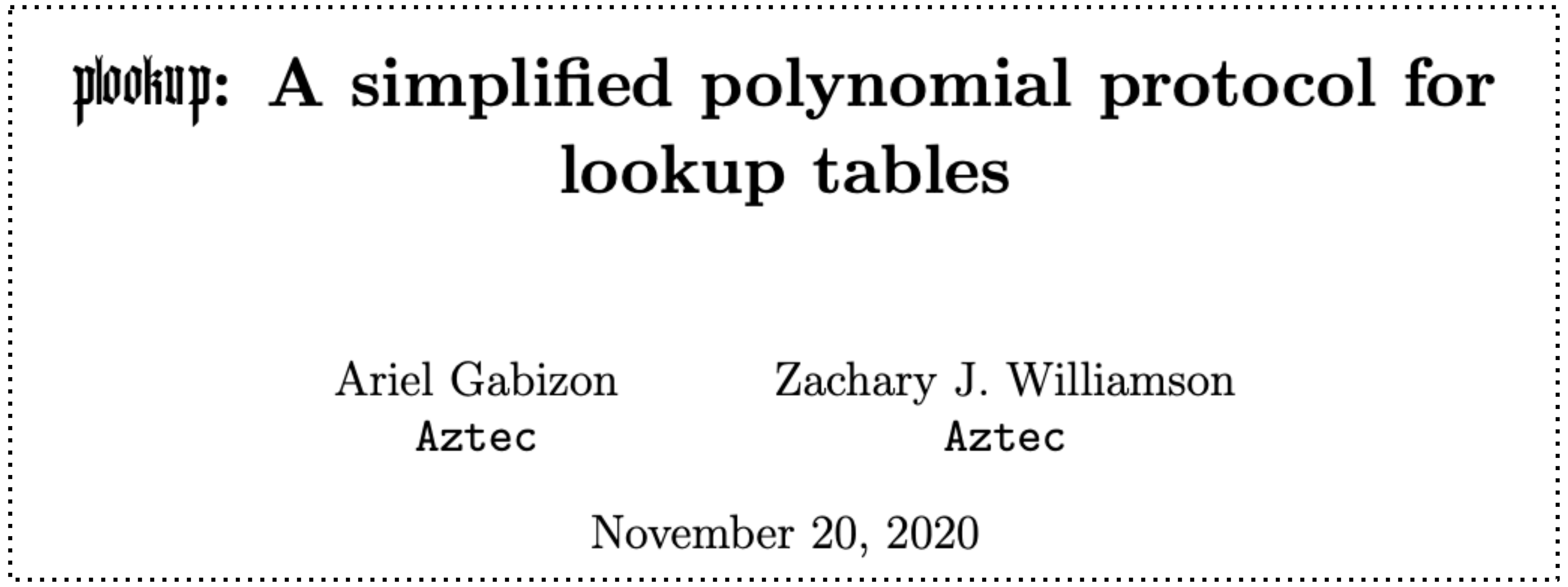
lookup

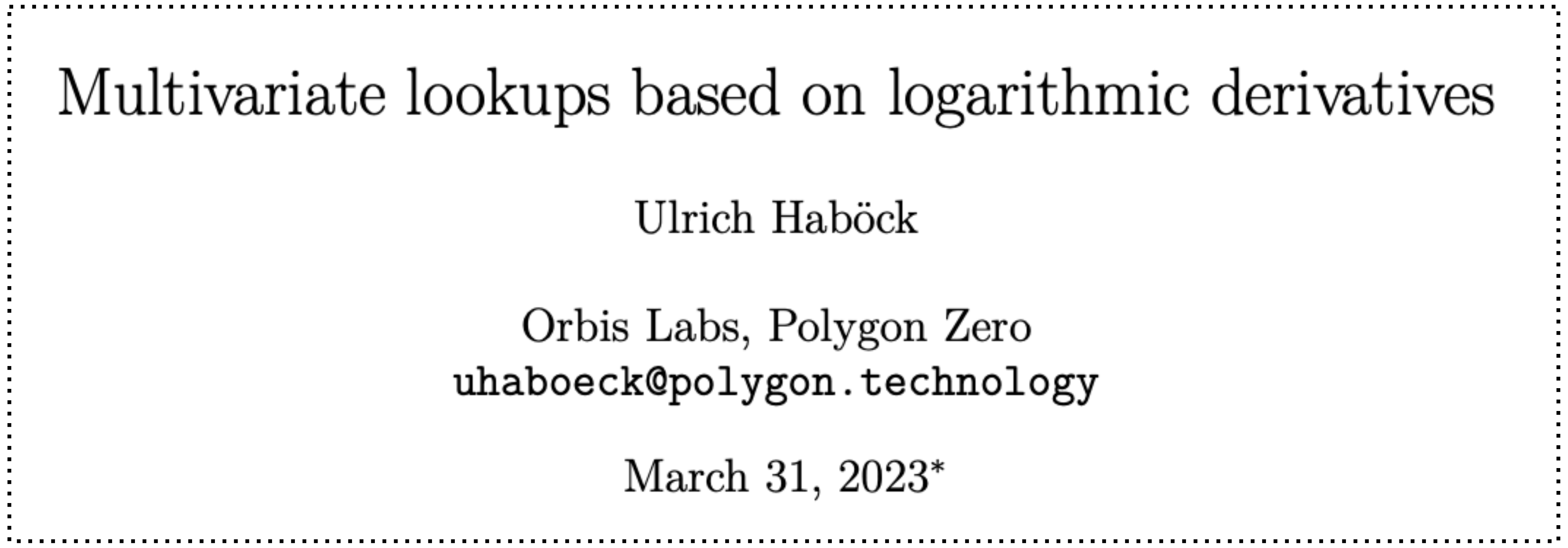


plookup: A simplified polynomial protocol for lookup tables

Ariel Gabizon
Aztec

Zachary J. Williamson
Aztec

November 20, 2020



Multivariate lookups based on logarithmic derivatives

Ulrich Haböck

Orbis Labs, Polygon Zero
uhaboeck@polygon.technology

March 31, 2023*

Fengrun Liu(刘冯润) 2023.9

Outline

- Permutation check in Plonk
- Plookup in high level: multiset check in plonk and plookup
- Plookup in detail: reduced to one multiset check
- Batch-column lookups: logUp
    - logUp
    - Multivariate PlookUp

Plonk

Gadgets for univariate polynomial over a multiplicative subgroup.

Let $\omega \in \mathbb{F}_p$ be a primitive k-th root of unity so that $\omega^k = 1$.

If $\Omega = \{1, \omega, \omega^2, \dots, \omega^{k-1}\} \subseteq \mathbb{F}_p$ then $Z_\Omega(X) = X^k - 1$.

Def: Vanishing Polynomial of Ω:

The vanishing polynomial of Ω is

$$Z_\Omega(X) := \prod_{a \in \Omega} (X - a)$$

such that $\deg(Z_\Omega) = k$.

By definition, the vanishing polynomial is a univariate polynomial to be 0 everywhere on subset Ω.

Let $f \in \mathbb{F}_p^{(\leq d)}[X]$ where $d \geq k$ and verifier has the commitment to f.

We can construct efficient Poly-IOPs for the following tasks.

- **Zero Test**: prove that f is identically zero on Ω.
- **Sum Check**: prove that $\sum_{a \in \Omega} f(a) = 0$.
- **Prod Check**: prove that $\prod_{a \in \Omega} f(a) = 1$.
  - $\rightarrow$ prove for rational functions that $\prod_{a \in \Omega} f(a)/g(a) = 1$
- **Permutation Check**: prove that $g(\Omega)$ is the same as $f(\Omega)$, just permuted.
- **Prescribed Permutation Check**: prove that $g(\Omega)$ is the same as $f(\Omega)$, permuted by the prescribed W.

refer to https://f7ed.com/2023/07/21/zkp-lec5/

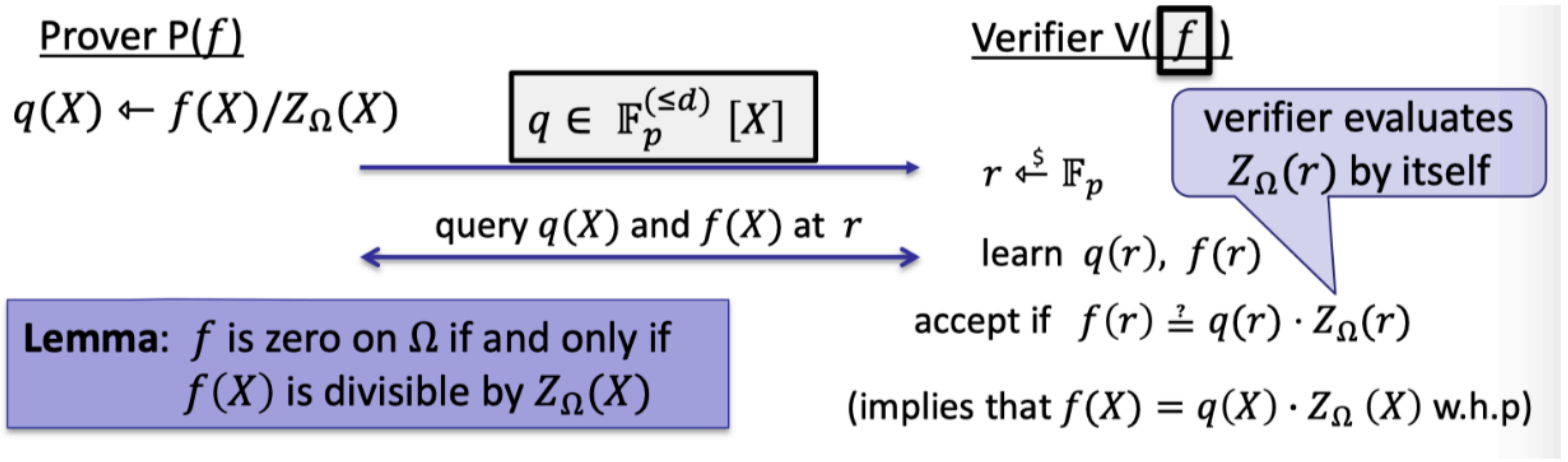
Plonk

Gadget 1: ZeroTest

Lemma:

f is zero on Ω if and only if $f(X)$ is divisible by $Z_\Omega(X)$.

The IOP of zero test is depicted as follows.



Prover P(f)
$q(X) \leftarrow f(X)/Z_\Omega(X)$
$q \in \mathbb{F}_p^{(\leq d)}[X]$
Verifier V($\boxed{f}$)
$r \xleftarrow{\$} \mathbb{F}_p$
verifier evaluates $Z_\Omega(r)$ by itself
query $q(X)$ and $f(X)$ at r
learn $q(r)$, $f(r)$
accept if $f(r) \overset{?}{=} q(r) \cdot Z_\Omega(r)$
(implies that $f(X) = q(X) \cdot Z_\Omega(X)$ w.h.p)
Lemma: f is zero on Ω if and only if $f(X)$ is divisible by $Z_\Omega(X)$

refer to https://f7ed.com/2023/07/21/zkp-lec5/

Plonk

Gadget 2: Prod Check / Sum Check

In product check, prover wants to convince verifier that the products of all evaluations over Ω equals to 1, i.e.

$$\prod_{a\in\Omega} f(a) = 1$$

We construct a degree-k polynomial to prove it.

Set $t \in \mathbb{F}_p^{(\leq k)}[X]$ to be the degree-k polynomial such that

$$t(1) = f(1),\ t(\omega^s) \qquad = \prod_{i=0}^{s} f(\omega^i) \text{ for } s = 1, \ldots, k$$

- $t(\omega) = f(1) \cdot f(\omega)$,
- $t(\omega^2) = f(1) \cdot f(\omega) \cdot f(\omega^2)$
- … …
- $t(\omega^{k-1}) = \prod_{a\in\Omega} f(a) = 1$

Lemma: If

(i) $t(\omega^{k-1}) = 1$ and

(ii) $t(\omega \cdot x) - t(x) \cdot f(\omega \cdot x) = 0$ for all $x \in \Omega$

then $\prod_{a\in\Omega} f(a) = 1$.

refer to https://f7ed.com/2023/07/21/zkp-lec5/

Plonk

Gadget 3: Prod Check / Sum Check

In product check, prover wants to convince verifier that the products of all evaluations over Ω equals to 1, i.e.

$$\prod_{a \in \Omega} f(a) = 1$$

We construct a degree-k polynomial to prove it.

Set $t \in \mathbb{F}_p^{(\le k)}[X]$ to be the degree-k polynomial such that

$$t(1) = f(1),\ t(\omega^s) \quad = \prod_{i=0}^{s} f(\omega^i) \text{ for } s = 1, \ldots, k$$

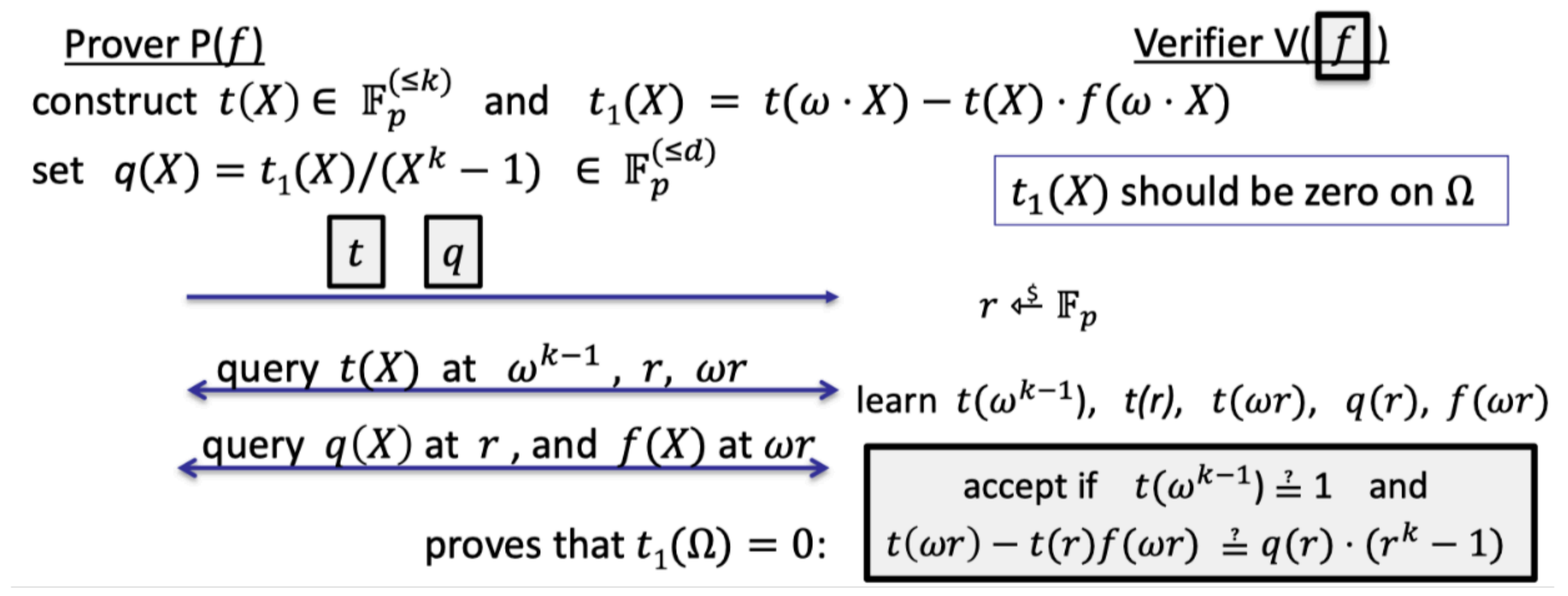
- $t(\omega) = f(1) \cdot f(\omega)$,
- $t(\omega^2) = f(1) \cdot f(\omega) \cdot f(\omega^2)$
- … …
- $t(\omega^{k-1}) = \prod_{a \in \Omega} f(a) = 1$

Lemma: If

(i) $t(\omega^{k-1}) = 1$ and

(ii) $t(\omega \cdot x) - t(x) \cdot f(\omega \cdot x) = 0$ for all $x \in \Omega$

then $\prod_{a \in \Omega} f(a) = 1$.



Prover P(f)
Verifier V($\boxed{f}$)
construct $t(X) \in \mathbb{F}_p^{(\le k)}$ and $t_1(X) = t(\omega \cdot X) - t(X) \cdot f(\omega \cdot X)$
set $q(X) = t_1(X)/(X^k - 1) \in \mathbb{F}_p^{(\le d)}$
$t_1(X)$ should be zero on Ω
$\boxed{t}$ $\boxed{q}$
$r \xleftarrow{\$} \mathbb{F}_p$
query $t(X)$ at ω^{k-1}, r, ωr
learn $t(\omega^{k-1})$, $t(r)$, $t(\omega r)$, $q(r)$, $f(\omega r)$
query $q(X)$ at r, and $f(X)$ at ωr
proves that $t_1(\Omega) = 0$:
accept if $t(\omega^{k-1}) \overset{?}{=} 1$ and $t(\omega r) - t(r) f(\omega r) \overset{?}{=} q(r) \cdot (r^k - 1)$

IOP for product check

refer to https://f7ed.com/2023/07/21/zkp-lec5/

Plonk

Gadget 3: Prod Check for rational functions

Likewise, it works to **prove the products on rational functions:**

$$\prod_{a\in\Omega}(f/g)(a) = 1$$

We construct a similar degree-k polynomial to prove it.

Set $t \in \mathbb{F}_p^{(\le k)}[X]$ to be the degree-k polynomial such that

$$t(1) = f(1)/g(1),\ t(\omega^s) \qquad = \prod_{i=0}^{s} f(\omega^i)/g(\omega^i) \text{ for } s = 1, \ldots, k$$

We write the prefix-product in an iterative way:

$$t(\omega\cdot x) = t(x)\cdot\frac{f(\omega\cdot x)}{g(\omega\cdot x)} \text{ for all } x\in\Omega$$

Lemma: If

(i) $t(\omega^{k-1}) = 1$ and

(ii) $t(\omega\cdot x)\cdot g(\omega\cdot x) - t(x)\cdot f(\omega\cdot x) = 0$ for all $x \in \Omega$

then $\prod_{a\in\Omega} f(a)/g(a) = 1$.

refer to https://f7ed.com/2023/07/21/zkp-lec5/

Plonk

Gadget 4: Permutation Check

Let f, g be polynomials in $\mathbb{F}_p^{(\le d)}[X]$.

Verifier has commitments to f and g.

In permutation check, that **goal** is that prover wants to prove that $(f(1), f(\omega), f(\omega^2), \ldots, f(\omega^{k-1}) \in \mathbb{F}_p^k$ is a **permutation** of $(g(1), g(\omega), g(\omega^2), \ldots, g(\omega^{k-1})) \in \mathbb{F}_p^k$.

The **thing to notice** is that prover **cannot** just commits to $\hat{f}$ and $\hat{g}$, then verifier checks if $\hat{f}(r) = \hat{g}(r)$.

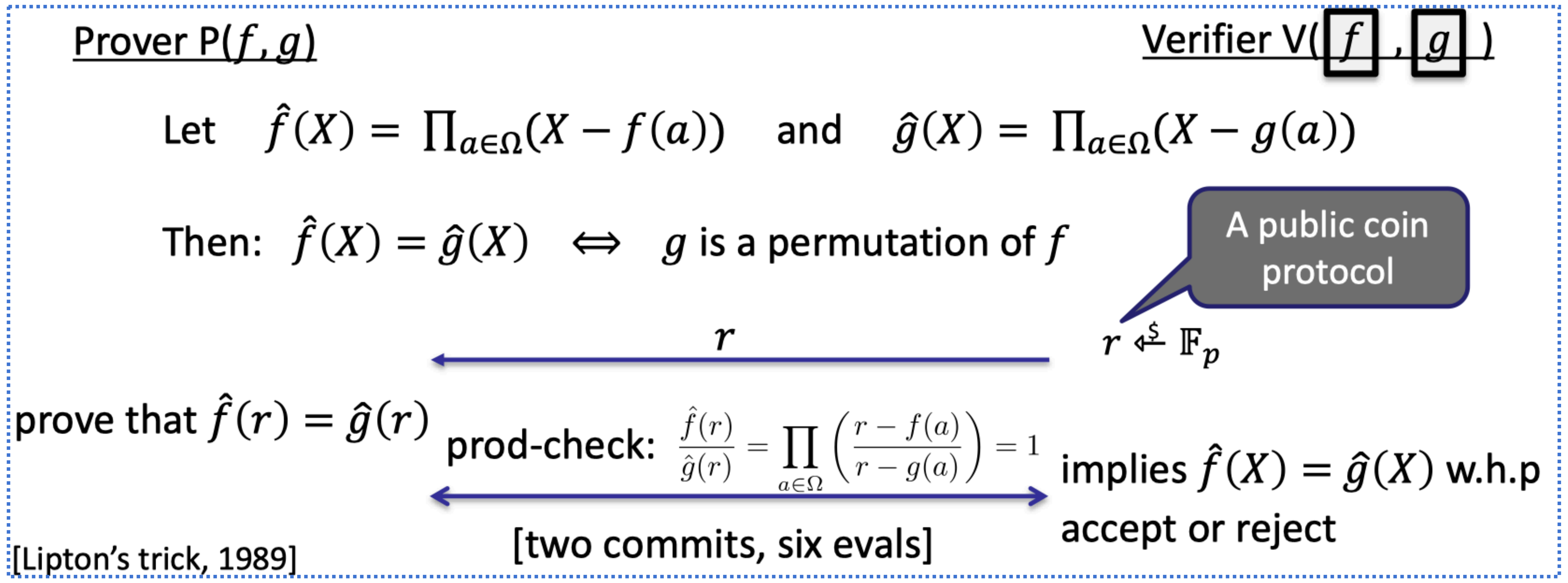
Because there is a missed proof that $\hat{f}$ is honestly constructed by the committed f.

