0.1 Plonk Arithmetization

Consider we have a certain relation $\mathcal{R}$, which we would like to write down into a processing-prone format over the field $\mathbb{F}$. Plonk arithmetizes this relation into a set of *8 polynomials*, which are then used to verify the witness knowledge. Let us start with the concrete example.

Example. To begin with, observe this fairly simple relation $\mathcal{R}_{\text{example}}$: suppose we have a public input $x \in \mathbb{F}$ and public output $y \in \mathbb{F}$, and we want to prove the knowledge of $e \in \mathbb{F}$ such that $e \times x + x - 1 = y$. Formally, we have the following relation:

$$\mathcal{R}_{\text{example}} = \left\{ \begin{matrix} \textbf{Public Statement:} & x, y \in \mathbb{F} \\ \textbf{Witness}: & e \in \mathbb{F} \end{matrix} \;\middle|\; e \times x + x - 1 = y \right\}$$

Remark. Note that of course, from x and y, it is fairly simple to find e: simply take $\frac{1-x+y}{x}$. However, the Plonk arithmetization is not limited to this simple example, and can be applied to more complex relations, such as hash function pre-image knowledge or any NP statement.

0.1.1 Execution Trace

Standard Plonk is defined as a system with two types of gates: **addition** and **multiplication**. We would explain how to build custom gates later. So, let us consider our program in terms of gates with left, right operands and output.

Example. We need **three gates** to encode our program:

1. **Gate #1**: left e, right x, output $u = e \times x$
2. **Gate #2**: left u, right x, output $v = u + x$
3. **Gate #3**: left v, right x, output $w = v + (-1)$

You might have glanced the intuitive formation of what is called *execution trace table* — a matrix T with columns L, R and O (it is common to denote those as A, B, C to distinguish from another matrix we will discuss later). Moreover, we will mark columns **A**, **B** and **C** in bold to indicate that they are vectors from $\mathbb{F}^N$, where here and hereafter, unless stated otherwise, N is the number of gates in the program.

Example. We might visualize the execution trace table T for the example program as follows:

A	**B**	**C**
2	3	6
6	3	9
9	✗	8

Notice how the last row has no value in **B** column (marked by ✗) — this is reasoned by the fact it is not a variable, but rather a constant, meaning it doesn't depend on execution. Also note that the number of gates in this particular circuit is $N = 3$.

Remark. As you might notice, in contrast to classic R1CS (which we used for Groth16), the standard Plonk arithmetization as is only allows two input values to be processed at a time. This way, if Groth16 requires only one constraint for verifying $x_1(x_2 + x_3 + x_4) = x_5$, Plonk would need three constraints to verify the same statement. Custom gates partially solve this problem as we will see later, but it is important to keep in mind.

0.1.2 Encode the program

It is essential to distinguish the definition of the program and its specific evaluation for the sake of simplicity and efficiency — once having established encoding for the program, you might apply it for any reasonable inputs. Therefore, let us at first focus on what defines whether execution trace table will be considered valid for our circuit, because having a table by itself does not tell much, since it can be populated with any values.

For that reason, we would define two matrices — $Q \in \mathbb{F}^{N\times 5}$ and $V \in \mathbb{Z}_{\geq 0}^{N\times 3}$ where $N \in \mathbb{N}$, again, is the number of gates in the program:

- Q is the **Gate Matrix**, which encodes the values of the gates and stores all the intermediate values computed.
- V is the **Wiring Matrix**, which encodes the wiring of the gates, i.e., how the output of one gate is carried as input to another.

Definition 0.1. The **gate matrix** $Q \in \mathbb{F}^{N\times 5}$ has one row per each gate with columns $\mathbf{Q}_L$, $\mathbf{Q}_R$, $\mathbf{Q}_O$, $\mathbf{Q}_M$, $\mathbf{Q}_C$ from $\mathbb{F}^N$. If columns $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C} \in \mathbb{F}^N$ of the execution trace table form valid evaluation of the circuit, then the following holds:

$$A_i(Q_L)_i + B_i(Q_R)_i + A_iB_i(Q_M)_i + C_i(Q_O)_i + (Q_C)_i = 0,\ \forall i \in [N]$$

