-1} y)^{2^m} \sum_{i \in [0, 2^{m-j}-1]} f\left(r_1 \cdots r_{j-1} y w^{2^j i}\right) \prod_{\ell \in [m]} \left(\left(r_1 \cdots r_{j-1} y w^{2^j i}\right)^{-2^{\ell-1}} z_\ell + 1\right) \right).
$$

Note how the last terms in the product over ℓ are independent of i as $\left(w^{2^j i}\right)^{-2^{\ell-1}}$ is 1 for $\ell \geq m - j + 1$. If we define

$$
q_j'(y) = \sum_{i \in [0, 2^{m-j}-1]} f\left(r_1 \cdots r_{j-1} y w^{2^j i}\right)
$$

$$
q_j''(y) = \sum_{i \in [0, 2^{m-j}-1]} f\left(r_1 \cdots r_{j-1} y w^{2^j i}\right) \prod_{\ell \in [m-j]} \left(\left(r_1 \cdots r_{j-1} y w^{2^j i}\right)^{-2^{\ell-1}} z_\ell + 1\right),
$$

this already means that

$$
p_j(y) = 2^{-m} \left(\left(1 - (r_1 \cdots r_{j-1} y)^{2^m}\right) q_j'(y) \right.
$$

$$
\left. + (r_1 \cdots r_{j-1} y)^{2^m} \left(\prod_{\ell \in [m-j+1, m]} \left(\left(r_1 \cdots r_{j-1} y\right)^{-2^{\ell-1}} z_\ell + 1\right) \right) q_j''(y) \right).
$$

So, it remains to show that

$$
q_j'(y) = p_j'\left((r_1 \cdots r_{j-1} y)^{2^{m-j}}\right)
$$

$$
q_j''(y) = p_j''\left((r_1 \cdots r_{j-1} y)^{2^{m-j}}\right)
$$

for polynomials p_j' and p_j'' with the degrees stated in the proposition. Here, we may follow similar steps as for Proposition 1. In particular, we have

$$
q_j'(y) = \sum_{k \in [0, 2^m - 1]} f[k] (r_1 \cdots r_{j-1} y)^k \sum_{i \in [0, 2^{m-j}-1]} w^{2^j i k}
$$

$$
= \sum_{k \in [0, 2^m - 1] \,:\, k \equiv 0 \pmod{2^{m-j}}} f[k] (r_1 \cdots r_{j-1} y)^k \cdot 2^{m-j}
$$

9

which is a linear combination of at most 2^j powers of $(r_1 \cdots r_{j-}y)^{2^{m-j}}$ as desired. For q_j'', on the other hand, we have

$$
\begin{aligned}
q_j''(y) &= \sum_{i \in [0, 2^{m-j}-1]} f\left(r_1 \cdots r_{j-1} y w^{2^j i}\right) \sum_{\ell \in [0, 2^{m-j}-1]} \left(r_1 \cdots r_{j-1} y w^{2^j i}\right)^{-\ell} z_1^{\ell_1} \cdots z_{m-j}^{\ell_{m-j}} \\
&= \sum_{k \in [0, 2^m-1], \ell \in [0, 2^{m-j}-1]} f[k] (r_1 \cdots r_{j-1} y)^{k-\ell} z_1^{\ell_1} \cdots z_{m-j}^{\ell_{m-j}} \sum_{i \in [0, 2^{m-j}-1]} w^{2^j i (k-\ell)} \\
&= \sum_{k \in [0, 2^m-1], \ell \in [0, 2^{m-j}-1] \,:\, k-\ell=0 \pmod{2^{m-j}}} f[k] (r_1 \cdots r_{j-1} y)^{k-\ell} z_1^{\ell_1} \cdots z_{m-j}^{\ell_{m-j}} \cdot 2^{m-j}.
\end{aligned}
$$

Again, we have at most 2^j terms in $(r_1 \cdots r_{j-1} y)^{2^{m-j}}$ as desired. This completes the proof of Proposition 2. $\qquad\square$

Proposition 2 suggests a protocol where the prover does not send oracles to the round polynomials p_j directly but instead sends oracles to p_j' and p_j''. The verifier is clearly able to evaluate p_j at any point r by querying p_j' and p_j'' at $(r_1 \cdots r_{j-1} r)^{2^{m-j}}$ and then performing $O(m)$ operations to assemble the result. In other words, the protocol works with *virtual oracles* to the round polynomials.

Prover algorithm. Let us now assume that instantiating oracles requires the prover to internally compute explicit representations of p_j' and p_j'' in the form of 2^j evaluations. We proceed to describe a linear-time algorithm to compute the evaluations at all points $w^{2^{m-j} k}$ for $k \in [0, 2^j - 1]$. Interestingly, the algorithm will be entirely independent of the verifier's randomness. This means the prover can pre-compute all protocol messages before any verifier interaction, resulting in a single-round protocol. The idea of the linear-time algorithm is to perform the computations for the last round $j = m$ first and for the first round $j = 1$ last. It will be ensured that the work that was already performed for round $j+1$ can be re-used for round j and only *half* of the amount of work needs to be performed in addition.