Instead, prover is needed to prove $\hat{f}(r) = \hat{g}(r)$ by performing a product check on a rational function $\frac{r-f(X)}{r-g(X)}$.

The **main idea** is to construct auxiliary polynomials that have the evaluations as its root.

Let $\hat{f}(X) = \prod_{a\in\Omega}(X - f(a))$ and $\hat{g}(X) = \prod_{a\in\Omega}(X - g(a))$.

Then $\hat{f}(X) = \hat{g}(X)$ if and only if g is a permutation of f on Ω.



Prover P(f, g)

Verifier V($\boxed{f}$, $\boxed{g}$)

Let $\hat{f}(X) = \prod_{a\in\Omega}(X - f(a))$ and $\hat{g}(X) = \prod_{a\in\Omega}(X - g(a))$

Then: $\hat{f}(X) = \hat{g}(X) \iff g$ is a permutation of f

A public coin protocol

$r \xleftarrow{\$} \mathbb{F}_p$

r

prove that $\hat{f}(r) = \hat{g}(r)$

prod-check: $\frac{\hat{f}(r)}{\hat{g}(r)} = \prod_{a\in\Omega}\left(\frac{r-f(a)}{r-g(a)}\right) = 1$

implies $\hat{f}(X) = \hat{g}(X)$ w.h.p

accept or reject

[two commits, six evals]

[Lipton's trick, 1989]

refer to https://f7ed.com/2023/07/21/zkp-lec5/

Multiset checks in Plonk and Plookup

Reduce Multiset Equality Check (Permutation) to Grand Products

Given two vectors $a = (a_1, \ldots, a_n), b = (b_1, \ldots, b_n)$ check they contain the same elements, counting repetitions, possibly in different order.

For example, $(1, 1, 2, 3)$ is multiset-equal to $(2, 1, 1, 3)$ but not multiset-equal to $(1, 2, 3, 3)$ or $(1, 1, 2, 4)$.

Let's see the reduction from multiset equality to grand product. The verifier simply chooses a random $\gamma \in \mathbb{F}$, and runs the grand product check on the randomly shifted vectors $a' \triangleq a + \gamma, b' \triangleq b + \gamma$ (γ is added to all coordinates).
This grand product check corresponds to

$$\prod_{i \in [n]} (a_i + \gamma) \stackrel{?}{=} \prod_{i \in [n]} (b_i + \gamma)$$

refer to https://hackmd.io/@arielg/ByFgSDA7D

Multiset checks in Plonk and Plookup

Prescribed Permutation

Given a permutation $\sigma : [n] \to [n]$ suppose we want to check that $b = \sigma(a)$, in the sense that for each i, $b_i = a_{\sigma(i)}$. See the PLONK paper (and other SNARK papers) as to why permutation

pairs

$$((a_i, i))_{i \in [n]}, ((b_i, \sigma(i))_{i \in [n]}$$

Thinking about it for a minute, you can see that they are multiset-equal if and only if $b = \sigma(a)$.

Reduced to multiset-equal to vectors of elements

For this we choose random β, and define vectors a', b' by

$$a'_i \triangleq a_i + \beta \cdot i, b'_i \triangleq b_i + \beta \cdot \sigma(i)$$

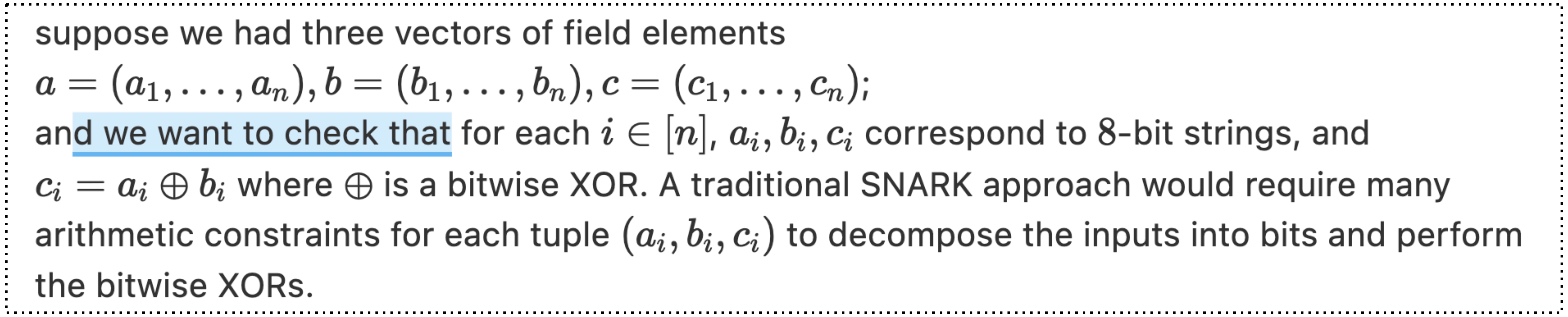
think of each element as formal polynomials in β,

$a'_i = b'_i$ implies $(a_i, i) = (b_i, \sigma(i))$ w.h.p.

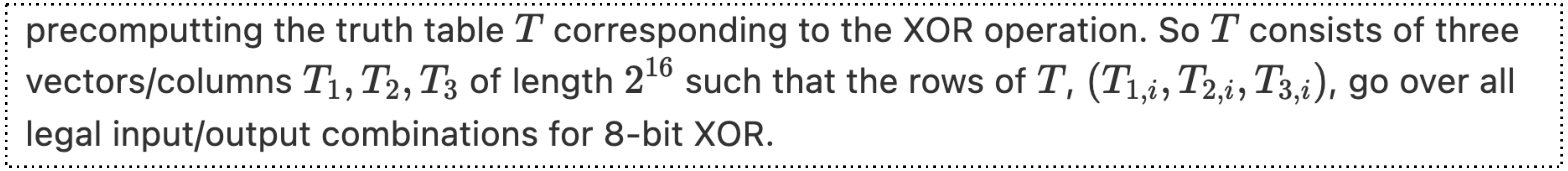
refer to https://hackmd.io/@arielg/ByFgSDA7D

Multiset checks in Plonk and Plookup

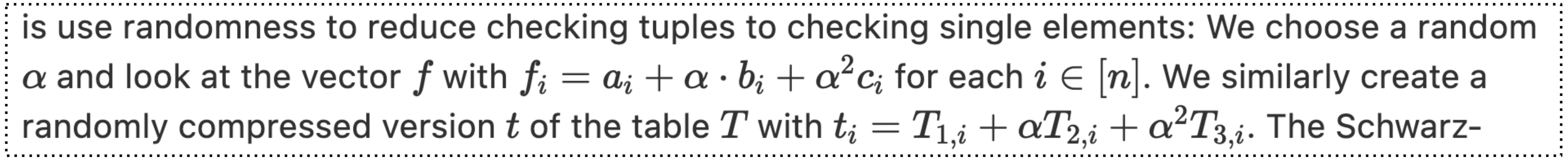
Motivation of lookup


suppose we had three vectors of field elements
$a=(a_1,\ldots,a_n), b=(b_1,\ldots,b_n), c=(c_1,\ldots,c_n)$;
and we want to check that for each $i\in[n]$, a_i,b_i,c_i correspond to 8-bit strings, and $c_i=a_i\oplus b_i$ where $\oplus$ is a bitwise XOR. A traditional SNARK approach would require many arithmetic constraints for each tuple (a_i,b_i,c_i) to decompose the inputs into bits and perform the bitwise XORs.

Lookup tables are an alternative approach to operations requiring many constraints.


precomputting the truth table T corresponding to the XOR operation. So T consists of three vectors/columns T_1,T_2,T_3 of length 2^{16} such that the rows of T, $(T_{1,i},T_{2,i},T_{3,i})$, go over all legal input/output combinations for 8-bit XOR.

Vector lookups: reduce tuples to a single element by randomization


is use randomness to reduce checking tuples to checking single elements: We choose a random α and look at the vector f with $f_i=a_i+\alpha\cdot b_i+\alpha^2c_i$ for each $i\in[n]$. We similarly create a randomly compressed version t of the table T with $t_i=T_{1,i}+\alpha T_{2,i}+\alpha^2T_{3,i}$. The Schwarz-

refer to https://hackmd.io/@arielg/ByFgSDA7D

Multiset checks in Plonk and Plookup

Main idea of Plookup: sort and compare difference

Goal: to check that every element of f is an element of t. Let's denote this by $f \subset t$

Let's describe this on a concrete example: $f = (2,2,1,1,5), t = (1,2,5)$. The prover will create an additional vector s - which is a "sorted by t" version of the concatenation (f,t) of f and t. Sorted by t means elements appear in s in the same order they appear in t. In our example, we have $s = (1,1,1,2,2,2,5,5)$.

Now the point is to look at the *difference vectors* s', t' of s and t; i.e. the vectors consisting of the differences of adjacent elements. We have $s' = (0,0,1,0,0,3,0), t' = (1,3)$.

Compare the set of non-zero difference set:

- **Captured by a multiset check** with $t'' = (1,3,0,0,0,0,0)$ and s'.
- where $t'' = t'$ concatenated with $|f|$ zeros

Observation (with issues):

following observation: Suppose we have sorted versions of two vectors, and further know they start at the same element. Then, they contain the same distinct elements if and only if they contain the same sequence of *non-zero differences* between adjacent elements.

A simpler reduction using two multiset checks:

1. Between s and (f,t)
2. Between s' and t''

Counterexample with different order:

$s = (1,1,4,4,4,4,5,5)$ then we have $s' = (0,3,0,0,0,1,0)$ with different order to t'.

Proof of (1+2 imply $f \subset t$)

- 1 implies $f \subset s$: to prove $s \subset t$
- 2 implies s' has at most $|t| - 1$ elem.
  - implies s has at most $|t|$ elem.
- 1 implies $t \subset s$
  - implies s has more than $|t|$ elem.

refer to https://hackmd.io/@arielg/ByFgSDA7D

Multiset checks in Plonk and Plookup

Plookup reduction: only uses one multiset check with randomization

Goal: to check that every element of f is an element of t. Let's denote this by $f \subset t$

A simpler reduction using two multiset checks:

1. Between s and (f, t)
2. Between s' and t''

$\longrightarrow$

Plookup reduction using only one multiset checks:

1. Define s', t' to be the "randomized difference vectors" of s and t.
2. Multiset check between s' and $((1+\beta)f, t')$

Multiset check between randomized difference vectors s' and $((1+\beta)f, t')$:

- **Main idea:** randomize each pair of adjacent elements between s (sorted version) and (f, t).
- For each element $s_i' = s_i + s_{i+1} \cdot \beta$ (can be thought of as the polynomial in β)
  - $s_i = s_{i+1}$: $s_i' = (1+\beta)f_j = f_j + f_j \cdot \beta$ implies $s_i = s_{i+1} = f_j$ implies $f \subset s$
  - $s_i \neq s_{i+1}$: $s_i' = t_j' = t_j + t_{j+1} \cdot \beta$ implies $(s_i, s_{i+1}) = (t_j, t_{j+1})$ implies s_{i+1} is contained in t implies $s \subset t$

```
f: 2 2 1 1 5
t: 1 2 5
s: 1 1 1 2 2 2 5 5
```

Reduce the multiset check to grand product:

$$F(\beta,\gamma) = \prod_j (\gamma + f_j + f_j \cdot \beta) \cdot \prod_j (\gamma + t_j + t_{j+1} \cdot \beta) \qquad G(\beta,\gamma) = \prod_i (\gamma + s_i + s_{i+1} \cdot \beta)$$

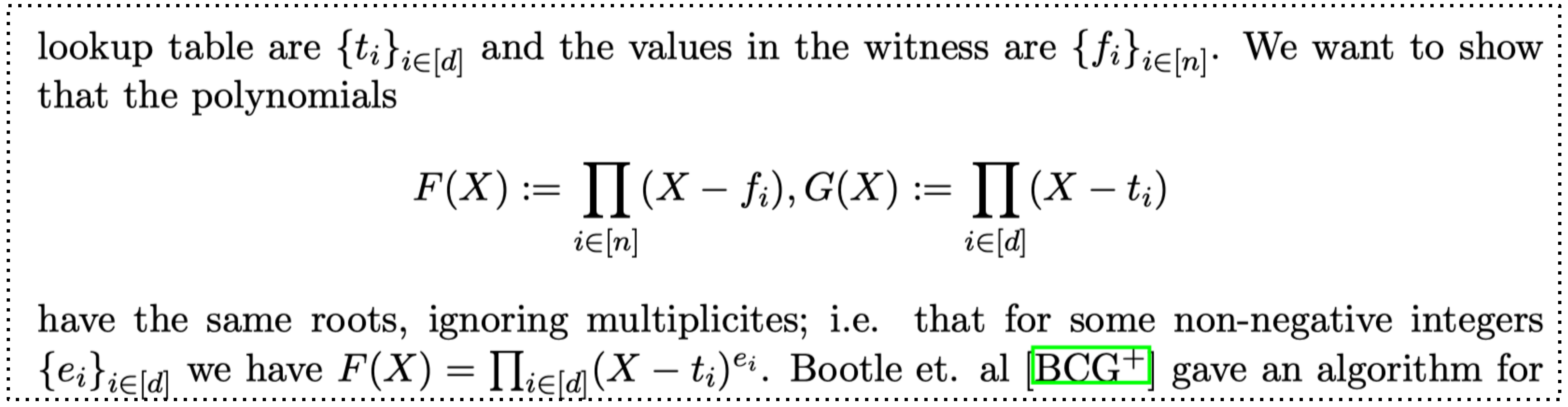
refer to https://hackmd.io/@arielg/ByFgSDA7D

https://eprint.iacr.org/2020/315.pdf

Plookup: details

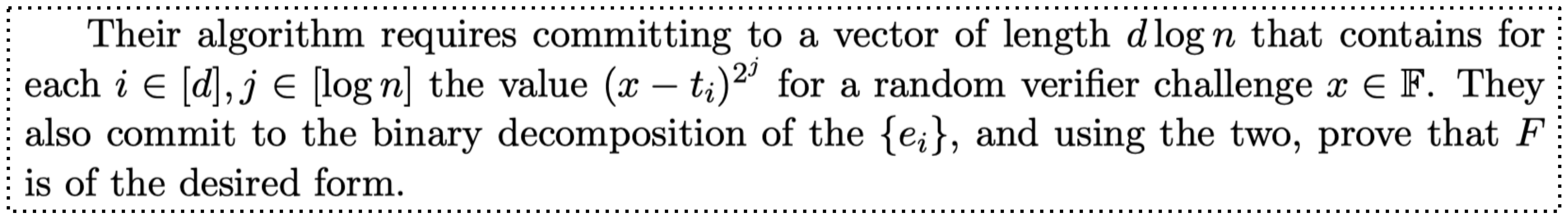
Notations

Exact problem in [BCG+]:

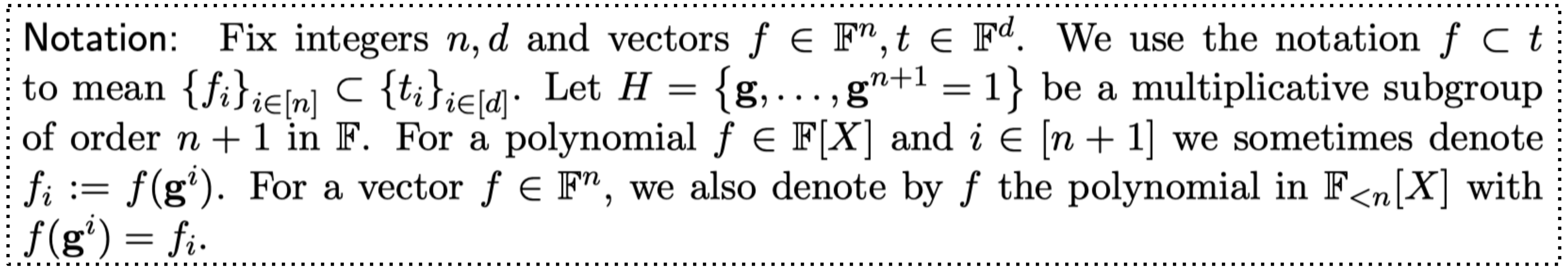

lookup table are $\{t_i\}_{i\in[d]}$ and the values in the witness are $\{f_i\}_{i\in[n]}$. We want to show that the polynomials

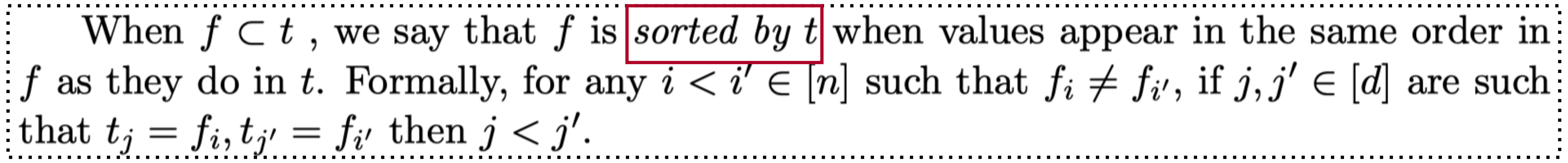
$$F(X) := \prod_{i\in[n]}(X-f_i), G(X) := \prod_{i\in[d]}(X-t_i)$$

have the same roots, ignoring multiplicites; i.e. that for some non-negative integers $\{e_i\}_{i\in[d]}$ we have $F(X) = \prod_{i\in[d]}(X-t_i)^{e_i}$. Bootle et. al [BCG+] gave an algorithm for

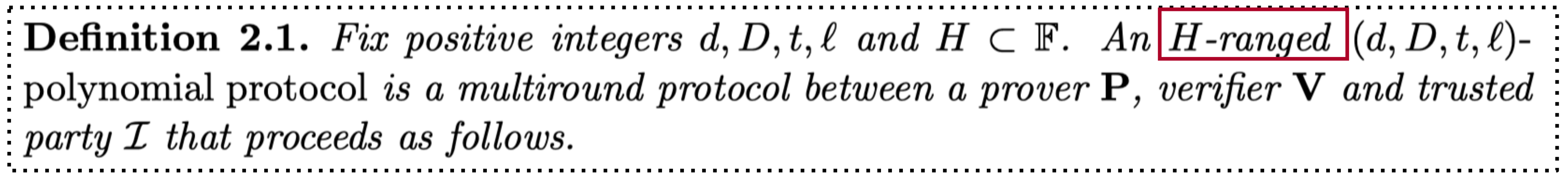

Their algorithm requires committing to a vector of length $d\log n$ that contains for each $i\in[d], j\in[\log n]$ the value $(x-t_i)^{2^j}$ for a random verifier challenge $x\in\mathbb{F}$. They also commit to the binary decomposition of the $\{e_i\}$, and using the two, prove that F is of the desired form.

