ブロックチェーン公開講座 第17回
ゼロ知識証明その1（概要・理論）

芝野恭平

東京大学大学院工学系研究科技術経営戦略学専攻
ブロックチェーンイノベーション寄付講座
特任研究員
shibano@tmi.t.u-tokyo.ac.jp

ゼロ知識証明（ZK）講義予定

- 前回：
  - データアベイラビリティ
    - KZGコミットメント

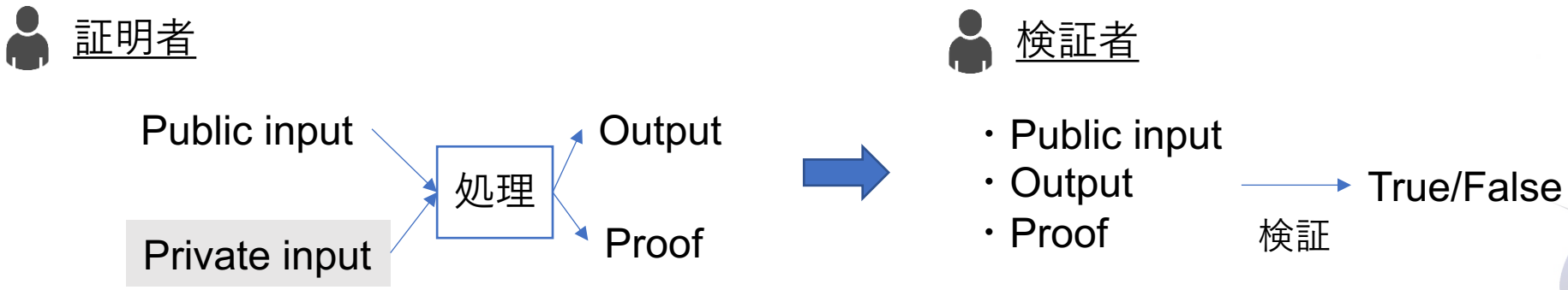
- 本日：
  ZKの概要，何を実現する技術なのか
  ZK理論，PlonKの仕組みを解説
  ※ 講義で取り扱う情報は実際の実装とは異なり，一部不十分な場合があります．

- 来週：
  実習．circomを使って簡単な回路を作ってみる．

- 再来週：
  ZKを使った実プロジェクトの紹介：
  ZCash/ZKEvm/ZKEmail

ゼロ知識証明(zkSNARKs)とは

- 証明者と検証者の2人を想定.
  - ブロックチェーンの場合，多くの場合検証者はコントラクト.
- とある処理を考える.
- 証明者がその処理の出力値に対する入力値を知っていることを，入力値そのものを伝えずに検証者に伝えることができる技術がゼロ知識証明.
  - 入力値は検証者に共有することもできる.
- 検証者は，OutputとProofを持って，その「処理」がされた結果が「Output」であることを「Proof」を用いて検証できる.
- この際，どんなに複雑な処理だったとしても，その検証コストは一定に抑えられる.
  - スマートコントラクトで使用するのに適している.



※「処理」ではなく「問題」と説明されている文献も多い.
ソフトウェア的な関数処理をイメージしやすいように講義では「処理」と記述を統一しています.
関数の処理を問題としてその出力値に対応するインプット値を知っているかどうかを検証する.

ゼロ知識証明(zkSNARKs)とは

- ZKの特長は「入力の秘匿化」と「検証コストの簡略化」の2つ
- 入力の秘匿化の応用事例：パスポートのスキャン画像を用いて20歳以上かどうかを判定

```
function generateAgeProof(photo, name, today){
    const containsName = checkName(photo, name);

    if (containsName) {
        const age = extractAgeFromPhoto(photo, today);
        return age >= 20;
    }
    return false;
}
```

この例では，パスポートのスキャン画像を入力値として，入力した名前がそのスキャン画像内に含まれており，かつ生年月日を読み取り今日の日付に照らし合わせて20歳以上だった場合にTrueが返る．
パブリックインプット：name, today
プライベートインプット：photo
とすることで，個人情報が含まれているパスポート写真そのものは検証者に渡さずに，名前とその人が20歳以上かどうかの判定ができる．
※ 実際には写真を偽造される可能性もあるので一つの単純な例だと思ってください．

ゼロ知識証明(zkSNARKs)とは

- 処理が大きすぎて，オンチェーンで処理しきれない処理を，ZKを用いてオンチェーンで検証可能にすることも応用例の一つ.
- 例：複数のTxを一括で実行.

```
function aggregateTransactions(transactions, beforeState){
    // 署名の検証
    const isValidSigniture = checkSigniture(transactions);

    // beforeStateからtxをすべて実行した結果のstate treeを取得
    const after = processTransactions(transactions, beforeState);

    return after;
}
```

この例では，複数のTxの署名を検証し，その後実行したあとのステートツリーを返す.
この処理のプルーフをオンチェーンで検証するだけで，たくさんのTxを実行することなくステートの更新が安全に行える.

ZKをブロックチェーンで応用することで・・？

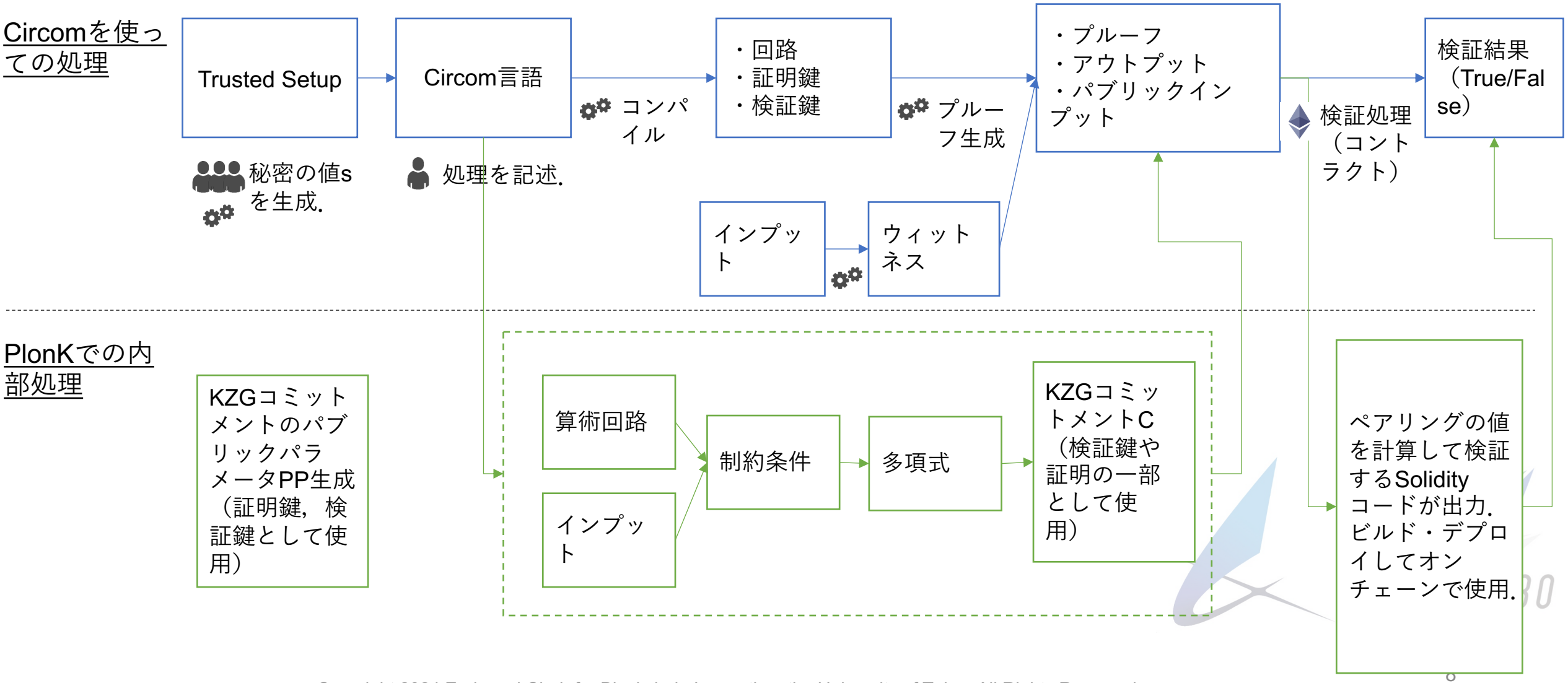
- 情報の秘匿化
  - パスワード認証のコントラクトウォレットを実現できる.
    - ユーザー固有のパスワードと，そのハッシュ値が一致していることでログイン認証.
    - パスワードを隠したままそのパスワードを知っていることを，ハッシュ値だけで確認.
    - Tx内にパスワードを入れてしまうと，Txは公開されてしまうのでそのままでは利用できない.
  - 匿名送金
    - 送金者，送金金額の秘匿化
- 計算の検証コストを削減
  - zkRollup
    - 送金した金額と，残高の変化，そしてその送金Txへの電子署名をいくつもまとめる.
    - 送金金額と残高だけだと処理は少ないが，署名の検証をオンチェーンでしなくてもよくなる.
    - 検証の計算量はO(1)なので，まとめ上げる送金Tx数によらずガスコストは一定に抑えられる.
  - マイナンバーカードを用いたコントラクトウォレット
    - マイナンバーカードで利用されているRSA形式の電子署名の検証をZKで.
    - オンチェーンで直接RSAの検証を行うとガスコストがかかるところをZKで一定まで抑えている.

