July 22, 2024

Introduction to Zerocheck

Outline:

1. Mathematical Preliminaries
   - Multilinear Polynomials
   - Lagrange vs Monomial Basis
   - equality indicator
   - Multilinear Extensions + Evaluation Formula
   - Polynomial Identity Testing
2. Why zerocheck matters
3. The zerocheck to sumcheck reduction

(7/15/24) Last Time

4. Sumcheck Protocol Overview
5. Computing the round polynomial

Today (7/22/24)

6. Notes on Performance
7. Simple performance optimizations
8. Section 3 performance optimizations
9. Zerocheck Batching

(7/29/24) Next Time

Goal: To teach concepts behind zero-check and jargon bust the mathematical language behind Binius

Recap

- We discussed mathematical preliminaries
- Importantly:
  - Multilinear Extension of a function $f: \{0,1\}^n \to \mathbb{F}$ is an n-variate multilinear polynomial $\tilde{f}(X_0, \ldots, X_{n-1})$
  - individual degree and total degree
  - equality testing univariate polynomials by checking if their evaluations at a random point are equal.

More mathematical machinery:

- **Multilinear Composite**

A multivariate polynomial $f(X_0, X_1, \ldots, X_{n-1})$ can be represented as a "composition of multilinears"

e.g.

given $f(X_0, X_1, X_2) = X_0 X_1^2 + X_1^3 X_2^3$

let $h_1(X_0, X_1, X_2) = X_0 X_1$

$h_2(X_0, X_1, X_2) = X_1 X_2$

$g(Y_1, Y_2) = Y_1^2 + Y_2^3$

then $f(\vec{X}) = g(h_1(\vec{X}), h_2(\vec{X}))$

where g = composition of total degree 3

h_1, h_2 = multilinears

Fact: max individual degree of a multilinear composite is at most total degree of g

in above example • $\deg_{x_0}(f) = 2$, $\deg_{x_1}(f) = 3$, $\deg_{x_2}(f) = 3$

- max individual degree of $F = \max\{2,3,3\} = 3$
- total degree of $g = 3$

- How to evaluate $f(\vec{x}) = g(h_1(\vec{x}), \ldots, h_c(\vec{x}))$
  - evaluate $y_i := h_i(\vec{x})$
  - evaluate $g(y_1, \ldots, y_c)$

// in this notation, c underlying multilinear polynomials

Now, we will ignore the context of the SNARK and of zerocheck. We will today only focus on understand sumcheck, and these three questions.

- what is it?
- how it works?
- why it works?

(4) Overview of (unoptimized) Sumcheck Protocol:

Defn: Sumcheck is an interactive protocol where a prover convinces a verifier that a specific polynomial on n-variables $f(X_0, \dots, X_{n-1})$ has hypercube evals that sum to a claimed value S.

i.e.

$$\sum_{\vec{v} \in \{0,1\}^n} f(\vec{v}) \stackrel{?}{=} S$$

↖ $f(0,\dots,0) + f(1,0,\dots,0) + \dots + f(1,\dots,1) \stackrel{?}{=} S$

Assume the verifier is able to query f at a point $r = (r_0, \dots, r_{n-1}) \in \mathbb{F}^n$

Key: the verifier does not want to do 2^n queries and add up the results themselves (inefficient).

Okay with querying at one point. // N.B. via polynomial commitment scheme opening proof

Assume The input polynomial f is represented as a composition of multilinears (as a MultilinearComposite)

$$f(X_0, \ldots, X_{n-1}) := g(h_1(X_0, \ldots, X_{n-1}), \ldots, h_c(X_0, \ldots, X_{n-1}))$$

where
- g is a c-variate composition function of total degree d
- $\forall i \in \{1, \ldots, c\}$, h_i is n-variate multilinear polynomial

we can say that f is an n-variate polynomial of "max individual degree" $\leq d$

Preview: What is going to happen?