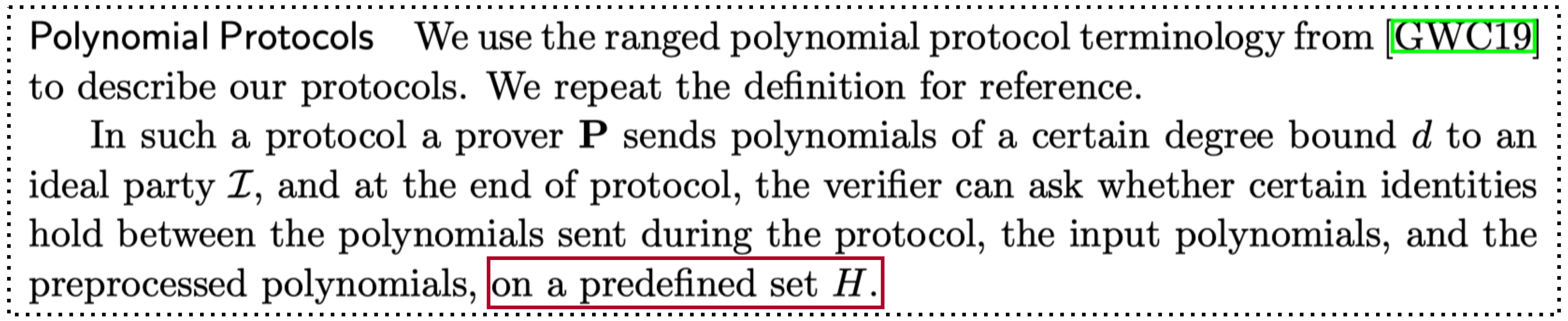
Notations:


Notation: Fix integers n, d and vectors $f\in\mathbb{F}^n, t\in\mathbb{F}^d$. We use the notation $f\subset t$ to mean $\{f_i\}_{i\in[n]}\subset\{t_i\}_{i\in[d]}$. Let $H=\{\mathbf{g},\ldots,\mathbf{g}^{n+1}=1\}$ be a multiplicative subgroup of order $n+1$ in $\mathbb{F}$. For a polynomial $f\in\mathbb{F}[X]$ and $i\in[n+1]$ we sometimes denote $f_i := f(\mathbf{g}^i)$. For a vector $f\in\mathbb{F}^n$, we also denote by f the polynomial in $\mathbb{F}_{<n}[X]$ with $f(\mathbf{g}^i)=f_i$.


When $f\subset t$, we say that f is *sorted by t* when values appear in the same order in f as they do in t. Formally, for any $i<i'\in[n]$ such that $f_i\neq f_{i'}$, if $j,j'\in[d]$ are such that $t_j=f_i, t_{j'}=f_{i'}$ then $j<j'$.

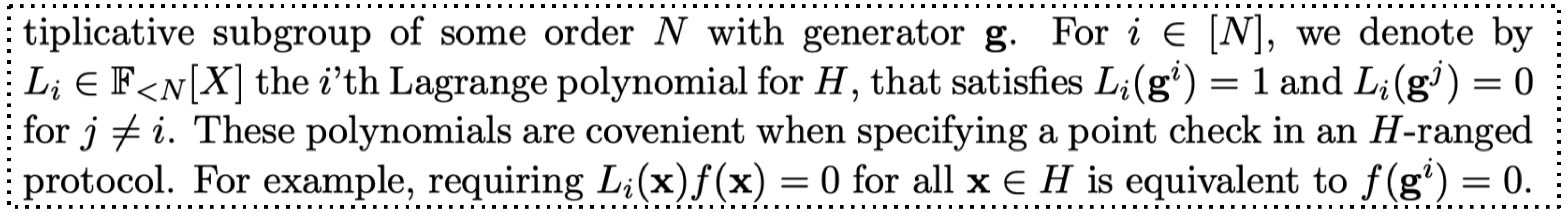
H-ranged Polynomial Protocol:


Definition 2.1. *Fix positive integers d, D, t, ℓ and $H\subset\mathbb{F}$. An H-ranged (d, D, t, ℓ)-*polynomial protocol *is a multiround protocol between a prover* **P**, *verifier* **V** *and trusted party $\mathcal{I}$ that proceeds as follows.*

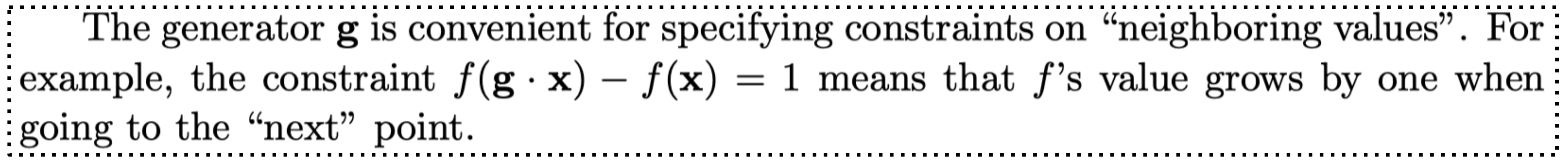

Polynomial Protocols We use the ranged polynomial protocol terminology from [GWC19] to describe our protocols. We repeat the definition for reference.

In such a protocol a prover **P** sends polynomials of a certain degree bound d to an ideal party $\mathcal{I}$, and at the end of protocol, the verifier can ask whether certain identities hold between the polynomials sent during the protocol, the input polynomials, and the preprocessed polynomials, on a predefined set H.

Reduced a point check to an H-ranged equality check:


tiplicative subgroup of some order N with generator $\mathbf{g}$. For $i\in[N]$, we denote by $L_i\in\mathbb{F}_{<N}[X]$ the i'th Lagrange polynomial for H, that satisfies $L_i(\mathbf{g}^i)=1$ and $L_i(\mathbf{g}^j)=0$ for $j\neq i$. These polynomials are covenient when specifying a point check in an H-ranged protocol. For example, requiring $L_i(\mathbf{x})f(\mathbf{x})=0$ for all $\mathbf{x}\in H$ is equivalent to $f(\mathbf{g}^i)=0$.

"neighboring values" in multiplicative group


The generator $\mathbf{g}$ is convenient for specifying constraints on "neighboring values". For example, the constraint $f(\mathbf{g}\cdot\mathbf{x})-f(\mathbf{x})=1$ means that f's value grows by one when going to the "next" point.

https://eprint.iacr.org/2020/315.pdf

Plookup: details

Main scheme

Now, given $t \in \mathbb{F}^d, f \in \mathbb{F}^n, s \in \mathbb{F}^{n+d}$, define bi-variate polynomials F, G as

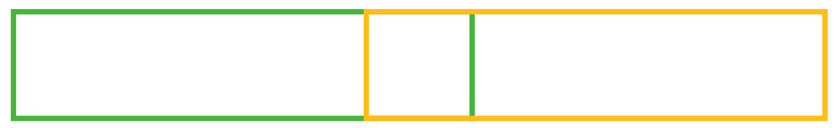
$$F(\beta,\gamma) := (1+\beta)^n \cdot \prod_{i\in[n]} (\gamma + f_i) \prod_{i\in[d-1]} (\gamma(1+\beta) + t_i + \beta t_{i+1})$$

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

Claim 3.1. *$F \equiv G$ if and only if*

1. *$f \subset t$, and*
2. *s is (f,t) sorted by t.*

Proof is referred to in [GW20]

S1. 

S3-5.

Grand Product Check:

$$\frac{F(\beta,\gamma)}{G(\beta,\gamma)} = \prod_{i=1}^{n} \ldots = 1$$

Divide the computation to $n+1$ rounds.

Claim 3.1 motivates the following protocol. It will be convenient to assume $d = n+1$. (If $d \le n$ pad t with $n-d+1$ repetitions of the last element.)

Preprocessed polynomials: The polynomial $t \in \mathbb{F}_{<n+1}[X]$ describing the lookup values.

Inputs: $f \in \mathbb{F}_{<n}[X]$

Protocol:

1. Let $s \in \mathbb{F}^{2n+1}$ be the vector that is (f,t) sorted by t. We represent s by $h_1, h_2 \in \mathbb{F}_{<n+1}[X]$ as follows. $h_1(\mathbf{g}^i) = s_i$ for $i \in [n+1]$; and $h_2(\mathbf{g}^i) = s_{n+i}$ for each $i \in [n+1]$.
2. $\mathbf{P}$ computes the polynomials $\boxed{h_1, h_2}$ and sends them to the ideal party $\mathcal{I}$.
3. $\mathbf{V}$ chooses random $\beta, \gamma \in \mathbb{F}$ and sends them to $\mathbf{P}$.
4. $\mathbf{P}$ computes a polynomial $\boxed{Z \in \mathbb{F}_{<n+1}[X]}$ that aggregates the value $F(\beta,\gamma)/G(\beta,\gamma)$ where F, G are as described above. Specifically, we let
   (a) $Z(\mathbf{g}) = 1$,
   (b) For $2 \le i \le n$
   $$Z(\mathbf{g}^i) = \frac{(1+\beta)^{i-1} \prod_{j<i}(\gamma + f_j) \cdot \prod_{1\le j<i}(\gamma(1+\beta) + t_j + \beta t_{j+1})}{\prod_{1\le j<i}(\gamma(1+\beta) + s_j + \beta s_{j+1})(\gamma(1+\beta) + s_{n+j} + \beta s_{n+j+1})},$$
   and
   (c) $Z(\mathbf{g}^{n+1}) = 1$.
5. $\mathbf{P}$ sends Z to $\mathcal{I}$.

https://eprint.iacr.org/2020/315.pdf

Plookup: details

Main scheme

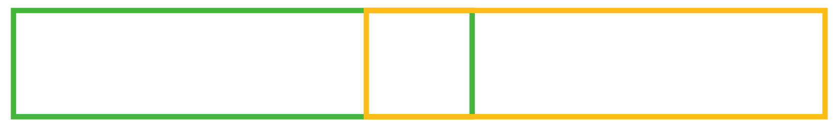
Claim 3.1 motivates the following protocol. It will be convenient to assume $d = n+1$. (If $d \leq n$ pad t with $n-d+1$ repetitions of the last element.)

Preprocessed polynomials: The polynomial $t \in \mathbb{F}_{<n+1}[X]$ describing the lookup values.

Inputs: $f \in \mathbb{F}_{<n}[X]$

Protocol:

1. Let $s \in \mathbb{F}^{2n+1}$ be the vector that is (f,t) sorted by t. We represent s by $h_1, h_2 \in \mathbb{F}_{<n+1}[X]$ as follows. $h_1(\mathbf{g}^i) = s_i$ for $i \in [n+1]$; and $h_2(\mathbf{g}^i) = s_{n+i}$ for each $i \in [n+1]$.
2. $\mathbf{P}$ computes the polynomials $\boxed{h_1, h_2}$ and sends them to the ideal party $\mathcal{I}$.
3. $\mathbf{V}$ chooses random $\beta, \gamma \in \mathbb{F}$ and sends them to $\mathbf{P}$.
4. $\mathbf{P}$ computes a polynomial $\boxed{Z \in \mathbb{F}_{<n+1}[X]}$ that aggregates the value $F(\beta,\gamma)/G(\beta,\gamma)$ where F, G are as described above. Specifically, we let
   (a) $Z(\mathbf{g}) = 1$,
   (b) For $2 \leq i \leq n$
   $$Z(\mathbf{g}^i) = \frac{(1+\beta)^{i-1}\prod_{j<i}(\gamma+f_j)\cdot\prod_{1\leq j<i}(\gamma(1+\beta)+t_j+\beta t_{j+1})}{\prod_{1\leq j<i}(\gamma(1+\beta)+s_j+\beta s_{j+1})(\gamma(1+\beta)+s_{n+j}+\beta s_{n+j+1})},$$
   and
   (c) $Z(\mathbf{g}^{n+1}) = 1$.
5. $\mathbf{P}$ sends Z to $\mathcal{I}$.

S1. 

S3-5. **Grand Product Check:**

$$\frac{F(\beta,\gamma)}{G(\beta,\gamma)} = \prod_{i=1}^{n} \ldots = 1$$

Divide the computation to $n+1$ rounds.

Verification:

6. $\mathbf{V}$ checks that Z is indeed of the form described above, and that $Z(\mathbf{g}^{n+1}) = 1$. More precisely, $\mathbf{V}$ checks the following identities for all $\mathbf{x} \in H$.
   (a) $L_1(\mathbf{x})(Z(\mathbf{x})-1) = 0$.
   (b)
   $$(\mathbf{x}-\mathbf{g}^{n+1})Z(\mathbf{x})(1+\beta)\cdot(\gamma+f(\mathbf{x}))(\gamma(1+\beta)+t(\mathbf{x})+\beta t(\mathbf{g}\cdot\mathbf{x}))$$
   $$=(\mathbf{x}-\mathbf{g}^{n+1})Z(\mathbf{g}\cdot\mathbf{x})(\gamma(1+\beta)+h_1(\mathbf{x})+\beta h_1(\mathbf{g}\cdot\mathbf{x}))(\gamma(1+\beta)+h_2(\mathbf{x})+\beta h_2(\mathbf{g}\cdot\mathbf{x}))$$
   (c) $L_{n+1}(\mathbf{x})(h_1(\mathbf{x})-h_2(\mathbf{g}\cdot\mathbf{x})) = 0$.
   (d) $L_{n+1}(\mathbf{x})(Z(\mathbf{x})-1) = 0$.

   and outputs acc iff all checks hold.

1. Only commits to 3 auxiliary polynomials of degree n. -> 3n+3
2. The polynomial computed in H-ranged equality is of degree at most (3n+1)-(n+1)=2n. -> 2n+1

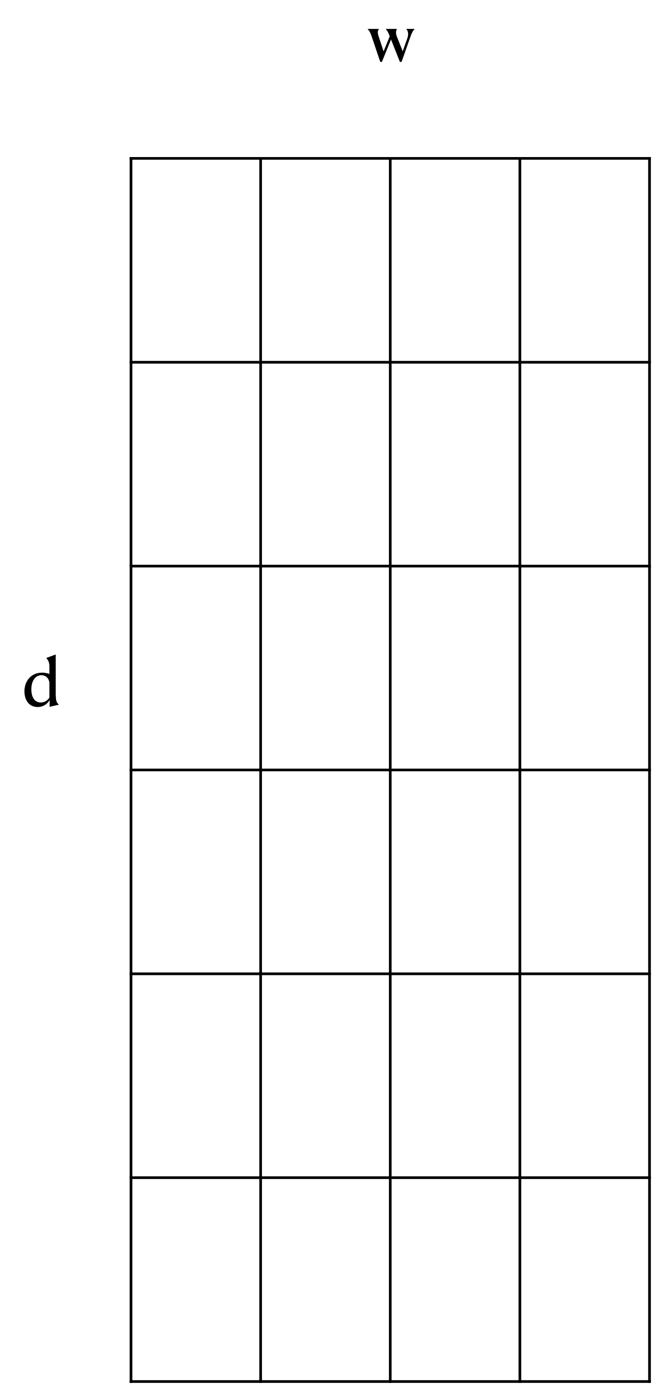
https://eprint.iacr.org/2020/315.pdf

Plookup: details

Vector lookups and multiple tables

Vector lookups:

Suppose we have several witness polynomials $f_1, \ldots, f_w \in \mathbb{F}_{<n}[X]$, and a table of values $t^* \in (\mathbb{F}^w)^d$. We wish to check that for each $j \in [n]$ $(f_1(\mathbf{g}^j), \ldots, f_w(\mathbf{g}^j)) \in t^*$. We can use randomization to efficiently reduce to the case of Section 3.



Main idea: view each row as a univariate polynomial in α.

The verifier will choose random $\alpha \in \mathbb{F}$.
Then we will define $t := \sum_{i \in [w]} \alpha^i t_i$, $f := \sum_{i \in [w]} \alpha^i f_i$.

soundness probability is $d \cdot w / |\mathbb{F}|$

for some $j \in [n]$, $(f_1(\mathbf{g}^j), \ldots, f_w(\mathbf{g}^j)) \notin t^*$.

Question: What is the difference between the univariate poly. and multivariate poly. in randomization ?

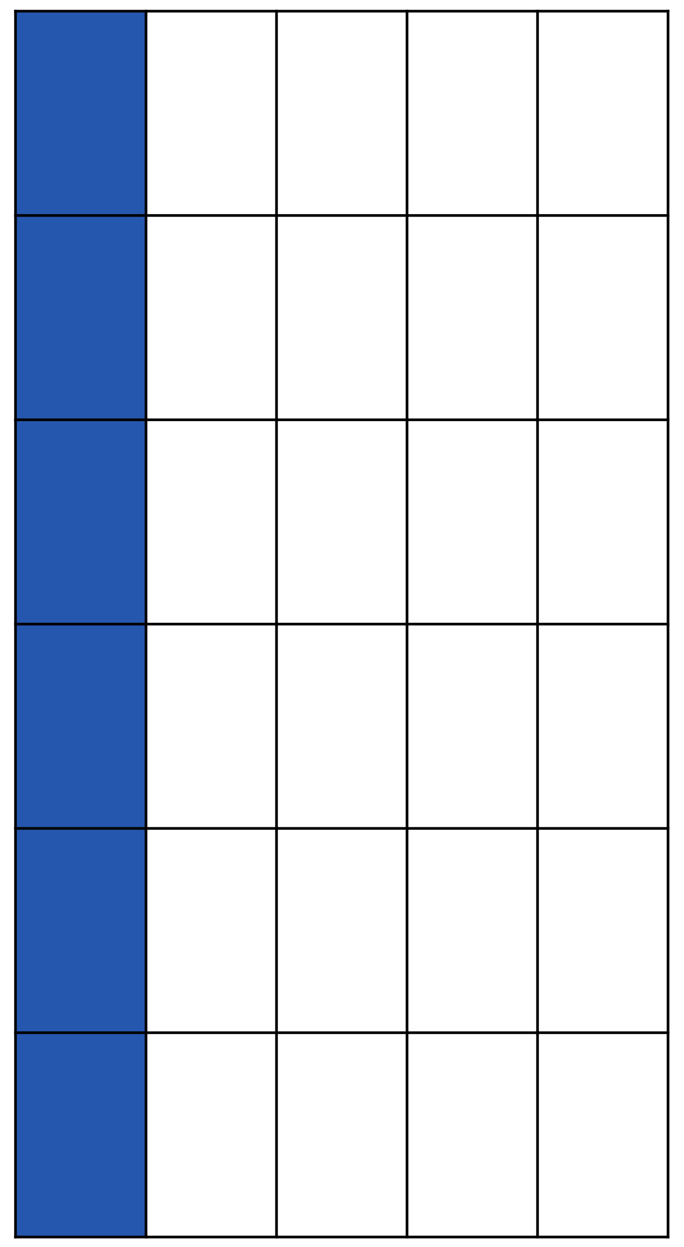
Application: Key-value setting (XOR example)

As alluded to in the introduction, a natural use case for this vector lookup primitive is a key-value setting, where we have a function f with $w-1$ inputs, and wish to verify a vector is of the form $(x_1, \ldots, x_{w-1}, f(x_1, \ldots, x_{w-1}))$ for some input $(x_1, \ldots, x_{w-1})$.