ZKの使用上の注意

- 「処理」は算術回路（次に説明していきます）で表現する必要がある.
  - プログラミング言語でいつも記述している感覚では一部表現が難しいところがある.
    - すべての処理を足し算と掛け算で表現.
    - 動的なループなどは効率的にできない.
      - 過剰なループ処理をして，不要なループ内ではなにもしない，など.
    - ビットシフトなど一般的な処理も，足し算と掛け算で表現し直す必要がある.
      - そのため，SHA256の計算はものすごく時間がかかる.

- ZKのプルーフ生成は思ったより時間がかかる.
  - オンチェーンで直接計算するよりは計算できる能力があがる，とはいえ，やはり通常のPCで計算を行っている感覚でやると扱えるデータサイズや計算の複雑さは限定的.
  - しかしながら，オンチェーンとは違い処理を実行する（プルーフ生成をする）PCのスペックを強化することで計算は可能.
    - プルーフ生成のPCのスペックを増強するのみでOK.
    - Proofの検証はオンチェーンで$O(1)$で可能なので，いかにプルーフを生成するかが問題になる.

ZKをコントラクトで使用する流れとPlonK内部処理



ゼロ知識証明のプロトコルの一例

- Groth16 (2016)
  - zkSNARKsの一種，回路ごとのTrusted Setupが必要，小さな証明サイズ，高速な検証時間.
  - circom
- PlonK (2019)
  - zkSNARKsの一種，ユニバーサルTrusted Setup.
  - circom
- Halo，Halo2 (2019-)
  - リカージョン（回路の中でプルーフ検証が可能）が得意.
  - Rust

本講義では，多くのプロトコルがある中，昨今の仕組みでの基礎となっているPlonKをもとにゼロ知識証明がどうやって実現しているのかを解説します.

Circom

- https://docs.circom.io/

- 独自のプログラミング言語で処理内容を記述できる.
- snarkjsというJSのライブラリを用いて，コンパイル後のファイルを取り扱うことができ，それを用いてZKのプルーフ生成や検証が可能.
- 検証用のコントラクトのSolidityコードも自動生成してくれる.

```
pragma circom 2.0.0;

include
"../node_modules/circomlib/circuits/eddsamimc.circom";

template Hash ()   {

   signal input in;
   signal output out;

   component hasher;
   hasher = MiMC7(91);
   hasher.x_in <== in;
   hasher.k <== 0;

   out <== hasher.out;

}

component main = Hash();
```

Circomでハッシュ値を計算する例

PlonK

- Circomで使用可能なゼロ知識証明のプロトコル.
- ユニバーサルトラステッドセットアップを実現しているのが特長.
  - 一つのトラステッドセットアップのPPは様々な回路で使用可能.
  - 回路を表現している制約の数はPPの制約の数以下である必要がある.
    - つまり，複雑な回路を実行したい場合は，それだけ大きな制約数に対応したPPが必要.

- Gabizon, Ariel, Zachary J. Williamson, and Oana Ciobotaru. "Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge." Cryptology ePrint Archive (2019).
  - https://eprint.iacr.org/2019/953

PlonKの処理の流れ

- KZGコミットメントを適用させるためにいくつかの変換を施す.

算術回路 → 制約条件 → 多項式 → KZGコミットメント

入出力値とゲートからなる遷移図

多変数x_iから構成される連立方程式.
証明者はその制約を満たす解を知っている.
そのため，ここで作られている制約式はすべて具体的な数字のセットとなっている.

一変数xから構成される多項式$f(x)$を複数作る.

制約条件で表現されている各パラメータ値セット（数字のセット）をもとにユニークな多項式を構築する.
すなわち，多項式は制約と入出力値から作られる.

それにより，その多項式そのものを知っていることは元の問題の解も知っていることにつながる.

複数の多項式に対して，複数の特定の評価点αで$f(\alpha) = \beta$であることを検証者に示す.

評価点について評価するために，フィアットシャミアにより得られる乱数を使う.
乱数で評価した点で解があう，ということで多項式そのものを知っていることの代替としてる.

→ 評価者が多項式について一切しらないと適当な値が返ってきてもなんとも言えないが，制約は事前共有されており，同じ多項式を復元できる.

制約 → 検証者に事前に1回共有.

パブリック値（パブリックインプットとアウトプット） → 検証者に証明ごとに共有.

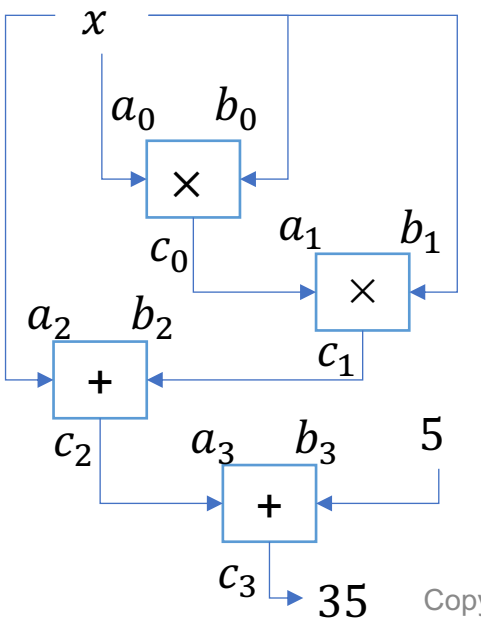
算術回路で表現された処理を，制約条件，多項式と変換している.
最終的にKZGコミットメントを使用するために，「処理」を表現可能な形式である算術回路をまず構築する.