P and V will interact for n rounds, in each round, we reduce a sumcheck claim to a simpler sumcheck claim on one fewer variable

- Before Round 0: claim is about a sum on the n-dimensional hypercube

e.g. $$\sum_{\vec{v} \in \{0,1\}^n} f_0(\vec{v}) \stackrel{?}{=} S_0$$

$f_0 := f$, multivariate poly on n vars

i.e. after round 0

- Before Round 1: claim is about a sum on the $(n-1)$-dimension hypercube

e.g. $$\sum_{\vec{v} \in \{0,1\}^{n-1}} f_1(\vec{v}) \stackrel{?}{=} S_1$$

f_1 is a multivariate poly on $n-1$ vars

...

- Before Round k: Claim is about a sum on $(n-k)$-dimension hypercube

e.g.

$$\sum_{v \in \{0,1\}^{n-k}} f_k(\vec{r}) \stackrel{?}{=} s_k$$

↖ f_k is a multivariate poly on $n-k$ vars

...

- Before Round $n-1$: Claim is about a sum on the 1-dimension hypercube

e.g.

$$\sum_{v \in \{0,1\}} f_{n-1}(\vec{r}) \stackrel{?}{=} s_{n-1}$$

↖ f_{n-1} is a multivariate poly on 1 vars
[this is also called univariate poly]

- Final (After Round $n-1$): Claim is about a sum on 0-dimension hypercube

e.g.

$$f_n(\cdot) \stackrel{?}{=} s_n$$

↖ f_n is a multivariate poly on 0 vars
it will look like $f(r'_0, \ldots, r'_{n-1})$

So, at the end of sumcheck n rounds, we have an evalcheck:

$$f(r'_0, \ldots, r'_{n-1}) \stackrel{?}{=} s_n$$

at this point V will query f @ $\vec{r} = (r_0, \ldots, r_{n-1})$
in total, V does only 1 query instead of 2^n queries

Remains: How do these reductions work? What is r_i? s_i? What makes protocol valid?

Defn: Completeness

An interactive protocol is complete when an honest prover can convince a verifier of any true fact.

Defn: Soundness

An interactive protocol is sound when a dishonest prover will fail to convince a verifier of any false fact

---

Technical Note: many forms of completeness and soundness (perfect, statistical, computational. Also adaptive vs non-adaptive)

We use:

- perfect completeness → with 100% prob P convinces V of any true fact
- adaptive computational soundness → a cheating prover will have $\leq 0.0\ldots01\%$ chance of finding an untrue statement to prove to a verifier convincingly.

Now, let's dive into what a "Round" of Sumcheck actually looks like.

Round K: The goal is to reduce a sumcheck claim about a multivariate polynomial on $n-K$ variables to a sumcheck claim about a multivariate polynomial on $n-K-1$ variables

Claim before round K

$$\sum_{\vec{v} \in \{0,1\}^{n-K}} f_K(\vec{v}) \stackrel{?}{=} S_K$$

Claim after round K and before round K+1

$$\sum_{\vec{v} \in \{0,1\}^{n-K-1}} f_{K+1}(\vec{v}) \stackrel{?}{=} S_{K+1}$$

- If round K claim true $\Rightarrow$ round K+1 claim true [completeness]
  if you followed protocol honestly
- if round K claim false $\Rightarrow$ round K+1 claim false [soundness]
  w.h.p. no matter what dishonest prover tries

Steps (for an honest prover)

1) Prover sends univariate round polynomial $R_k(X)$ to the verifier claimed to equal

$$R_k(X) = \sum_{\vec{v} \in \{0,1\}^{n-k-1}} f_k(X, \vec{v})$$

this is what honest prover is supposed to send. can send any degree $\leq d$ polynomial though.