Multiple tables:

Suppose further that we have in fact have multiple tables $t_1^*, \ldots, t_\ell^*$ and for each $i \in [n]$ wish to check that for some predefined $j = j(i) \in [\ell]$ $(f_1(\mathbf{g}^i), \ldots, f_w(\mathbf{g}^i)) \in t_j^*$. We

Main idea: add a column specifying the table index

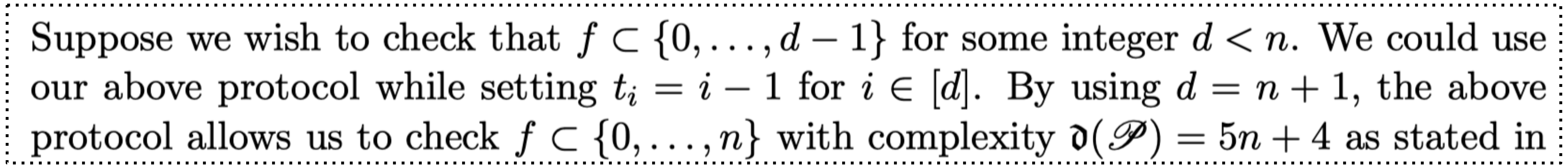


https://eprint.iacr.org/2020/315.pdf

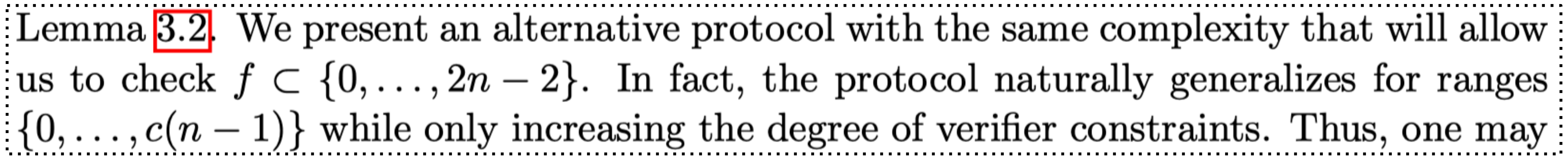
Plookup: details

Optimization for continuous ranges

Tables with continuous ranges:


Suppose we wish to check that $f \subset \{0, \ldots, d-1\}$ for some integer $d < n$. We could use our above protocol while setting $t_i = i - 1$ for $i \in [d]$. By using $d = n + 1$, the above protocol allows us to check $f \subset \{0, \ldots, n\}$ with complexity $\mathfrak{d}(\mathscr{P}) = 5n + 4$ as stated in

Optimizations: protocol is parameterized by c


Lemma 3.2. We present an alternative protocol with the same complexity that will allow us to check $f \subset \{0, \ldots, 2n-2\}$. In fact, the protocol naturally generalizes for ranges $\{0, \ldots, c(n-1)\}$ while only increasing the degree of verifier constraints. Thus, one may

Recall:

A simpler reduction using two multiset checks:

1. Between s and (f, t) implies $f \subset s$
2. Between s' and t''

Main idea: the constraints of S1-S3 imply $s \subset t$

Simple case: $c = 1$

Let $t_i = i - 1$ for $i \in [n]$.

1. $s_1 = 0$
2. $s_{2n+1} = n - 1$
3. $s_{i+1} - s_i \leq 1$ for each $i \in [2n]$
4. Multiset check between s and (f, t)

General case: c

Let $t_i = c(i - 1)$ for $i \in [n]$.

1. $s_1 = 0$
2. $s_{2n+1} = c(n - 1)$
3. $s_{i+1} - s_i \leq c$ for each $i \in [2n]$
4. Multiset check between s and (f, t)

Constraint $s_{i+1} - s_i \leq c$:
can be plugged in its difference into the degree $c + 1$ poly. that vanishes on $\{0,\ldots,c\}$.

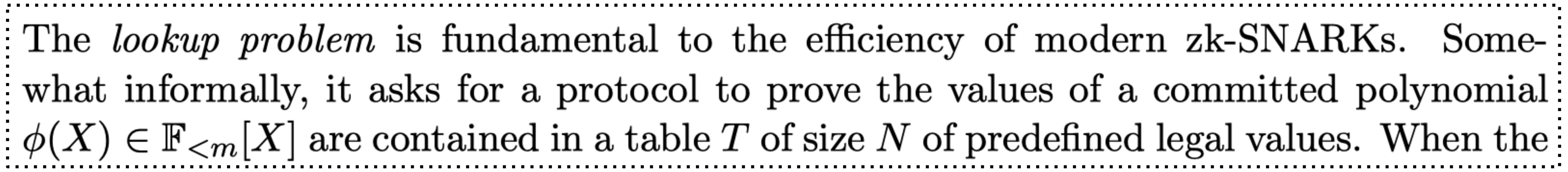
complexity: $(3 + c)n + 2$

https://eprint.iacr.org/2022/1530

logUp

Comparison to previous work

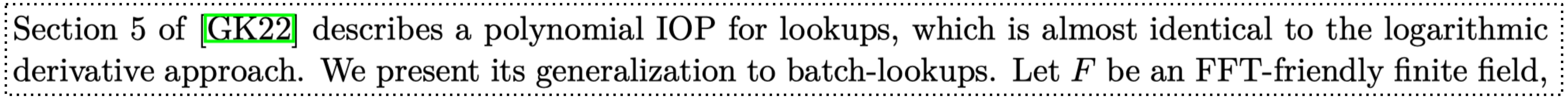
Lookup problem: a column of size m and a table of size N


The *lookup problem* is fundamental to the efficiency of modern zk-SNARKs. Somewhat informally, it asks for a protocol to prove the values of a committed polynomial $\phi(X) \in \mathbb{F}_{<m}[X]$ are contained in a table T of size N of predefined legal values. When the

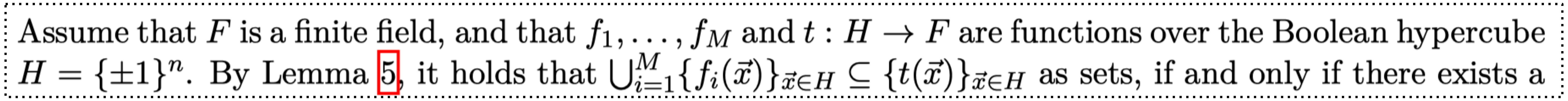
- Plookup [GW20]: quasilinear in both m and N in polynomial-IOP (univariate)

One intriguing question: whether the dependence on N could be made sub-linear after performing a preprocessing step for the table T.

- Caulk [ZBK+22]: $O(m^2 + m \log N)$
- Caulk+[PK22]: $O(m^2)$ getting rid of the dependence on table size completely.
- Flookup[GK22]: quasilinear in m and has no dependence on N after preprocessing step.
- logup[This]: batch-lookups over hypercube


Section 5 of [GK22] describes a polynomial IOP for lookups, which is almost identical to the logarithmic derivative approach. We present its generalization to batch-lookups. Let F be an FFT-friendly finite field,

Batch-lookup problem: M column of size $|H|$ and a table of size $|H|$


Assume that F is a finite field, and that $f_1, \ldots, f_M$ and $t : H \to F$ are functions over the Boolean hypercube $H = \{\pm 1\}^n$. By Lemma 5, it holds that $\bigcup_{i=1}^{M} \{f_i(\vec{x})\}_{\vec{x} \in H} \subseteq \{t(\vec{x})\}_{\vec{x} \in H}$ as sets, if and only if there exists a

https://eprint.iacr.org/2022/1530

logUp

Main idea

Recall the exact problem in [BCG+18]

logarithmic derivatives: Given a sequence of field elements $(a_i)_{i=1}^N$ and another sequence $(t_j)_{j=1}^M$, then $\{a_i : i = 1, \ldots, N\} \subseteq \{t_j : j = 1, \ldots, M\}$ as sets, if and only if there exists a sequence of field elements $(m_j)_{j=1}^M$ (the multiplicities) such that

$$\prod_{i \in [N]} (X - a_i) = \prod_{j \in [M]} (X - t_j)^{m_j} \xrightarrow[\text{Turns products into sums of reciprocals.}]{\text{logarithmic derivatives}} \sum_{i=1}^{N} \frac{1}{X - a_i} = \sum_{j=1}^{M} \frac{m_j}{X - t_j}.$$

- Reduced to one grand product check with product = 1
- H-ranged polynomial-IOP (H is a multiplicative subgroup)
- In the univariate setting

- Reduced to sumcheck with sum = 0
- Lagrange-IOP over hypercube $H = \{\pm 1\}^n$
- In the multivariate setting

https://eprint.iacr.org/2022/1530

logUp

Preliminaries: Lagrange Kernel

Notations:

Let F denote a finite field, and F^* its multiplicative group. Throughout the document we regard the boolean hypercube $H = \{\pm 1\}^n$ as a multiplicative subgroup of $(F^*)^n$. For a multivariate function $f(X_1, \ldots, X_n)$, we will often use the vector notation $\vec{X} = (X_1, \ldots, X_n)$ for its arguments, writing $f(\vec{X}) := f(X_1, \ldots, X_n)$.

Lagrange Kernel of H:

The *Lagrange kernel* of H is the multilinear polynomial

$$L_H(\vec{X}, \vec{Y}) = \frac{1}{2^n} \cdot \prod_{j=1}^{n} (1 + X_j \cdot Y_j). \tag{1}$$

Notice that $L_H(\vec{X}, \vec{Y})$ is symmetric in $\vec{X}$ and $\vec{Y}$, i.e. $L_H(\vec{X}, \vec{Y}) = L_H(\vec{Y}, \vec{X})$, and that (1) is evaluated within only $\mathcal{O}(\log |H|)$ field operations. Whenever $\vec{y} \in H$ we have that $L_H(\vec{X}, \vec{y})$ is the Lagrange polynomial on H, which is the unique multilinear polynomial which satisfies $L_H(\vec{x}, \vec{y}) = 1$ at $\vec{x} = \vec{y}$, and zero elsewhere on H. In particular for a function $f : H \to F$ the inner product evaluation formula

$$\langle f, L_H(\,.\,, \vec{y}) \rangle_H := \sum_{\vec{x} \in H} f(\vec{x}) \cdot L_H(\vec{x}, \vec{y}) = f(\vec{y}).$$

Sketch of properties:

1. symmetric
2. Lagrange polynomial on H associated with $\vec{y} \in H$
3. **Point evaluation** on $\vec{y} \in H$ is reduced on sum over hypercube

Unique Multilinear Extension

Lemma 1. *Let $p(\vec{X})$ be the unique multilinear extension of $f : H \to F$. Then for every $\vec{y} \in F^n$,*

$$\langle f, L_H(\,.\,, \vec{y}) \rangle_H = \sum_{x \in H} f(\vec{x}) \cdot L_H(\vec{x}, \vec{y}) = p(\vec{y}). \tag{2}$$

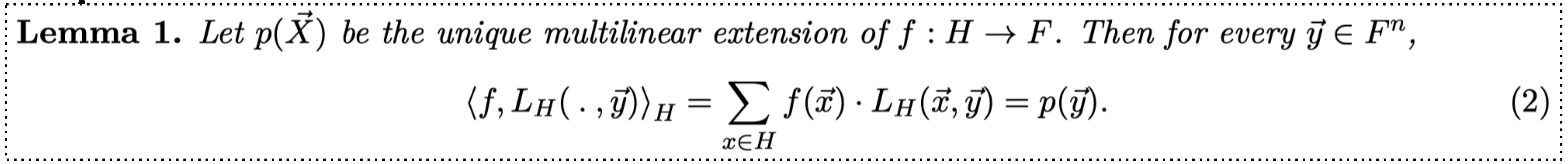
1. Lagrange polynomial on H associated with $\vec{y} \in F^n$
2. **Point evaluation** on $\vec{y} \in F^n$ is reduced on sum over hypercube

https://eprint.iacr.org/2022/1530

logUp

Preliminaries: Lagrange IOP over the boolean hypercube $H = \{\pm 1\}^n$

Unique Multilinear Extension



Lemma 1. *Let $p(\vec{X})$ be the unique multilinear extension of $f : H \to F$. Then for every $\vec{y} \in F^n$,*

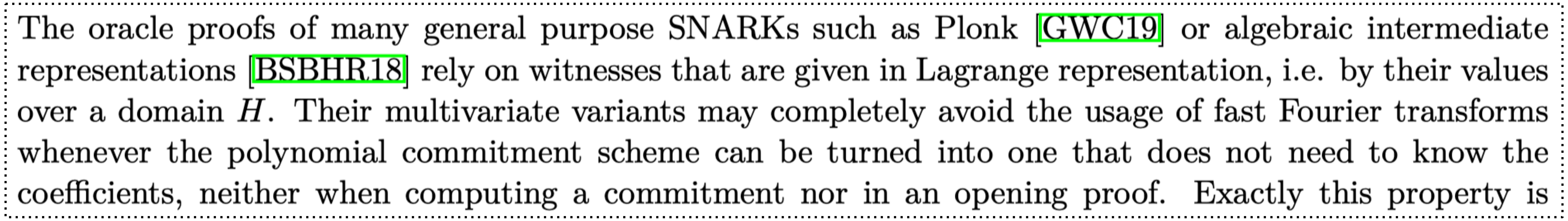
$$\langle f, L_H(\,.\,,\vec{y})\rangle_H = \sum_{x\in H} f(\vec{x}) \cdot L_H(\vec{x},\vec{y}) = p(\vec{y}). \qquad (2)$$

1. Lagrange polynomial on H associated with $\vec{y} \in F^n$
2. **Point evaluation** on $\vec{y} \in F^n$ is reduced on sum over hypercube

Lagrange IOP between prover and verifier:

1. verifier sends a random challenge $y \in F^n$
2. prover computes one or several functions over the boolean hypercube —— vector of $f(\vec{x})$ for all $x \in H$
   and gives the verifier **oracle access** to them.
3. verifier is allowed to **query** the oracles for **their inner products** with the Lagrange kernel $L_H(\,\cdot\,,\vec{y})$ associated with any $y \in F^n$
   —— vector of $L_H(\vec{x},\vec{y})$ for all $x \in H$

Motivation of Lagrange IOP: (or advantage of multivariate setting)



The oracle proofs of many general purpose SNARKs such as Plonk [GWC19] or algebraic intermediate representations [BSBHR18] rely on witnesses that are given in Lagrange representation, i.e. by their values over a domain H. Their multivariate variants may completely avoid the usage of fast Fourier transforms whenever the polynomial commitment scheme can be turned into one that does not need to know the coefficients, neither when computing a commitment nor in an opening proof. Exactly this property is

https://eprint.iacr.org/2022/1530

logUp

Preliminaries: Lagrange IOP over the boolean hypercube $H = \{\pm 1\}^n$

Unique Multilinear Extension

Lemma 1. *Let $p(\vec{X})$ be the unique multilinear extension of $f : H \to F$. Then for every $\vec{y} \in F^n$,*

$$\langle f, L_H(\,.\,,\vec{y})\rangle_H = \sum_{x\in H} f(\vec{x}) \cdot L_H(\vec{x},\vec{y}) = p(\vec{y}). \quad (2)$$

1. Lagrange polynomial on H associated with $\vec{y} \in F^n$
2. **Point evaluation** on $\vec{y} \in F^n$ is reduced on sum over hypercube

Lagrange IOP between prover and verifier:

1. verifier sends a random challenge $y \in F^n$
2. prover computes one or several functions over the boolean hypercube —— vector of $f(\vec{X})$ for all $X \in H$ and gives the verifier **oracle access** to them.
3. verifier is allowed to **query** the oracles for **their inner products** with the Lagrange kernel $L_H(\,\cdot\,,\vec{y})$ associated with any $y \in F^n$ —— vector of $L_H(\vec{X},\vec{y})$ for all $X \in H$

The domain evaluation of $L_H(\vec{X},\vec{y})$ over H can be computed in $O(|H|)$ rather than $O(\log|H| \cdot |H|)$

Main idea: split the computation into $\log|H|$ rounds

- $p_k(X_1, \ldots, X_k, y_1, \ldots, y_k) = \frac{1}{2^n}\prod_{j=1}^{k}(1 + X_j \cdot y_j)$ for 2^k values
- $p_{k+1} = p_k \cdot (1 + X_{k+1} \cdot y_{k+1})$ for 2^{k+1} values

logUp

Preliminaries: Sumcheck for a batch of polynomials

Claims for $i = 0,\dots,L$

- $\sum p_i(X_1,\dots,X_n) = s_i$

Reduced to a single sum

viewed as a multivariate polynomial

Claims for a single sum: $s_0 + \sum_{i=1}^{L} \lambda_i s_i$

$L+1$ sumcheck protocol for $p_0,\dots,p_L$

1 sumcheck protocol for $\bar{p}(X_1,\dots,X_n) = p_0(X_1,\dots,X_n) + \sum^{L} \lambda_i \cdot p_i(X_1,\dots,X_n)$

Soundness probability: $\varepsilon_{sumcheck} \leq \frac{1}{|F|} \cdot \left(1 + \sum_{i=1}^{n} \deg_{X_i} p(X_1,\dots,X_n)\right)$.

- randomization with soundness prob. $1/F|$
- a single sumcheck protocol with soundness prob.

$$\varepsilon_{sumcheck} \leq \frac{1}{|F|} \cdot \sum_{i=1}^{n} \deg_{X_i} p(X_1,\dots,X_n).$$

https://eprint.iacr.org/2022/1530

logUp

Preliminaries: Sumcheck

Let us discuss the prover cost of the sumcheck protocol for the case that $p(\vec{X}) = p(X_1, \dots, X_n)$ is of the form

$$p(\vec{X}) = Q(w_1(\vec{X}), \dots, w_\nu(\vec{X})),$$

with each $\boxed{w_i(\vec{X}) \in F[X_1, \dots, X_n] \text{ being multilinear,}}$ and

$$Q(Y_1, \dots, Y_\nu) = \sum_{(i_1,\dots,i_\nu)\in\{0,1\}^\nu} c_{i_1,\dots,i_\nu} \cdot Y_1^{i_1} \cdots Y_\nu^{i_\nu}$$

convenient for counting the degree

is $\boxed{\text{a multivariate polynomial}}$ having (a typically low) absolute degree d. We denote the arithmetic complex-

Notations:
- A: time in field additions/subtractions
- M: time in field multiplications
- $|Q|_A$: # field adds for Q
- $|Q|_M$: # field multiplications for Q

Computational Cost in each step $i = 1, \dots, n$ of computing the evaluation of a univariate polynomial $s_i(X)$ over a set $D \supseteq \{\pm 1\}$ of size $d + 1$.

domain evaluations over each $H_i = \{\pm 1\}^{n-i}$ for each $s_i(z) = \sum_{(x_{i+1},\dots,x_n)\in H_i} Q(r_1, \dots, r_{i-1}, z, x_{i+1}, \dots, x_n)$.

$$w_j(r_1, \dots, r_{i-1}, \pm 1, X_{i+1}, \dots, X_n) \longrightarrow \begin{matrix} w_j(r_1, \dots, r_{i-1}, z, X_{i+1}, \dots, X_n), \\ w_j(r_1, \dots, r_{i-1}, r_i, X_{i+1}, \dots, X_n) \end{matrix}$$

Question ?

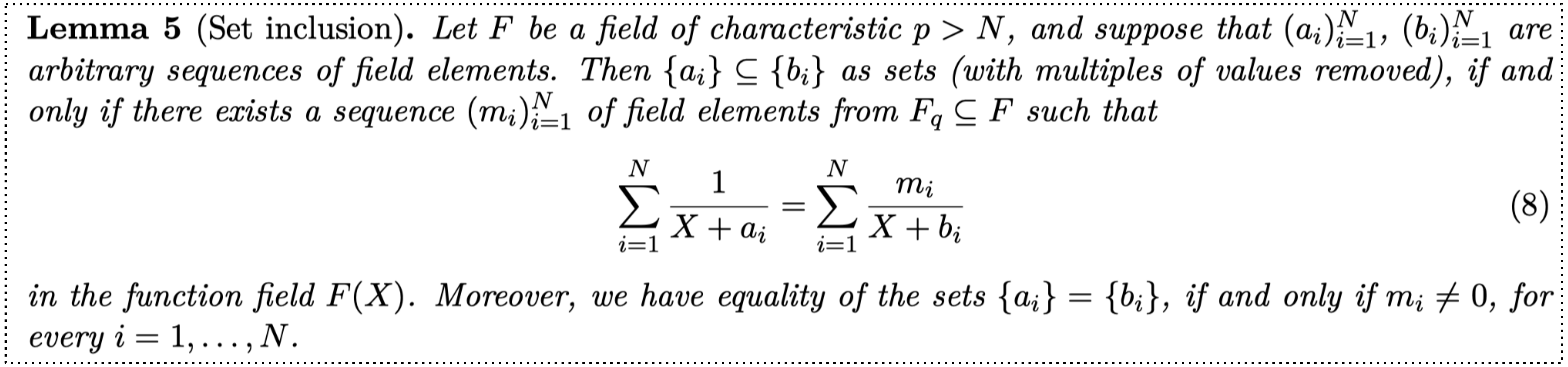
Step i: $\boxed{\nu \cdot |H_i| \cdot \mathsf{A}} + \nu \cdot (|D| - 1) \cdot |H_i| \cdot (\mathsf{M} + \mathsf{A}) + |D| \cdot |H_i| \cdot (|Q|_\mathsf{M} \cdot \mathsf{M} + |Q|_\mathsf{A} \cdot \mathsf{A}) + |D| \cdot |H_i| \cdot \mathsf{A},$

Overall cost: $|H| \cdot (d + 1) \cdot ((\nu + |Q|_\mathsf{M}) \cdot \mathsf{M} + (\nu + |Q|_\mathsf{A}) \cdot \mathsf{A})$

https://eprint.iacr.org/2022/1530

logUp

Main lemma (set inclusion)

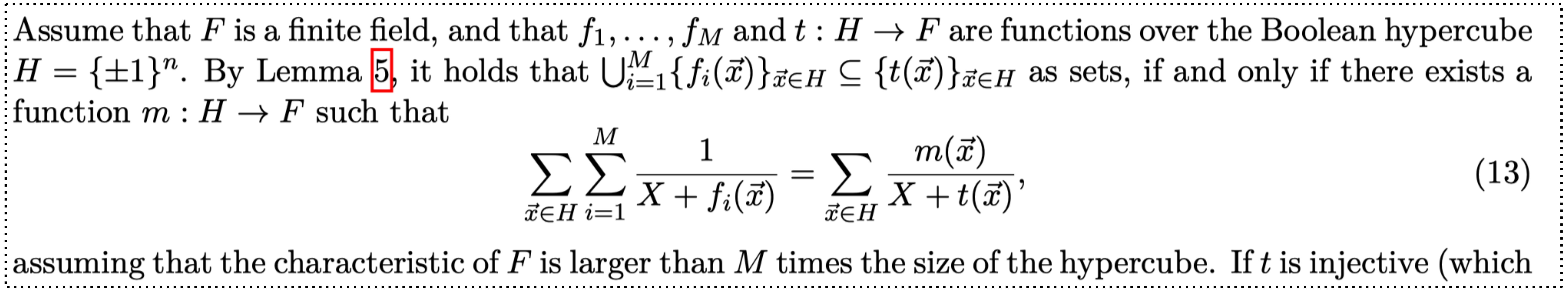


Lemma 5 (Set inclusion). *Let F be a field of characteristic $p > N$, and suppose that $(a_i)_{i=1}^N$, $(b_i)_{i=1}^N$ are arbitrary sequences of field elements. Then $\{a_i\} \subseteq \{b_i\}$ as sets (with multiples of values removed), if and only if there exists a sequence $(m_i)_{i=1}^N$ of field elements from $F_q \subseteq F$ such that*

$$\sum_{i=1}^{N} \frac{1}{X + a_i} = \sum_{i=1}^{N} \frac{m_i}{X + b_i} \tag{8}$$

in the function field $F(X)$. Moreover, we have equality of the sets $\{a_i\} = \{b_i\}$, if and only if $m_i \neq 0$, for every $i = 1, \ldots, N$.