<u>すべての処理はKZGコミットメントを適用するために行われる.</u>

算術回路について

- PlonKの起点は算術回路.
- 算術回路は，2入力1出力のゲートの組み合わせで表現される．ゲートは加算「+」もしくは乗算「×」で表される
- 各ゲートiの左入力をa_i, 右入力をb_i，出力をc_iとする．取り扱う値はすべて有限体$F_r = 0, \ldots, r-1$上の点．入出力値をワイヤーとも呼ぶ.
- 証明したい処理をまずは算術回路で表現するところから始まる.

算術回路の例： $x^3 + x + 5 = 35$.
これは左辺の式で表される処理を行って35になる入力値を知っているかを示す例



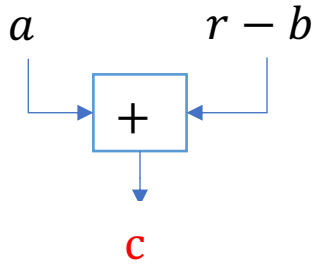
0番ゲート：左も右も入力がxで，出力はx^2
1番ゲート：左入力はx^2，右入力はx，出力はx^3
2番ゲート：左入力はx，右入力はx^3，出力は$x^3 + x$
3番ゲート：左入力は$x^3 + x$，右入力は5，出力は35

https://scryptplatform.medium.com/how-plonk-works-part-1-bc8050f4805e

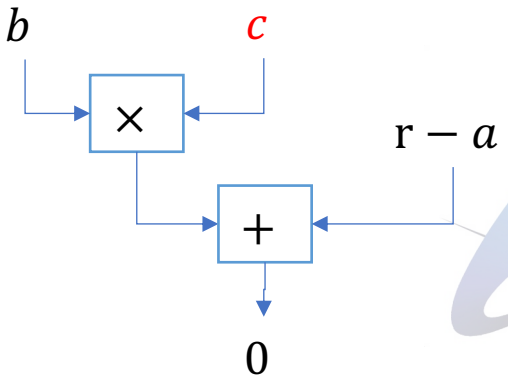
算術回路の例

- ゲートでは足し算＋と掛け算×しか定義されていないが，以下のように設定することで減算と除算も表現可能.
- つまり，算術回路では四則演算すべてが実行可能.
  - 注意として，入力値は事前に計算をして適切な値を入れる必要がある.
  - つまり，全体の処理を回路で表現していく中で，減算ゲート，除算ゲートのように取り扱うことはできない.

減算：有限体$F_r = \{0,1,\dots,r-1\}$上では，$b + (r-b) = r = 0 \bmod r$が成り立つため，$-b$は$r-b$である.
つまり，$c = a - b$はaと$r-b$の加算として計算できる．bの代わりに$r-b$を使用する.



除算：$c = a/b$を計算する代わりにその出力 cを新たな入力として与え $bc - a$を計算する．検証者は $bc - a = 0$であることを確かめることで，$bc = a$すなわち $c = a/b$であることを確かめられる.

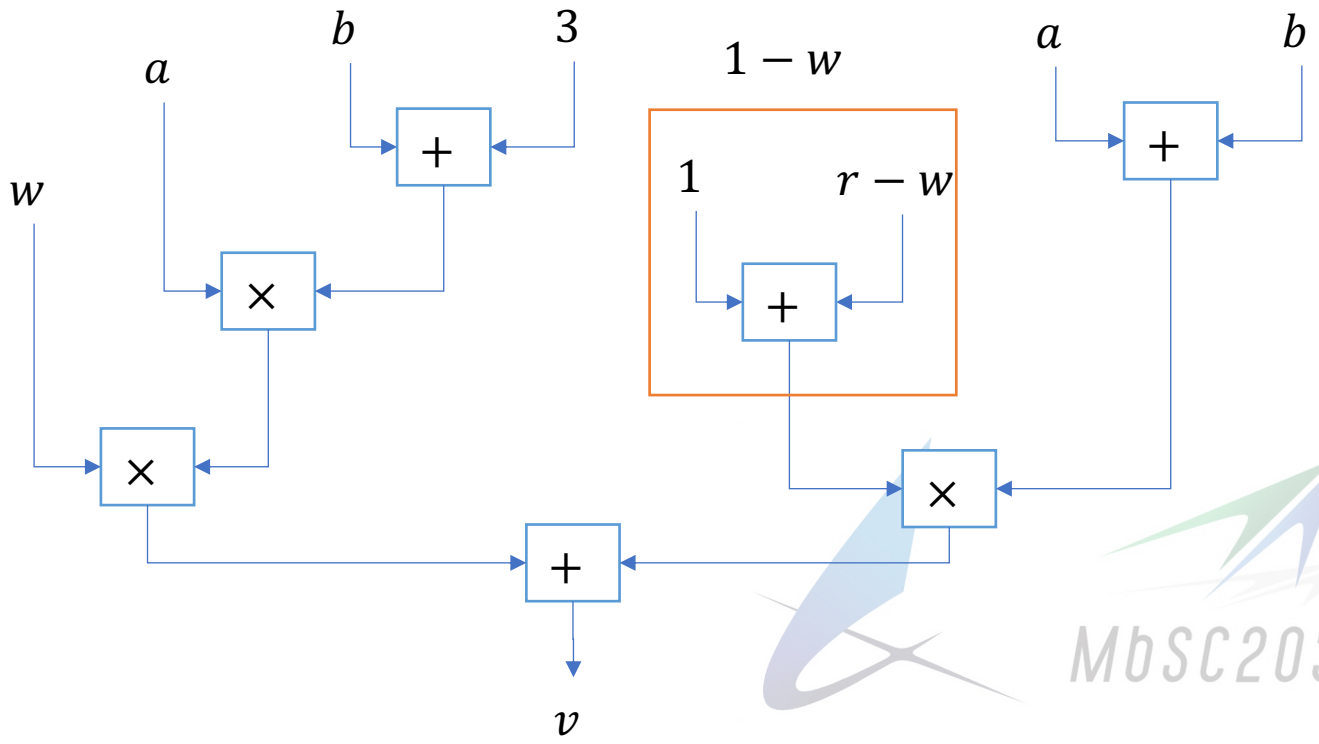


算術回路の例

- 以下のようにプログラミング言語で表現される関数を算術回路に変換してみましょう.

```
function myfunc(w, a, b){
    if (w == 1)
        return a * (b + 3);
     else
        return a + b;
}
```