Using Hadamard product notation, this can be concisely rewritten as:

$$\mathbf{A} \odot \mathbf{Q}_L + \mathbf{B} \odot \mathbf{Q}_R + \mathbf{A} \odot \mathbf{B} \odot \mathbf{Q}_M + \mathbf{C} \odot \mathbf{Q}_O + \mathbf{Q}_C = \mathbf{0}$$

Example. For our program, we would have a following Q table:

$\mathbf{Q}_L$	$\mathbf{Q}_R$	$\mathbf{Q}_M$	$\mathbf{Q}_O$	$\mathbf{Q}_C$
0	0	1	−1	0
1	1	0	−1	0
1	0	0	−1	−1

You can verify that our claim holds for aforementioned trace matrix:

$$2 \times 0 + 3 \times 0 + 2 \times 3 \times 1 + 6 \times (-1) + 0 = 0$$
$$6 \times 1 + 3 \times 1 + 6 \times 3 \times 0 + 9 \times (-1) + 0 = 0$$
$$9 \times 1 + 0 \times 0 + 9 \times 0 \times 0 + 8 \times (-1) + (-1) = 0$$

Recall that columns of trace matrix T are $\mathbf{A} = \begin{bmatrix}2\\6\\9\end{bmatrix}$, $\mathbf{B} = \begin{bmatrix}3\\3\\\times\end{bmatrix}$, $\mathbf{C} = \begin{bmatrix}6\\9\\8\end{bmatrix}$.

Now, we do have a way of encoding gates separately, yet in order to guarantee how result of one gate is carried in as input of the other (*wirings*), we need another matrix — V.

Definition 0.2. The **wiring matrix** $V \in \mathbb{Z}_{\geq 0}^{N \times 3}$ consists of indices of all inputs and intermediate values, so that if T is a valid trace,

$$\forall (i,j)\ \forall (k,\ell) :\ V_{i,j} = V_{k,\ell} \implies T_{i,j} = T_{k,\ell}$$

Put more simply, if two values are equal in V, then the corresponding values (corresponding to these indices) in T must be equal as well.

Example. For our program, V can be defined as follows:

L	R	O
0	1	2
2	1	3
3	✗	4

Here 0 is an index of e, 1 is an index of x, 2 — of intermediate value u, 3 — of v and finally 4 — of output w.

0.1.3 Custom Gates

In order to reach beyond classical operations such as addition and multiplication, one may consider composing a custom gate. The main streamliner of this functionality is a matrix Q, using 5 basic columns of which, you already may build custom logic.

Example. Our entire program may be encoded as one custom gate.

$$Q = \begin{array}{|c|c|c|c|c|} \hline Q_L & Q_R & Q_M & Q_O & Q_C \\ \hline 0 & 1 & 1 & -1 & -1 \\ \hline \end{array} \qquad V = \begin{array}{|c|c|c|} \hline \mathrm{L} & \mathrm{R} & \mathrm{O} \\ \hline 0 & 1 & 2 \\ \hline \end{array} \qquad T = \begin{array}{|c|c|c|} \hline \mathrm{A} & \mathrm{B} & \mathrm{C} \\ \hline 2 & 3 & 8 \\ \hline \end{array}$$

$$2 \times 0 + 3 \times 1 + 2 \times 3 \times 1 + 8 \times (-1) + (-1) = 0$$

As you can see, custom gates is a good way to reduce the number of constraints needed for the same program.

Remark. Real-world PlonK applications commonly have additional columns in the Q matrix, enabling an even broader set of custom functionality.

0.1.4 Public Inputs

With the current design, we can prove that the computations were done correctly, but we have no restrictions on the values of inputs. For example, when the prover wants to convince the verifier that he knows e for $x = 3$ and $y = 7$, the verifier does not even check whether x is 3 (not to mention whether the result of execution $y = 7$ is correct) in the trace table T. One way of doing this is by incorporating them in three previously defined matrices Q, V, T.

