Lookup Checks. Plookup. Logup

August 14, 2025

Distributed Lab

zkdl-camp.github.io

github.com/ZKDL-Camp

Introduction

Motivation

Suppose you want to implement the AES-128 or SHA-256 using arithmetical circuits (for instance, for national passport verification).

As the part of such algorithm, assume given $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in \{0, 1\}^n$, you need to implement XORing:

$$\boldsymbol{c} = \boldsymbol{a} \oplus \boldsymbol{b}$$

Arithmetical Circuit, informally

- Bit-decompose $\boldsymbol{a}$ to $\{a_i\}_{i \in [n]}$ and check $a_i(1 - a_i) = 0, i \in [n]$.
- Bit-decompose $\boldsymbol{b}$ to $\{b_i\}_{i \in [n]}$ and check $b_i(1 - b_i) = 0, i \in [n]$.
- Verify $c_i = a_i + b_i - 2a_ib_i$ (which is precisely $c_i = a_i \oplus b_i$).

In Total: $3n$ constraints for *one XORing operation*. E.g., for 8-bit XORing we would have approximately 24 constraints per operation.

Motivation (cont.)

Suppose you need to compute **10k** 8-bit XOR operations during hashing (that corresponds to $\approx$ four SHA256 512-bit blocks).

$$\text{Total constraints} = \text{10k} \times 24 = 240k \text{ constraints}$$

Lookup

With lookup, you pay 2^{16} constraints to commit to the lookup table, and then you use only 1 constraint per XOR. Thus in total you would have $2^{16} + 10k \approx 75k$ constraints. So we get $3.2\times$ boost!

More generally, lookup check allows to reduce complexity from $\mathcal{O}(mn)$ (where m is the cost of each operation and n is the number of operations) to $\mathcal{O}(n + d)$, where d is the lookup size.

Application to Bionetta

In *Bionetta*, lookup checks allow to reduce the number of constraints $\mathcal{O}(n)$ (where n is the circuit size in Groth16) to sublinear $\mathcal{O}(n/\log n)$.

More specifically, implementing $\text{ReLU}(x) = \max\{0, x\}$ naively requires roughly $\mathcal{O}(b)$ constraints where $b = \log_2 |\mathbb{F}|$.

In *UltraGroth*, by splitting the integer into w-bit limbs, one can reduce the complexity per ReLU down to b/w with the lookup commitment cost of 2^w constraints. Thus, in total, one gets $\mathcal{O}(2^w + \ell b/w)$ constraints with ℓ being the number of ReLUs.

For optiomal value of w, this reduces to $\mathcal{O}(\ell b/\log(\ell b))$.

Formalizing the problem

Definition

Lookup check consits in proving/verifying that $\{z_i\}_{i\in[n]} \subseteq \{t_j\}_{j\in[d]}$. We will denote multisets as $\vec{z} := \{z_i\}_{i\in[n]}$ and $\vec{t} = \{t_j\}_{j\in[d]}$ and write inclusion check $\vec{z} \subseteq \vec{t}$ for short.

Example

- $\vec{t} = \{0, 1\}$. $\vec{z} \subseteq \vec{t}$ means “all z_i’s are binary”.
- $\vec{z} = \{10, 6, 7, 1, 1, 6, 10, 7, 1\} \subseteq \{1, 6, 7, 10\} = \vec{t}$. On the other hand, we have $\vec{z}^* := \{1, 6, 10, 5\} \not\subseteq \vec{t}$.
- For $\vec{t} = \{t_j\}_{j\in[2^w]}$ with $t_j = j$, condition “$\vec{z} \subseteq \vec{t}$” means “every element z_i is a w-bit integer”.

Working with tuples

In particular, XOR example can also be reduced to this check.

- First initialize all tuples $\vec{t} := \{(a_i, b_i, c_i)\}_{i \in [2^{2n}]}$ such that $a_i \oplus b_i = c_i$ (perceive each a_i, b_i, c_i as a field element from $\mathbb{F}$).
- We need to check whether $\vec{z} := \{(x_i, y_i, w_i)\}_{i \in [m]} \subseteq \vec{t}$. The verifier samples the challenge $\beta \leftarrow\$ \mathbb{F}$, and lookup table is perceived as $\vec{t}_\beta = \{a_i + \beta b_i + \beta^2 c_i\}_{i \in [2^{2n}]}$, witness as $\vec{z}_\beta := \{x_i + \beta y_i + \beta^2 w_i\}_{i \in [m]}$, and then the check is $\vec{z}_\beta \subseteq \vec{t}_\beta$ as usual.