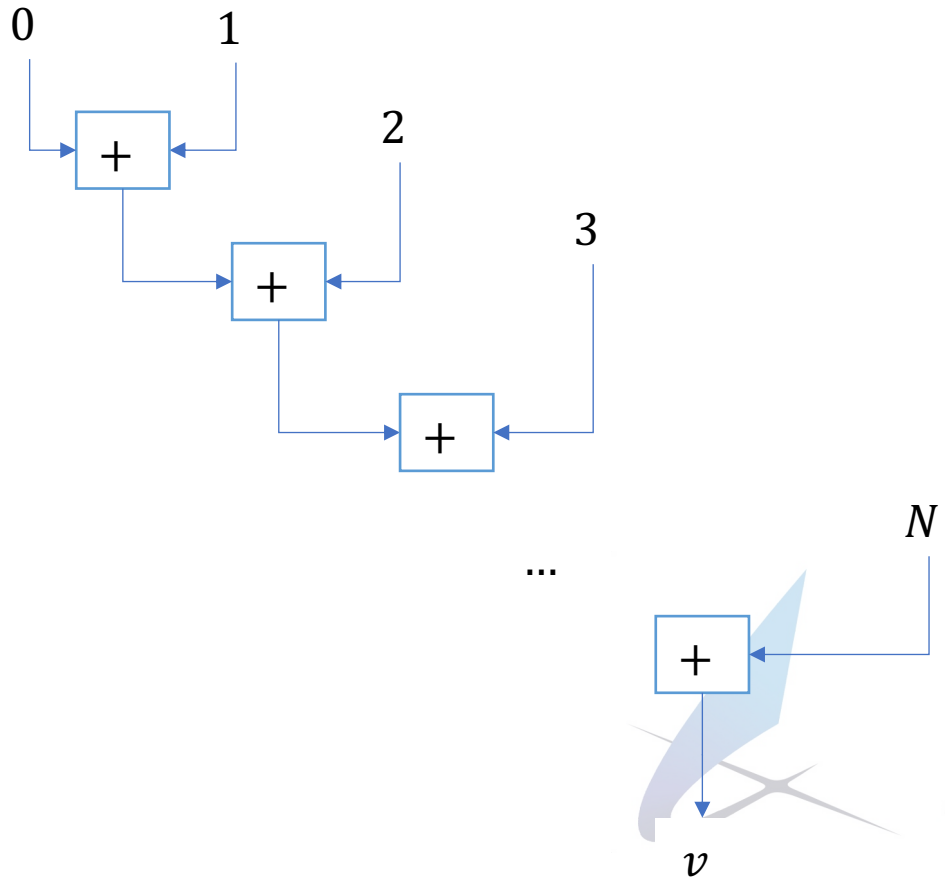
出力はvとする



算術回路の例

- 以下のようにプログラミング言語で表現される関数を算術回路に変換してみましょう.

```
N = 10
function sumUpTo(){
    let sum = 0;
    for (let i = 1; i <= N; i++){
        sum += i;
    }
    return sum;
}
```



for文はN個分の制約を繰り返すことで表現できる.
ここで，算術回路は固定で作成する必要があるため関数の引数にNを入れて動的に変化させることはできない.

出力はvとする

算術回路から制約条件へ

- 次に，算術回路で表現された処理を，制約条件へ変換する．
- 制約条件は連立方程式．

- ゲート制約
- コピー制約

- という2つの制約をもって，算術回路を制約条件に変換する．

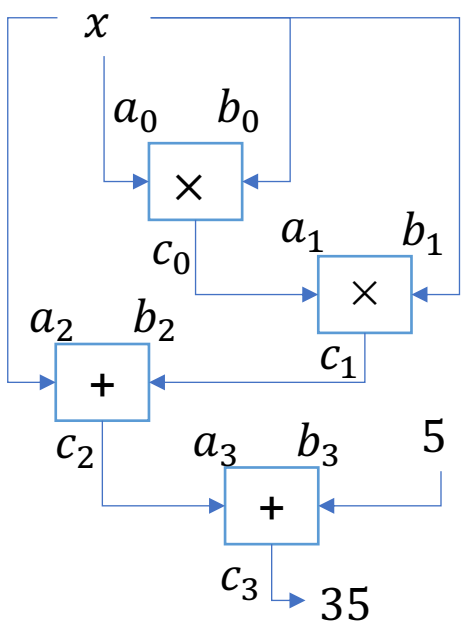
- 連立方程式の形で構成されているものの，証明者は各値を知っている．
- すなわち，方程式のパラメータや解はすべて知っている状態．

算術回路から制約条件へ：ゲート制約

- 加算・乗算ゲートの左入力、右入力、出力の値をそれぞれ x_a, x_b, x_c とすると，次の制約条件を満たす.
- 今は，ゲートでの計算のみを表現することを考え，具体的に入力値・出力値にどんな値が入るかは考えない

  - 加算: $x_a + x_b - x_c = 0$
  - 乗算: $x_a\, x_b - x_c = 0$

$x^3 + x + 5 = 35$
の例の場合



$$x_{a_0}\, x_{b_0} - x_{c_0} = 0$$
$$x_{a_1}\, x_{b_1} - x_{c_1} = 0$$
$$x_{a_2} + x_{b_2} - x_{c_2} = 0$$
$$x_{a_3} + x_{b_3} - x_{c_3} = 0$$

※ 各入力値を入れると・・・

$$x\, x - x_{c_0} = 0 \quad (i = 0)$$
$$x_{a_1}\, x - x_{c_1} = 0 \quad (i = 1)$$
$$x + x_{b_2} - x_{c_2} = 0 \quad (i = 2)$$
$$x_{a_3} + 5 - 35 = 0 \quad (i = 3)$$

算術回路から制約条件へ：ゲート制約

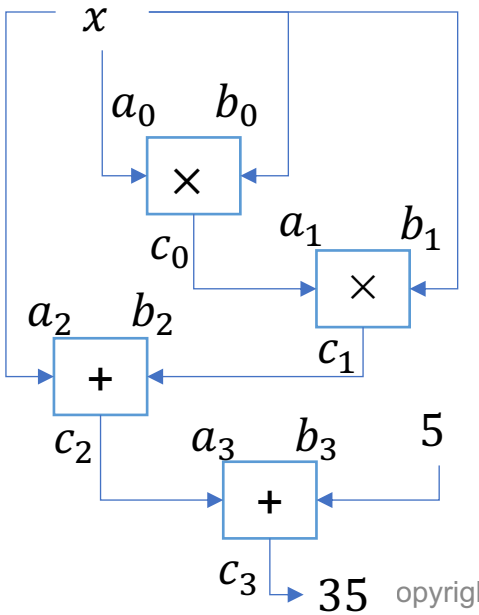
- 加算でも乗算でも同じ表現での方程式で表されるようにする.
- ゲートごとに決まるパラメータ $(\mathrm{q}_L, \mathrm{q}_R, \mathrm{q}_O, \mathrm{q}_M, \mathrm{q}_C) \in (\mathrm{F}_r^n)^5$ を導入して、式の形式を次のように統一.

$$q_{L_i}\, x_{a_i} + q_{R_i}\, x_{b_i} + q_{O_i}\, x_{c_i} + q_{M_i} x_{a_i} x_{b_i} + q_{C_i} = 0 \text{ for all } i$$

加算，乗算，定数ゲートの場合，パラメータの値をそれぞれ
$(\mathrm{q}_{L_i}, \mathrm{q}_{R_i}, \mathrm{q}_{O_i}, \mathrm{q}_{M_i}, \mathrm{q}_{C_i}) = (1,1,-1,0,0), (0,0,-1,1,0), (0,0,-1,0,q_{C_i})$ とする.

※ ここで，nはゲート数を含んだ総制約数．必ずしもゲート数とは一致しないことに注意.
後ほど出てくるパラメータ値の制約も含む.

$x^3 + x + 5 = 35$の例



$$x_{a_0}\, x_{b_0} - x_{c_0} = 0$$
$$x_{a_1}\, x_{b_1} - x_{c_1} = 0$$
$$x_{a_2} + x_{b_2} - x_{c_2} = 0$$
$$x_{a_3} + 5 - 35 = 0$$

