0.1 Plonk Arithmetization

Consider we have a certain program P, which we would like to write down into a processing-prone format. For that reason, we would arithmetize it into a set of 8 polynomials.

> **Example.** To begin with, observe this simple example P:
>
> Input: x
> Private input: e
> Output: e * x + x - 1

0.1.1 Execution Trace

Standard Plonk is defined as a system with two types of gates for 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.**
> 1. Gate: left e, right x, output $u = e \times x$
> 2. Gate: left u, right x, output $v = u \times x$
> 3. Gate: 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).

Example.

A	B	C
2	3	6
6	3	9
9		8

Notice how the last row has no value in B column - this is reasoned by the fact it is not a variable, but rather a constant, meaning it doesn't depend on execution.

0.1.2 Encode the program

It is important to distinguish between definition of the program, and it's 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 and V.

Definition 0.1. Q matrix has one row per each gate with columns Q_L, Q_R, Q_O, Q_M, Q_C. If columns A, B and C of the execution trace table form valid evaluation of the circuit,

$$A_iQ_{Li} + B_iQ_{Ri} + A_iB_iQ_{Mi} + C_iQ_{Oi} + Q_{ci} = 0$$

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

Q_L	Q_R	Q_M	Q_o	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:

$$\begin{aligned} 2 \times 0 + 3 \times 0 + 2 \times 3 \times 1 + 6 \times (-1) + 0 = 0 \\ 6 \times 1 + 3 \times 1 + 6 \times 3 \times 0 + 9 \times (-1) + 0 = 0 \\ 9 \times 1 + 0 \times 0 + 9 \times 0 \times 0 + 8 \times (-1) + (-1) = 0 \end{aligned}$$

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. V consists of indices of all inputs and intermediate values, so that if T is a valid trace,

$$\forall i, j, k, l : (V_{i,j} = V_{k,l}) \Rightarrow (T_{i,j} = T_{k,l})$$

Example. For our program, V would look like following:

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 - u, 3 - v and 4 - 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:

Q_L	Q_R	Q_M	Q_o	Q_c
0	1	1	-1	-1

V:

L	R	O
0	1	2

T:

A	B	C
2	3	8

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

As you can see, custom gates can optimize quite a lot.

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 current design, there is no way to demonstrate that appropriate inputs are used in our program. One way of doing this is by incorporating them in three matrices.

Example. Consider, as if we added new *selector* rows to Q and tied them in V and T:

Q:

Q_L	Q_R	Q_M	Q_o	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	

The problem with this approach, is that now we have lost agnosticism in Q and V of concrete evaluations. In order to resolve this, we would define a separate one-column matrix named PI (public inputs).

Example. With only Q modified, we have:

PI
3
8
0
0

Q:

Q_L	Q_R	Q_M	Q_o	Q_c
-1	0	0	0	0
-1	0	0	0	0
1	1	1	-1	1
1	-1	0	0	0

Definition 0.3. Matrix T with columns A, B and C encodes correct execution of the program, if:

1. $\forall i : A_i Q_{Li} + B_i Q_{Ri} + A_i B_i Q_{Mi} + C_i Q_{Oi} + Q_{ci} + PI_i = 0$
2. $\forall i, j, k, l : (V_{i,j} = V_{k,l}) \Rightarrow (T_{i,j} = T_{k,l})$

0.1.5 Matrices to Polynomials

Now we can traduce the sets of constraints on matrices to just a few equations on polynomials. Let w be a primitive N-th root of unity and let $H = w^i : 0 \leq i < N$. Let $a, b, c, q_L, q_R, q_M, q_O, q_C, pi$ be polynomials of degree at most N that interpolate corresponding columns from matrices at the domain H. This means, that $\forall i : a(w^i) = A_i$ and the same for other columns.
Now we can reduce down our first condition to checking valid execution trace into the following claim over polynomials:

$$\exists t : aq_L + bq_R + abq_M + cq_O + q_C + pi = z_H t,$$

where z_H is the vanishing polynomial $X^n - 1$.
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 is commonly denoted as σ.

> **Example.** A permutation is a rearrangement of the set, which is in our case:
>
> $$I = \{(i,j) : \text{such that } 0 \leq i < N, \text{ and } 0 \leq j < 3\}$$
>
> The matrix V induces a permutation of this set where $\sigma((i,j))$ is equal to the 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 | |
>
> $$\sigma((0,0)) = (2,1), \sigma((0,1)) = (0,3), \sigma((0,2)) = (0,2)$$
> $$\sigma((0,3)) = (0,1), \sigma((2,1)) = (0,0), \sigma((3,1)) = (2,2)$$

Having defined permutation, we can now reduce a condition 2 of valid execution trace matrix into the following check:

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

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

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

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

We can reduce this check down to polynomial equations.