Conclusion

Lookup checks are cool, so let us study them!

Plookup

plookup

Definition

Plookup is the Poly-IOP-based lookup check protocol that uses rather *exotic* multiset equality check. Why I call it *exotic* you will see in just a moment.

Reminder. When constructing $\mathcal{P}\text{lon}\mathcal{K}$, we considered the so-called *permutation check*, which checked whether multisets $\{a_i\}_{i\in[n]}$ and $\{b_i\}_{i\in[n]}$ are equal. This was done by running the grand product:

$$\prod_{i\in[n]}(\gamma + a_i) = \prod_{i\in[n]}(\gamma + b_i), \quad \gamma \xleftarrow{\$} \mathbb{F}$$

Can we use it for checking $\{z_i\}_{i\in[n]} \subseteq \{t_j\}_{j\in[d]}$*?* No, consider:

$$Z(\gamma) = \prod_{i\in[n]}(\gamma + z_i), \quad T(\gamma) = \prod_{j\in[d]}(\gamma + t_j)$$

We can merely state that roots of $Z(\gamma)$ and $T(\gamma)$ are the same.

plookup: problematic solution

Core problem

How to reduce lookup check to the multiset equality check?

Definition

Given $\vec{s} = \{s_i\}_{i \in [n]}$, denote by $\partial\vec{s}$ the **diference set** $\{s_{i+1} - s_i\}_{i \in [n-1]}$ without zero elements. For example, $\partial\{1, 1, 2, 5, 5\} = \{1, 3\}$.

Attempt #1. Suppose we sort witness $\vec{z}$ and get $\vec{s}$ as a result. We might reasonably expect that $\partial\vec{s} = \partial\vec{t}$.

Example

Suppose $\vec{t} = \{1, 4, 8\}$ and $\vec{s} = \{1, 1, 4, 8, 8, 8\}$. Notice that $\vec{s} \subseteq \vec{t}$ and moreover $\partial\vec{t} = \partial\vec{s} = \{3, 4\}$.

Problem. Converse is false. Consider $\vec{s}^* := \{1, 5, 5, 5, 8, 8\}$.

plookup: solution #1

Attempt #2. Construct $\vec{s}$ — sorted concatenation $(\vec{z}, \vec{t})$. First assert permutation check of $\vec{s}$ and $(\vec{z}, \vec{t})$. Then, assert that $\partial\vec{s} = \partial\vec{t}$.

Lemma

This is necessary and sufficient condition for $\vec{z} \subseteq \vec{t}$.

Example

Suppose $\vec{t} = \{1, 4, 8\}$, $\vec{z} = \{1, 1, 4, 8, 8, 8\}$, and $\vec{z}^* = \{1, 5, 5, 5, 8, 8\}$. For $\vec{t}$ and $\vec{z}$ we see that $\vec{s} = \{1, 1, 1, 4, 4, 8, 8, 8, 8\}$ and clearly $\partial\vec{s} = \{3, 4\} = \partial\vec{t}$. As for $\vec{z}^*$, the sorted concatenation is $\vec{s}^* = \{1, 1, 4, 5, 5, 5, 8, 8, 8\}$ and so $\partial\vec{s}^* = \{1, 3\} \neq \partial\vec{t}$.

Thus our protocol might simply be running two permutation checks: (a) on $\vec{s}$ and $(\vec{z}, \vec{t})$, and (b) on $\partial\vec{s}$ and $\partial\vec{t}$. However, that's still **two** grand products. We can reduce this to a single grand product.

plookup: solution #2

Attempt #3. Why do we need this strage difference set $\partial\vec{s}$? Let us make it more general!

Definition

The **randomized difference set** of $\vec{s} = \{s_i\}_{i\in[n]}$ for randomness β, denoted as $\partial_\beta\vec{s}$, is defined as $\{s_i + \beta s_{i+1}\}_{i\in[n-1]}$.

Lemma

Necessary and sufficient condition for lookup check is $\partial_\beta\vec{s} = ((1+\beta)\vec{z}, \partial_\beta\vec{t})$ *for randomly chosen* $\beta \xleftarrow{\$} \mathbb{F}$.

Intuition. Note that for two same consecutive integers s_i, s_{i+1}, instead of zero, one gets $(1+\beta)s_i$. Other than that, we also have elements of form $t_i + \beta t_{i+1}$, which obviously form $\partial_\beta\vec{t}$. Thus, concatenating all elements of form $(1+\beta)s_i$ and $\partial_\beta\vec{t}$ gives $\partial_\beta\vec{s}$.