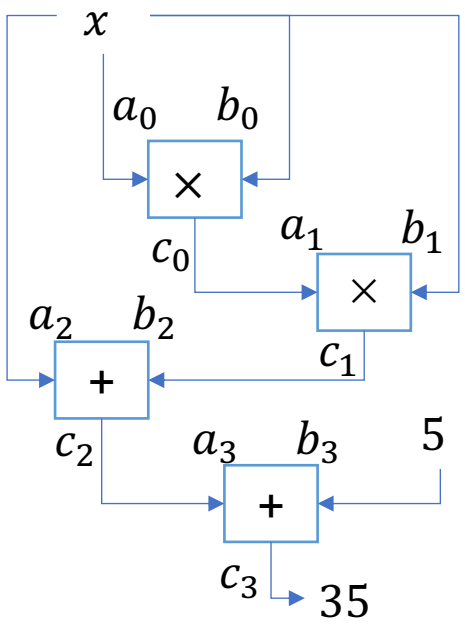
$$0 * x_{a_0} + 0 * x_{b_0} + (-1) * x_{c_0} + 1 * x_{a_0} x_{b_0} + 0 = 0$$
$$0 * x_{a_1} + 0 * x_{b_1} + (-1) * x_{c_1} + 1 * x_{a_1} x_{b_1} + 0 = 0$$
$$1 * x_{a_2} + 1 * x_{b_2} + (-1) * x_{c_2} + 0 * x_{a_2} x_{b_2} + 0 = 0$$
$$1 * x_{a_3} + 1 * x_{b_3} + (-1) * x_{c_3} + 0 * x_{a_3} x_{b_3} + 0 = 0$$
$$0 * x_{a_3} + 1 * x_{b_3} + 0 * x_{c_3} + 0 * x_{a_3} x_{b_3} - 5 = 0$$
$$0 * x_{a_3} + 0 * x_{b_3} + (-1) * x_{c_3} + 0 * x_{a_3} x_{b_3} + 35 = 0$$

$(\mathrm{q}_L, \mathrm{q}_R, \mathrm{q}_O, \mathrm{q}_M, \mathrm{q}_C) = ($
$(0,0,-1,1,0),$
$(0,0,-1,1,0),$
$(1,1,-1,0,0),$
$(1,1,-1,0,0),$
$(0,1,0,0,-5),$
$(0,0,-1,0,35))$

算術回路から制約条件へ：コピー制約

- これは，とあるゲートの出力が別のゲートの入力と等しいことを表す制約.

$x^3 + x + 5 = 35$
の例の場合



$$x_{c_0} = x_{a_1}$$
$$x_{c_1} = x_{b_2}$$
$$x_{c_2} = x_{a_3}$$

算術回路から制約条件へ：コピー制約

- 効率的に表現することを考える.
- 前のページの表現方法だと，ワイヤーの数に比例して式の数が増えてしまうため好ましくない.
  - $n * n$行列
- 1式でまとめて累積値を用いて表現する.

β, γは証明者が予測できないような乱数（フィアットシャミア，ハッシュ値）.
便宜的に $0 \leq i < n, n \leq i < 2n, 2n \leq i < 3n$ の場合，x_iはそれぞれ $x_{a_i}, x_{b_i}, x_{c_i}$ を表すとし， $\sigma(i)$はi番目のワイヤの値がコピーされるワイヤを表すとする.
次のF, Gを計算し，$F = G$となればよい.

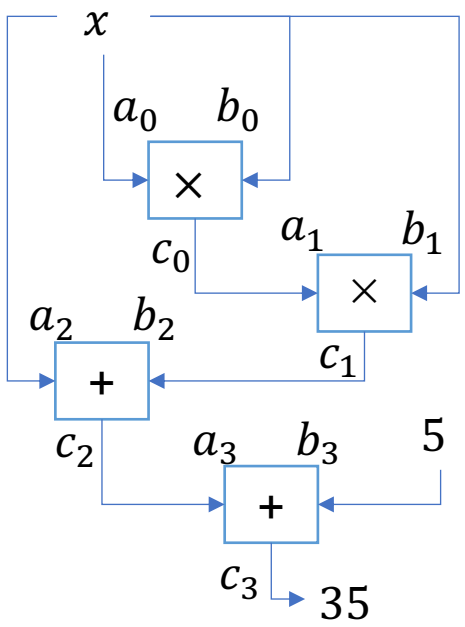
$$F = \prod_{i=0}^{3n-1} (x_i + \beta \cdot i + \gamma)$$

$$G = \prod_{i=0}^{3n-1} (x_i + \beta \cdot \sigma(i) + \gamma)$$

$x_i = x_{\sigma(i)}$ が全ての iで成り立つならば累積値の値は x_iと $x_{\sigma(i)}$を入れ替えても同じように計算されるため $F = G$ が成り立つ.
逆に $F = G$ならば，高い確率で $x_i = x_{\sigma(i)}$ が全ての i で成り立つ．直感的には $j = \sigma(i)$ ならば $i = \sigma(j)$であるかつ、β, γのランダム性により F, G の因数で β の係数が同じもの同士がそれぞれ等しくなる必要があるので、 $x_i + \beta \cdot i + \gamma = x_{\sigma(i)} + \beta \cdot i + \gamma$ が高い確率で成り立つため.

算術回路から制約条件へ：コピー制約

$x^3 + x + 5 = 35$の例



i	0	1	2	3	4	5	6	7	8	9	10	11
	a_0	a_1	a_2	a_3	b_0	b_1	b_2	b_3	c_0	c_1	c_2	c_3
	a_0	c_0	a_2	c_2	b_0	b_1	c_1	b_3	a_1	b_2	a_3	c_3
$\sigma(i)$	0	8	2	10	4	5	9	7	1	6	3	11

$$F = \prod_{i=0}^{3n-1}(x_i + \beta \cdot i + \gamma)$$

$$= (x_{a_0} + \beta \cdot 0 + \gamma)(x_{a_1} + \beta \cdot 1 + \gamma)(x_{a_2} + \beta \cdot 2 + \gamma)(x_{a_3} + \beta \cdot 3 + \gamma)(x_{b_0} + \beta \cdot 4 + \gamma)(x_{b_1} + \beta \cdot 5 + \gamma)(x_{b_2} + \beta \cdot 6 + \gamma)(x_{b_3} + \beta \cdot 7 + \gamma)(x_{c_0} + \beta \cdot 8 + \gamma)(x_{c_1} + \beta \cdot 9 + \gamma)(x_{c_2} + \beta \cdot 10 + \gamma)(x_{c_3} + \beta \cdot 11 + \gamma)$$

$$G = \prod_{i=0}^{3n-1}(x_i + \beta \cdot \sigma(i) + \gamma)$$

$$= (x_{a_0} + \beta \cdot 0 + \gamma)(x_{a_1} + \beta \cdot 8 + \gamma)(x_{a_2} + \beta \cdot 2 + \gamma)(x_{a_3} + \beta \cdot 10 + \gamma)(x_{b_0} + \beta \cdot 4 + \gamma)(x_{b_1} + \beta \cdot 5 + \gamma)(x_{b_2} + \beta \cdot 9 + \gamma)(x_{b_3} + \beta \cdot 7 + \gamma)(x_{c_0} + \beta \cdot 1 + \gamma)(x_{c_1} + \beta \cdot 6 + \gamma)(x_{c_2} + \beta \cdot 3 + \gamma)(x_{c_3} + \beta \cdot 11 + \gamma)$$

$$\frac{F}{G} = \frac{(x_{a_1} + \beta \cdot 1 + \gamma)(x_{a_3} + \beta \cdot 3 + \gamma)(x_{b_2} + \beta \cdot 6 + \gamma)(x_{c_0} + \beta \cdot 8 + \gamma)(x_{c_1} + \beta \cdot 9 + \gamma)(x_{c_2} + \beta \cdot 10 + \gamma)}{(x_{a_1} + \beta \cdot 8 + \gamma)(x_{a_3} + \beta \cdot 10 + \gamma)(x_{b_2} + \beta \cdot 9 + \gamma)(x_{c_0} + \beta \cdot 1 + \gamma)(x_{c_1} + \beta \cdot 6 + \gamma)(x_{c_2} + \beta \cdot 3 + \gamma)}$$

$$= 1$$

算術回路から制約条件へ：パブリックインプットと検証者への共有情報

- アウトプットとパブリックインプットは，パブリック値として証明生成時に検証者にそのまま共有される．
- パブリック値はp個の値$(x_1, \dots, x_p)$で構成されるとする．
- その時，ゲート制約とコピー制約として以下を追加：
  - ※ もともとあったゲート・コピー制約に加えて制約式が増える．
  - $i \in \{1, \dots, p\}, (\mathrm{q}_{L_i}, \mathrm{q}_{R_i}, \mathrm{q}_{O_i}, \mathrm{q}_{M_i}, \mathrm{q}_{C_i}) = (1, 0, 0, 0, 0)$

$$1 * x_{a_i} + 0 * x_{b_i} + 0 * x_{c_i} + 0 * x_{a_i} x_{b_i} + 0 = 0$$

  - もし，iがx_{a_i}の値であればこのままでよく，x_{b_j}やx_{c_j}の値であれば，$x_{a_i} = x_{b_j}$や$x_{a_i} = x_{c_j}$というコピー制約をつければ良い．
    - ※ 直接Q_{R_i}やQ_{O_i}に設定してもいいはず．

