Whirlaway

Thomas Coratger, Tom Wambsgans

1 Introduction

Whirlaway is a hash-based SNARK focusing on lightweight proofs, powered by the WHIR Polynomial Commitment Scheme [1]. The implementation is still in progress and can be found here. In this document, we explain how the proof system works. Nothing is fundamentally new, simply the combination of several recent techniques (WHIR, Ring-Switching, Sumcheck / Univariate Skip).

2 Notations and Symbols

- log is always in base 2
- $[i]_2$: big-endian bit decomposition of an integer i
- $eq(x, y) := \prod_{i=1}^{n}(x_i y_i + (1-x_i)(1-y_i))$, for x and y in $\mathbb{F}^n$. This "equality multilinear polynomial" verifies: $eq(x, y) = 1$ if $x = y$, 0 otherwise, for x and y both sampled on the hypercube $\{0, 1\}^n$.
- $\mathbb{F}_p$: base field, typically KoalaBear ($p = 2^{31} - 2^{24} + 1$), or BabyBear ($p = 2^{31} - 2^{27} + 1$)
- $\mathbb{F}_q$: extension field ($q = p^{2^\kappa}$)
- M (resp. M'): number of columns (resp. non-preprocessed columns) in the AIR table
- m (resp. m'): smallest integer such that $2^m \geq M$ (resp. $2^{m'} \geq M'$)
- $N = 2^n$: number of rows in the AIR table
- $h_1, \dots, h_u$: transition constraints
- H: batched constraint ($H := \sum_{i=0}^{u-1} h_i \alpha^i$)
- T: multilinear polynomial in $\mathbb{F}_p$ encoding all the (non-preprocessed) columns, with $n + m'$ variables

3 Arithmetization

3.1 AIR

We use AIR arithmetization (Algebraic Intermediate Representation). The witness consists of a list of M columns $c_0, \dots, c_{M-1}$. Each column contains $N = 2^n$ elements if $\mathbb{F}_p$ (we use a power of 2 for simplicity). The goal of the prover is to convince the verifier that the table respects a set of u transition constraints $h_0, \dots h_{u-1}$. Each constraint h is a polynomial in $2M$ variables, which is respected if for all rows $r \in \{0, \dots, N-2\}$:

$$h(c_0[r], \dots, c_{M-1}[r], c_0[r+1], \dots, c_{M-1}[r+1]) = 0$$

3.2 Preprocessed columns

Traditional AIR systems allow the verifier to fix certain cells in the table (see "boundary conditions" here). For technical reasons, we use a slightly different approach: we allow the verifier to fix certain columns, potentially sparse (called "preprocessed columns"). The work of the verifier associated to each preprocessed column is proportional to its number of nonzero rows. We denote by $c_0, \dots, c_{M'-1}$ the non-preprocessed columns and $c_{M'}, \dots, c_{M-1}$ the preprocessed ones.

3.3 Example: Fibonacci sequence

Let's say the prover wants to convince the verifier that the N-th values of the Fibonacci sequence equals F_N. We use $M = 4$ columns:

The first $M' = 2$ columns c_0 and c_1 contain the values of the Fibonacci sequence, which is guaranteed by the constraints:

- $h_0(X_0^{\text{up}}, X_1^{\text{up}}, -, -, -, X_1^{\text{down}}, -, -) = X_1^{\text{down}} - (X_0^{\text{up}} + X_1^{\text{up}})$
- $h_1(-, X_1^{\text{up}}, -, -, X_0^{\text{down}}, -, -, -) = X_0^{\text{down}} - X_1^{\text{up}}$

The last two columns c_2 and c_3 are "preprocessed": their content is enforced by the verifier. In our case we set $c_2 = [1, 0, \dots, 0]$ and $c_3 = [0, \dots, 0, 1]$. We finally use the following constraints, to ensure that the 2 initial values of the sequence are correct (0 and 1), and that the final value equals F_N:

- $h_2(X_0^{\text{up}}, -, X_2^{\text{up}}, -, -, -, -, -) = X_2^{\text{up}} \cdot X_0^{\text{up}}$
- $h_3(-, X_1^{\text{up}}, X_2^{\text{up}}, -, -, -, -, -) = X_2^{\text{up}} \cdot (X_1^{\text{up}} - 1)$