Extension for Batch-column Lookup

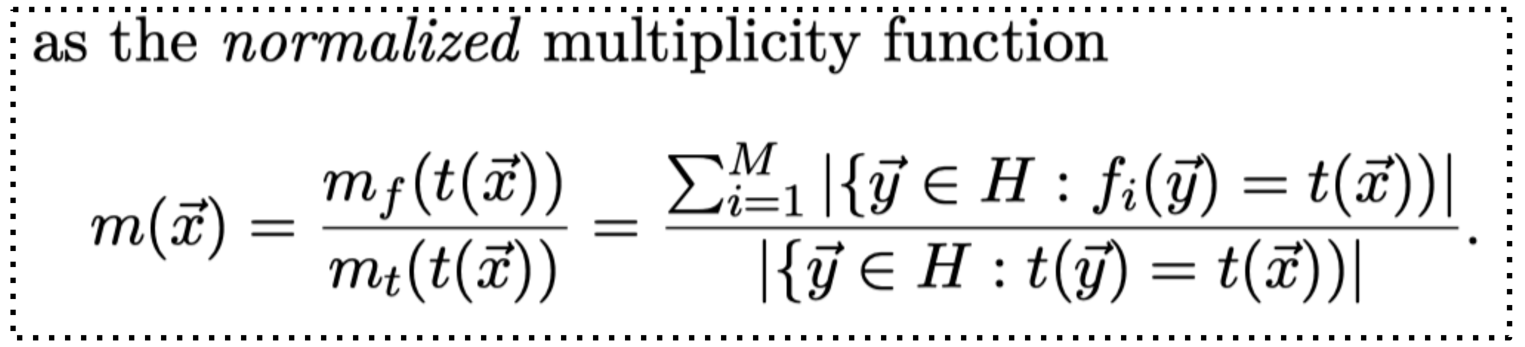


Assume that F is a finite field, and that $f_1, \ldots, f_M$ and $t : H \to F$ are functions over the Boolean hypercube $H = \{\pm 1\}^n$. By Lemma 5, it holds that $\bigcup_{i=1}^M \{f_i(\vec{x})\}_{\vec{x} \in H} \subseteq \{t(\vec{x})\}_{\vec{x} \in H}$ as sets, if and only if there exists a function $m : H \to F$ such that

$$\sum_{\vec{x} \in H} \sum_{i=1}^{M} \frac{1}{X + f_i(\vec{x})} = \sum_{\vec{x} \in H} \frac{m(\vec{x})}{X + t(\vec{x})}, \tag{13}$$

assuming that the characteristic of F is larger than M times the size of the hypercube. If t is injective (which

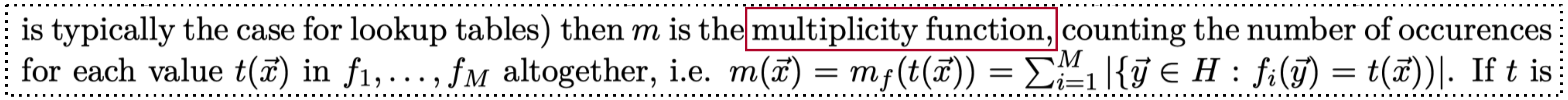
Or not one-to-one:



as the *normalized* multiplicity function

$$m(\vec{x}) = \frac{m_f(t(\vec{x}))}{m_t(t(\vec{x}))} = \frac{\sum_{i=1}^M |\{\vec{y} \in H : f_i(\vec{y}) = t(\vec{x})\}|}{|\{\vec{y} \in H : t(\vec{y}) = t(\vec{x})\}|}.$$

t is injective (one-to-one):



is typically the case for lookup tables) then m is the $\boxed{\text{multiplicity function,}}$ counting the number of occurences for each value $t(\vec{x})$ in $f_1, \ldots, f_M$ altogether, i.e. $m(\vec{x}) = m_f(t(\vec{x})) = \sum_{i=1}^M |\{\vec{y} \in H : f_i(\vec{y}) = t(\vec{x})\}|$. If t is

https://eprint.iacr.org/2022/1530

logUp

Batch-column Lookup

Assume that F is a finite field, and that $f_1, \dots, f_M$ and $t : H \to F$ are functions over the Boolean hypercube $H = \{\pm 1\}^n$. By Lemma 5, it holds that $\bigcup_{i=1}^{M} \{f_i(\vec{x})\}_{\vec{x} \in H} \subseteq \{t(\vec{x})\}_{\vec{x} \in H}$ as sets, if and only if there exists a function $m : H \to F$ such that

$$\sum_{\vec{x} \in H} \sum_{i=1}^{M} \frac{1}{X + f_i(\vec{x})} = \sum_{\vec{x} \in H} \frac{m(\vec{x})}{X + t(\vec{x})}, \tag{13}$$

assuming that the characteristic of F is larger than M times the size of the hypercube. If t is injective (which

Rational Equality:

Given a random challenge $x \leftarrow_{\$} F$ from the verifier, the prover shows that the rational identity (13) holds at $X = x$, i.e.

$$\sum_{\vec{x} \in H} \sum_{i=1}^{M} \frac{1}{x + f_i(\vec{x})} - \frac{m(\vec{x})}{x + t(\vec{x})} = 0, \tag{15}$$

But sumcheck protocol only works for polynomial one.

Main Idea:

- splits the sum of $M + 1$ terms into partial sums of roughly same number of terms ℓ.
- provides multilinear help functions for each sum
- each multilinear help function is subject to a domain identity

$$\frac{m(x)}{x+t(\vec{x})} - \frac{1}{x+f_1(\vec{x})} - \dots - \frac{1}{x+f_M(\vec{x})},$$

https://eprint.iacr.org/2022/1530

logUp

Main idea of batch-column lookup

Main Idea:

- splits the sum of $M+1$ terms into partial sums of roughly same number of terms ℓ.
- provides multilinear help functions for each sum
- each multilinear help function is subject to a domain identity

$$m_0(\vec{x}) = m(\vec{x}), \quad \varphi_0(\vec{x}) = x + t(\vec{x}),$$
$$m_i(\vec{x}) = -1, \quad \varphi_i(\vec{x}) = x + f_i(\vec{x}), \quad i = 1, \ldots, M.$$

Let ℓ be the chosen sum size, $[0, M] = \bigcup_{k=1}^{K} I_k$ the decomposition of $[0, M]$ into $K = \left\lceil \frac{M+1}{\ell} \right\rceil$ subintervals $I_k = [(k-1)\cdot\ell, k\cdot\ell) \cap [0, M]$, $k = 1, \ldots, K$. Let

$$h_k(\vec{x}) = \sum_{i \in I_k} \frac{m_i(\vec{x})}{\varphi_i(\vec{x})}, \quad k = 1, \ldots, K, \tag{16}$$

be the respective partial sum of consecutive terms in the overall expression $\frac{m(x)}{x+t(\vec{x})} - \frac{1}{x+f_1(\vec{x})} - \ldots - \frac{1}{x+f_M(\vec{x})}$,

The prover provides the oracles for $h_1, \ldots, h_k$, subject to

$$\sum_{\vec{x} \in H} h_1(\vec{x}) + \ldots + h_K(\vec{x}) = 0,$$

and the domain identities

$$h_k(\vec{x}) \cdot \prod_{i \in I_k} \varphi_i(\vec{x}) = \sum_{i \in I_k} m_i(\vec{x}) \cdot \prod_{j \in I_k \setminus \{i\}} \varphi_j(\vec{x})$$

Reduce domain identity to sumchecks:

- $h(x) = 0$ for $x \in H = \{\pm 1\}^n$
- MLE $p(z) = \sum_{x \in H} L_H(x, z) \cdot h(x)$

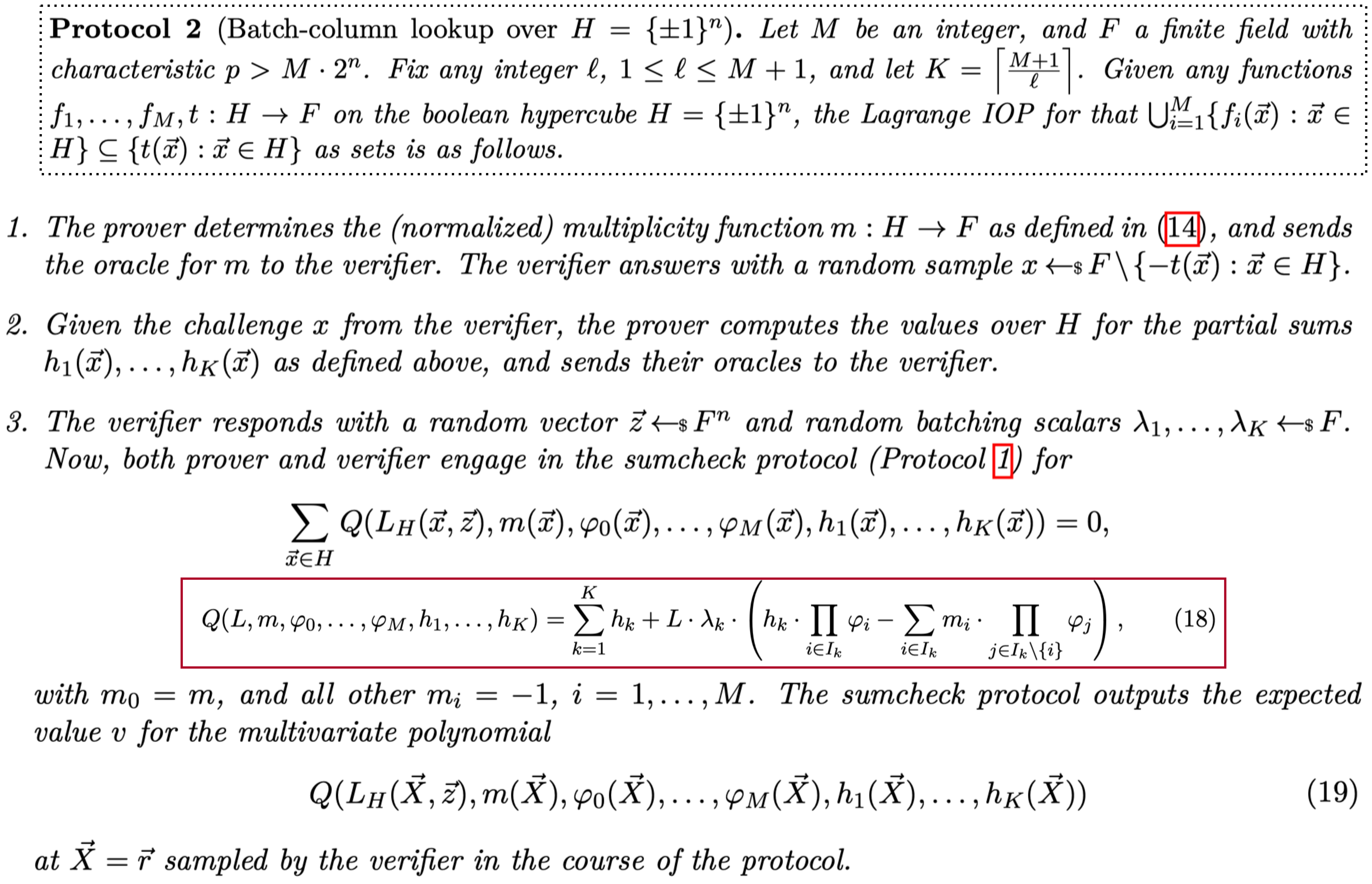
1. verifier sends a random challenge $z \in F^n$
2. run sumcheck for $L_H(X, z) \cdot h(X)$

Reduce $K+1$ sumchecks to a single one.

https://eprint.iacr.org/2022/1530

logUp

Protocol for batch-column lookup



Protocol 2 (Batch-column lookup over $H = \{\pm 1\}^n$). *Let M be an integer, and F a finite field with characteristic $p > M \cdot 2^n$. Fix any integer ℓ, $1 \leq \ell \leq M+1$, and let $K = \left\lceil \frac{M+1}{\ell} \right\rceil$. Given any functions $f_1, \ldots, f_M, t : H \to F$ on the boolean hypercube $H = \{\pm 1\}^n$, the Lagrange IOP for that $\bigcup_{i=1}^{M} \{f_i(\vec{x}) : \vec{x} \in H\} \subseteq \{t(\vec{x}) : \vec{x} \in H\}$ as sets is as follows.*

1. *The prover determines the (normalized) multiplicity function $m : H \to F$ as defined in (14), and sends the oracle for m to the verifier. The verifier answers with a random sample $x \leftarrow_\$ F \setminus \{-t(\vec{x}) : \vec{x} \in H\}$.*

2. *Given the challenge x from the verifier, the prover computes the values over H for the partial sums $h_1(\vec{x}), \ldots, h_K(\vec{x})$ as defined above, and sends their oracles to the verifier.*

3. *The verifier responds with a random vector $\vec{z} \leftarrow_\$ F^n$ and random batching scalars $\lambda_1, \ldots, \lambda_K \leftarrow_\$ F$. Now, both prover and verifier engage in the sumcheck protocol (Protocol 1) for*

$$\sum_{\vec{x} \in H} Q(L_H(\vec{x}, \vec{z}), m(\vec{x}), \varphi_0(\vec{x}), \ldots, \varphi_M(\vec{x}), h_1(\vec{x}), \ldots, h_K(\vec{x})) = 0,$$

$$\boxed{Q(L, m, \varphi_0, \ldots, \varphi_M, h_1, \ldots, h_K) = \sum_{k=1}^{K} h_k + L \cdot \lambda_k \cdot \left(h_k \cdot \prod_{i \in I_k} \varphi_i - \sum_{i \in I_k} m_i \cdot \prod_{j \in I_k \setminus \{i\}} \varphi_j \right), \qquad (18)}$$

with $m_0 = m$, and all other $m_i = -1$, $i = 1, \ldots, M$. The sumcheck protocol outputs the expected value v for the multivariate polynomial

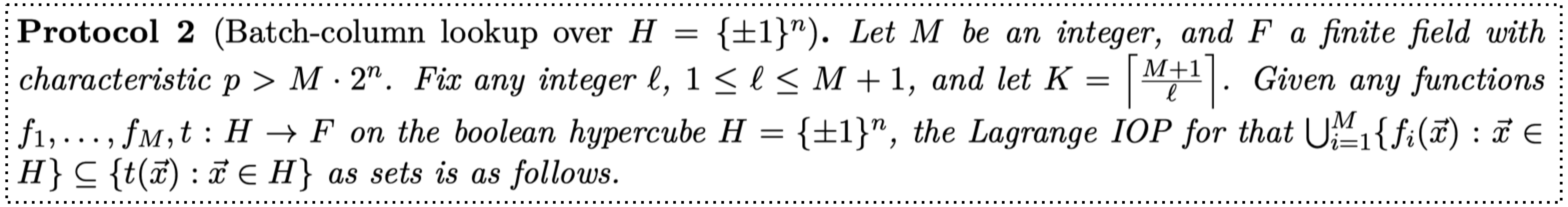
$$Q(L_H(\vec{X}, \vec{z}), m(\vec{X}), \varphi_0(\vec{X}), \ldots, \varphi_M(\vec{X}), h_1(\vec{X}), \ldots, h_K(\vec{X})) \qquad (19)$$

at $\vec{X} = \vec{r}$ sampled by the verifier in the course of the protocol.

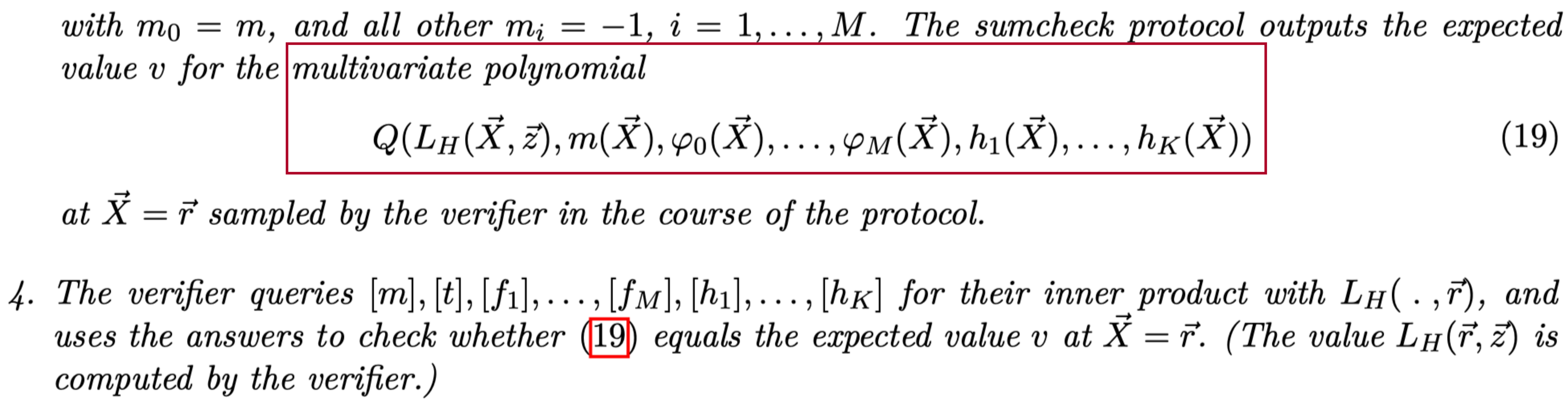
https://eprint.iacr.org/2022/1530

logUp

Protocol for batch-column lookup



Protocol 2 (Batch-column lookup over $H = \{\pm 1\}^n$). *Let M be an integer, and F a finite field with characteristic $p > M \cdot 2^n$. Fix any integer ℓ, $1 \leq \ell \leq M+1$, and let $K = \left\lceil \frac{M+1}{\ell} \right\rceil$. Given any functions $f_1, \ldots, f_M, t : H \to F$ on the boolean hypercube $H = \{\pm 1\}^n$, the Lagrange IOP for that $\bigcup_{i=1}^{M} \{f_i(\vec{x}) : \vec{x} \in H\} \subseteq \{t(\vec{x}) : \vec{x} \in H\}$ as sets is as follows.*



with $m_0 = m$, and all other $m_i = -1$, $i = 1, \ldots, M$. The sumcheck protocol outputs the expected value v for the multivariate polynomial

$$\boxed{Q(L_H(\vec{X}, \vec{z}), m(\vec{X}), \varphi_0(\vec{X}), \ldots, \varphi_M(\vec{X}), h_1(\vec{X}), \ldots, h_K(\vec{X}))} \tag{19}$$

at $\vec{X} = \vec{r}$ sampled by the verifier in the course of the protocol.