- ちなみに，ゲート制約$(\mathrm{q}_L, \mathrm{q}_R, \mathrm{q}_O, \mathrm{q}_M, \mathrm{q}_C) \in (\mathrm{F}_r^n)^5$や，コピー制約$\sigma(i)$ $(\forall\, i \leq n)$ は回路生成時の1回，検証者に事前に共有される．
- パブリック値は証明生成次に毎回一緒に共有される．

制約条件から多項式へ

- 算術回路から制約条件ができた.
- 次に，これをKZGコミットメントを利用するための多項式へ変換する.
- ゲート制約，コピー制約それぞれで多項式を求める.
  - 多項式は，それぞれの制約から一意に決まる形で求められる.
  - 「多項式がとある点でとある評価値である」 ⇒ 「それぞれの制約を満たす」という多項式を作る.
  - 多項式を知っていることをKZGコミットメントを使って証明することで，制約を満たしていることを示していることになる.（十分に無視できる確率を除いて）

- ゲート制約は，ゲートの制約式$\mathrm{q}_{L_i}, \mathrm{q}_{R_i}, \mathrm{q}_{O_i}, \mathrm{q}_{M_i}, \mathrm{q}_{C_i}$（$i$は各ゲート）で5つの多項式
  - 多項式補間を使う.
    - n個の点を通る$n-1$次の多項式を求められる.
  - さらに，ゲートの入出力3つ$x_{a_i}, x_{b_i}, x_{c_i}$で3つの多項式
  - パブリック値からも多項式PI_iを生成
  - 注意：ここでの各値は何かしらの具体的な値．証明者はその値を知っている.

- コピー制約は，$F = G$を表現するための多項式1つ

制約条件から多項式へ：ゲート制約

- 多項式補間：
  - n個の点を通る$n-1$次の多項式を求められる.
  - 各値をx, y座標上の点に変換する．その値をy座標に取り，x座標は$g^n = 1$を満たす1の累乗根とする.
  - 例えば$(x_{a_i})_{i\in\{0,\dots,n-1\}}$の場合は$(g^0, x_{a_0}), \dots, (g^{n-1}, x_{a_{n-1}})$ を通る多項式を求める.
  - KZGコミットメントで使用する多項式を知っている，ということで制約条件を満たす入出力を知っている，ということを表現.
  - つまり，制約条件からユニークな多項式を生成することが必要で，各値をy座標においた点で表現すれば十分.
  - 原理的には，x座標は何でも良いが累乗根で表現すると計算の高速化ができる.

制約条件から多項式へ：ゲート制約

- パラメータ $(q_L, q_R, q_O, q_M, q_C) \in (F_r^n)^5$ を多項式補間して、 $n-1$次多項式 $L(x), R(x), O(x), M(x), C(x)$を求める。
  - 全ての $i \in \{0, \dots, n-1\}$ でそれぞれ $L(g^i) = q_{L_i}, R(g^i) = q_{R_i}, O(g^i) = q_{o_i}, M(g^i) = q_{M_i}, C(g^i) = q_{C_i}$ が成立する.
- ワイヤの値$x_a = \left(x_{a_i}\right)_{i\in\{0,\dots,n-1\}}$, $x_b = \left(x_{b_i}\right)_{i\in\{0,\dots,n-1\}}$, $x_c = \left(x_{c_i}\right)_{i\in\{0,\dots,n-1\}}$を多項式補間して $n-1$次多項式 $a(x), b(x), c(x)$を求める。
  - 全ての $i \in \{0, \dots, n-1\}$ でそれぞれ $a(g^i) = x_{a_i}, b(g^i) = x_{b_i}, c(g^i) = x_{c_i}$ が成立する.
- パブリック値も同様に多項式補間し，$p-1$次の多項式$PI(x)$を求める.
  - 全ての $i \in \{1, \dots, p\}$ でそれぞれ $PI(g^i) = -x_i$
- ここまでで，n-1次の多項式は9つできた.
  - $L(x), R(x), O(x), M(x), C(x), a(x), b(x), c(x), PI(x)$
- 各$x = g^i$で多項式の等式

$$L(x)a(x) + R(x)b(x) + O(x)c(x) + M(x)a(x)b(x) + C(x) + PI(x) = 0$$

- が成り立つことをKZGコミットメントで証明する.

制約条件から多項式へ：コピー制約

- コピー制約を多項式に変換する際は，$i \geq n$についても1の累乗根で表せるようにするために，定数 $k_1, k_2 \in F_q$ を用いて $n \leq i < 2n$を $k_1 \mathrm{g}^i$，$2n \leq i < 3n$を $k_2 \mathrm{g}^i$ で表す．
- なお，集合 $(\mathrm{g}^0, \ldots, \mathrm{g}^{n-1}), (k_1\,\mathrm{g}^0, \ldots, k_1\,\mathrm{g}^{n-1}), (k_2\,\mathrm{g}^0, \ldots, k_2\,\mathrm{g}^{n-1})$に対して，$i \in \{0, \ldots, 3n-1\}$ は各要素に1対1で対応している．そして，累積値に対応する多項式$z(x)$を次のように定義する．
- ただしラグランジュ基底 $L_i(x) = \prod_{j \in \{0,\ldots,n-1, i \neq j\}} \frac{x - g^j}{g^i - g^j}$ であり $L_i(\mathrm{g}^i) = 1$，$j \neq i$ で $L_i(\mathrm{g}^j) = 0$ であるという特徴を持つ．

$$z(x) = L_1(x) + \sum_{i \in [1, n-1]} L_{i+1}(x) \prod_{j \in [1,i]} \frac{(x_j + \beta\,\mathrm{g}^j + \gamma)(x_{n+j} + \beta\,k_1\,\mathrm{g}^j + \gamma)(x_{2n+j} + \beta\,k_2\,\mathrm{g}^j + \gamma)}{(x_j + \beta\,\sigma(j) + \gamma)(x_{n+j} + \beta\,\sigma(n+j) + \gamma)(x_{2n+j} + \beta\,\sigma(2n+j) + \gamma))}$$

- このとき$z(x)$は次の関係を満たす漸化式になっている．

$$z(1) = 1$$

$$z(\mathrm{g}^{i+1}) = z(\mathrm{g}^i) \frac{(x_j + \beta\,\mathrm{g}^j + \gamma)(x_{n+j} + \beta\,k_1\,\mathrm{g}^j + \gamma)(x_{2n+j} + \beta\,k_2\,\mathrm{g}^j + \gamma)}{(x_j + \beta\,\sigma(j) + \gamma)(x_{n+j} + \beta\,\sigma(n+j) + \gamma)(x_{2n+j} + \beta\,\sigma(2n+j) + \gamma))}$$

