

Core Program Week 4

zkSTARKの基礎



目次

- zkSTARKの概観
- 数学的基礎①～FFT～
- 数学的基礎②～Reed-Solomon符号～
- AIR
- コミットメント
- FRI(Fast Reed-Solomon IOP)
- STARKプロトコル

zkSTARKの概観



STARKとは

STARK (Scalable Transparent ARgument of Knowledge)

- Scalable
 - ステートメントを直接計算するのに必要な時間を T とした時、証明の生成時間は $O(T \log T)$ であり、検証時間は $O(\log T)$ である
- Transparent
 - Trusted setupを必要としない
- ARgument of Knowledge
 - ある知識を持っていることの証明ができる

STARKとSNARKの比較

	STARK	SNARK
セットアップ	不要(Traasparent)	必要(Trasted Setup)
証明サイズ	大きい(数十～数百 KB)	小さい(数百 bytes)
検証時間	やや遅い	高速
証明生成時間	中程度	高速～中程度
量子耐性	あり	なし

量子耐性について

- STARK: ハッシュ関数の安全性のみ
→ 理論的に破ることができる量子アルゴリズムがまだ見つかっていない
- SNARK: 離散対数問題の困難性
→ Shorのアルゴリズムにより量子コンピュータで破られる

参照: https://en.wikipedia.org/wiki/Shor%27s_algorithm

STARKのzkVMでの使用

zkVM の多くのプロジェクトで採用

コンポーネントの比較

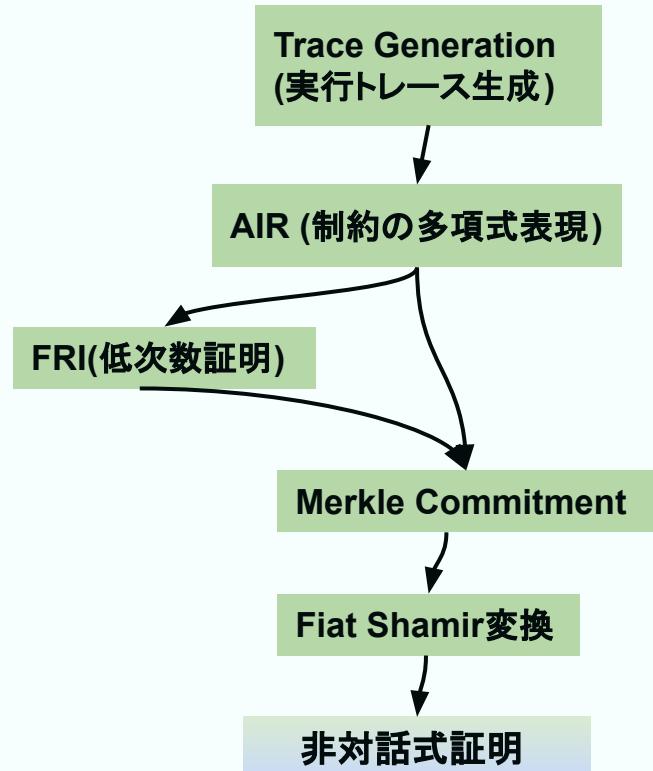
	Style	Arithmetization Schemes	Commitment Scheme	Memory Consistency Check
Risc 0	STARK(zk-STARK)	AIR	FRI	Merkle Tree
SP1	STARK(Plonky 3)	AIR	FRI	Merkle Tree
Nexus	Folding(Nova)	CCS	Pedersen commitment	Merkle Tree
Ceno	GKR	CCS	FRI	Offline Memory Checking
Jolt	Lookup(Lasso)	R1CS	Hyrax	Offline Memory Checking

主要なzkVMとその特徴

<https://www.youtube.com/watch?v=XDx4YngodU0>

STARKプロトコルの全体像

1. **Trace Generation(実行トレースの生成)**
計算の各ステップを記録し、実行の履歴を作成
2. **AIR(制約の多項式表現)**
実行の正しさを代数的制約として多項式形式で表現
3. **FRI(低次数証明)**
関連する多項式が低次数であることを効率的に証明
4. **Merkle Commitment**
データの整合性を暗号学的ハッシュツリーで保証
5. **Fiat-Shamir変換**
対話式プロトコルを非対話式に変換してランダム性を除去し非対話式証明を生成



STARK理解の数学的基礎①

～FFT～



DFT(離散フーリエ変換)

多項式の2つの表現形式

係数表現

$$f(x) = a_0 + a_1x + a_2x^2 + \dots + a_{n-1}x^{n-1}$$

評価値表現

$$[(x_0, f(x_0)), (x_1, f(x_1)), \dots, (x_{n-1}, f(x_{n-1}))]$$

DFT(離散フーリエ変換)