Illustration

Suppose we, again, have the following witness and table:

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

The sorted array is $\vec{s} = \{1, 1, 1, 4, 4, 8, 8, 8, 8\}$. Now sample random $\beta \leftarrow\$ \mathbb{F}$. Then we have:

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

On the other hand, we also have:

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

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

Protocol Specifics

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

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

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

Theorem

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

Similarly to permutation check equation $\prod_j (\gamma + a_j) = \prod_j (\gamma + b_j)$, the rest of the protocol is done to ensure that $Z = T$ in the $\mathcal{P}$lon$\mathcal{K}$ish manner. See lecture notes for concrete details.

Logup

Derivatives

Similarly to calculus, we can define derivative operations over polynomials and rational function over arbitrary fields.

Definition

Given a polynomial $q(X) := \sum_{j=0}^{d} q_j X^j$ over $\mathbb{F}[X]$, the **formal derivative**, denoted by $q'(X)$,is given by $\sum_{j=1}^{d} j q_j X^{j-1}$.

Definition

For a function $q(X)/r(X)$ from rational function field $\mathbb{F}(X)$, the **formal derivative** is given by

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

Logarithmic Derivative

Definition

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

Motivation

What is the derivative for $\log f(x)$ given $f : \mathbb{R} \to \mathbb{R}$? By the chain rule, $f'(x)/f(x)$, which is given by the definition above.

Lemma

$\text{logD}[qr] = \text{logD}[q] + \text{logD}[r]$.

Consequently, we can infer:

$$\text{logD}\left[\prod_{j=1}^{n}(X + a_j)\right] = \sum_{j=1}^{n} \text{logD}\left[X + a_j\right] = \sum_{j=1}^{n} \frac{1}{X + a_j}$$

Finally something useful

Theorem (On fractional permutation check)

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

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

for randomly chosen $\gamma \xleftarrow{\$} \mathbb{F}$.

Intuition. Take logarithmic derivative from both sides of a permutation check equation $\prod_{j=1}^{n}(\gamma + a_j) = \prod_{j=1}^{n}(\gamma + b_j)$ and you are done (see prev. slide). Other direction is a bit trickier to prove, but still trivial enough.

Central Equation

Theorem (On fractional lookup check)

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

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

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

Note

This is the central equation used in a large number of studies on its own. What follows is just one variation of how to apply SumCheck using this equation.

Bringing Sum-Check

Suppose the lookup table size is $d = 2^\nu$ for some ν. Then, the table $\vec{t} = \{t_j\}_{j\in[d]}$ can be viewed as a function $t : \{0,1\}^\nu \to \mathbb{F}$.

Problem: attempting to define $\vec{z} = \{z_i\}_{i\in[n]}$ in a similar manner fails since in practice $n > d$, so the function $\{0,1\}^\nu \to \mathbb{F}$ has a too-small domain. Instead, split $\vec{z}$ into $m = \lceil n/d \rceil$ f-ns $z_1, \dots, z_m : \{0,1\}^\nu \to \mathbb{F}$.

Note

In other words, we reduced the problem of $\{z_i\}_{i\in[n]} \subseteq \{t_j\}_{j\in[d]}$ to $\bigcup_{i\in[m]}\{z_i(\mathbf{x})\}_{\mathbf{x}\in\{0,1\}^\nu} \subseteq \{t(\mathbf{x})\}_{\mathbf{x}\in\{0,1\}^\nu}$.

Sum-Check equation is as follows:

$$\boxed{\sum_{\mathbf{x}\in\{0,1\}^\nu}\sum_{i\in[m]}\frac{1}{\gamma+z_i(\mathbf{x})} = \sum_{\mathbf{x}\in\{0,1\}^\nu}\frac{\mu(\mathbf{x})}{\gamma+t(\mathbf{x})}},$$

where $\mu(\mathbf{x}) = \sum_{i\in[m]}\#\{\mathbf{y}\in\{0,1\}^\nu : z_i(\mathbf{y}) = t(\mathbf{x})\}$.

Running Sum-Check. Or Not?