制約条件から多項式へ：コピー制約

- 最後の式を変形すると

$$z(g^{i+1})(a(g^i)+\beta\,\sigma(i)+\gamma)(b(g^i)+\beta\,\sigma(n+i)+\gamma)(c(g^i)+\beta\,\sigma(2n+i)+\gamma)$$
$$= z(g^i)(a(g^i)+\beta\,g^i+\gamma)(b(g^i)+\beta\,k_1\,g^i+\gamma)(c(g^i)+\beta\,k_2\,g^i+\gamma) \quad \cdots(*)$$

- が成り立つ．$g^n = 1$ であるため，その総積をとると

$$\prod_{i=0}^{n-1}(a(g^i)+\beta\,\sigma(i)+\gamma)(b(g^i)+\beta\,\sigma(n+i)+\gamma)(c(g^i)+\beta\,\sigma(2n+i)+\gamma)$$
$$= \prod_{i=0}^{n-1}(a(g^i)+\beta\,g^i+\gamma)(b(g^i)+\beta\,k_1\,g^i+\gamma)(c(g^i)+\beta\,k_2\,g^i+\gamma)$$

- が成り立つが，これは制約における累積値の等式 $F = G$ に対応していることがわかる．
- つまり(*)が成立するとき，$F = G$ が成り立ち，コピー制約が満たされていることがわかる．
- そのため $z(x)$ をコピー制約に対する多項式として採用する．

求められた多項式とKZGコミットメント適用時の問題点

- ここまでで，ゲート制約，コピー制約は制約を表現しているユニークな多項式に変換できた．
- $\forall x \in \{g^0, \ldots, g^{n-1}\}$,
  - $L(x) * a(x) + R(x) * b(x) + O(x) * c(x) + M(x) * a(x) * b(x) + C(x) + PI(x) = 0$
  - $z(gx) * (a(x) + \beta\, \sigma(i) + \gamma) * (b(x) + \beta\, \sigma(n+i) + \gamma) * (c(x) + \beta\, \sigma(2n+i) + \gamma)$
    $- z(x) * (a(x) + \beta\, x + \gamma) * (b(x) + \beta\, k_1\, x + \gamma) * (c(x) + \beta\, k_2\, x + \gamma) = 0$

が成り立つことを示せばよい．

→ KZGコミットメントを適用できれば，多項式そのものは検証者に渡さず上記n個$(g^0, \ldots, g^{n-1})$の評価点での評価値が0になることが示せる．

一部検証者が持っている情報もあり，それが含まれることを検証者が確認しつつ上記多項式を構築していく必要がある．つまり，証明者と検証者が一緒にこれらの多項式を構築する．

この条件下で，これらの多項式の関係にKZGコミットメントを適用しようとすると，次の3つの問題がある．

1. 複数の評価点 $\forall x \in \{g^0, \ldots, g^{n-1}\}$で評価結果が0になることを証明しなければならない．さらに証明のサイズはnによらず一定にしたい．
2. 検証者が指定する多項式 (例: $L(x), \sigma(i)$)と証明者が提供する多項式 $a(x), b(x), c(x)$の積を扱う必要がある．しかしKZGコミットメントはどちらもG_1上の要素になるため，2つのKZGコミットメントを直接かけることは不可能である．
3. 共通の多項式 $a(x), b(x), c(x)$がそれぞれの式で使われていることを証明する必要がある．しかし、これらの多項式は証明者の秘密情報 (例: 算術回路への入力)を含むため直接検証者に公開することはできない．

複数の評価点でのKZGコミットメント

- 一つ目の問題は，複数の評価点での評価結果に対して証明が行えるようにKZGコミットメントを変形することで解決.
- KZGコミットメントの最も基本的な形は，ある多項式 $f(x)$のコミットメントcについて $f(\alpha) = \beta$であることを証明するためには，商の多項式

$$q(x) = \frac{f(x) - f(\alpha)}{x - \alpha}$$

- を計算し，それをG_1上でランダムな点（トラステッドセットアップで得られた点）で評価した結果である $\pi \leftarrow \sum_{i=0}^{n} q_i(s^i P)$を証明として提出する.
- そして，検証者は $e(c - \beta P, Q) = e(\pi, sQ - \alpha Q)$が成り立つ場合に検証に成功する.

- ここで $\forall\, \alpha' \in \{\alpha_1, \ldots, \alpha_n\}$で $f(\alpha') = 0$のとき，因数定理より次の式を満たす多項式 $q(x)$が存在する．すなわち：

$$f(x) = q(x)(x - \alpha_1)(x - \alpha_2) \cdots (x - \alpha_n)$$

- いま，$\{\alpha_1, \ldots, \alpha_n\} = \{g^0, \ldots, g^{n-1}\}$のときを考える．累乗根であるので：

$$(x - \alpha_1)(x - \alpha_2) \cdots (x - \alpha_n) = (x - g^0)(x - g^1) \cdots (x - g^{n-1}) = x^n - 1$$

- が成り立つ．すなわち$\forall\, x \in \{g^0, \ldots, g^{n-1}\}$ で$f(x) = 0$が成り立つことと以下の式 (**)は同値.

$$f(x) = q(x)(x^n - 1) \quad \therefore q(x) = \frac{f(x)}{x^n - 1} \cdots (**)$$

- 証明者は(**)に対する証明 $\pi \leftarrow \sum_{i=0}^{n} q_i(s^i P)$を提出し，検証者は

$$e(c, Q) = e(\pi, s^n Q - Q)$$

- が成り立つかを確かめることで，$\forall\, x \in \{g^0, \ldots, g^{n-1}\}$ で $f(x) = 0$ であることを証明・検証できる.

n回KZGコミットメントの評価をしなくても１回で済む.

複数の評価点でのKZGコミットメント

$$e(c, Q) = e(\pi, s^n Q - Q)$$

について

$$\begin{aligned} e(c, Q) &= e(\Sigma_{i=0}^{n} f_i s^i P, Q) && (\because\ c = \Sigma_{i=0}^{n} f_i s^i P) \\ &= e(f(s)P, Q) && (\because\ f(s) = \Sigma_{i=0}^{n} f_i s^i) \\ &= e(P, Q)^{f(s)} && (\because\ e(aP, bQ) = e(P, Q)^{ab}) \end{aligned}$$

$$\begin{aligned} e(\pi, s^n Q - Q) &= e(\Sigma_{i=0}^{n} q_i s^i P, (s^n - 1)Q) && (\because \pi = \Sigma_{i=0}^{n} q_i s^i P) \\ &= e(q(s)P, (s^n - 1)Q) && (\because\ q(s) = \Sigma_{i=0}^{n} q_i s^i) \\ &= e(P, Q)^{q(s)(s^n - 1)} && (\because\ e(aP, bQ) = e(P, Q)^{ab}) \end{aligned}$$

左辺＝右辺なので，$f(s) = q(s)(s^n - 1)$ が成り立つ．
今，sはランダムな点であり，前のページの通り，$\forall x \in \{g^0, \ldots, g^{n-1}\}$ で $f(x) = 0$ が言える．

ランダム評価点による多項式の定数化と線形化

- 2つ目と3つ目の問題（証明者が持っている多項式が式の一部に含まれていて，さらに積で表現）は，多項式をランダムな点で評価した結果（定数）に置き換え，多項式の積を含む等式を複数の多項式の線形和に変換することで解決できる.
- 例として，証明者はある3つの多項式 $h_1(x), h_2(x), h_3(x)$について、それらのKZGコミットメント

$$c_1 = h_1(s)P, c_2 = h_2(s)P, c_3 = h_3(s)P$$

を事前に検証者に渡しこれらが多項式として $h_1(x)h_2(x) - h_3(x) = 0$を満たすことを検証者に証明したいとしよう.

