QAP, PCP, POE

Oct 1, 2024

Distributed Lab

Plan

Recap

Recap. ZK-SNARK

Definition

zk-SNARK – Zero-Knowledge Succinct Non-interactive ARgument of Knowledge.

Recap. ZK-SNARK

Definition

zk-SNARK – Zero-Knowledge Succinct Non-interactive ARgument of Knowledge.

- **Argument of Knowledge** — a proof that the prover knows the data (witness) that resolves a certain problem, and this knowledge can be "extracted".

Recap. ZK-SNARK

Definition

zk-SNARK – Zero-Knowledge Succinct Non-interactive ARgument of Knowledge.

- **Argument of Knowledge** — a proof that the prover knows the data (witness) that resolves a certain problem, and this knowledge can be "extracted".
- **Succinctness** — the proof size and verification time is relatively small to the computation size and typically does not depend on the size of the data or statement.

Recap. ZK-SNARK

Definition

zk-SNARK – Zero-Knowledge Succinct Non-interactive ARgument of Knowledge.

- **Argument of Knowledge** — a proof that the prover knows the data (witness) that resolves a certain problem, and this knowledge can be "extracted".
- **Succinctness** — the proof size and verification time is relatively small to the computation size and typically does not depend on the size of the data or statement.
- **Non-interactiveness** — to produce the proof, the prover does not need any interaction with the verifier.

Recap. ZK-SNARK

Definition

zk-SNARK – Zero-Knowledge Succinct Non-interactive ARgument of Knowledge.

- **Argument of Knowledge** — a proof that the prover knows the data (witness) that resolves a certain problem, and this knowledge can be "extracted".
- **Succinctness** — the proof size and verification time is relatively small to the computation size and typically does not depend on the size of the data or statement.
- **Non-interactiveness** — to produce the proof, the prover does not need any interaction with the verifier.
- **Zero-Knowledge** — the verifier learns nothing about the data used to produce the proof, despite knowing that this data resolves the given problem and that the prover possesses it.

Recap. Arbitrary Program To Circuits

We can do that in a way like the computer does it - boolean circuits.

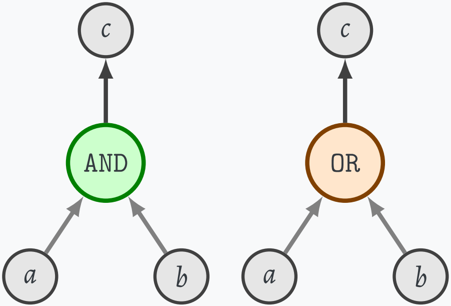

Figure: Boolean AND and OR Gates

But nothing stops us from using something more powerful instead of boolean values...

Recap. Arbitrary Program To Circuits

We can do that in a way like the computer does it - boolean circuits.

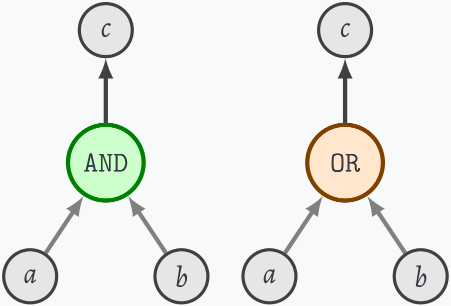

Figure: Boolean AND and OR Gates

> 100000 gates just for SHA256...

Recap. Arbitrary Program To Circuits

We can do that in a way like the computer does it - boolean circuits.

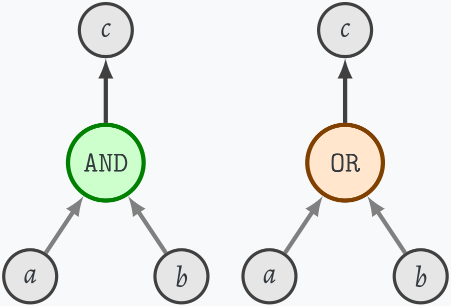

Figure: Boolean AND and OR Gates

> 100000 gates just for SHA256... But nothing stops us from using something more powerful instead of boolean values, gates.

Recap. Arbitrary Program To Circuits

Similar to Boolean Circuits, the **Arithmetic circuits** consist of gates and wires.

- Wires: elements of some finite field $\mathbb{F}_p$.
- Gates: addition ($\oplus$) and multiplication ($\odot$) corresponding to the field.