Proposition 0.3. One way to solve this is to use the **equality gates**. Introduce two gadgets:

- **Constant Equality Gate:** Suppose we want to check whether the certain variable equals to the constant value $\alpha \in \mathbb{F}$ at gate with index i. For ith gate, set $(Q_L)_i = -1$, $(Q_C)_i = \alpha$ and other columns to 0. Then, add a row to V with $L = i$, $R = \text{✗}$ and $O = \text{✗}$. Then, to satisfy the condition, the ith left input **must** be equal to α.
- **Nodes Equality Gate:** Suppose we want to check whether the ith and jth gates have equal outputs in the kth gate. Set $(Q_L)_k = 1$, $(Q_R)_k = -1$ with other columns to 0. Add a row to V with $L = i$, $R = j$ and $O = \text{✗}$. Then, to satisfy the condition, the ith and jth outputs **must** be equal.

Example. Suppose the prover wants to prove that he knows e for the public statement $(x, y) = (3, 8)$. We can encode this as follows:

$Q =$

$\mathbf{Q}_L$	$\mathbf{Q}_R$	$\mathbf{Q}_M$	$\mathbf{Q}_O$	$\mathbf{Q}_C$
-1	0	0	0	3
-1	0	0	0	8
1	1	1	-1	1
1	-1	0	0	0

$V =$

L	**R**	**O**
0	✗	✗
1	✗	✗
2	0	3
1	3	✗

$T =$

A	**B**	**C**
3	✗	✗
8	✗	✗
2	3	8
8	8	✗

As can be seen, besides the original program gate, inscribed in the third row, we have three additional gates:

- The first two gates "allocate" two nodes with indices 0 and 1 to the values 3 and 8 respectively. This is done through the *constant equality gates*.
- The last gate checks whether the result of the third gate is equal to the index 1, corresponding to the allocated value 8. This is done through the *nodes equality gate*.

The primary problem with this approach, is that now we have lost agnosticism in Q and V of concrete evaluations. In other words, our circuit is now "hardcoded" to the specific values of public inputs. In order to resolve this, we would define a separate one-column matrix named $\mathbf{\Pi} \in \mathbb{F}^N$, in which we would encode the public inputs.

Example. With only Q modified, we now have:

$\mathbf{\Pi}$
3
8
0
0

$Q =$

$\mathbf{Q}_L$	$\mathbf{Q}_R$	$\mathbf{Q}_M$	$\mathbf{Q}_O$	$\mathbf{Q}_C$
-1	0	0	0	0
-1	0	0	0	0
1	1	1	-1	1
1	-1	0	0	0

Proposition 0.4 (Wrap-up)**.** The matrix T with columns **A**, **B** and **C** $\in \mathbb{F}^N$ encodes correct execution of the program, if the following two conditions hold:

1. $\forall i \in [N] : A_i(Q_L)_i + B_i(Q_R)_i + A_iB_i(Q_M)_i + C_i(Q_O)_i + (Q_C)_i + \Pi_i = 0$
2. $\forall (i, j)\, \forall (k, \ell) : V_{i,j} = V_{k,\ell} \implies T_{i,j} = T_{k,\ell}$

0.1.5 Matrices to Polynomials

Gates Satisfability. Now we can traduce the sets of constraints on matrices to just a few equations on polynomials, as we have already done for Groth16. Let ω be a primitive N-th root of unity[1] and let $\Omega = \{\omega^j\}_{0 \leq j < N}$. Although currently the choice of set Ω might seem totally random, in the next sections we will see how the usage of Fast-Fourier Transform (FFT) will make this choice convenient.

Let $a, b, c, q_L, q_R, q_M, q_O, q_C, \pi \in \mathbb{F}^{(\leq N)}[X]$ be polynomials of degree at most N that interpolate corresponding columns from matrices at the domain Ω. In other words, we have $\forall j \in [N] : a(\omega^j) = A_j$ and the same holds for other polynomials.

Notice that if our trace matrix is correct, then the first condition of Proposition 0.4 can be reduced to the following polynomial equation:

$$a(\omega^j)q_L(\omega^j) + b(\omega^j)q_R(\omega^j) + a(\omega^j)b(\omega^j)q_M(\omega^j) + c(\omega^j)q_O(\omega^j) + q_C(\omega^j) + \pi(\omega^j) = 0, \ \forall j \in [N]$$

Notice that this essentially means that the left polynomial $aq_L + bq_R + abq_M + cq_O + q_C + \pi$ has roots at ω^j for all $j \in [N]$. This is equivalent to stating that the polynomial $z_\Omega(X) = \prod_{j=0}^{N-1}(X - \omega^j)$ divides the left hand side. Now, the interesting fact...