Idea #1: Run Sum-Check on the sum:

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

Problem. $\zeta(\mathbf{x})$ is a fraction, so we can't quite run the Sum-Check yet. Idea of logup is to split the sum into ℓ terms:

$$\zeta(\mathbf{x}) = \underbrace{\frac{\mu(\mathbf{x})}{\gamma + t(\mathbf{x})} - \frac{1}{\gamma + z_0(\mathbf{x})} - \dots - \frac{1}{\gamma + z_{\ell-2}(\mathbf{x})}}_{\zeta_0(\mathbf{x}), \ell \text{ terms}}$$

$$\underbrace{- \frac{1}{\gamma + z_{\ell-1}(\mathbf{x})} - \dots - \frac{1}{\gamma + z_{2\ell-2}(\mathbf{x})}}_{\zeta_1(\mathbf{x}), \ell \text{ terms}} - \dots$$

We form $k \approx m/\ell$ *helper columns* $\{h_i(\mathbf{x})\}_{i \in [k]}$ that satisfy: (a) $h_i(\mathbf{x})$ agrees with $\zeta_i(\mathbf{x})$ over $\{0,1\}^\nu$, (b) $\sum_{\mathbf{x} \in \{0,1\}^\nu} \sum_{i=1}^{k} h_i(\mathbf{x}) = 0$.

Enforcing correct helper columns

Idea #2: Combine k zero-checks using random scalars $\{\lambda_i\}_{i\in[k]} \leftarrow\$ \mathbb{F}$ and merge into a single Sum-Check protocol.

For simplicity, assume each $\zeta_i(\mathbf{x}) = \sum_{j\in\mathcal{I}_i} \frac{q_j(\mathbf{x})}{r_j(\mathbf{x})}$. Note that:

$$\zeta_i(\mathbf{x}) = \sum_{j\in\mathcal{I}_i} \frac{q_j(\mathbf{x})}{r_j(\mathbf{x})} = \frac{\sum_{j\in\mathcal{I}_i} q_j \prod_{k\in\mathcal{I}_i\setminus\{j\}} r_k(\mathbf{x})}{\prod_{j\in\mathcal{I}_i} r_j(\mathbf{x})} = h_i(\mathbf{x})$$

$$\implies h_i(\mathbf{x}) \prod_{j\in\mathcal{I}_i} r_j(\mathbf{x}) = \sum_{j\in\mathcal{I}_i} q_j \prod_{k\in\mathcal{I}_i\setminus\{j\}} r_k(\mathbf{x})$$

Example

For $\ell = 2$, $h_0(\mathbf{x}) = \frac{\mu(\mathbf{x})}{\gamma+t(\mathbf{x})} - \frac{1}{\gamma+z_0(\mathbf{x})} = \frac{\mu(\mathbf{x})(\gamma+z_0(\mathbf{x}))-\gamma-t(\mathbf{x})}{(\gamma+t(\mathbf{x}))(\gamma+z_0(\mathbf{x}))}$. Thus, we enforce the following equality:

$$h_0(\mathbf{x})(\gamma + t(\mathbf{x}))(\gamma + z_0(\mathbf{x})) = \mu(\mathbf{x})(\gamma + z_0(\mathbf{x})) - \gamma - t(\mathbf{x})$$

Final Sum-Check

This way, one runs the Sum-Check on the following function for randomly sampled $\boldsymbol{a} \leftarrow\$ \mathbb{F}^n$:

$$\boxed{\sum_{r\in[k]} h_r(\mathbf{x}) + \mathsf{eq}(\mathbf{x};\boldsymbol{a})\beta_r\left(h_r(\mathbf{x})\prod_{i\in\mathcal{I}_r} r_i(\mathbf{x}) - \sum_{i\in\mathcal{I}_r} q_i(\mathbf{x})\prod_{j\in\mathcal{I}_r\setminus\{i\}} r_j(\mathbf{x})\right) = 0}$$

- The first term enforces that $\sum_{\mathbf{x}\in\{0,1\}^\nu}\sum_{r\in[k]} h_r(\mathbf{x}) = 0$.
- The second term checks whether the helper columns are consistent with each $\zeta_i(\mathbf{x})$ (see prev. slide). For this, we check the equality $h_i(\mathbf{x})\prod_{j\in\mathcal{I}_i} r_j(\mathbf{x}) = \sum_{j\in\mathcal{I}_i} q_j\prod_{k\in\mathcal{I}_i\setminus\{j\}} r_k(\mathbf{x})$ at a random point $\boldsymbol{a}$ by multiplying by $\mathsf{eq}(\mathbf{x};\boldsymbol{a})$ and taking a linear combination of these equations.

Lemma

The larger ℓ is (size of each chunk), the more complex computations are involved but smaller commitment sizes are required.

Thank you for your attention

♥

zkdl-camp.github.io
github.com/ZKDL-Camp