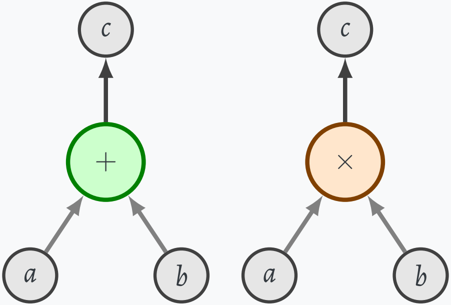

Figure: Addition and Multiplication Gates

Recap. Arbitrary Program To Circuits

Example

How can we translate `if` statements?

```
def example(a: bool, b: F, c: F) -> F:
    if a:
        return b * c
    else:
        return b + c
```

Recap. Arbitrary Program To Circuits

Example

How can we translate `if` statements?

```
def example(a: bool, b: F, c: F) -> F:
    if a:
        return b * c
    else:
        return b + c
```

We can transform such a function into the next expression:

$$r = a \times (b \times c) + (1 - a) \times (b + c)$$

Recap. Arbitrary Program To Circuits

Example

How can we translate `if` statements?

```
def example(a: bool, b: F, c: F) -> F:
    if a:
        return b * c
    else:
        return b + c
```

We can transform such a function into the next expression:

$$r = a \times (b \times c) + (1 - a) \times (b + c)$$

Corresponding equations for the circuit are:

$$r_1 = b \times c, \qquad r_3 = 1 - a, \qquad r_5 = r_3 \times r_2$$
$$r_2 = b + c, \qquad r_4 = a \times r_1, \qquad r = r_4 + r_5$$

Recap. Arbitrary Program To Circuits

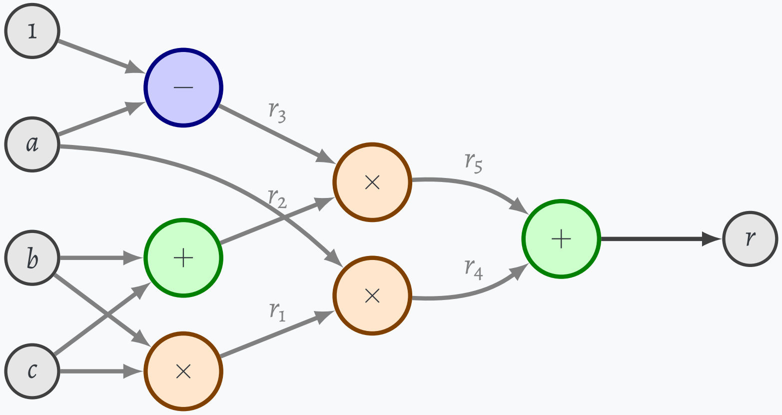

Figure: Example of a circuit evaluating the `if` statement logic.

Recap. Arbitrary Program To Circuits

Example

How can we translate `if` statements?

```
def example(a: bool, b: F, c: F) -> F:
    if a:
        return b * c
    else:
        return b + c
```

Recap. Arbitrary Program To Circuits

Example

How can we translate `if` statements?

```
def example(a: bool, b: F, c: F) -> F:
    if a:
        return b * c
    else:
        return b + c
```

We can transform such a function into the next expression:

$$r = a \times (b \times c) + (1 - a) \times (b + c)$$

Recap. Arbitrary Program To Circuits

Example

How can we translate `if` statements?

```
def example(a: bool, b: F, c: F) -> F:
    if a:
        return b * c
    else:
        return b + c
```

We can transform such a function into the next expression:

$$r = a \times (b \times c) + (1 - a) \times (b + c)$$

Corresponding equations for the circuit are:

$$r_1 = b \times c, \qquad r_3 = 1 - a, \qquad r_5 = r_3 \times r_2$$
$$r_2 = b + c, \qquad r_4 = a \times r_1, \qquad r = r_4 + r_5$$

Recap. R1CS

Each **constraint** in the Rank-1 Constraint System must be in the form:

$$\langle \mathbf{a}, \mathbf{w} \rangle \times \langle \mathbf{b}, \mathbf{w} \rangle = \langle \mathbf{c}, \mathbf{w} \rangle$$

Recap. R1CS

Each **constraint** in the Rank-1 Constraint System must be in the form:

$$\langle \mathbf{a}, \mathbf{w} \rangle \times \langle \mathbf{b}, \mathbf{w} \rangle = \langle \mathbf{c}, \mathbf{w} \rangle$$

Where $\langle \mathbf{u}, \mathbf{v} \rangle$ is a dot product.