Lemma 0.5. It so happens that if Ω is a set of N-th roots of unity, then the polynomial $z_\Omega(X) = X^N - 1$ is the vanishing polynomial of Ω.

Proof Idea. If ω is the Nth primitive root, then for any $h \in \Omega$ we have $h^N = 1$ and therefore all elements of Ω are the roots of $X^N - 1$. There are precisely N such roots, so $X^N - 1$ can be decomposed as a product of linear factors $c \cdot \prod_{j=0}^{N-1}(X - \omega^j)$. It is easy to see that $c = 1$ by comparing the leading coefficient.

Aha! So we have that $X^N - 1$ must divide the left polynomial. Let us wrap this up in the following proposition.

Proposition 0.6. Now we can reduce down our first condition of Proposition 0.4 to checking valid execution trace into the following claim over polynomials:

$$\exists t \in \mathbb{F}^{(\leq 3N)}[X] : \ aq_L + bq_R + abq_M + cq_O + q_C + \pi = z_\Omega t,$$

where $z_\Omega(X)$ is the vanishing polynomial $X^N - 1$.

Wiring Satisfability. The next step is to shrink the second condition imposed by the V matrix. This may be achieved by introducing the concept of permutation.

Remark. Permutation of the set S is commonly denoted as $\sigma : S \to S$. This function is bijective, meaning that for every $s \in S$ there exists a unique $s' \in S$ such that $\sigma(s) = s'$.

[1] Suppose such ω exists, then $\omega^N = 1$ and $\omega^j \neq 1$ for $0 \leq j < N$.

Example. A permutation is a rearrangement of the set, which is in our case:

$$\mathcal{I} = \{(i,j) : \text{such that } 0 \leq i < N, \text{ and } 0 \leq j < 3\}$$

Naturally, the matrix V induces a permutation σ of this set where $\sigma((i,j))$ equals to the pair of indices of the next occurrence of the value at position (i,j). So, for our example:

$V =$

L	R	O
0	✗	✗
1	✗	✗
2	**0**	3
1	3	✗

We have the following permutation:

$$\sigma(\textbf{(0,0)}) = \textbf{(2,1)},\ \sigma((0,1)) = (0,3),\ \sigma((0,2)) = (0,2)$$

$$\sigma((0,3)) = (0,1),\ \sigma((2,1)) = (0,0),\ \sigma((3,1)) = (2,2)$$

For demonstration purposes, we marked in **green** the index of the first and second occurance of the value 0. For proper σ definition (as it has to be bijective), the application of σ to the last occurance outputs the first one.

Permutation Check. This is probably the most tedious part of PlonK. We split the following derivation into two parts:

- **Set Equality using Polynomials.** We will show how to check whether two sets of field elements are equal using polynomials.
- **Permutation Check using Polynomials.** We will show how to check whether a given function is a permutation using polynomials in several forms.

Set equality. Having defined permutation, we can now reduce the second condition of Proposition 0.4 of valid execution trace matrix into the following check:

$$\forall (i,j) \in \mathcal{I} : T_{i,j} = T_{\sigma(i,j)}$$

You may have noticed how this can be reformulated as equality of two sets A and B:

$$A := \{((i,j), T_{i,j}) : (i,j) \in \mathcal{I}\}$$
$$B := \{(\sigma((i,j)), T_{i,j}) : (i,j) \in \mathcal{I}\}$$

We can reduce this check down to polynomial equations! Here is how: suppose for simplicity we have two sets with two elements $A = \{a_0, a_1\}$ and $B = \{b_0, b_1\}$. Introduce two sets of polynomials $A' = \{a_0 + X, a_1 + X\}$ and $B' = \{b_0 + X, b_1 + X\}$.