4. *The verifier queries $[m], [t], [f_1], \ldots, [f_M], [h_1], \ldots, [h_K]$ for their inner product with $L_H(\,.\,, \vec{r})$, and uses the answers to check whether ($\boxed{19}$) equals the expected value v at $\vec{X} = \vec{r}$. (The value $L_H(\vec{r}, \vec{z})$ is computed by the verifier.)*

https://eprint.iacr.org/2022/1530

logUp

Protocol for batch-column lookup

Protocol 2 (Batch-column lookup over $H = \{\pm 1\}^n$). *Let M be an integer, and F a finite field with characteristic $p > M \cdot 2^n$. Fix any integer ℓ, $1 \leq \ell \leq M+1$, and let $K = \left\lceil \frac{M+1}{\ell} \right\rceil$. Given any functions $f_1, \ldots, f_M, t : H \to F$ on the boolean hypercube $H = \{\pm 1\}^n$, the Lagrange IOP for that $\bigcup_{i=1}^{M} \{f_i(\vec{x}) : \vec{x} \in H\} \subseteq \{t(\vec{x}) : \vec{x} \in H\}$ as sets is as follows.*

1. *The prover determines the (normalized) multiplicity function $m : H \to F$ as defined in ([14]), and sends the oracle for m to the verifier. The verifier answers with a random $\boxed{\text{sample } x \leftarrow_{\$} F \setminus \{-t(\vec{x}) : \vec{x} \in H\}.}$*
2. *Given the challenge x from the verifier, the prover computes the values over H for the partial sums $h_1(\vec{x}), \ldots, h_K(\vec{x})$ as defined above, and sends their oracles to the verifier.*
3. *The verifier responds with a random vector $\vec{z} \leftarrow_{\$} F^n$ and random batching scalars $\lambda_1, \ldots, \lambda_K \leftarrow_{\$} F$. Now, both prover and verifier engage in the sumcheck protocol (Protocol [1]) for*

$$\sum_{\vec{x} \in H} Q(L_H(\vec{x}, \vec{z}), m(\vec{x}), \varphi_0(\vec{x}), \ldots, \varphi_M(\vec{x}), h_1(\vec{x}), \ldots, h_K(\vec{x})) = 0,$$

$$\boxed{Q(L, m, \varphi_0, \ldots, \varphi_M, h_1, \ldots, h_K) = \sum_{k=1}^{K} h_k + L \cdot \lambda_k \cdot \left(h_k \cdot \prod_{i \in I_k} \varphi_i - \sum_{i \in I_k} m_i \cdot \prod_{j \in I_k \setminus \{i\}} \varphi_j \right), \qquad (18)}$$

with $m_0 = m$, and all other $m_i = -1$, $i = 1, \ldots, M$. The sumcheck protocol outputs the expected value v for the multivariate polynomial

$$Q(L_H(\vec{X}, \vec{z}), m(\vec{X}), \varphi_0(\vec{X}), \ldots, \varphi_M(\vec{X}), h_1(\vec{X}), \ldots, h_K(\vec{X})) \qquad (19)$$

at $\vec{X} = \vec{r}$ sampled by the verifier in the course of the protocol.

Remark.

$\boxed{\text{condition } x \notin \{-t(\vec{x})\}_{\vec{x} \in H}}$

Two methods to handle this.

1. omit the constraint on x, letting the verifier sample x and prover set h_0 arbitrary.
   —> with completeness error prob. $|H|/|F|$
2. modify the identity for h_0 over H

$$\left(h_0 \cdot \prod_{i \in I_0} \varphi_i - \sum_{i \in I_0} m_k \cdot \prod_{j \neq i} \varphi_j \right) \cdot \varphi_0 = 0$$

which imposes no condition on h_0 whenever $\phi_0(\vec{x}) = 0$.

$$m_0(\vec{x}) = m(\vec{x}), \quad \varphi_0(\vec{x}) = x + t(\vec{x}),$$
$$m_i(\vec{x}) = -1, \quad \varphi_i(\vec{x}) = x + f_i(\vec{x}), \quad i = 1, \ldots, M.$$

Question:

why not consider the condition $x \notin \{f_i(\vec{x})\}_{x \in H}$

$\boxed{\text{As for completeness, honest prover has guaranteed } f \subset t.}$

https://eprint.iacr.org/2022/1530

logUp

Protocol for batch-column lookup: variants

Variant 1: single-column lookup

Let us point out two variations of Protocol 2. In the single-column case $M = 1$ the lookup argument can be turned into a multiset check for the ranges of f_1 and t, by setting m as the constant function $m(\vec{x}) = 1$. In this case only h_0 needs to be provided by the prover. More interestingly, Protocol 2 is

expression $\frac{m(x)}{x+t(\vec{x})} - \frac{1}{x+f_1(\vec{x})} - \ldots - \frac{1}{x+f_M(\vec{x})}$

$$m_0(\vec{x}) = m(\vec{x}), \quad \varphi_0(\vec{x}) = x + t(\vec{x}),$$
$$m_i(\vec{x}) = -1, \quad \varphi_i(\vec{x}) = x + f_i(\vec{x}), \quad i = 1, \ldots, M.$$

Question.

Variant 2: range equality

easily extended to a proof of range equality, showing that $\bigcup_{i=1}^{M}\{f_i(\vec{x})\}_{\vec{x}\in H} = \{t(\vec{x})\}_{\vec{x}\in H}$ as sets. For this the prover additionally shows that $m \neq 0$ over H, which is done by providing another auxiliary function $g : H \to F$ subject to $g \cdot m = 1$ over H. However, we are not aware of any application of this fact.

Question.

https://eprint.iacr.org/2022/1530

logUp

Soundness

ϵ_1 from rational equality

Given a random challenge $x \leftarrow_\$ F$ from the verifier, the prover shows that the rational identity (13) holds at $X = x$, i.e.

$$\sum_{\vec{x} \in H} \sum_{i=1}^{M} \frac{1}{x + f_i(\vec{x})} - \frac{m(\vec{x})}{x + t(\vec{x})} = 0, \tag{15}$$

multiplying it with the common denominator

$$\prod_{\vec{x} \in H} (X + t(\vec{x})) \cdot \prod_{i=1}^{M} (X + f_i(\vec{x})).$$

Since we sample x from a set of size at least $|F| - |H|$, the soundness error this step is at most

$$\varepsilon_1 \leq \frac{(M+1) \cdot |H| - 1}{|F| - |H|}.$$

ϵ_2 from K domain identities

and the domain identities

$$h_k(\vec{x}) \cdot \prod_{i \in I_k} \varphi_i(\vec{x}) = \sum_{i \in I_k} m_i(\vec{x}) \cdot \prod_{j \in I_k \setminus \{i\}} \varphi_j(\vec{x})$$

Reduce domain identity to sumchecks:

- $h(x) = 0$ for $x \in H = \{\pm 1\}^n$
- MLE $p(z) = \sum_{x \in H} L_H(x, z) \cdot h(x)$

1. verifier sends a random challenge $z \in F^n$
2. run sumcheck for $L_H(X, z) \cdot h(X)$

$$\varepsilon_2 \leq \frac{K}{|F|},$$

$\epsilon_3 = 1/|F|$ from batching $K+1$ sumchecks

Reduce $K+1$ sumchecks to a single one.

$\epsilon_{sumcheck}$ from the single sumcheck protocol

$$\varepsilon_{sumcheck} \leq \frac{1}{|F|} \cdot \sum_{i=1}^{n} \deg_{X_i} p(X_1, \ldots, X_n).$$

total soundness error

$$\varepsilon < \frac{(M+1) \cdot |H| - 1}{|F| - |H|} + \frac{K+1}{|F|} + \varepsilon_{sumcheck},$$

https://eprint.iacr.org/2022/1530

logUp

Computational cost

Overall cost: $|H| \cdot (d+1) \cdot ((\nu + |Q|_{\mathsf{M}}) \cdot \mathsf{M} + (\nu + |Q|_{\mathsf{A}}) \cdot \mathsf{A})$

$$\sum_{\vec{x} \in H} Q(L_H(\vec{x}, \vec{z}), m(\vec{x}), \varphi_0(\vec{x}), \dots, \varphi_M(\vec{x}), h_1(\vec{x}), \dots, h_K(\vec{x})) = 0,$$

where

Q has $v = M + K + 3$ variables with absolute degree $d = \ell + 2$.

$$Q(L, m, \varphi_0, \dots, \varphi_M, h_1, \dots, h_K) = \sum_{k=1}^{K} h_k + L \cdot \lambda_k \cdot \left(h_k \cdot \prod_{i \in I_k} \varphi_i - \sum_{i \in I_k} m_i \cdot \prod_{j \in I_k \setminus \{i\}} \varphi_j \right), \quad (18)$$

Notations:

- A: time in field additions/subtractions
- M: time in field multiplications
- $|Q|_A$: # field adds for Q
- $|Q|_M$: # field multiplications for Q

Domain evaluation strategy for Q using batch inversion.

$|Q_{\mathsf{M}}| = 3 \cdot M + K \cdot (\ell - 1), \quad |Q_{\mathsf{A}}| = K \cdot (\ell + 1),$

formula $(\boxed{12})$. Assume that in each group I_k the inverses of φ_i are given, except for one distinct $i_k \in I_k$. Then we may evaluate the domain identity terms in Q by the fractional representation

$$h_k \cdot \prod_{i \in I_k} \varphi_i - \sum_{i \in I_k} m_i \cdot \prod_{j \in I_k \setminus \{i\}} \varphi_j = \prod_{i \in I_k \setminus \{i_k\}} \varphi_j \cdot \left(\varphi_{i_k} \cdot \left(h_k - \sum_{i \in I_k \setminus \{i_k\}} \frac{m_i}{\varphi_i} \right) - m_{i_k} \right) \quad \ell^2(M+A)$$

For $k \geq 2$, all involved $m_i = -1$, and we have

$$\prod_{i \in I_k \setminus \{i_k\}} \varphi_j \cdot \left(\varphi_0 \cdot \left(h_k + \sum_{i \in I_k \setminus \{i_k\}} \frac{1}{\varphi_i} \right) + 1 \right), \quad \ell(M+A) \quad (23)$$

regardless of the choice of i_k. For the first group $k = 1$, we chose $i_1 = 0$, so that

$$\prod_{i \in I_1 \setminus \{0\}} \varphi_j \cdot \left(\varphi_0 \cdot \left(h_k + \sum_{i \in I_1 \setminus \{0\}} \frac{1}{\varphi_i} \right) - m \right). \quad \ell(M+A) \quad (24)$$

overall cost for evaluation of Q

$K \cdot (\ell + 2) \cdot \mathsf{M} + K \cdot (\ell + 1) \cdot \mathsf{A}.$

$+3(M-K)$ for batch inversion of $M - K$ φ_i

$|Q_{\mathsf{M}}| = 3 \cdot M + K \cdot (\ell - 1), \quad |Q_{\mathsf{A}}| = K \cdot (\ell + 1),$

https://eprint.iacr.org/2022/1530

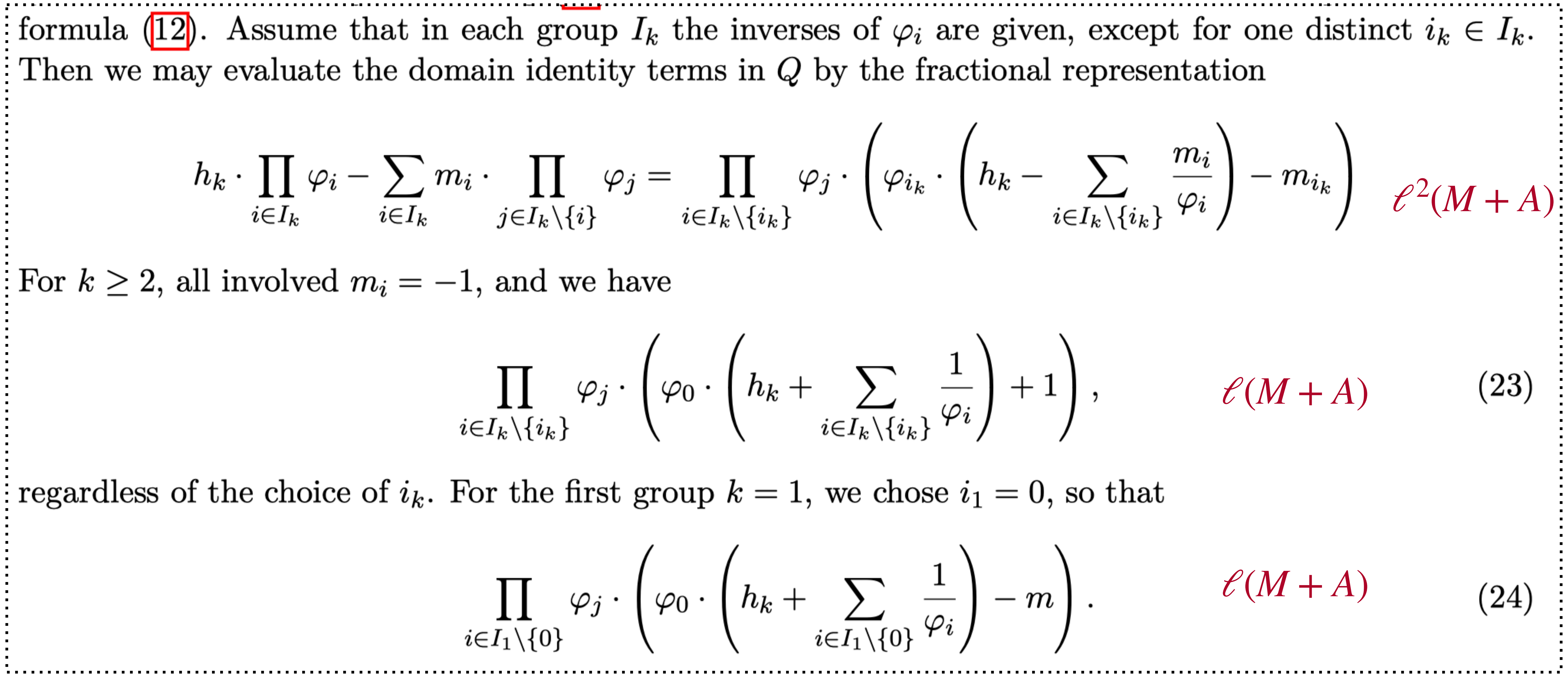
logUp

Computational cost

Sumcheck cost: $|H| \cdot (d+1) \cdot ((\nu + |Q|_{\mathsf{M}}) \cdot \mathsf{M} + (\nu + |Q|_{\mathsf{A}}) \cdot \mathsf{A})$

Domain evaluation strategy for Q using batch inversion.

$$\boxed{|Q_{\mathsf{M}}| = 3 \cdot M + K \cdot (\ell - 1), \quad |Q_{\mathsf{A}}| = K \cdot (\ell + 1),}$$


formula ([12]). Assume that in each group I_k the inverses of φ_i are given, except for one distinct $i_k \in I_k$. Then we may evaluate the domain identity terms in Q by the fractional representation

$$h_k \cdot \prod_{i \in I_k} \varphi_i - \sum_{i \in I_k} m_i \cdot \prod_{j \in I_k \setminus \{i\}} \varphi_j = \prod_{i \in I_k \setminus \{i_k\}} \varphi_j \cdot \left(\varphi_{i_k} \cdot \left(h_k - \sum_{i \in I_k \setminus \{i_k\}} \frac{m_i}{\varphi_i} \right) - m_{i_k} \right) \qquad \ell^2(M+A)$$

For $k \geq 2$, all involved $m_i = -1$, and we have

$$\prod_{i \in I_k \setminus \{i_k\}} \varphi_j \cdot \left(\varphi_0 \cdot \left(h_k + \sum_{i \in I_k \setminus \{i_k\}} \frac{1}{\varphi_i} \right) + 1 \right), \qquad \ell(M+A) \qquad (23)$$

regardless of the choice of i_k. For the first group $k = 1$, we chose $i_1 = 0$, so that

$$\prod_{i \in I_1 \setminus \{0\}} \varphi_j \cdot \left(\varphi_0 \cdot \left(h_k + \sum_{i \in I_1 \setminus \{0\}} \frac{1}{\varphi_i} \right) - m \right). \qquad \ell(M+A) \qquad (24)$$

overall cost for evaluation of Q

$K \cdot (\ell + 2) \cdot \mathsf{M} + K \cdot (\ell + 1) \cdot \mathsf{A}.$

$+3(M - K)$ for batch inversion of $M - K$ φ_i

$$\boxed{|Q_{\mathsf{M}}| = 3 \cdot M + K \cdot (\ell - 1), \quad |Q_{\mathsf{A}}| = K \cdot (\ell + 1),}$$

Batch inversion for computing the inverses for a sequence $(a_i)_{i=1}^N$.

1. compute the cumulative products $p_i = a_1 \ldots a_i$ for $i = 1, \ldots, n$.
2. compute their inverses reversely, starting from $q_n = \dfrac{1}{p_n}$, and putting $q_{i-1} = q_i \cdot a_i$ for $i = n, \ldots, 2$
3. compute the inverses derived via $a_i^{-1} = p_{i-1} \cdot q_i$

Overall cost for batch inversion:
$3 \cdot (N-1)$ multiplications and a single inversion.

https://eprint.iacr.org/2022/1530

logUp

Computational cost and optimal choice of ℓ

- arithmetic costs of
$$|H| \cdot (K + 5 + (\ell + 3) \cdot (4 \cdot M + 3 + \ell \cdot K)) \cdot \mathsf{M}, \tag{26}$$
neglecting field additions and substractions, and
- oracle costs of
$$K + 1 = \left\lceil \frac{M+1}{\ell} \right\rceil \boxed{+1} \tag{27}$$
oracles of size $|H|$.

Question:
The reason of not committing to f_i is that we've committed to $m(x)$, which is computed by all f_i.

Optimal choice of ℓ:

- $\ell = 1$: $M + 1$ helper functions (commitments) but with low degree of Q, $d = 3$.
- $\ell = M + 1$: a single commitment but with a degree of $d = M + 3$
- optimal choice for ℓ is trade-off between arithmetic degree and the number of commitments.

$$\sum_{\vec{x} \in H} Q(L_H(\vec{x}, \vec{z}), \boxed{m(\vec{x})}, \varphi_0(\vec{x}), \ldots, \varphi_M(\vec{x}), \boxed{h_1(\vec{x}), \ldots, h_K(\vec{x})}) = 0,$$

where

Q has $v = M + K + 3$ variables with absolute degree $d = \ell + 2$.

$$Q(L, m, \varphi_0, \ldots, \varphi_M, h_1, \ldots, h_K) = \sum_{k=1}^{K} h_k + L \cdot \lambda_k \cdot \left(h_k \cdot \prod_{i \in I_k} \varphi_i - \sum_{i \in I_k} m_i \cdot \prod_{j \in I_k \setminus \{i\}} \varphi_j \right), \tag{18}$$

https://eprint.iacr.org/2022/1530

logUp

Vector-valued lookups

Vector lookups in logUp

As Plookup, Protocol [2] is easily generalized to functions with multilinear values,

$$t(\vec{x}) = \sum_{(j_1,\dots,j_k)\in\{0,1\}^k} t_{j_1,\dots,j_k}(\vec{x}) \cdot Y_1^{i_1} \cdots Y_k^{j_k},$$

$$f_i(\vec{x}) = \sum_{(j_1,\dots,j_k)\in\{0,1\}^k} f_{i,j_1,\dots,j_k}(\vec{x}) \cdot Y_1^{i_1} \cdots Y_k^{j_k},$$

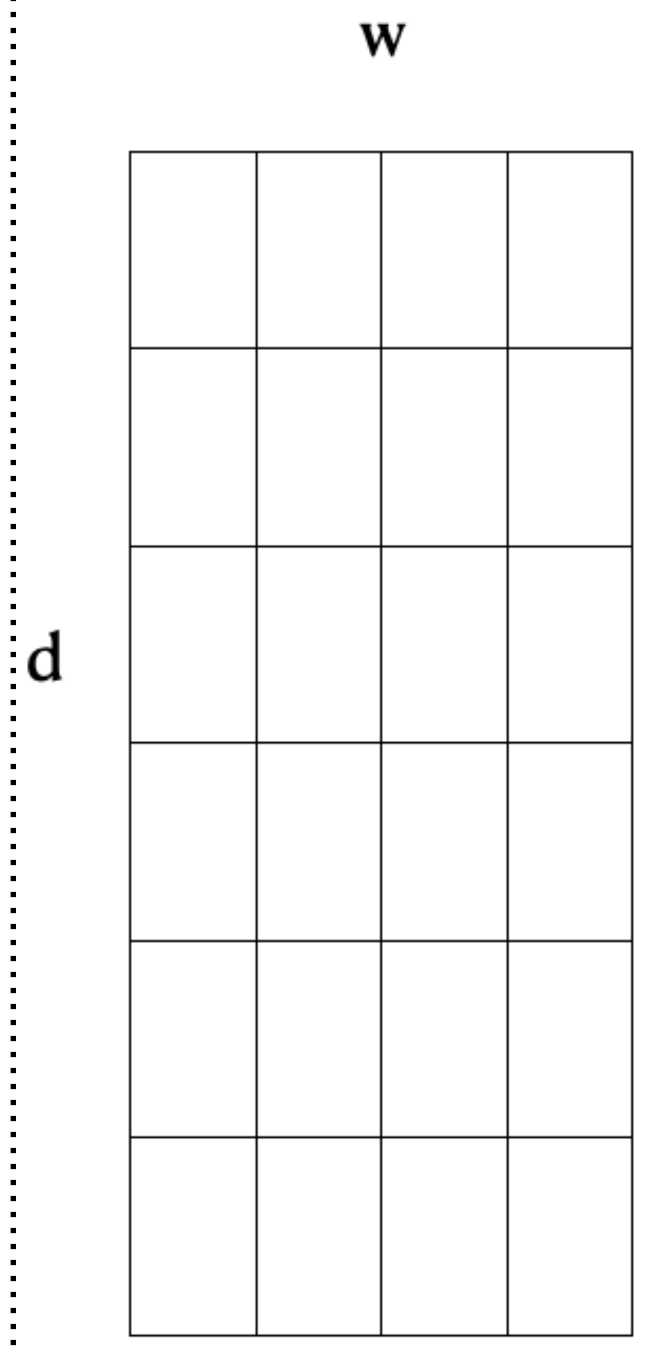
Question:

what is the difference of choosing multivariate and univariate randomization.