$$\langle \mathbf{u}, \mathbf{v} \rangle := \mathbf{u}^\top \mathbf{v} = \sum_{i=1}^{n} u_i v_i$$

Recap. R1CS

Each **constraint** in the Rank-1 Constraint System must be in the form:

$$\langle \mathbf{a}, \mathbf{w} \rangle \times \langle \mathbf{b}, \mathbf{w} \rangle = \langle \mathbf{c}, \mathbf{w} \rangle$$

Where $\langle \mathbf{u}, \mathbf{v} \rangle$ is a dot product.

$$\langle \mathbf{u}, \mathbf{v} \rangle := \mathbf{u}^\top \mathbf{v} = \sum_{i=1}^{n} u_i v_i$$

Thus

$$\left(\sum_{i=1}^{n} a_i w_i \right) \times \left(\sum_{j=1}^{n} b_j w_j \right) = \sum_{k=1}^{n} c_k w_k$$

That is, actually, a quadratic equation with multiple variables.

Recap. R1CS

Example

Consider the most basic circuit with one multiplication gate: $x_1 \times x_2 = r$. The witnes vector $\mathbf{w} = (r, x_1, x_2)$. So

$$w_2 \times w_3 = w_1$$
$$(0 + w_2 + 0) \times (0 + 0 + w_3) = w1 + 0 + 0$$
$$(0w_1 + 1w_2 + 0w_3) \times (0w_1 + 0w_2 + 1w_3) = 1w_1 + 0w_2 + 0w_3$$

Therefore the coefficients vectors are:

$$\mathbf{a} = (0, 1, 0), \quad \mathbf{b} = (0, 0, 1), \quad \mathbf{c} = (1, 0, 0).$$

The general form of our constraint is:

$$(a_1w_1 + a_2w_2 + a_3w_3)(b_1w_1 + b_2w_2 + b_3w_3) = c_1w_1 + c_2w_2 + c_3w_3$$

Recap. R1CS

$$r = x_1 \times (x_2 \times x_3) + (1 - x_1) \times (x_2 + x_3)$$

Recap. R1CS

$$r = x_1 \times (x_2 \times x_3) + (1 - x_1) \times (x_2 + x_3)$$

Thus, the next constraints can be build:

$$x_1 \times x_1 = x_1 \quad \text{(binary check)} \tag{1}$$

$$x_2 \times x_3 = \mathsf{mult} \tag{2}$$

$$x_1 \times \mathsf{mult} = \mathsf{selectMult} \tag{3}$$

$$(1 - x_1) \times (x_2 + x_3) = r - \mathsf{selectMult} \tag{4}$$

Recap. R1CS

$$r = x_1 \times (x_2 \times x_3) + (1 - x_1) \times (x_2 + x_3)$$

Thus, the next constraints can be build:

$$x_1 \times x_1 = x_1 \quad \text{(binary check)} \tag{1}$$

$$x_2 \times x_3 = \mathsf{mult} \tag{2}$$

$$x_1 \times \mathsf{mult} = \mathsf{selectMult} \tag{3}$$

$$(1 - x_1) \times (x_2 + x_3) = r - \mathsf{selectMult} \tag{4}$$

The witness vector: $\mathbf{w} = (1, r, x_1, x_2, x_3, \mathsf{mult}, \mathsf{selectMult})$.

Recap. R1CS

$$r = x_1 \times (x_2 \times x_3) + (1 - x_1) \times (x_2 + x_3)$$

Thus, the next constraints can be build:

$$x_1 \times x_1 = x_1 \quad \text{(binary check)} \tag{1}$$

$$x_2 \times x_3 = \mathsf{mult} \tag{2}$$

$$x_1 \times \mathsf{mult} = \mathsf{selectMult} \tag{3}$$

$$(1 - x_1) \times (x_2 + x_3) = r - \mathsf{selectMult} \tag{4}$$

The witness vector: $\mathbf{w} = (1, r, x_1, x_2, x_3, \mathsf{mult}, \mathsf{selectMult})$.