// i.e. an honest prover simplifies this RHS expression into monomial coefficients $a_0, \ldots, a_d$

$R_k(X)$ is guaranteed to have degree $\leq d$ ← total degree of composition poly g

2) If the prover is honest about $R_k(X)$, and the round k claim is true, then $R_k(0) + R_k(1) = S_k$

The verifier checks this is true, if not, rejects claim

3) The verifier samples a random challenge $r'_k \leftarrow_R L$ and sends r'_k to the prover

4) The prover and verifier both compute $S_{k+1} := R_k(r'_k)$

5) Round $k+1$ claim poly is $f_{k+1}(X) = f_k(r'_k, X)$

- This process requires V to evaluate $R_k(X) = a_0 + \cdots + a_d X^d$ @ the point r'_k
- The complicated, compute-intensive step is Step 1, where the prover computes the monomial coefficients for the round polynomial $R_k(X) = a_0 + a_1 X + \cdots + a_d X^d$

coeffs: $[a_0, \ldots, a_d]$ basis: $[1, X, X^2, \ldots, X^d]$

Before we dive into how the prover does Step 1, let's first ask why this "round" protocol is mathematically desirable.

Want:

- if round k claim true $\Rightarrow$ round $k+1$ claim true [completeness]
- if round k claim false $\Rightarrow$ round $k+1$ claim false with high probability [soundness]

Completeness:

Suppose round k claim true and $R_k(x)$ is honest, then

- $R_k(0) + R_k(1) = \sum_{v \in \{0,1\}^{n-k-1}} f_k(0, \vec{v}) + \sum_{v \in \{0,1\}^{n-k-1}} f_k(1, \vec{v}) = \sum_{v \in \{0,1\}^{n-k}} f_k(\vec{v}) = s_k$

  ↑ verifier check will pass

Let $r'_k \leftarrow_R L$ be any sampled value

- $s_{k+1} = R_k(r'_k) = \sum_{v \in \{0,1\}^{n-k-1}} f_k(r'_k, \vec{v}) = \sum_{v \in \{0,1\}^{n-k-1}} f_{k+1}(\vec{v})$

So Round $k+1$ claim on f_{k+1}, s_{k+1} is true

Soundness:

Suppose round K claim is false, then the dishonest prover selects any round polynomial $R_K(X)$

Case 1: $R_K(X)$ is the real round polynomial

i.e. $R_K(X) = \sum_{\vec{v} \in \{0,1\}^{n-K-1}} f_K(X, \vec{v})$

since Round K claim is false

then $R_K(0) + R_K(1) = \sum_{\vec{v} \in \{0,1\}^{n-K}} f_K(\vec{v}) \neq S_K$

So the verifier check will fail!

This means the dishonest prover MUST try lying about $R_K(X)$

Case 2: $R_K(X)$ is NOT the real round polynomial

i.e. $R_K(X) \neq \sum_{\vec{v} \in \{0,1\}^{n-K-1}} f_K(X, \vec{v})$

[By polynomial identity testing, for random $r'_K \leftarrow_R L$ with high prob]

$$\Rightarrow R_K(r'_K) \neq \sum_{\vec{v} \in \{0,1\}^{n-K-1}} f_K(r'_K, \vec{v})$$

$$\Rightarrow S_{K+1} \neq \sum_{\vec{v} \in \{0,1\}^{n-K-1}} f_{K+1}(\vec{v})$$

So Round $K+1$ claim (f_{K+1}, S_{K+1}) is false

We have now detailed how a round of Sumcheck works.

If original claim is true

→ do n rounds of sumcheck, knocking out 1 variable at a time. After every round you have a simpler (one fewer variable) sumcheck claim that is both

- true and
- whose truthiness is a good proxy for the previous claim's truthiness

→ each round verifier

- receives degree d univariate round poly
- samples a random challenge from large field L
- evaluates round poly at challenge, creating the next round's claim

If original claim is false

→ Do n rounds of sumcheck, and at some round you need to hope Round K claim is false but round K+1 claim is true, otw you have failed as a malicious prover

→ Polynomial identity testing must fail at some round, which happens with probability $\leq \frac{d}{|L|}$

→ Prob you cheat successfully in Round 0 or Round 1 or ... or Round n-1

$\leq$ Prob[Cheat Round 0] + Pr[Cheat Round 1] + ... + Pr[Cheat Round n-1]

$= n \cdot$ Prob[Cheat Round i]

$= n \cdot \frac{d}{|L|}$ → "soundness error"