  (When the selector $c_2 \neq 0$ (which turns out to be the case at the initial row), we necessarily have $c_0 = 0$ and $c_1 = 1$)
- $h_4(X_0^{\text{up}}, -, -, X_3^{\text{up}}, -, -, -, -) = X_3^{\text{up}} \cdot (X_0^{\text{up}} - F_n)$

  (When the selector $c_3 \neq 0$ (which turns out to be the case at the final row), we necessarily have $c_0 = F_n$)

Note that c_2 and c_3 are sparse, both contain only one non-zero index. As a consequence, they have a negligible impact on the verification time.

Table 1: Fibonacci Sequence AIR Example

Row	c_0	c_1	c_2 (preproc.)	c_3 (preproc.)
0	0	1	1	0
1	1	1	0	0
2	1	2	0	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
N-1	F_N	F_{N+1}	0	1

4 Proving system

4.1 Commitment

Contrary to most of the STARK systems, which use a univariate Polynomial Commitment Scheme (PCS), like FRI or KZG, we use instead a multilinear[1] PCS: WHIR [1]. The entire AIR table is encoded and committed as a single multilinear polynomial T (except for the preprocessed columns, which are not committed). T has $n + m'$ variables, where $n = \log N = \log(\text{number of rows})$ and $m' = \lceil \log M' \rceil = \lceil \log(\text{number of non-preprocessed columns}) \rceil$. T is defined in the lagrange basis (by its evaluations on the hypercube).

For every (non-preprocessed) column i ($0 \leq i < M'$), for every row r ($0 \leq r < N$):

$$T([i]_2[r]_2) := c_i[r]$$

Where $[i]_2$ and $[r]_2$ are the corresponding bit decomposition (big-endian) of i and r (e.g. $M' = 20, N = 128, i = 3, r = 33, [i]_2[r]_2 = (00011|0100001)$).

The undefined evaluations ($M' \leq i < 2^{m'}$) are irrelevant and can be set to zero.

[1]a multivariate polynomial with degree at most one in each variable

Note that the coefficients of T are in the base field $\mathbb{F}_p$. The random evaluation point at which T will be queried later by the verifier is in the extension field $\mathbb{F}_q$ (for soundness). To avoid the "embedding overhead" of committing in the extension field, we use the **ring-switching** protocol (see section 3 of [2]) to commit in the base field and open in the extension field.

4.2 Batching the constraints

After receiving the commitment to T, the verifier sends a random scalar $\alpha \in \mathbb{F}_q$ to the prover. Except with small soundness error, we can replace the u transition constraints by a single one: $H := \sum_{i=0}^{u-1} h_i \alpha^i$.

4.3 Zerocheck

The main argument comes from [3] (see also [4]).

For each column c, we define the multilinear polynomials c^{up} in n variables by:

$$c^{\text{up}}([r]_2) = \begin{cases} c[r] & \text{if } r \in \{0, \dots, N-2\} \\ c[N-2] & \text{if } r = N-1 \end{cases}$$

Similarly, we define the multilinear polynomials c^{down} in n variables by:

$$c^{\text{down}}([r]_2) = \begin{cases} c[r+1] & \text{if } r \in \{0, \dots, N-2\} \\ c[N-1] & \text{if } r = N-1 \end{cases}$$

The batched constraint H is respected on the table if and only if:

$$\forall r \in \{0, \dots, N-2\}, \quad H(c_0[r], \dots, c_{M-1}[r], c_0[r+1], \dots, c_{M-1}[r+1]) = 0$$
$$\Leftrightarrow$$
$$\forall r \in \{0, \dots, N-1\}, \quad H(c_0^{\text{up}}([r]_2), \dots, c_{M-1}^{\text{up}}([r]_2), c_0^{\text{down}}([r]_2), \dots, c_{M-1}^{\text{down}}([r]_2)) = 0$$

The last equality can be proven using a zerocheck (see [5]), assuming the verifier has oracle access to $c_0^{\text{up}}, \dots, c_{M-1}^{\text{up}}$ and $c_0^{\text{down}}, \dots, c_{M-1}^{\text{down}}$, which will be addressed in 4.4. The zerocheck is performed as follows:

- The verifier sends a random vector $r \in (\mathbb{F}_q)^n$
- Prover and verifier run the sumcheck protocol to prove that:

$$\sum_{b \in \{0,1\}^n} eq(b, r) \cdot H(c_0^{\text{up}}(b), \dots, c_{M-1}^{\text{up}}(b), c_0^{\text{down}}(b), \dots, c_{M-1}^{\text{down}}(b)) = 0$$

- Let $\beta \in (\mathbb{F}_q)^n$ be the vector of random challenges sent during the sumcheck. The verifier needs to evaluate the expression inside the sum above for $b \leftarrow \beta$.

  $eq(\beta, r)$ can be easily computed.

  To handle the other factor, the prover sends the claimed values of $c_0^{\text{up}}(\beta), \dots, c_{M-1}^{\text{up}}(\beta)$ and $c_0^{\text{down}}(\beta), \dots, c_{M-1}^{\text{down}}(\beta)$ (correctness will be addressed in 4.4). Given these $2M$ values, the verifier can finally evaluate H, which concludes the zerocheck.

4.4 Oracle access to c^{up} and c^{down}

In 4.3, for each column c_i, the prover has sent two values: $\text{claim}_i^{\text{up}}$ and $\text{claim}_i^{\text{down}}$ respectively equal to $c_i^{\text{up}}(\beta)$ and $c_i^{\text{down}}(\beta)$ in the honest case. It is now time to prove the correctness of these $2M$ evaluations.

First, the verifier sends a random challenge $\gamma \in \mathbb{F}_q$. Except with small soundness error, the $2M$ claims can be reduced to the following:

$$\sum_{i=0}^{M-1} (\gamma^i \cdot \text{claim}_i^{\text{up}} + \gamma^{i+M} \cdot \text{claim}_i^{\text{down}}) \overset{?}{=} \sum_{i=0}^{M-1} (\gamma^i \cdot c_i^{\text{up}}(\beta) + \gamma^{i+M} \cdot c_i^{\text{down}}(\beta)) \tag{1}$$

The verifier can easily compute the left side. To handle the right side, the following protocol is used:

4.4.1 Expression of $c^{\mathbf{up}}$

For every column c, for every $r \in (\mathbb{F}_q)^n$, we have:

$$c^{\text{up}}(r) = \sum_{b \in \{0,1\}^n} \underbrace{[eq(b,r) \cdot (1 - eq(r, (\underbrace{1, \dots, 1}_{n \text{ times}})) + eq((r,b), (\underbrace{1, \dots, 1}_{2n-1 \text{ times}}, 0)]}_{\text{shift}^{\text{up}}(r,b)} \cdot \tilde{c}(b))$$

Where $\tilde{c}$ represents the multilinear extension (MLE) of c.

4.4.2 Expression of $c^{\mathbf{down}}$

For every column c, for every $r \in (\mathbb{F}_q)^n$, we have:

$$c^{\text{down}}(r) = \sum_{b \in \{0,1\}^n} \underbrace{[\text{next}(r,b) + eq((r,b), (\underbrace{1, \dots, 1}_{2n \text{ times}}))]}_{\text{shift}^{\text{down}}(r,b)} \cdot \tilde{c}(b)$$

Where "next" is the multilinear polynomial in $2n$ variables defined on the hypercube by:

$$\text{next}([x]_2[y]_2) = \begin{cases} 1 & \text{if } y = x + 1 \\ 0 & \text{otherwise} \end{cases} \text{ for every pair of n-bit integers } (x, y)$$

See section 5.1 of [3] for more details.

4.4.3 Yet another sumcheck

The right side of (1) can thus be expressed as:

$$\sum_{b \in \{0,1\}^n} \underbrace{\sum_{i=0}^{M-1} [\gamma^i \cdot \text{shift}^{\text{up}}(\beta, b) + \gamma^{i+M} \cdot \text{shift}^{\text{down}}(\beta, b)] \cdot \tilde{c}_i(b)}_{\text{expr}(\beta,b)}$$

A second sumcheck (with respect to b) is used to compute this sum. Let $\delta \in (\mathbb{F}_q)^n$ be the corresponding vector of challenges. The verifier must finally evaluate $\text{expr}(\beta, \delta)$. Both shift^{up} and $\text{shift}^{\text{down}}$ can be succinctly computed. It remains $(\tilde{c}_i(\delta))_{0 \le i < M}$, the evaluations of the columns MLEs on a common point δ.