We start by writing out the following two expressions for p_j' and p_j'' that both come from the proof of Proposition 2:

$$
p_j'\left(x^{2^{m-j}}\right) = \sum_{i \in [0, 2^m-1] \,:\, i=0 \pmod{2^{m-j}}} 2^{m-j} f[i] x^i
$$

$$
p_j''\left(x^{2^{m-j}}\right) = \sum_{i \in [0, 2^{m-j}-1]} f\left(x w^{2^j i}\right) \prod_{\ell \in [m-j]} \left(\left(x w^{2^j i}\right)^{-2^{\ell-1}} z_\ell + 1\right).
$$

Notice that p_j' is, in fact, the $(m-j)$-fold even part of f. That is, p_m' has exactly the values of f, p_{m-1}' has the values of the even part of f and so on. But it is well-known that the values of any polynomial's even part can be computed from values of the polynomial itself. Thus, explicit representations of all p_j' are obtainable in linear

10

time via repeated application of the formula

$$p'_j\left(w^{2^{m-j}k}\right) = 2^{-1}\left(p'_{j+1}\left(w^{2^{m-j-1}k}\right) + p'_{j+1}\left(-w^{2^{m-j-1}k}\right)\right).$$

It remains to discuss p''_j. In round $j = m$, it is clear that p''_m also has the same values as f on all $w^k, k \in [0, 2^m - 1]$. For $j = m-1, \ldots, 1$, the goal is to compute the values of p''_j based on p''_{j+1}. We rearrange as follows:

$$p''_j\left(w^{2^{m-j}k}\right) = \sum_{i \in [0, 2^{m-j}-1]} f\left(w^{k+2^j i}\right) \prod_{\ell \in [m-j]}\left(\left(w^{k+2^j i}\right)^{-2^{\ell-1}} z_\ell + 1\right)$$

$$= \sum_{i \in [0, 2^{m-j}-1]} f\left(w^{k+2^j i}\right)\left(\prod_{\ell \in [m-j-1]}\left(\left(w^{k+2^j i}\right)^{-2^{\ell-1}} z_\ell + 1\right)\right)$$
$$\left(\left(w^{k+2^j i}\right)^{-2^{m-j-1}} z_{m-j} + 1\right)$$

$$= \sum_{i \in [0, 2^{m-j}-1]} f\left(w^{k+2^j i}\right)\left(\prod_{\ell \in [m-j-1]}\left(\left(w^{k+2^j i}\right)^{-2^{\ell-1}} z_\ell + 1\right)\right)$$
$$\left((-1)^i w^{-2^{m-j-1}k} z_{m-j} + 1\right)$$

where the second equation separated out the last term of the product over ℓ and the last equation used that $w^{-2^{m-1}} = -1$. We continue by separating the even terms of the sum, where i has the form $2i'$, and the odd terms, where i has the form $2i'+1$. This way, $(-1)^i$ can be replaced by ± 1 in the above expression, which quickly leads to the desired recursive formula:

$$p''_j\left(w^{2^{m-j}k}\right) = \sum_{i \in [0, 2^{m-j-1}-1]} f\left(w^{k+2^{j+1} i}\right)\left(\prod_{\ell \in [m-j-1]}\left(\left(w^{k+2^{j+1} i}\right)^{-2^{\ell-1}} z_\ell + 1\right)\right)$$
$$\left(w^{-2^{m-j-1}k} z_{m-j} + 1\right)$$

$$+ \sum_{i \in [0, 2^{m-j-1}-1]} f\left(w^{k+2^{j+1} i+2^j}\right)\left(\prod_{\ell \in [m-j-1]}\left(\left(w^{k+2^{j+1} i+2^j}\right)^{-2^{\ell-1}} z_\ell + 1\right)\right)$$
$$\left(-w^{-2^{m-j-1}k} z_{m-j} + 1\right)$$

$$= p''_{j+1}\left(\left(w^k\right)^{2^{m-j-1}}\right)\left(\left(w^k\right)^{-2^{m-j-1}} z_{m-j} + 1\right)$$
$$+ p''_{j+1}\left(\left(w^{k+2^j}\right)^{2^{m-j-1}}\right)\left(\left(w^{k+2^j}\right)^{-2^{m-j-1}} z_{m-j} + 1\right)$$

$$= p''_{j+1}\left(w^{2^{m-j-1}k}\right)\left(w^{-2^{m-j-1}k} z_{m-j} + 1\right)$$
$$+ p''_{j+1}\left(-w^{2^{m-j-1}k}\right)\left(-w^{-2^{m-j-1}k} z_{m-j} + 1\right).$$

Overall, the prover is able to compute and send (virtual) oracles for all round polynomials with only $O(2^m)$ effort. With the prover's computations being independent of any verifier interaction, we furthermore have a single-round protocol. That is, the prover starts by using the linear-time algorithm described above and directly sends oracles to all p'_j and p''_j. Only then does the verifier choose random points $r_1, \ldots, r_m$ and query the oracles at the required points to verify the sumcheck protocol transcript.