By "Union Bound"
$Pr[A \cup B] \leq Pr[A] + Pr[B]$

(5) Computing the Round Polynomial (as honest prover)

Recall, in Kth round of sumcheck, the prover must send a "round polynomial" to the verifier, represented in monomial coeffs

$$R_K(X) = \sum_{\vec{v} \in \{0,1\}^{n-K-1}} f_K(X, \vec{r})$$

$$= \sum_{v_{K+1}, \dots, v_{n-1} \in \{0,1\}} f(r_0, \dots, r_{K-1}, X, v_{K+1}, \dots, v_{n-1})$$

Recall: r_i is the ith round verifier challenge

Observe that $\deg_X(R_K) = \deg_{X_K}(f) \leq \deg(g) =: d$

So, we can represent R_K with $d+1$ coefficients

$$R_K(X) = a_0 + a_1 X + \cdots + a_d X^d$$

or $[a_0, \dots, a_d]$ // coordinate vector w.r.t. monomial basis ↑ with respect to

Recall: a degree d univariate polynomial uniquely determined by $d+1$ evaluations

So, we can evaluate $R_K(X)$ at $X = 0, \dots, d$

$$R_K(0) = \sum_{\vec{v} \in \{0,1\}^{n-K-1}} f(\vec{r}, 0, \vec{v})$$

$$\dots$$

$$R_K(d) = \sum_{\vec{v} \in \{0,1\}^{n-K-1}} f(\vec{r}, d, \vec{v})$$

Recall $f(\vec{x}) = g(h_1(\vec{x}), \ldots, h_\ell(\vec{x}))$

Algorithm (Take 1)

- evals = $[0, \ldots, 0]$ // length $d+1$ vector
  // ith index represents accumulation
  // of calculating $R_k(i)$
- for $\vec{v}$ in $\{0,1\}^{n-k-1}$:
  - for X in $\{0, \ldots, d\}$:
    - multilin_evals = $[0, \ldots, 0]$ // length c
    - for j in $[1, \ldots, c]$:
      - multilin_evals$[j-1] = h_j(\vec{r}, X, \vec{v})$
    - evals$[X]$ += g(multilin_evals)
- // now evals is the coordinate vector representation
  // of $R_k(X)$ w.r.t. univariate Lagrange Basis over domain $\{0, \ldots, d\}$
- mon_coeffs = $M \cdot$ evals
  ↖ change of basis matrix

return mon_coeffs

This works, but we can be "smart" about how we do ☆, specifically for a fixed hypercube point $\vec{v}^*$, evaluating $f(\vec{r}, X, \vec{v}^*)$ for $X \in [0, \ldots, d]$

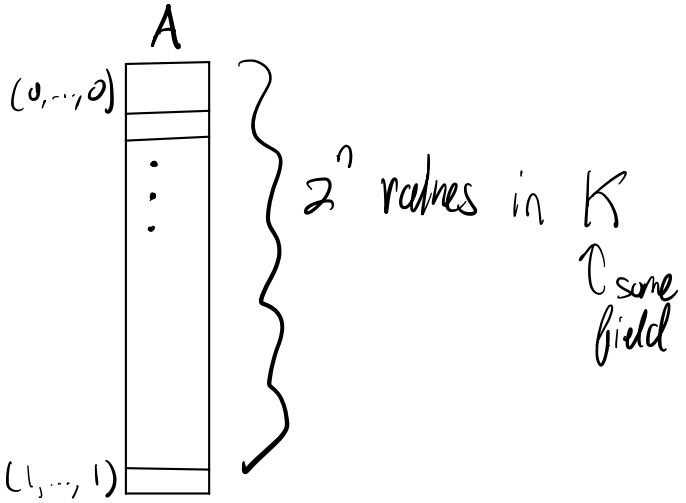
This is called process_vertex in our SumcheckEvaluator trait in Irreducible oss/Binius

Specifically we can be "smart" about evaluating $h_j(\vec{r}, X, \vec{v}^*)$ $\forall X \in [0, \ldots, d]$ $\forall j \in [1, \ldots, c]$

How to evaluate a multilinear:

(wrt n-dimensional boolean hypercube)