When do we have the set equality $A' = B'$? Well, $(a_0 + X)(a_1 + X) = (b_0 + X)(b_1 + X)$ works fine. This is true because of linear polynomial unique factorization property, working as prime factors. Now, we can utilize Schwartz-Zippel lemma to replace the latter formula with $(a_0 + \gamma)(a_1 + \gamma) = (b_0 + \gamma)(b_1 + \gamma)$ for some random $\gamma \xleftarrow{R} \mathbb{F}$ with overwhelming probability, being at least $1 - 2/|\mathbb{F}|$. If we wish to generalize this for arbitrary sets $A = \{a_0, \ldots, a_{k-1}\}$ and $B = \{b_0, \ldots, b_{k-1}\}$, apply the following equivalent check:

$$\prod_{i=0}^{k-1}(a_i+\gamma)=\prod_{i=0}^{k-1}(b_i+\gamma)$$

Let Ω be a domain of the form $\{1,\omega,\ldots,\omega^{k-1}\}$ for some k-th root of unity ω. Let f and g be polynomials that we interpolate at Ω as follows:

$$f(\omega^j)=a_j+\gamma,\quad g(\omega^j)=b_j+\gamma,\quad j\in[k]$$

Then, $\prod_{i=0}^{k-1}(a_i+\gamma)=\prod_{i=0}^{k-1}(b_i+\gamma)$ holds if and only if there is a polynomial $Z\in\mathbb{F}[X]$ such that for all $h\in\Omega$ we have $Z(\omega^0)=1$ and $Z(h)f(h)=g(h)Z(\omega h)$.

Now that we can encode equality of sets of field elements, let's expand this to sets of tuples of field elements. Let $A=\{(a_0,a_1),(a_2,a_3)\}$ and $B=\{(b_0,b_1),(b_2,b_3)\}$. Then, similarly, if

$$A'=\{a_0+a_1Y+X,a_2+a_3Y+X\},\quad B'=\{b_0+b_1Y+X,b_2+b_3Y+X\},$$

then $A=B$ if and only if $A'=B'$. As before, we can leverage Schwartz-Zippel lemma to reduce this down into sampling two random β and $\gamma\xleftarrow{R}\mathbb{F}$ and checking equality of:

$$(a_0+\beta a_1+\gamma)(a_2+\beta a_3+\gamma)=(b_0+\beta b_1+\gamma)(b_2+\beta b_3+\gamma)$$

Permutation Check. Now, to go back to the second condition of Proposition 0.4 which we are trying to formulate in the polynomial domain, it becomes clear that if we somehow encoded inner indices tuple (i,j) into a one field element, we could use the above fact. Recall that $i\in[N]$ and $j\in\{0,1,2\}$. Thus, take the $3N$-th primitive root of unity η and define the bijective map $((i,j),v)\mapsto(\eta^{3i+j},T_{i,j})$. Thus, consider the modified sets:

$$\begin{aligned}A&=\{(\eta^{3i+j},T_{i,j}):(i,j)\in\mathcal{I}\}\\B&=\{(\eta^{3k+\ell},T_{i,j}):(i,j)\in\mathcal{I},\sigma((i,j))=(k,\ell)\}\end{aligned}$$

Sample two random field elements β and $\gamma\xleftarrow{R}\mathbb{R}$. Let $\mathcal{D}=\{1,\eta,\eta^2,\ldots,\eta^{3N-1}\}$. Then, interpolate two polynomials f and g over the defined set $\mathcal{D}$ as follows:

$$\begin{aligned}f(\eta^{3i+j})&=T_{i,j}+\eta^{3i+j}\beta+\gamma,\quad (i,j)\in\mathcal{I}\\g(\eta^{3k+\ell})&=T_{i,j}+\eta^{3k+\ell}\beta+\gamma,\quad (i,j)\in\mathcal{I},\quad \sigma((i,j))=(k,\ell)\end{aligned}$$

Similarly to our previous discussion, there should be a polynomial $Z\in\mathbb{F}[X]$ such that $\forall d\in\mathcal{D}$, we have $Z(\eta^0)=1$ and $Z(d)f(d)=g(d)Z(\eta d)$. This would imply the set equality $A=B$ with overwhelming probability according to Schwartz-Zippel lemma.