Vector lookups in Plookup

Vector lookups:

Suppose we have several witness polynomials $f_1, \dots, f_w \in \mathbb{F}_{<n}[X]$, and a table of values $t^* \in (\mathbb{F}^w)^d$. We wish to check that for each $j \in [n]$ $(f_1(\mathbf{g}^j), \dots, f_w(\mathbf{g}^j)) \in t^*$. We can use randomization to efficiently reduce to the case of Section [3].



Main idea: view each row as a univariate polynomial in α.

The verifier will choose random $\alpha \in \mathbb{F}$.
Then we will define $t := \sum_{i\in[w]} \alpha^i t_i$, $f := \sum_{i\in[w]} \alpha^i f_i$.

soundness probability is $d \cdot w / |\mathbb{F}|$
for some $j \in [n]$, $(f_1(\mathbf{g}^j), \dots, f_w(\mathbf{g}^j)) \notin t^*$.

Question: What is the difference between the univariate poly. and multivariate poly. in randomization ?

Application: Key-value setting (XOR example)

As alluded to in the introduction, a natural use case for this vector lookup primitive is a key-value setting, where we have a function f with $w-1$ inputs, and wish to verify a vector is of the form $(x_1, \dots, x_{w-1}, f(x_1, \dots, x_{w-1}))$ for some input $(x_1, \dots, x_{w-1})$.

https://eprint.iacr.org/2022/1530

logUp

Batch-column lookups using multivariate Plookup

$$\prod_{i\in[N]}(X-a_i)=\prod_{j\in[M]}(X-t_j)^{m_j} \xrightarrow[\text{Turns products into sums of reciprocals.}]{\text{logarithmic derivatives}} \sum_{i=1}^{N}\frac{1}{X-a_i}=\sum_{j=1}^{M}\frac{m_j}{X-t_j}.$$

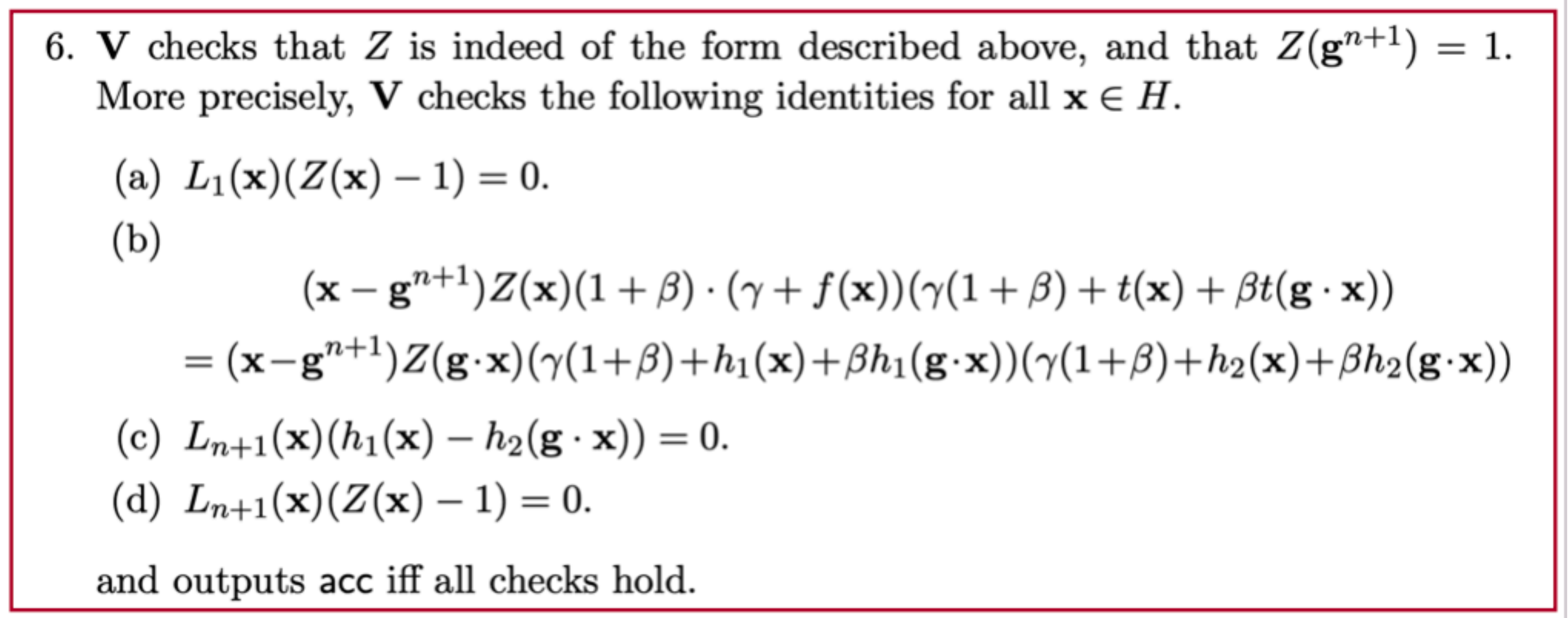
- Reduced to one grand product check with product = 1
- H-ranged polynomial-IOP (H is a multiplicative subgroup)
- In the univariate setting

- Reduced to sumcheck with sum = 0
- Lagrange-IOP over hypercube $H=\{\pm 1\}^n$
- In the multivariate setting

Bridge Plookup strategy with sumcheck protocol using **multivariate time shift** in Hyperplonk [CBBZ22].

"time shift" in Plookup strategy in a cyclic subgroup with a generator

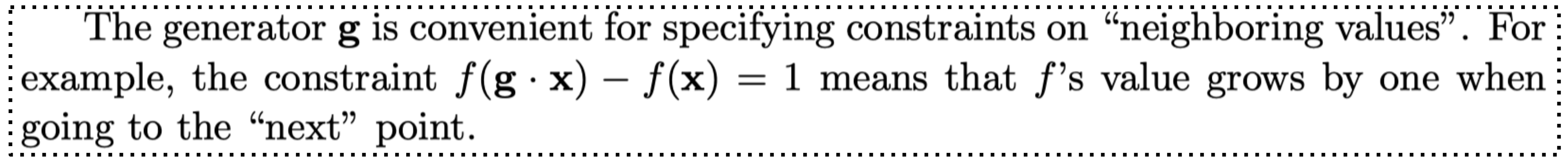
Verification:



6. **V** checks that Z is indeed of the form described above, and that $Z(\mathbf{g}^{n+1})=1$. More precisely, **V** checks the following identities for all $\mathbf{x}\in H$.
   (a) $L_1(\mathbf{x})(Z(\mathbf{x})-1)=0$.
   (b)
   $$(\mathbf{x}-\mathbf{g}^{n+1})Z(\mathbf{x})(1+\beta)\cdot(\gamma+f(\mathbf{x}))(\gamma(1+\beta)+t(\mathbf{x})+\beta t(\mathbf{g}\cdot\mathbf{x}))$$
   $$=(\mathbf{x}-\mathbf{g}^{n+1})Z(\mathbf{g}\cdot\mathbf{x})(\gamma(1+\beta)+h_1(\mathbf{x})+\beta h_1(\mathbf{g}\cdot\mathbf{x}))(\gamma(1+\beta)+h_2(\mathbf{x})+\beta h_2(\mathbf{g}\cdot\mathbf{x}))$$
   (c) $L_{n+1}(\mathbf{x})(h_1(\mathbf{x})-h_2(\mathbf{g}\cdot\mathbf{x}))=0$.
   (d) $L_{n+1}(\mathbf{x})(Z(\mathbf{x})-1)=0$.

   and outputs acc iff all checks hold.

"neighboring values" in multiplicative group



The generator **g** is convenient for specifying constraints on "neighboring values". For example, the constraint $f(\mathbf{g}\cdot\mathbf{x})-f(\mathbf{x})=1$ means that f's value grows by one when going to the "next" point.

https://eprint.iacr.org/2022/1530

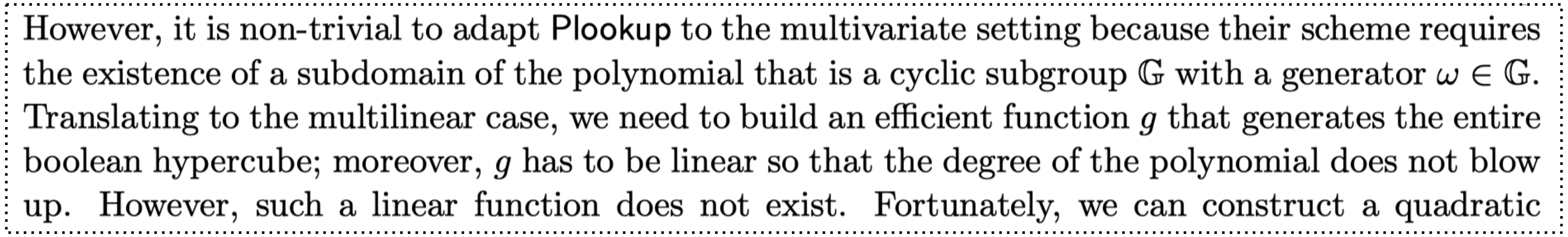
https://eprint.iacr.org/2022/1355.pdf

logUp

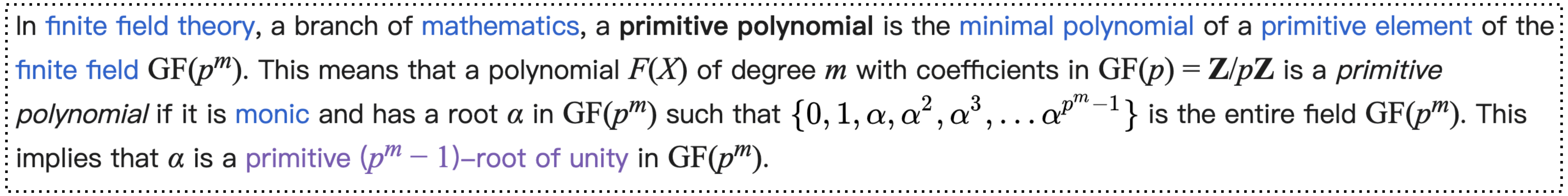
Multivariate time shift function

Bridge Plookup strategy with sumcheck protocol using **multivariate time shift** in Hyperplonk [CBBZ22].

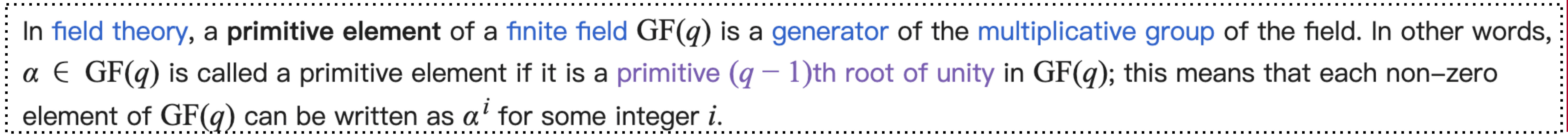
It has to build an efficient function g that generates the entire boolean hypercube.


However, it is non-trivial to adapt Plookup to the multivariate setting because their scheme requires the existence of a subdomain of the polynomial that is a cyclic subgroup $\mathbb{G}$ with a generator $\omega \in \mathbb{G}$. Translating to the multilinear case, we need to build an efficient function g that generates the entire boolean hypercube; moreover, g has to be linear so that the degree of the polynomial does not blow up. However, such a linear function does not exist. Fortunately, we can construct a quadratic

primitive polynomial Wiki


In finite field theory, a branch of mathematics, a **primitive polynomial** is the minimal polynomial of a primitive element of the finite field $\mathrm{GF}(p^m)$. This means that a polynomial $F(X)$ of degree m with coefficients in $\mathrm{GF}(p) = \mathbf{Z}/p\mathbf{Z}$ is a *primitive polynomial* if it is monic and has a root α in $\mathrm{GF}(p^m)$ such that $\{0, 1, \alpha, \alpha^2, \alpha^3, \ldots \alpha^{p^m-1}\}$ is the entire field $\mathrm{GF}(p^m)$. This implies that α is a primitive $(p^m - 1)$–root of unity in $\mathrm{GF}(p^m)$.

primitive element


In field theory, a **primitive element** of a finite field $\mathrm{GF}(q)$ is a generator of the multiplicative group of the field. In other words, $\alpha \in \mathrm{GF}(q)$ is called a primitive element if it is a primitive $(q-1)$th root of unity in $\mathrm{GF}(q)$; this means that each non–zero element of $\mathrm{GF}(q)$ can be written as α^i for some integer i.

Generating hypercube via primitive polynomial.

- $GF(2^2)$ with primitive polynomial $q(x) = x^2 + x + 1$

1. Initialize a set $\{0,1,x\}$
2. Multiply x to the last element a added in the set.
    1. If degree of ax is 2, $a = ax \,\%\, q(x)$
    2. Else, $a = ax$.
3. Add a in the set until the size of set is 2^2.

https://blog.csdn.net/cyq6239075/article/details/105738710

https://eprint.iacr.org/2022/1530

https://eprint.iacr.org/2022/1355.pdf

logUp

Multivariate time shift function

A quadratic generator in boolean hypercube.

A quadratic generator in $\mathbb{F}_{2^\mu}$. For every $\mu \in \mathbb{N}$, we fix a *primitive polynomial* $p_\mu \in \mathbb{F}_2[X]$ where $p_\mu := X^\mu + \sum_{s \in S} X^s + 1$ for some set $S \subseteq [\mu - 1]$, so that $\mathbb{F}_2[X]/(p_\mu) \cong \mathbb{F}_2^\mu[X] \cong \mathbb{F}_{2^\mu}$. By definition of primitive polynomials, $X \in \mathbb{F}_2^\mu[X]$ is a generator of $\boxed{\mathbb{F}_2^\mu[X] \setminus \{0\}}$. This naturally defines a generator function $g_\mu : B_\mu \to B_\mu$ as

$$g_\mu(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_\mu) = (\boldsymbol{b}_\mu, \boldsymbol{b}'_1, \ldots, \boldsymbol{b}'_{\mu-1}),$$

where $\boldsymbol{b}'_i = \boldsymbol{b}_i \oplus \boldsymbol{b}_\mu$ $(i \leq 1 < \mu)$ if $i \in S$, and $\boldsymbol{b}'_i = \boldsymbol{b}_i$ otherwise. Essentially, for a polynomial $f \in \mathbb{F}_2^\mu[X]$ with coefficients $\boldsymbol{b}$, $g_\mu(\boldsymbol{b})$ is the coefficient vector of $X \cdot f(X)$. Hence the following lemma is straightforward.

Lemma 3.8. *Let $g_\mu : B_\mu \to B_\mu$ be the generator function defined above. For every $\mathbf{x} \in B_\mu \setminus \{0^\mu\}$, it holds that $\{g_\mu^{(i)}(\mathbf{x})\}_{i \in [2^\mu - 1]} = B_\mu \setminus \{0^\mu\}$, where $g_\mu^{(i)}(\cdot)$ denotes i repeated application of g_μ.*

Linearizing the generator. For a multivariate polynomial $f \in \mathcal{F}_\mu^{(\leq d)}$, we define $f_{\Delta_\mu} \in \mathcal{F}_\mu^{(\leq d)}$ as

$$f_{\Delta_\mu}(\mathbf{X}_1, \ldots, \mathbf{X}_\mu) := \mathbf{X}_\mu \cdot f(1, \mathbf{X}'_1, \ldots, \mathbf{X}'_{\mu-1}) + (1 - \mathbf{X}_\mu) \cdot f(0, \mathbf{X}_1, \ldots, \mathbf{X}_{\mu-1})$$

where $\mathbf{X}'_i := 1 - \mathbf{X}_i$ $(i \leq 1 < \mu)$ if $i \in S$, and $\mathbf{X}'_i := \mathbf{X}_i$ otherwise.

it holds that $f_{\Delta_\mu}(\mathbf{x}) = f(g_\mu(\mathbf{x}))$ for every $\mathbf{x} \in B_\mu$. *f_{Δ_μ} has individual degree d and one can evaluate f_{Δ_μ} from 2 evaluations of f.*

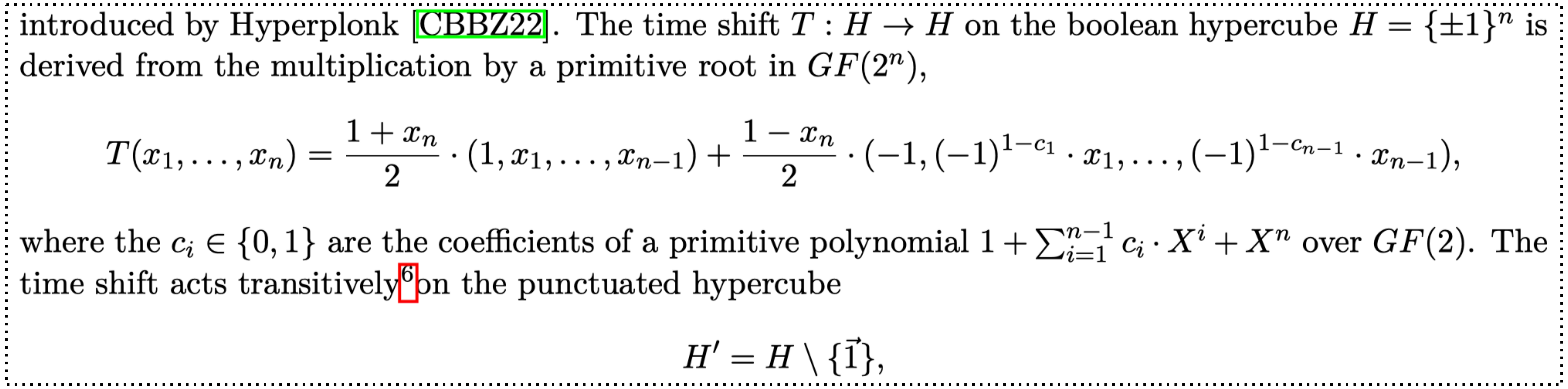
https://eprint.iacr.org/2022/1530

https://eprint.iacr.org/2022/1355.pdf

logUp

Multivariate time shift function

Time shift function T on the punctuated hypercube

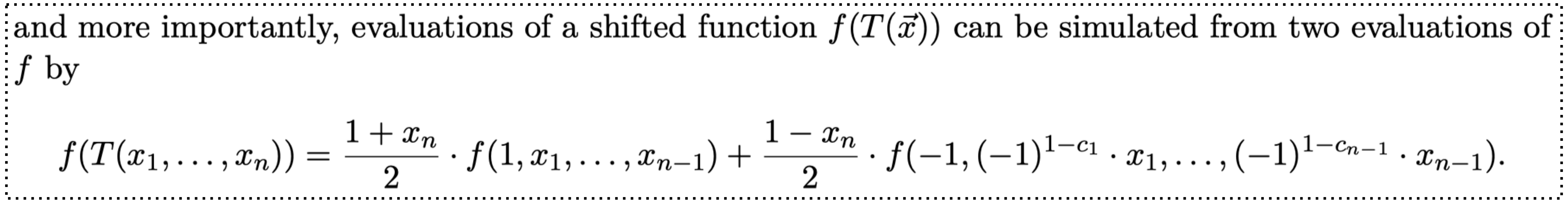

introduced by Hyperplonk [CBBZ22]. The time shift $T: H \to H$ on the boolean hypercube $H = \{\pm 1\}^n$ is derived from the multiplication by a primitive root in $GF(2^n)$,

$$T(x_1, \ldots, x_n) = \frac{1 + x_n}{2} \cdot (1, x_1, \ldots, x_{n-1}) + \frac{1 - x_n}{2} \cdot (-1, (-1)^{1-c_1} \cdot x_1, \ldots, (-1)^{1-c_{n-1}} \cdot x_{n-1}),$$

where the $c_i \in \{0, 1\}$ are the coefficients of a primitive polynomial $1 + \sum_{i=1}^{n-1} c_i \cdot X^i + X^n$ over $GF(2)$. The time shift acts transitively[6] on the punctuated hypercube

$$H' = H \setminus \{\vec{1}\},$$

"neighboring values" in punctuated hypercube (multiplicative groups)


and more importantly, evaluations of a shifted function $f(T(\vec{x}))$ can be simulated from two evaluations of f by

$$f(T(x_1, \ldots, x_n)) = \frac{1 + x_n}{2} \cdot f(1, x_1, \ldots, x_{n-1}) + \frac{1 - x_n}{2} \cdot f(-1, (-1)^{1-c_1} \cdot x_1, \ldots, (-1)^{1-c_{n-1}} \cdot x_{n-1}).$$

https://eprint.iacr.org/2022/1530

logUp

Batch-column Plookup

Notations

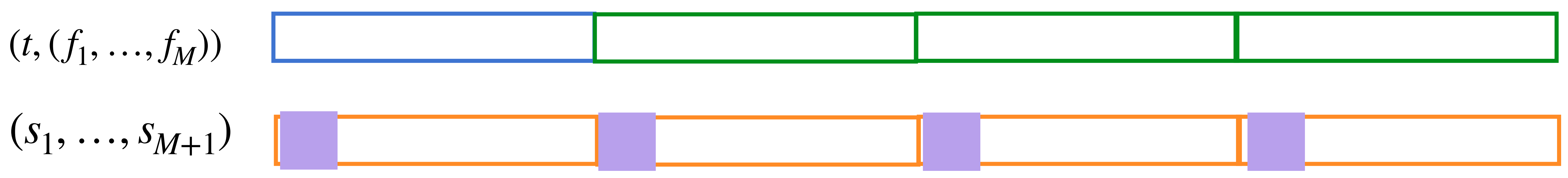
Let $t : H' \to F$ be the lookup table, and $f_i : H' \to F$, $i = 1, \ldots, M$, the functions subject to the lookup. Although the functions are defined over the punctuated hypercube H', we assume arbitrary values at $\vec{1}$.