Assume we have the Lagrange Basis representation of a multilinear $\tilde{A}(X_0, \ldots, X_{n-1})$



Now we wish to evaluate @ a point $\vec{r} = (r_0, \ldots, r_{n-1}) \in L^n$

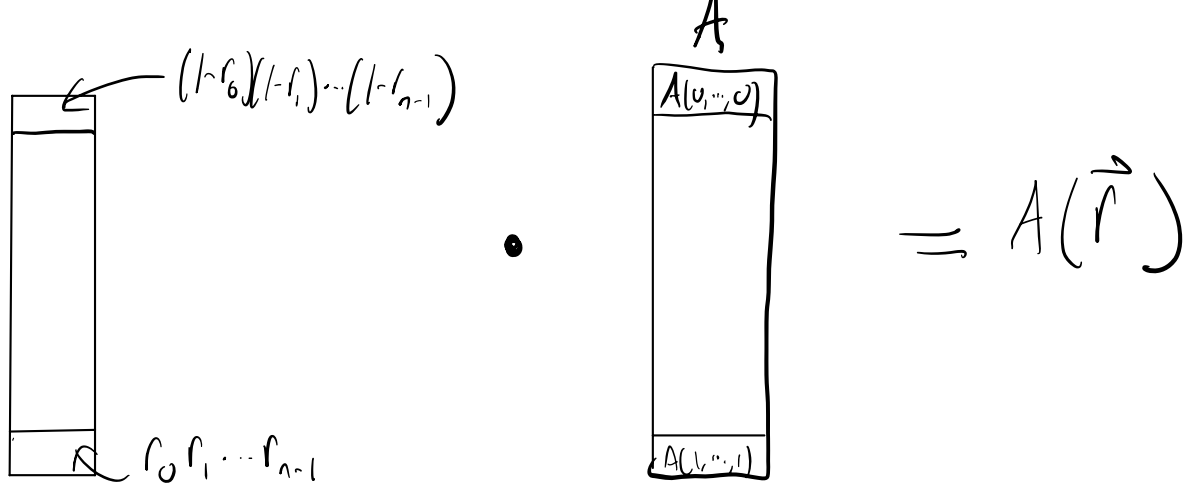
↑ some extension field

Method 1: Expand Tensor then Dot Product

- evaluate L.B. polys @ r

$$\simeq \bigotimes_{i=0}^{n-1} [1 - r_i, r_i]$$

- dot product with L.B. coeffs



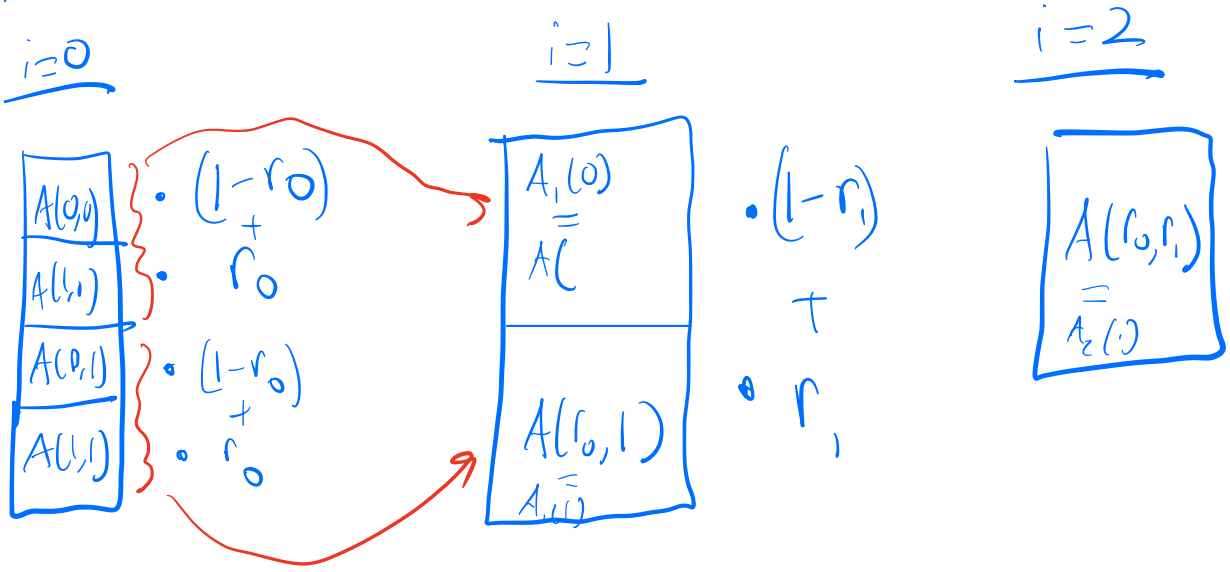
Method 2: Specialize each variable through linear interpolations