4 Variations and Extensions

It is intended for this article's full version to include further variants of the generic and specialized protocols. For example, we would like to discuss alternative choices of H' or φ (which might allow for reduced communication complexity etc.). As another example, we also plan to describe a tweak to the multilinear evaluation protocol that would apply to *multilinear extensions* (by replacing the function K with another).

5 Related Work

It is intended for this article's full version to compare the protocols presented here with the explicit or implicit PIOPs from works like [1, 3, 8, 5, 11] (for sumcheck) and [3, 6, 10, 12] (for multilinear evaluation).

References

[1] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. "Aurora: Transparent Succinct Arguments for R1CS". In: *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part I*. Ed. by Yuval Ishai and Vincent Rijmen. Vol. 11476. Lecture Notes in Computer Science. Springer, 2019, pp. 103–128. DOI: 10.1007/978-3-030-17653-2_4 (cited on page 12).

[2] Jonathan Bootle, Alessandro Chiesa, and Jens Groth. "Linear-Time Arguments with Sublinear Verification from Tensor Codes". In: *Theory of Cryptography - 18th International Conference, TCC 2020, Durham, NC, USA, November 16-19, 2020, Proceedings, Part II*. Ed. by Rafael Pass and Krzysztof Pietrzak. Vol. 12551. Lecture Notes in Computer Science. Springer, 2020, pp. 19–46. DOI: 10.1007/978-3-030-64378-2_2 (cited on page 5).

[3] Jonathan Bootle, Alessandro Chiesa, Yuncong Hu, and Michele Orrù. "Gemini: Elastic SNARKs for Diverse Environments". In: *Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part II*. Ed. by Orr Dunkelman and Stefan Dziembowski. Vol. 13276. Lecture Notes in Computer Science. Springer, 2022, pp. 427–457. DOI: 10.1007/978-3-031-07085-3_15 (cited on pages 1, 12).

[4]   Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. "HyperPlonk: Plonk with Linear-Time Prover and High-Degree Custom Gates". In: *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part II*. Ed. by Carmit Hazay and Martijn Stam. Vol. 14005. Lecture Notes in Computer Science. Springer, 2023, pp. 499–530. DOI: 10.1007/978-3-031-30617-4_17 (cited on page 3).

[5]   Justin Drake, Ariel Gabizon, and Izaak Meckler. *Checking univariate identities in linear time*. 2023. URL: https://hackmd.io/@relgabizon/ryGTQXWri (cited on page 12).

[6]   Tohru Kohrita and Patrick Towa. "Zeromorph: Zero-Knowledge Multilinear-Evaluation Proofs from Homomorphic Univariate Commitments". In: *J. Cryptol.* 37.4 (2024), p. 38. DOI: 10.1007/S00145-024-09519-0 (cited on page 12).

[7]   Abhiram Kothapalli and Bryan Parno. "Algebraic Reductions of Knowledge". In: *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part IV*. Ed. by Helena Handschuh and Anna Lysyanskaya. Vol. 14084. Lecture Notes in Computer Science. Springer, 2023, pp. 669–701. DOI: 10.1007/978-3-031-38551-3_21 (cited on pages 2, 3).

[8]   Helger Lipmaa, Janno Siim, and Michal Zajac. "Counting Vampires: From Univariate Sumcheck to Updatable ZK-SNARK". In: *Advances in Cryptology - ASIACRYPT 2022 - 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5-9, 2022, Proceedings, Part II*. Ed. by Shweta Agrawal and Dongdai Lin. Vol. 13792. Lecture Notes in Computer Science. Springer, 2022, pp. 249–278. DOI: 10.1007/978-3-031-22966-4_9 (cited on pages 5, 12).

[9]   Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. "Algebraic Methods for Interactive Proof Systems". In: *J. ACM* 39.4 (1992), pp. 859–868. DOI: 10.1145/146585.146605 (cited on page 2).

[10] Shahar Papini and Ulrich Haböck. "Improving logarithmic derivative lookups using GKR". In: *IACR Cryptol. ePrint Arch.* (2023), p. 1284. URL: https://eprint.iacr.org/2023/1284 (cited on page 12).

[11] Yuncong Zhang, Shifeng Sun, and Dawu Gu. "Efficient KZG-Based Univariate Sum-Check and Lookup Argument". In: *Public-Key Cryptography - PKC 2024 - 27th IACR International Conference on Practice and Theory of Public-Key Cryptography, Sydney, NSW, Australia, April 15-17, 2024, Proceedings, Part II*. Ed. by Qiang Tang and Vanessa Teague. Vol. 14602. Lecture Notes in Computer Science. Springer, 2024, pp. 400–425. DOI: 10.1007/978-3-031-57722-2_13 (cited on page 12).

[12] Jiaxing Zhao, Srinath T. V. Setty, and Weidong Cui. "MicroNova: Folding-based arguments with efficient (on-chain) verification". In: *IACR Cryptol. ePrint Arch.* (2024), p. 2099. URL: https://eprint.iacr.org/2024/2099 (cited on page 12).