Plookup identity for mutliset check

(These will be ignored by the lookup argument.) The prover provides the ordered union of the f_i together with t in a piecewise manner, via the additional functions $s_i : H' \to F$, $i = 1, \ldots, M + 1$. The *Plookup identity* (in the [Gab22] style) is then

$$\prod_{\vec{x} \in H'} \prod_{i=1}^{M} (X + s_i(\vec{x}) + s_{i+1}(\vec{x}) \cdot Y) \cdot (X + s_{M+1}(\vec{x}) + s_1(T(\vec{x})) \cdot Y)$$

$$= \prod_{\vec{x} \in H'} \prod_{i=1}^{M} (X + f_i(\vec{x}) + f_i(\vec{x}) \cdot Y) \cdot (X + t(\vec{x}) + t(T(\vec{x})) \cdot Y).$$



https://eprint.iacr.org/2022/1530

logUp

Batch-column Plookup

Notations

Let $t : H' \to F$ be the lookup table, and $f_i : H' \to F$, $i = 1, \ldots, M$, the functions subject to the lookup. Although the functions are defined over the punctuated hypercube H', we assume arbitrary values at $\vec{1}$.

1. Plookup identity for mutliset check

(These will be ignored by the lookup argument.) The prover provides the ordered union of the f_i together with t in a piecewise manner, via the additional functions $s_i : H' \to F$, $i = 1, \ldots, M+1$. The *Plookup identity* (in the [Gab22] style) is then

$$\prod_{\vec{x} \in H'} \prod_{i=1}^{M} (X + s_i(\vec{x}) + s_{i+1}(\vec{x}) \cdot Y) \cdot (X + s_{M+1}(\vec{x}) + s_1(T(\vec{x})) \cdot Y)$$
$$= \prod_{\vec{x} \in H'} \prod_{i=1}^{M} (X + f_i(\vec{x}) + f_i(\vec{x}) \cdot Y) \cdot (X + t(\vec{x}) + t(T(\vec{x})) \cdot Y).$$

2. Reduce it to a grand product identity over H'

The identity is reduced to a grand product over H' by random samples $\alpha, \beta \leftarrow_\$ F$ for X and Y, yielding

$$\prod_{\vec{x} \in H'} h(\vec{x}) = 1, \tag{28}$$

where

$$h(\vec{x}) = \frac{\alpha + s_{M+1}(\vec{x}) + s_1(T(\vec{x})) \cdot \beta}{\alpha + t(\vec{x}) + t(T(\vec{x})) \cdot \beta} \cdot \prod_{i=1}^{M} \frac{(\alpha + s_i(\vec{x}) + s_{i+1}(\vec{x}) \cdot \beta)}{(\alpha + f_i(\vec{x}) + f_i(\vec{x}) \cdot \beta)}. \tag{29}$$

https://eprint.iacr.org/2022/1530

logUp

Batch-column Plookup

3. Split the product into K partial products for controlling the degree of the grand product identity.

$$h(\vec{x}) = \prod_{k=1}^{K} h_k(\vec{x}),$$

$$h_k(\vec{x}) = \prod_{i \in I_k} \frac{\sigma_i(\vec{x})}{\varphi_i(\vec{x})},$$

As in Protocol 2, we control the algebraic degree of the resulting identity for (28) by splitting the product (29) into $K = \left\lceil \frac{M+1}{\ell} \right\rceil$ partial products of size ℓ, where $1 \leq \ell \leq M+1$. For this we use the notation

$$\varphi_0(\vec{x}) = \alpha + t(\vec{x}) + \beta \cdot t(T(\vec{x})),$$
$$\sigma_0(\vec{x}) = \alpha + s_{M+1}(\vec{x}) + \beta \cdot s_1(T(\vec{x})),$$

$$\varphi_i(\vec{x}) = \alpha + (1+\beta) \cdot f_i(\vec{x}),$$
$$\sigma_i(\vec{x}) = \alpha + s_i(\vec{x}) + \beta \cdot s_{i+1}(\vec{x}).$$

4. Compute the cumulative products along the orbit of the time shift on H'

where $I_k = [(k-1)\cdot\ell, k\cdot\ell) \cap [0, M]$. For each $k = 1, \ldots, K$, the prover computes the cumulative products of the values $h_k(\vec{x})$ along the orbit of the time shift T on H', starting with $\phi_k(-\vec{1}) = 1$, and setting

$$\phi_k(T^j(-\vec{1})) = \phi_k(T^{j-1}(-\vec{1})) \cdot h(T^{j-1}(-\vec{1})),$$

typo: $h \rightarrow h_k$

for $j = 1, \ldots, |H'| - 1$. At the remaining point $\vec{x} = \vec{1}$ outside H', the prover sets ϕ_k to zero. Correctness

5. Reduce domain identities and point identities (for correctness of the grand product) to sumchecks over H by Lagrange polynomial.

$$\phi_k(T(\vec{x})) \cdot \prod_{i \in I_k} \varphi_i(\vec{x}) - \phi_k(\vec{x}) \cdot \prod_{i \in I_k} \sigma_i(\vec{x}) = 0,$$

$$\phi_1(-1) = 1,$$
$$\phi_k(T^{-1}(-\vec{1})) = \phi_{(k \bmod K)+1}(-\vec{1}),$$

containing the resulting grand product when $k = K$

$L_H(\,.\,, -\vec{1})$ and $L_H(\,.\,, \vec{y})$, where $\vec{y} \leftarrow_{\$} F^n$,

6. Batch all sumchecks to one single sumcheck.

https://eprint.iacr.org/2022/1530

logUp

Batch-column Plookup

7. The resulting overall sumcheck in batch-column Plookup

$$\sum_{\vec{x}\in H} Q(L_H(\vec{x},\vec{y}), L_H(\vec{x},-\vec{1}), \varphi_0(\vec{x}), \ldots, \varphi_M(\vec{x}), \sigma_0(\vec{x}), \ldots, \sigma_M(\vec{x}), \phi_1(\vec{x}), \ldots, \phi_K(\vec{x}), \phi_1(T(\vec{x})), \ldots \phi_K(T(\vec{x}))) = 0,$$

$$|Q_{\mathsf{M}}| = K\cdot(2\cdot\ell+3)+2,$$

$$|Q_{\mathsf{A}}| = 4\cdot K - 1.$$

where Q has $v = 2\cdot(M+1+K)+3$ variables with absolute degree $d = \ell + 2$.

$$Q(L_H, L, L_T, \varphi_0, \ldots, \varphi_M, \sigma_0, \ldots, \sigma_M, \phi_1, \ldots, \phi_K, \phi_{1,T}, \ldots, \phi_{K,T}) = \\ L\cdot(\phi-1) + \sum_{k=1}^{K} \lambda_k \cdot L_H \cdot \left(\phi_{k,T}\cdot\prod_{i\in I_k}\varphi_i - \phi_k\cdot\prod_{i\in I_k}\sigma_i\right) + \mu_k\cdot\left(L_T\cdot\phi_k - L\cdot\phi_{(k \bmod K)+1}\right). \quad (30)$$

Compared to logUp

$$|Q_{\mathsf{M}}| = 3\cdot M + K\cdot(\ell-1), \quad |Q_{\mathsf{A}}| = K\cdot(\ell+1),$$

$$\sum_{\vec{x}\in H} Q(L_H(\vec{x},\vec{z}), m(\vec{x}), \varphi_0(\vec{x}), \ldots, \varphi_M(\vec{x}), h_1(\vec{x}), \ldots, h_K(\vec{x})) = 0,$$

where

Q has $v = M + K + 3$ variables with absolute degree $d = \ell + 2$.

$$Q(L, m, \varphi_0, \ldots, \varphi_M, h_1, \ldots, h_K) = \sum_{k=1}^{K} h_k + L\cdot\lambda_k\cdot\left(h_k\cdot\prod_{i\in I_k}\varphi_i - \sum_{i\in I_k} m_i \cdot \prod_{j\in I_k\setminus\{i\}}\varphi_j\right), \quad (18)$$

https://eprint.iacr.org/2022/1530

logUp

Comparison of computational costs in batch-column lookups

Batch-column lookups using Multivariate Plookup

- arithmetic costs of
$$|H| \cdot \left(\left(2 \cdot \ell^2 + 13 \cdot \ell + 18\right) \left\lceil \frac{M+1}{\ell} \right\rceil + \ell \cdot (2 \cdot M + 7) + 8 \cdot (M+3) \right) \cdot \mathsf{M},$$
neglecting field additions and substractions, and
- oracle costs of
$$M + K + 1 = M + \left\lceil \frac{M+1}{\ell} \right\rceil + 1$$
functions of size $|H|$.

$$\sum_{\vec{x} \in H} Q(L_H(\vec{x}, \vec{y}), L_H(\vec{x}, -\vec{1}), \varphi_0(\vec{x}), \ldots, \varphi_M(\vec{x}), \sigma_0(\vec{x}), \ldots, \sigma_M(\vec{x}), \phi_1(\vec{x}), \ldots, \phi_K(\vec{x}), \phi_1(T(\vec{x})), \ldots \phi_K(T(\vec{x}))) = 0,$$

oracle costs:
$M+1$ for s_i (sorted union of table and witness), K for φ_k (cumulative products)

Batch-column lookups using logUp

- arithmetic costs of
$$|H| \cdot (K + 5 + (\ell + 3) \cdot (4 \cdot M + 3 + \ell \cdot K)) \cdot \mathsf{M},$$
neglecting field additions and substractions, and
- oracle costs of
$$K + 1 = \left\lceil \frac{M+1}{\ell} \right\rceil + 1$$
oracles of size $|H|$.

$$\sum_{\vec{x} \in H} Q(L_H(\vec{x}, \vec{z}), m(\vec{x}), \varphi_0(\vec{x}), \ldots, \varphi_M(\vec{x}), h_1(\vec{x}), \ldots, h_K(\vec{x})) = 0,$$

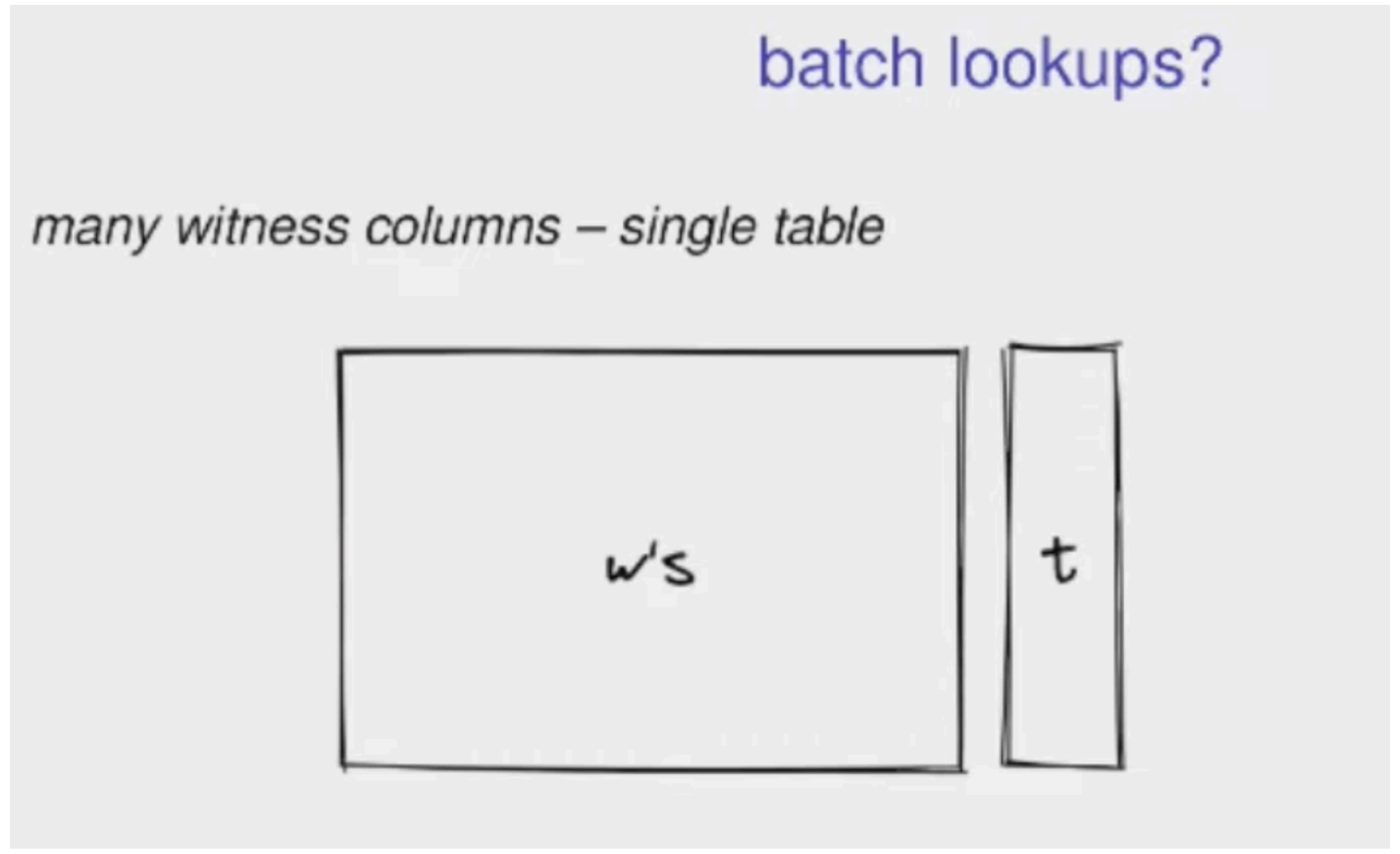
significant lower oracle costs:
K for h_i (helper functions), and 1 for m (multiplicities) ?

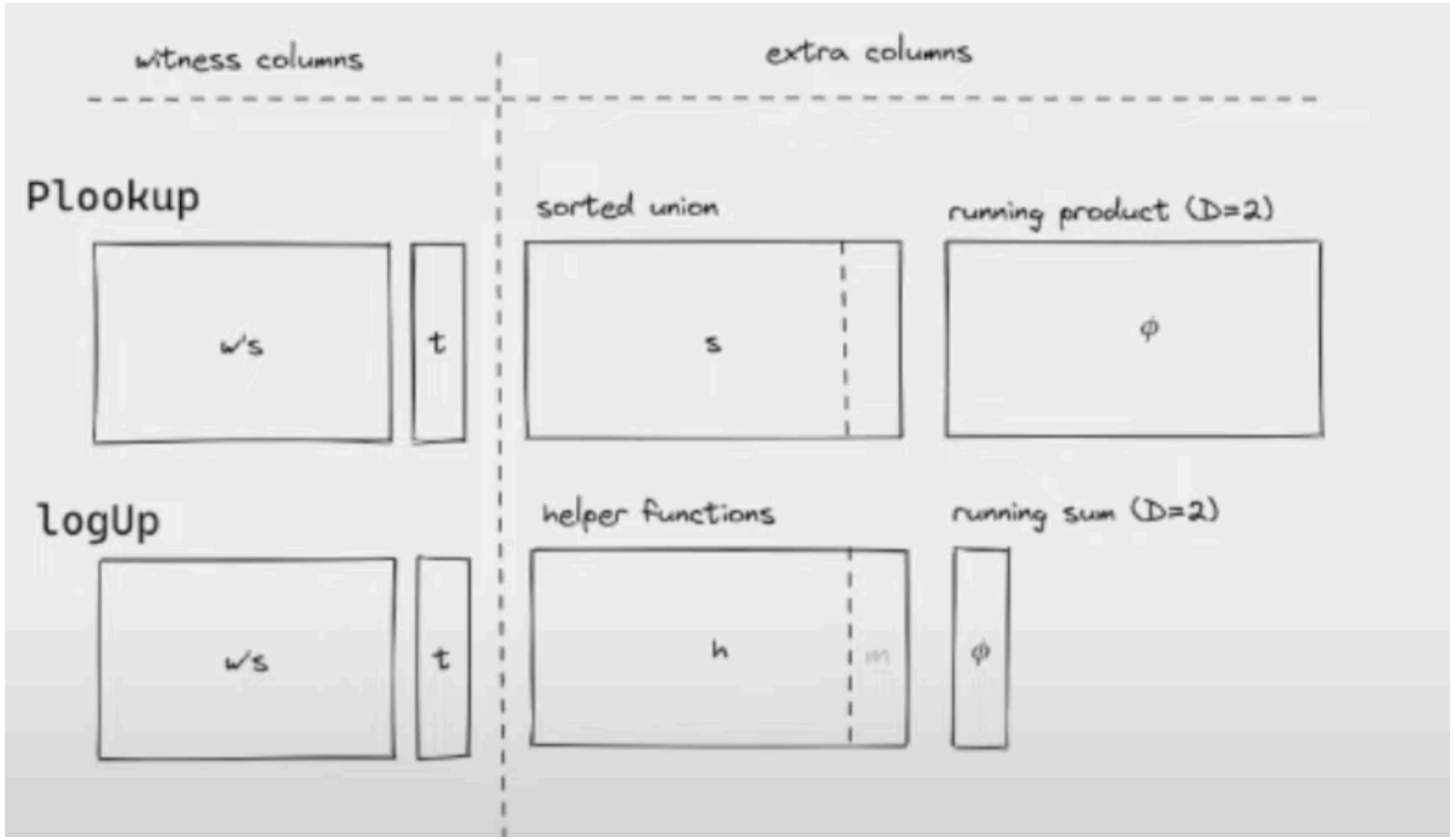
https://eprint.iacr.org/2022/1530

logUp

Comparison of computational costs

ZK9: logUp - Lookup arguments based on the logarithmic derivative - Ulrich Haböck:
https://www.youtube.com/watch?v=qv_5dF2_C4g&themeRefresh=1


batch lookups?
many witness columns – single table
w's
t


witness columns
extra columns
Plookup
sorted union
running product (D=2)
w's
t
s
φ
logUp
helper functions
running sum (D=2)
w's
t
h
m
φ