Shorter Form. Now, using the $3N$-th root of unity is a bit of overkill, so let us try compressing it down to $\Omega=\{\omega^j\}_{0\le j<N}$ where ω is the Nth root of unity. We will define three polynomials $S_{\sigma,1}$, $S_{\sigma,2}$, $S_{\sigma,3}\in\mathbb{F}[X]$, which are interpolated as follows:

$$\begin{aligned}S_{\sigma,1}(\omega^i)&=\eta^{3k+\ell},\quad (i,0)\in\mathcal{I},\quad \sigma((i,0))=(k,\ell)\\S_{\sigma,2}(\omega^i)&=\eta^{3k+\ell},\quad (i,1)\in\mathcal{I},\quad \sigma((i,1))=(k,\ell)\\S_{\sigma,3}(\omega^i)&=\eta^{3k+\ell},\quad (i,2)\in\mathcal{I},\quad \sigma((i,2))=(k,\ell)\end{aligned}$$

Let k_1 and k_2 be two field elements such that $\omega^i \neq \omega^j k_1 \neq \omega^\ell k_2$ for all possible triplets i, j, ℓ. Recall that β and γ are random field elements. Let f and g be the polynomials that interpolate, respectively, the following values at Ω:

$$f(\omega^i) = \left(T_{i,0} + \omega^i\beta + \gamma\right)\left(T_{i,1} + \omega^i k_1\beta + \gamma\right)\left(T_{i,2} + \omega^i k_2\beta + \gamma\right), \quad i \in [N]$$
$$g(\omega^i) = \left(T_{i,0} + S_{\sigma,1}(\omega^i)\beta + \gamma\right)\left(T_{i,0} + S_{\sigma,2}(\omega^i)\beta + \gamma\right)\left(T_{i,0} + S_{\sigma,3}(\omega^i)\beta + \gamma\right), \quad i \in [N]$$

That being said, there is a polynomial $Z \in \mathbb{F}[X]$ such that $\forall d \in \mathcal{D}$ we have $Z(\omega^0) = 1$ and $Z(d)f(d) = g(d)Z(\omega d)$, implying $A = B$ with overwhelming probability. That being said, we now can encode our program using 8 polynomials mentioned at the very beginning:

$$q_L, q_R, q_M, q_O, q_C, S_{\sigma,1}, S_{\sigma,2}, S_{\sigma,3}$$

These are typically called **common preprocessed input**.

0.1.6 Summary

Having a program for relation $\mathcal{R}$, we saw how it can be represented as a sequence of gates with left, right operands and output. The circuit may be encoded using two matrices Q — for capturing gates, and V — for encoding value carries (*wirings*). Upon execution, we get trace execution matrix T and Π for public inputs.

Definition 0.7. Let $T \in \mathbb{F}^{N\times 3}$ be a trace matrix with columns $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C} \in \mathbb{F}^N$ and let $\mathbf{\Pi} \in \mathbb{F}^N$ be a public input vector. They correspond to a valid execution instance with public input given by $\mathbf{\Pi}$ if and only if:

1. $\forall i \in [N] : A_i(Q_L)_i + B_i(Q_R)_i + A_iB_i(Q_M)_i + C_i(Q_O)_i + (Q_C)_i + \Pi_i = 0$
2. $\forall (i,j), \forall (k,\ell) : V_{i,j} = V_{k,\ell} \implies T_{i,j} = T_{k,\ell}$
3. $\forall i > n : \Pi_i = 0$, where n is the number of public inputs.

Then, we encode these conditions in terms of polynomials.

Definition 0.8. Let $z_\Omega = X^N - 1$ be a vanishing polynomial. Let $T \in \mathbb{F}^{N\times 3}$ be a trace matrix with columns $\mathbf{A}, \mathbf{B}, \mathbf{C} \in \mathbb{F}^N$ and $\mathbf{\Pi} \in \mathbb{F}^N$ be a vector of public signals. They correspond to a valid execution instance with public input given by $\mathbf{\Pi}$ if and only if:

1. $\exists t_1 \in \mathbb{F}[X] : aq_L + bq_R + abq_M + cq_O + q_C + \pi = z_\Omega t_1$
2. $\exists t_2, t_3, z \in \mathbb{F}[X] : zf - gz' = z_\Omega t_2$ and $(z-1)L_1 = z_\Omega t_3$, where $z'(X) = z(X\omega)$.

Remark. We can reduce every needed check down to one equation, if we introduce randomness. Let α be a random field element, then:

$$\begin{aligned} z_\Omega t &= aq_L + bq_R + abq_M + cq_O + q_C + \pi \\ &= \alpha(gz' - fz) \\ &= \alpha^2(z-1)L_1 \end{aligned}$$

The transition between second and third line is very unobvious and requires a bit of algebraic manipulation. Don't worry if you don't see it immediately.