Suppose we have sets $A = \{a_0, a_1\}$ and $B = \{b_0, b_1\}$. We can consider polynomials $A' = \{a_0 + X, a_1 + X\}, B' = \{b_0 + X, b_1 + X\}$.
So, $A' = B'$, only if $(a_0+X)(a_1+X) = (b_0+X)(b_1+X)$. 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 γ with high probability If we wish to generalize this for sets $A = \{a_0, \dots, a_{k-1}\}$ and $B = \{b_0, \dots, b_{k-1}\}$:

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

Let H be a domain of the form $\{1, \omega, \dots, \omega^{k-1}\}$ for some k-th root of unity ω. Let f and g be polynomials that interpolate the following values at H:

$$f : (a_0 + \gamma, \dots, a_{k-1} + \gamma)$$
$$g : (b_0 + \gamma, \dots, b_{k-1} + \gamma)$$

Then $\prod_{i=0}^{k-1}(a_i + \gamma) = \prod_{i=0}^{k-1}(b_i + \gamma)$ if only if: $\exists Z \forall h \in H : Z(\omega^0) = 1$ 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 to the previous:

$$A' = \{a_0 + a_1 Y + X, a_2 + a_3 Y + X\}$$

$$B' = \{b_0 + b_1 Y + X, b_2 + b_3 Y + X\}$$

$$A = B \leftrightarrow A' = B'$$

As before, we can leverage Schwartz-Zippel lemma to reduce this down into sampling random β and γ and checking equality of:

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

Now, if we would go back into condition 2 which we are trying to formulate in polynomial domain, it will become clear that first if we somehow encode inner indices tuple (i, j) into a one field element, we can use the above fact. Recall that $i \in [0; N-1]$ and $j \in [0; 2]$; we can take $3N$-th primitive root of unity η and define our field element as $a_0 = \eta^{3i+j}$:

$$A = \{(\eta^{3i+j}, T_{i,j}) : (i, j) \in I\}$$

$$B = \{(\eta^{3k+l}, T_{i,j}) : (i, j) \in I, \sigma((i, j)) = (k, l)\}$$

Then, summarizing: Let η be a $3N$-th primitive root of unity, β and γ random field elements. Let $D = \{1, \eta, \eta^2, \ldots, \eta^{3N-1}\}$. Then let f and g interpolate at D:

$$f : \{T_{i,j} + \eta^{3i+j}\beta + \gamma : (i, j) \in I\}$$

$$g : \{T_{i,j} + \eta^{3k+l}\beta + \gamma : (i, j) \in I, \sigma((i, j)) = (k, l)\}$$

So, $\exists Z \forall h \in H : Z(\eta^0) = 1$ and $Z(h)f(h) = g(h)Z(\eta h) \leftrightarrow A = B$ w.h.p.

Notice, that $\omega = \eta^3$ is a primitive N-th root of unity. Let $H = \{1, \omega, \omega^2, \ldots, \omega^{N-1}\}$. We will define three polynomials, which interpolate following sets:

$$S_{\sigma 1} : \{\eta^{3k+l} : (i, 0) \in I, \sigma((i, 0)) = (k, l)\}$$

$$S_{\sigma 2} : \{\eta^{3k+l} : (i, 1) \in I, \sigma((i, 0)) = (k, l)\}$$

$$S_{\sigma 3} : \{\eta^{3k+l} : (i, 2) \in I, \sigma((i, 0)) = (k, l)\}$$

Let ω be an N-th root of unity. Let $H = \{1, \omega, \omega^2, \ldots, \omega^{N-1}\}$. Let k_1 and k_2 be two field elements such that $\omega^i \neq \omega^j k_1 \neq \omega^l k_2$ for all i, j, l. Let β and γ be random field elements. Let f and g be the polynomials that interpolate, respectively, the following values at H:

$$f : \left\{\left(T_{0,j} + \omega^i\beta + \gamma\right)\left(T_{1,j} + \omega^i k_1\beta + \gamma\right)\left(T_{2,j} + \omega^i k_2\beta + \gamma\right) : 0 \leq i < N\right\}$$

$$g : \left\{\left(T_{0,j} + S_{0,1}(\omega^i)\beta + \gamma\right)\left(T_{0,j} + S_{0,2}(\omega^i)\beta + \gamma\right)\left(T_{0,j} + S_{0,3}(\omega^i)\beta + \gamma\right) : 0 \leq i < N\right\}$$

So, $\exists Z \forall h \in D : Z(\omega^0) = 1$ and $Z(d)f(d) = g(d)Z(\omega d) \leftrightarrow A = B$ w.h.p.

So, we now can encode our program using 8 polynomials mentioned at the beginning:

$$q_L, q_R, q_M, q_O, q_C, S_{01}, S_{02}, S_{03}$$

These are called *common preprocessed input*.

0.1.6 Summary

Having a program P 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 PI for public inputs.

Definition 0.4. Let T be a $N \times 3$ matrix with columns A, B, C and PI a $N \times 1$ matrix. They correspond to a valid execution instance with public input given by PI if and only if:

1. $\forall i : A_i Q_{Li} + B_i Q_{Ri} + A_i B_i Q_{Mi} + C_i Q_{Oi} + Q_{Ci} + PI_i = 0$
2. $\forall i, j, k, l : V_{i,j} = V_{k,l} \implies T_{i,j} = T_{k,l}$
3. $\forall i > n : PI_i = 0$

When, we encode those conditions in terms of polynomials.

Definition 0.5. Let $z_H = X^N - 1$. Let T be a $N \times 3$ matrix with columns A, B, C and PI a $N \times 1$ matrix. They correspond to a valid execution instance with public input given by PI if and only if:

1. $\exists t_1 : aq_L + bq_R + abq_M + cq_O + q_C + pi = z_H t_1$
2. $\exists t_2, t_3, z : zf - gz' = z_H t_2$ and $(z - 1)L_1 = z_H t_3$, 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:

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