- 単純な解決策は，検証者が選んだランダムな評価点 δ でそれぞれの多項式を評価した結果

$$t_1 = h_1(\delta), t_2 = h_2(\delta), t_3 = h_3(\delta)$$

- を証明者に計算・証明させ，検証者は各々KZGコミットメントの検証をする.
- そして検証者が，$t_1\, t_2 - t_3 = 0$という等式が成り立つことを計算して確かめるという方法.
- しかし，これでは多項式の数だけ，KZGコミットメントの証明・検証をする必要がある.
  - 上記の例だと3回必要.

ランダム評価点による多項式の定数化と線形化

- そこで，証明者と検証者はコミットメントが送信された後に次のような操作を行い，この関係を効率的に証明・検証する．

1. 検証者はランダムな評価点 $\delta \in F_r$ を取り，これを証明者に送信する．（実際にはフィアットシャミアーで非インタラクティブに）
2. 証明者は $h_1(\delta) = t_1$ という評価結果について，証明 π_1 を検証者に送信する．
3. さらに、証明者は成り立つことを証明したい全体の多項式 $h'(x) = t_1 h_2(x) - h_3(x)$ において，$h'(\delta) = 0$ という評価結果について証明 π_2 を検証者に送信する。
4. 検証者は π_1 をコミットメント c_1，π_2 をコミットメント $c' = t_1 c_2 + c_3$ でそれぞれ検証する．

- 4の検証が成り立つのは、KZGコミットメントが加法準同型性、すなわち「多項式の和のコミットメントは多項式のコミットメントの和と等しい」という線形の性質を満たすため．
- これにより証明者は，線形関係の多項式についてはそれらの線形和の多項式の評価結果を送るのみでよくなる．

ランダム評価点による多項式の定数化と線形化

- ゲート制約について考える.
- 前ページの例と同様に，PlonKにおける証明者は多項式 $a(x), b(x), c(x)$を，検証者が選んだランダムな評価点 δで評価し，その評価結果 $\bar{a} = a(\delta), \bar{b} = b(\delta), \bar{c} = c(\delta)$ とそれらに対するKZGコミットメントの証明を検証者に送信する.
- 他の多項式のコミットメントは，送信しても良いし検証者が自分で計算しても良い.
- さらに，複数点における検証のための商の多項式

$$q_1(x) = \frac{L(x)a(x) + R(x)b(x) + O(x)c(x) + M(x)a(x)b(x) + C(x) + PI(x)}{x^n - 1}$$

- に対応する証明 π_{q_1}も送信する.
- $L(x), R(x), O(x), M(x), C(x)$,$PI(x)$のコミットメントは，$L(s)P, R(s)P, O(s)P, M(s)P, C(s)P, PI(s)P$であり，検証者はKZGコミットメントの線形和

$$c' = \bar{a}L(s)P + \bar{b}R(s)P + \bar{c}O(s)P + \bar{a}\bar{b}M(s)P + C(s)P + PI(s)P$$

- に対して、

$$e(c', Q) = e(\pi_{q_1}, s^n Q - Q)$$

- が成り立つことを検証することでゲート制約に対応する多項式が正当であることを確認できる.

- なお，$\bar{a}, \bar{b}, \bar{c}$は単なるスカラーであるため，多項式 $a(x), b(x), c(x)$の情報を漏らさない．また，証明者から渡された$\bar{a}, \bar{b}, \bar{c}$を適切に使い回す，例えば $M(s)P$の係数には積$\bar{a}\bar{b}$ を使う，ゲート制約とコピー制約の多項式で共有の$\bar{a}, \bar{b}, \bar{c}$を用いることで，検証者は同じ多項式 $a(x), b(x), c(x)$を使うように強制することができる．したがって，この手法は，証明者が持っている多項式が式の一部に含まれていて，さらに積で表現されるという，2と3の問題の解決もしている.
- コピー制約についても同様の手法で証明・検証できる.
- 実際のPlonKのプロトコルでは，複数の評価点・複数の評価結果に対するKZGコミットメントのバッチ処理を行うことで，さらに証明のサイズと検証者の計算量を削減している.

全体の流れ：事前準備

- 以上を踏まえたうえで，全体の流れは以下の通り．

<u>事前準備</u>

1. 算術回路からゲート制約のパラメータ $(q_{L_i}, q_{R_i}, q_{O_i}, q_{M_i}, q_{C_i})$およびコピー制約が成り立つワイヤ間の関係 $\sigma(i)$ を求める．
2. 1の値に多項式補間を適用して，それぞれ多項式$L(x), R(x), O(x), M(x), C(x), S_{\sigma_1}(x), S_{\sigma_2}(x), S_{\sigma_3}(x)$を求める．
   ※$S_{\sigma_1}(x), S_{\sigma_2}(x), S_{\sigma_3}(x)$はコピー制約から導かれる多項式の中で$\sigma(i)$ から導出される箇所．
3. 2のそれぞれの多項式に対して，KZGコミットメントを求める．

- この事前準備は回路ごとに一度だけ行えば良い．
- また，検証者は3のコミットメントのみを保持すればいいので，検証者のストレージが限られている場合 (例: スマートコントラクト)でも，回路のサイズによらず一定のストレージ量しか必要としない．

全体の流れ：証明

証明

1. 算術回路への入力から，x_a, x_b, x_cを求める．ただし，x_aのうち最初のp個はパブリック値とする．
   パブリック値は検証者に共有する
2. x_a, x_b, x_cに多項式補間を適用して，それぞれ多項式 $a(x), b(x), c(x)$を求める．同様に，パブリック値から多項式 $PI(x)$を求める．
3. $PI(x)$を除く2の多項式に対してKZGコミットメントを求める．
4. 3のコミットメントのハッシュ値から乱数β, γを求める．
5. $x_a, x_b, x_c, \beta, \gamma$から多項式 $\mathrm{z}(X)$を求める．
6. 次の多項式の関係が成り立つことをKZGコミットメントで証明する．
   評価点δは「4」と同様にコミットメントのハッシュ値から求めれば良い．

$$\forall\, x \in \{g^0, \ldots, g^{n-1}\},$$
$$L(x)a(x) + R(x)b(x) + O(x)c(x) + M(x)a(x)b(x) + (c(x) + PI(x)) = 0$$

$$\mathrm{z}(1) = 1$$
$$z(gx)(a(x) + \beta\,\sigma(i) + \gamma)(b(x) + \beta\,\sigma(n+i) + \gamma)(c(x) + \beta\,\sigma(2n+i) + \gamma)$$
$$- z(x)(a(x) + \beta\,x + \gamma)(b(x) + \beta\,k_1\,x + \gamma)(c(x) + \beta\,k_2\,x + \gamma) = 0$$

全体の流れ：検証

検証

1. 与えられた$a(x), b(x), c(x)$ のKZGコミットメントのハッシュ値から，乱数 β, γを求める．
2. パブリック値から多項式 $PI(x)$を求める．
3. $L(x), R(x), O(x), M(x), C(x), S_{\sigma_1}(x), S_{\sigma_2}(x), S_{\sigma_3}(x), PI(x)$のKZGコミットメントと，$a(x), b(x), c(x), z(gx)$ については与えられたKZGコミットメントとδ による評価値$\bar{a}, \bar{b}, \bar{c}, \bar{z}$を用いて，証明プロセスのステップ6で述べられた多項式の関係が成り立つことを検証する．

- 本スライドの著作権は，東京大学ブロックチェーンイノベーション寄付講座に帰属しています．自己の学習用途以外の使用，無断転載・改変等は禁止します．