4.5 PCS opening

The verifier evaluates by himself $(\tilde{c}_i(\delta))_{M' \le i < M}$ on the preprocessed columns.

The prover sends to the verifier $(v_i)_{0 \le i < M'}$, equal to $(\tilde{c}_i(\delta))_{0 \le i < M'}$ in the honest case.

The verifier sends a vector z of $m' = \lceil \log M' \rceil$ random scalars in $\mathbb{F}_q$.

The verifier computes $\sum_{i \in \{0,1\}^{m'}} eq(i, z) \cdot v_i$, then requests a PCS opening for $T((z, \delta))$, and requires that both evaluations coincide.

5 Univariate skip

The sumcheck protocol reduces a sum over a multivariate polynomial by sending a sequence of univariate polynomials. Traditionally, this is done one variable at a time, across n rounds. However, only the first round happens fully in the base field $\mathbb{F}_p$ — later rounds require evaluating at verifier-sent points in the extension field $\mathbb{F}_q$, which is computationally more expensive.

The *univariate skip* optimization (from [6]) leverages this asymmetry: by reorganizing the domain and algebraic structure, we can perform the first k rounds of sumcheck all at once, entirely within $\mathbb{F}_p$. This leads to major efficiency improvements.

5.1 Changing the evaluation domain

Traditionally, all evaluations are performed over the Boolean hypercube $H^n = \{0,1\}^n$. To skip k variables, we restructure the domain to:

$$D \times H^{n-k} \tag{2}$$

where:

- $D \subset \mathbb{F}_p$ is a multiplicative subgroup of size 2^k
- H^{n-k} is the Boolean hypercube in the remaining $n-k$ variables

Each row of the table is now indexed by a pair $(x, \mathbf{y})$ with $x \in D$, $\mathbf{y} \in H^{n-k}$.

5.2 Table columns as polynomials

Each table column is reinterpreted as a polynomial:

$$f_i(x, \mathbf{y}) \in \mathbb{F}_p[x, y_0, \ldots, y_{n-k-1}] \tag{3}$$

such that:

- $\deg_x f_i \leq |D| - 1$ (i.e. low-degree over x)
- f_i is multilinear in each y_j

This is just a change of viewpoint: instead of thinking of table columns as defined over H^n, we interpolate them over $D \times H^{n-k}$.

5.3 Constraint polynomial and composition

Let the prover have ℓ table columns $f_0(x, \mathbf{y}), \ldots, f_{\ell-1}(x, \mathbf{y})$ defined over the domain $D \times H^{n-k}$. These are low-degree in x and multilinear in $\mathbf{y}$. A constraint polynomial is a function $C(z_0, \ldots, z_{\ell-1})$ that expresses relations among columns. Composing it with the table polynomials gives:

$$C(x, \mathbf{y}) := C(f_0(x, \mathbf{y}), \ldots, f_{\ell-1}(x, \mathbf{y})) \tag{4}$$

This multivariate polynomial should vanish on all points of the table if the constraint is satisfied.

Example. Consider $n = 2$, $k = 1$ (so $x \in D$ and $\mathbf{y} \in H^1 = \{0,1\}$), and $\ell = 2$ columns. Suppose the constraint is simply:

$$C(z_0, z_1) = z_1 - z_0 \tag{5}$$

This means, column 1 must equal column 0. Let $D = \{1, \omega\} \subset \mathbb{F}_p$ for some subgroup of size 2 (e.g., $\omega^2 = 1$). The table is indexed by $(x, y) \in D \times \{0,1\}$, and might look like:

Table 2: Constraint $f_1(x, y) = f_0(x, y)$

x	y	$f_0(x,y)$	$f_1(x,y)$	Constraint holds?
1	0	3	3	Yes
1	1	4	4	Yes
ω	0	5	5	Yes
ω	1	6	6	Yes

In this case, the composed polynomial is:

$$C(x, y) = f_1(x, y) - f_0(x, y) \tag{6}$$

which vanishes on all 4 points of $D \times H^1$. This confirms that the constraint is satisfied. This small example illustrates how table rows correspond to evaluations of low-degree polynomials, and how polynomial constraints can be reduced to checking vanishing over a structured domain.