for i in $[0..n-1]$

- Specialize X_i in $\tilde{A}(r_0, \ldots, r_{i-1}, X_i, X_{i+1}, \ldots, X_{n-1})$ to r_i
- New polynomial is $\tilde{A}(r_0, \ldots, r_i, X_{i+1}, \ldots, X_{n-1})$ has one fewer variable and coordinate vector length has halved from 2^{n-i} to 2^{n-i-1}

Now you have coordinate vector of length $2^0 = 1$ for the 0-variate "polynomial" $\tilde{A}(r_0, \ldots, r_{n-1})$. This one value is your answer.

ex// $n=2$



Linear Interpolations

$$A(r_0, 0) = (1-r_0)A(0,0) + r_0 A(1,0)$$

$$A(r_0, 1) = (1-r_0)A(0,1) + r_0 A(1,1)$$

$$A(r_0, r_1) = (1-r_1) \cdot A(r_0, 0) + r_1 \cdot A(r_0, 1)$$

interpolating $A(r_0, X)$ to $A(r_0, r_1)$ given $A(r_0, 0)$ and $A(r_0, 1)$

Notice:

1) we are evaluating multilinears in Round K

@ $(r'_0, \ldots, r'_{K-1}, X, \vec{v})$

$\forall X \in [0, \ldots, d], \forall \vec{v} \in \{0,1\}^{n-K-1}$

where $r'_0, \ldots, r'_{K-1}$ are from L

but $X_K, \ldots, X_{n-1}$ are not taking large field values

2) We evaluate multilinears in Round K+1

@ $(r'_0, \ldots, r'_{K-1}, r'_K, X, \vec{v})$

which is similar to $(r'_0, \ldots, r'_{K-1}, X, \vec{v})$

so this suggests we should reuse computation from Round K

Let's exploit observation 1

Fix $j^* \in [1, \ldots, c]$, so we're looking at one multilinear h_{j^*}

To evaluate $h_{j^*}(\underbrace{r_0, \ldots, r_{k-1}}_{\text{fixed big field values}}, X, \vec{v})$ $\forall \vec{v} \in \{0,1\}^{n-k-1}$ $\forall X \in [0, \ldots, d]$

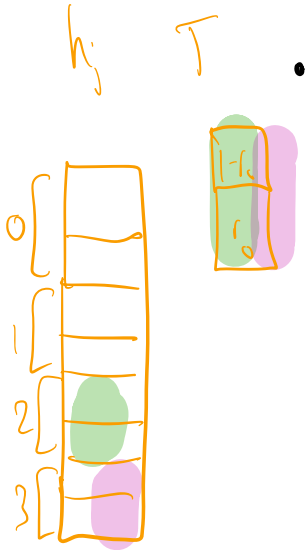
Method 1 (Improved)

- Create partial tensor $\bigotimes_{i=0}^{k-1} [1-r_i, r_i]$ // Size is 2^k
- For each i^* in $[0, \ldots, 2^{n-k-1}]$:

  // This corresponds with some $v^* \in \{0,1\}^{n-k-1}$

  - dot product partial tensor with the $(2 \cdot i)$th chunk of size 2^k to get $h_{j^*}(r_0, \ldots, r_{k-1}, 0, \vec{v}^*)$
  - dot product partial tensor with $(2 \cdot i + 1)$th chunk of size 2^k in h_{j^*} L.B. coordinate vector to get $h_{j^*}(r_0, \ldots, r_{k-1}, 1, \vec{v}^*)$
  - for X in $2, \ldots, d$:

    extrapolate $h_{j^*}(r_0, \ldots, r_{k-1}, X_k, \vec{v}^*)$ @ $X_k = X$ to get $h_{j^*}(r_0, \ldots, r_{k-1}, X, \vec{v}^*)$



How does "dot product partial tensor with $(2i)$th chunk" give you evaluation @ $(r'_0, \ldots, r'_{k-1}, 0, \vec{v})$

A

Suppose $k=2$, so we have the partial tensor T

T

We want to evaluate

$A(r'_0, r'_1, \underline{0}, \vec{v})$

and $A(r'_0, r'_1, \underline{1}, \vec{v})$ $\forall \vec{v} \in \{0,1\}^1$

for $\vec{v} = \{0\}$

- corresponds to $i = 0$
- Chunk $2i+0$ = chunk 0
- Chunk$_0$ • $T = A(r'_0, r'_1, 0, 0)$

why? $A_0(r'_0, r'_1) = A(r'_0, r'_1, 0, 0)$

↑ L.B. coeffs IS chunk 0

L.B. poly evals @ (r'_0, r'_1) IS partial tensor

- Chunk $2i+1$ = Chunk 1
- Chunk$_1$ • $T = A(r'_0, r'_1, 1, 0)$

why? $A_1(X_0, X_1) := A(X_0, X_1, 1, 0)$

$A_1(r'_0, r'_1)$ = A_1 LB coeffs

Chunk$_1$ →

•

L.B. polys evaluated @ (r'_0, r'_1)

T →

for $\vec{v} = \{1\}$

- corresponds to $i = 1$
- Chunk $2i+0$ = Chunk 2
- Chunk$_2$ • $T = A(r'_0, r'_1, 0, 1)$
- Chunk $2i+1$ = chunk 3
- Chunk$_3$ • $T = A(r'_0, r'_1, 1, 1)$

Key Insight: The L.B. coeffs of a multilinear on n variables can be interpreted as a size 2^{n-k} array of L.B. coeffs of multilinears on k variables where each smaller multilinear specializes $(n-k)$ variables to be a different $(n-k)$ dimensional hypercube point

Method 2 (Improved)

For i in $[0, \ldots, k-1]$:

- Specialize X_i in $h_{j^*}(r_0, \ldots, r_{i-1}, X_i, \ldots, X_{n-1})$ as r_i (linearly interpolate adjacent values)
  - New poly is $h_{j^*}(r_0, \ldots, r_{i-1}, r_i, X_{i+1}, \ldots, X_{n-1})$ on one fewer var

Now you have $H^{(k)}_{j^*}(X_k, \ldots, X_{n-1}) := h_{j^*}(r_0, \ldots, r_{k-1}, X_k, \ldots, X_{n-1})$ ("folded multilinear")

- For each $\vec{v}^* \in \{0,1\}^{n-k-1}$:
  - read $H^{(k)}_{j^*}(0, \vec{v})$ and read $H_{j^*}(1, \vec{v})$
  - for X in $2, \ldots, d$:
    - extrapolate $H^{(k)}_{j^*}(X_k, \vec{v})$ @ $X_k = X$

Let's exploit observation 2

If we are using method 1, we have already computed $\bigotimes_{i=0}^{K-1} [1-r_i^j, r_i^j]$ for round K.