The coefficients vectors:

$$\begin{aligned}
\mathbf{a}_1 &= (0,0,1,0,0,0,0), & \mathbf{b}_1 &= (0,0,1,0,0,0,0), & \mathbf{c}_1 &= (0,0,1,0,0,0,0) \\
\mathbf{a}_2 &= (0,0,0,1,0,0,0), & \mathbf{b}_2 &= (0,0,0,0,1,0,0), & \mathbf{c}_2 &= (0,0,0,0,0,1,0) \\
\mathbf{a}_3 &= (0,0,1,0,0,0,0), & \mathbf{b}_3 &= (0,0,0,0,0,1,0), & \mathbf{c}_3 &= (0,0,0,0,0,0,1) \\
\mathbf{a}_4 &= (1,0,-1,0,0,0,0), & \mathbf{b}_4 &= (0,0,0,1,1,0,0), & \mathbf{c}_4 &= (0,1,0,0,0,0,-1)
\end{aligned}$$

Quadratic Arithmetic Program

Problems we have for now:

Problems we have for now:

- Although Rank-1 Constraint Systems provide a powerful method for representing computations, they are not succinct.

Problems we have for now:

- Although Rank-1 Constraint Systems provide a powerful method for representing computations, they are not succinct.
- We need to transform our computations into a form that is more convenient for proving statements about them.

We finished with:

$$\mathbf{a_1}, \mathbf{a_2}, \ldots, \mathbf{a_m}, \quad \mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b_m}, \quad \mathbf{c_1}, \mathbf{c_2}, \ldots, \mathbf{c_m},$$

We finished with:

$$\mathbf{a_1}, \mathbf{a_2}, \ldots, \mathbf{a_m}, \quad \mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b_m}, \quad \mathbf{c_1}, \mathbf{c_2}, \ldots, \mathbf{c_m},$$

Of course, they form corresponding matrices:

$$A = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{bmatrix} \quad B = \begin{bmatrix} b_{11} & b_{12} & \ldots & b_{1n} \\ b_{21} & b_{22} & \ldots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \ldots & b_{mn} \end{bmatrix} \quad C = \begin{bmatrix} c_{11} & c_{12} & \ldots & c_{1n} \\ c_{21} & c_{22} & \ldots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \ldots & c_{mn} \end{bmatrix}$$

We finished with:

$$\mathbf{a_1}, \mathbf{a_2}, \ldots, \mathbf{a_m}, \quad \mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b_m}, \quad \mathbf{c_1}, \mathbf{c_2}, \ldots, \mathbf{c_m},$$

Of course, they form corresponding matrices:

$$A = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{bmatrix} \quad B = \begin{bmatrix} b_{11} & b_{12} & \ldots & b_{1n} \\ b_{21} & b_{22} & \ldots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \ldots & b_{mn} \end{bmatrix} \quad C = \begin{bmatrix} c_{11} & c_{12} & \ldots & c_{1n} \\ c_{21} & c_{22} & \ldots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \ldots & c_{mn} \end{bmatrix}$$

An example of a single “if“ statement:

$$\begin{aligned} \mathbf{a}_1 &= (0, 0, 1, 0, 0, 0, 0) \\ \mathbf{a}_2 &= (0, 0, 0, 1, 0, 0, 0) \\ \mathbf{a}_3 &= (0, 0, 1, 0, 0, 0, 0) \\ \mathbf{a}_4 &= (1, 0, -1, 0, 0, 0, 0) \end{aligned} \qquad A = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

We finished with:

$$\mathbf{a_1}, \mathbf{a_2}, \ldots, \mathbf{a_m}, \quad \mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b_m}, \quad \mathbf{c_1}, \mathbf{c_2}, \ldots, \mathbf{c_m},$$

Of course, they form corresponding matrices:

$$A = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{bmatrix} \quad B = \begin{bmatrix} b_{11} & b_{12} & \ldots & b_{1n} \\ b_{21} & b_{22} & \ldots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \ldots & b_{mn} \end{bmatrix} \quad C = \begin{bmatrix} c_{11} & c_{12} & \ldots & c_{1n} \\ c_{21} & c_{22} & \ldots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \ldots & c_{mn} \end{bmatrix}$$

An example of a single “if“ statement:

$$\begin{aligned} \mathbf{a}_1 &= (0, 0, 1, 0, 0, 0, 0) \\ \mathbf{a}_2 &= (0, 0, 0, 1, 0, 0, 0) \\ \mathbf{a}_3 &= (0, 0, 1, 0, 0, 0, 0) \\ \mathbf{a}_4 &= (1, 0, -1, 0, 0, 0, 0) \end{aligned} \qquad A = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Pleeeeeenty of zeroes, doesn’t it? And this is just one out of 3 matrices...