5.4 Constructing the skipped sumcheck polynomial

TODO

Let the verifier send a random challenge vector $\boldsymbol{\alpha} \in \mathbb{F}_q^{n-k}$ for the y-variables. The prover computes the following univariate polynomial:

$$v(x) := \sum_{\mathbf{y} \in H^{n-k}} eq(\mathbf{y}, \boldsymbol{\alpha}) \cdot C(x, \mathbf{y}) \tag{7}$$

This is a standard sumcheck-style reduction: the verifier will later check that $v(x_0)$ equals the expected value at a randomly chosen $x_0 \in D$. The key point: this polynomial $v(x)$ encodes all k skipped rounds at once. It is univariate, and its degree is at most:

$$\deg v(x) \leq d(|D| - 1) = d(2^k - 1) \tag{8}$$

The prover can compute this using at most $d \cdot 2^k$ evaluations of $C(x, \mathbf{y})$, all in the base field $\mathbb{F}_p$.

5.5 Why this is cheaper

Compare this to the naive approach where one attempts to skip k rounds by constructing a multivariate polynomial $v(X_0, \ldots, X_{k-1})$. That polynomial would require:

$$(d+1)^k \tag{9}$$

evaluations for interpolation — exponential in k. In contrast, the univariate skip produces a degree-$d(2^k - 1)$ polynomial that can be interpolated with only $O(2^k)$ evaluations, which scales much better.

5.6 Verifier check

The verifier:

- sends $\boldsymbol{\alpha} \in \mathbb{F}_q^{n-k}$
- receives the univariate polynomial $v(x)$ from the prover
- samples a random $x_0 \in D$
- evaluates or requests an opening at x_0

This single round replaces k rounds of classical sumcheck — and crucially, all the evaluations are in $\mathbb{F}_p$.

5.7 Cost comparison

Let $C_{\mathbb{F}_p}$ and $C_{\mathbb{F}_q}$ be the cost of evaluating the constraint in the base field and extension field respectively.

- Standard sumcheck (with k rounds):

$$\underbrace{2^{n-1}(d-1)C_{\mathbb{F}_p}}_{\text{round 0}} + \underbrace{2^{n-k}(d-1)(2^{k-1}-1)C_{\mathbb{F}_q}}_{\text{rounds 1 to } k-1} \tag{10}$$

- Univariate skip (skip k rounds):

$$\underbrace{2^{n-k}(d-1)(2^k-1)C_{\mathbb{F}_p}}_{\text{all done in base field}} \tag{11}$$

When $\mathbb{F}_q$ is significantly more expensive to evaluate in than $\mathbb{F}_p$ (as is typically the case), this optimization leads to substantial speedups.

References

[1] G. Arnon, A. Chiesa, G. Fenzi, and E. Yogev, “WHIR: Reed–solomon proximity testing with super-fast verification,” 2024. [Online]. Available: https://eprint.iacr.org/2024/1586

[2] B. E. Diamond and J. Posen, “Polylogarithmic proofs for multilinears over binary towers,” 2024. [Online]. Available: https://eprint.iacr.org/2024/504

[3] S. Setty, J. Thaler, and R. Wahby, “Customizable constraint systems for succinct arguments,” 2023. [Online]. Available: https://eprint.iacr.org/2023/552

[4] W. Borgeaud, “A simple multivariate air argument inspired by superspartan,” 2023. [Online]. Available: https://solvable.group/posts/super-air/

[5] B. Chen, B. Bünz, D. Boneh, and Z. Zhang, “HyperPlonk: Plonk with linear-time prover and high-degree custom gates,” 2022. [Online]. Available: https://eprint.iacr.org/2022/1355

[6] A. Gruen, “Some improvements for the PIOP for ZeroCheck,” 2024. [Online]. Available: https://eprint.iacr.org/2024/108