For round $K+1$, we need to compute the partial tensor on one more variable

$$\bigotimes_{i=0}^{K} [1-r_i^j, r_i^j] = \left(\bigotimes_{i=0}^{K-1} [1-r_i^j, r_i^j]\right) \otimes [1-r_K^j, r_K^j]$$

If we are using method 2, we have already, for Round K, computed $H_{j^*}^{(K)}(X_K, \ldots, X_{n-1})$. In Round $K+1$ we need to calculate

$H_{j^*}^{(K+1)}(X_{K+1}, \ldots, X_{n-1}) = H_{j^*}^{(K)}(X_K, \ldots, X_{n-1})$ specialized @ $X_K = r_K^j$

Recall $f(\vec{x}) = g(h_1(\vec{x}), \dots, h_c(\vec{x}))$

Algorithm (Take 2)

- evals = $[0, \dots, 0]$ // length $d+1$ vector
- for $\vec{v}$ in $\{0,1\}^{n-k-1}$:
  - zero_multilin_evals = $[0, \dots, 0]$ // length c, jth elt is $h_j(\vec{r}, 0, \vec{v})$
  - one_multilin_evals = $[0, \dots, 0]$ // length c, jth elt is $h_j(\vec{r}, 1, \vec{v})$
  - z_multilin_evals = $[0, \dots, 0]$ // length c, jth elt is $h_j(\vec{r}, z, \vec{v})$
  - for j in $[1, \dots, c]$
    - zero_multilin_evals$[j-1] = h_j(\vec{r}, 0, \vec{v})$
    - one_multilin_evals$[j-1] = h_j(\vec{r}, 1, \vec{v})$

evaluate g $2^{n-k-1} \cdot (d+1)$ times

  - evals[0] += g(zero_multilin_evals) ← g(zero_multilin_evals) $= f(\vec{r}, 0, \vec{v})$
  - evals[1] += g(one_multilin_evals) ← g(one_multilin_evals) $= f(\vec{r}, 1, \vec{v})$
  - for z in $[2, \dots, d]$
    - for j in $[1, \dots, c]$:
      z_multilin_evals = linear_extrapolate $h_j(\vec{r}, X_k, \vec{v})$ @ $X_k = z$
    - evals[z] += g(z_multilin_evals) ← g(z_multilin_evals) $= f(\vec{r}, z, \vec{v})$
- // now evals is the coordinate vector representation
  // of $R_k(X)$ w.r.t. univariate Lagrange Basis over domain $\{0, \dots, d\}$
- mon_coeffs = $M \cdot$ evals
  ↑ change of basis matrix

return mon_coeffs

The interesting thing is how one evaluates $h_j(\vec{r}, 0, \vec{v})$ and $h_j(\vec{r}, 1, \vec{v})$ $\forall v \in \{0,1\}^{n-k-1}$
(equiv how one evaluates $h_j(\vec{r}, \vec{v})$ $\forall v \in \{0,1\}^{n-k}$)
whether you use method 1, dot product with partial tensor, or method 2, folding and reading, should reuse data structure from previous round

N.B. • Method 1 is "pre-switchover" but after round 0

• Method 2 is "past-switchover"

• Round 0 is sort of a trivial case evaluating $h_j(\vec{r})$ $\forall r \in \{0,1\}^n$ is just reading.

(6) Some Notes on Performance

- Method 1 is space-efficient in earlier rounds because e.g. 2^{24} $\mathbb{F}_2$ elts is smaller than 2^{23} $\mathbb{F}_{2^{128}}$ elts. If $h_j(X_0, \ldots, X_{23})$ has $\mathbb{F}_2$ values, then $H_j^{(1)}(r_0', X_1, \ldots, X_{23})$ has $\mathbb{F}_{2^{128}}$ values if $r_0' \in \mathbb{F}_{2^{128}}$
- the degree of g, denoted as d, matters!
- many performance optimizations e.g.
  - no need to evaluate @ $X=0$ // still need to evaluate multilinears there, but not the composition poly g
  - First round zerocheck, do not evaluate round poly @ $X=0$ or $X=1$
  - If sumcheck instance $f(\vec{x}) = \widetilde{eq}(\vec{x}, \vec{r}) \cdot f(\vec{x})$ where $f(\vec{x}) = g(h_1(\vec{x}), \ldots, h_c(\vec{x}))$ and g has total degree d then normally round polynomials would have degree $d+1$ BUT, can be "smart" and make round polynomial degree d

These optimizations will be covered in depth in the next lecture, and we will talk about the implications for what this means for the performance of proving Binius circuits.