The previous witness vector:

$$\mathbf{w} = (1, r, x_1, x_2, x_3, \text{mult}, \text{selectMult})$$

Let's take a closer look at the matrix columns:

$$\begin{array}{c} \begin{array}{ccccccc} & & 3 & & & & \end{array} \\ \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Consider 4th constraint: $(1 - x_1) \times (x_2 + x_3) = r - \text{selectMult}$

$$\begin{array}{cc} & \begin{array}{ccccccc} & & 3 & & & & \end{array} \\ \begin{array}{c} \\ \\ \\ 4 \end{array} & \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & \boxed{-1} & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

So, every column is a mapping of constraint number to a coefficient for the witness element.

The previous witness vector:

$$\mathbf{w} = (1, r, \overset{3}{\boxed{x_1}}, x_2, x_3, \text{mult}, \text{selectMult})$$

Let's take a closer look at the matrix columns:

$$\begin{array}{c} \begin{matrix} & & 3 & & & & \end{matrix} \\ \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Consider 4th constraint: $(1 - x_1) \times (x_2 + x_3) = r - \text{selectMult}$

$$\begin{array}{cc} & \begin{matrix} & & 3 & & & & \end{matrix} \\ \begin{matrix} \\ \\ \\ 4 \end{matrix} & \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

The previous witness vector:

$$\mathbf{w} = (1, r, \overset{3}{\boxed{x_1}}, x_2, x_3, \text{mult}, \text{selectMult})$$

Let's take a closer look at the matrix columns:

$$\begin{array}{c} \begin{array}{ccccccc} & & 3 & & & & \end{array} \\ \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Consider 4th constraint: $(1 - x_1) \times (x_2 + x_3) = r - \text{selectMult}$

$$\begin{array}{cc} & \begin{array}{ccccccc} & & 3 & & & & \end{array} \\ \begin{array}{c} \\ \\ \\ 4 \end{array} & \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & \boxed{-1} & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

So, every column is a mapping of constraint number to a coefficient for the witness element.

As we know, such a mapping can be builds using Lagrange interpolation polynomial with the following formula:

$$L(x) = \sum_{i=0}^{n} y_i \ell_i(x), \quad \ell_i(x) = \prod_{j=0, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}.$$

As we know, such a mapping can be builds using Lagrange interpolation polynomial with the following formula:

$$L(x) = \sum_{i=0}^{n} y_i \ell_i(x), \quad \ell_i(x) = \prod_{j=0, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}.$$

There are n columns and m constraints. So, it results in n polynomials such that:

$$A_j(i) = a_{i,j},\ i \in \{1, 2, \ldots, m\},\ j \in \{1, 2, \ldots, n\}$$

As we know, such a mapping can be builds using Lagrange interpolation polynomial with the following formula:

$$L(x) = \sum_{i=0}^{n} y_i \ell_i(x), \quad \ell_i(x) = \prod_{j=0, j \neq i}^{n} \frac{x - x_j}{x_i - x_j}.$$

There are n columns and m constraints. So, it results in n polynomials such that:

$$A_j(i) = a_{i,j}, \; i \in \{1, 2, \dots, m\}, \; j \in \{1, 2, \dots, n\}$$

The same is true for matrices B and C, with $3n$ polynomials in total, n for each of the coefficients matrices:

$$A_1(x), A_2(x), \dots, A_n(x), B_1(x), B_2(x), \dots, B_n(x), C_1(x), C_2(x), \dots, C_n(x)$$

As we know, such a mapping can be builds using Lagrange interpolation polynomial with the following formula:

$$L(x) = \sum_{i=0}^{n} y_i \ell_i(x), \quad \ell_i(x) = \prod_{j=0, j \neq i}^{n} \frac{x - x_j}{x_i - x_j}.$$

There are n columns and m constraints. So, it results in n polynomials such that:

$$A_j(i) = a_{i,j}, \;\; i \in \{1, 2, \ldots, m\}, \;\; j \in \{1, 2, \ldots, n\}$$

The same is true for matrices B and C, with $3n$ polynomials in total, n for each of the coefficients matrices:

$$A_1(x), A_2(x), \ldots, A_n(x), B_1(x), B_2(x), \ldots, B_n(x), C_1(x), C_2(x), \ldots, C_n(x)$$

Note

We could have assigned any *unique* index from $\mathbb{F}$ to each constraint (say, t_i for each $i \in \{1, \ldots, m\}$) and interpolate through these points:

$$A_j(t_i) = a_{i,j}, \;\; i \in \{1, 2, \ldots, m\}, \;\; j \in \{1, 2, \ldots, n\}$$

Thanks for your attention!