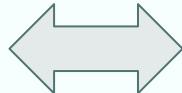
多項式の2つの表現形式

係数表現

$$f(x) = a_0 + a_1x + a_2x^2 + \dots + a_{n-1}x^{n-1}$$

評価値表現

$$[(x_0, f(x_0)), (x_1, f(x_1)), \dots, (x_{n-1}, f(x_{n-1}))]$$



DFTの役割

係数表現と評価値表現の相互変換

なぜ評価値表現が有用？

- ・多項式の乗算: 各点での値を掛けるだけ(O(N))
- ・多項式の補間: 評価値から係数を復元

$$f(x) = a_0 + a_1x + a_2x^2 + \dots + a_{n-1}x^{n-1}$$

$$g(x) = b_0 + b_1x + b_2x^2 + \dots + b_{n-1}x^{n-1}$$

FFT入門 - 多項式演算の課題

従来のDFTの計算コスト

$$f(x) = a_0 + a_1x + a_2x^2 + \dots + a_{n-1}x^{n-1}$$

$$g(x) = b_0 + b_1x + b_2x^2 + \dots + b_{n-1}x^{n-1}$$

主要な演算とその計算量

- ・評価 : $f(x_0), f(x_1), \dots, f(x_{N-1})$ を計算 $\rightarrow O(N^2)$
- ・補間 : N 個の点から多項式を復元 $\rightarrow O(N^2)$
- ・乗算 : $f(x) \times g(x) \rightarrow O(N^2)$

問題

データサイズ N が大きくなると計算時間が急激に増加

FFTとは何か

高速フーリエ変換(Fast Fourier Transform)

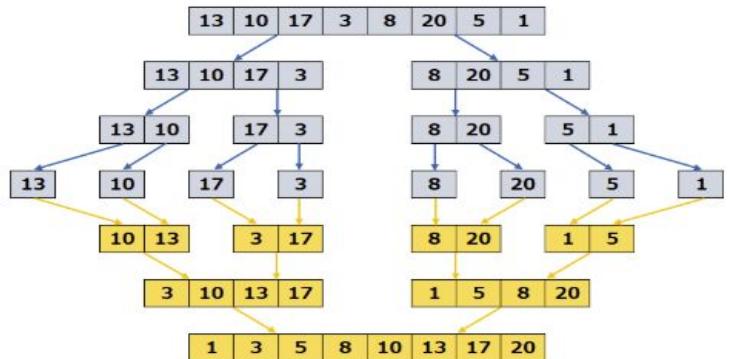
定義

- ・離散フーリエ変換(DFT)を高速に計算するアルゴリズム
- ・分割統治法を用いて計算量を劇的に削減

計算量の改善

従来のDFT: $O(N^2)$
FFT: $O(N \log N)$
→ 高速化

例: マージソート



<https://breeze-group.co.jp/202005/sorting-algorithm/>

N次単位根の重要性

N次単位根とは

定義

- ・ $\omega^N = 1$ を満たす複素数 ω
- ・有限体 \mathbb{F}_p では、位数Nの元として存在

性質

- ・ $\omega^0, \omega^1, \omega^2, \dots$ は全て異なる
- ・ $\omega^N = \omega^0$ (周期性)

STARKでの利用

- ・有限体 \mathbb{F}_p の乗法群から適切な巡回部分群を選択
- ・ N の形が効率的

FFTの基本原理

分割統治による高速化

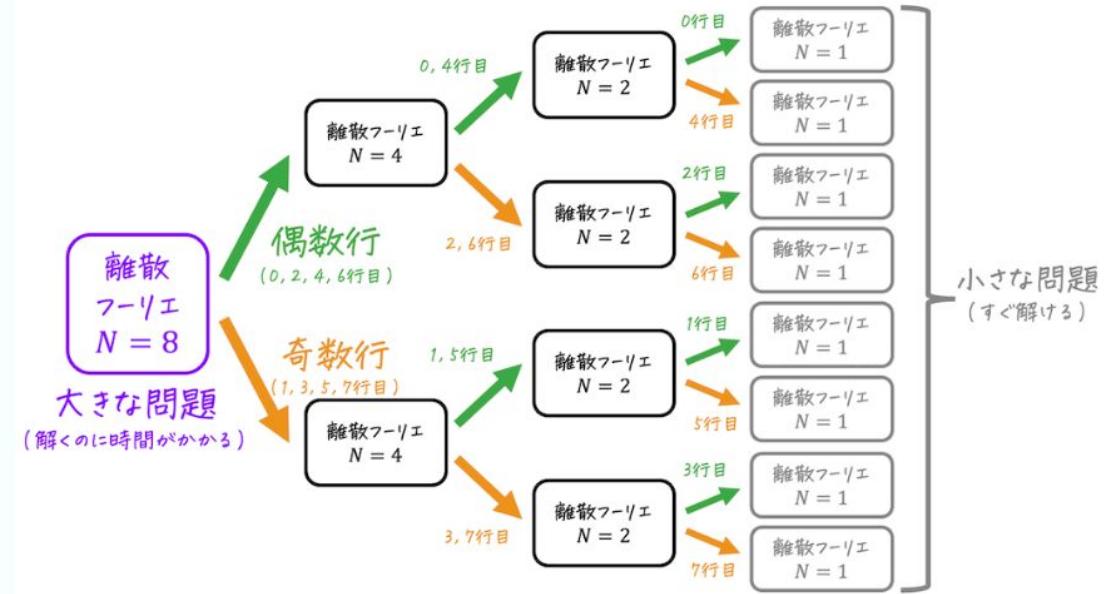
基本アイデア

- ・N点のDFTを2つのN/2点のDFTに分割
- ・再帰的に適用して計算量を削減

偶数・奇数による分割

$$f(x) = f_{\text{even}}(x^2) + x \cdot f_{\text{odd}}(x^2)$$

- ・ $f_{\text{even}}(x) = a_0 + a_2x + a_4x^2 + \dots$
- ・ $f_{\text{odd}}(x) = a_1 + a_3x + a_5x^2 + \dots$



<https://www.momoyama-usagi.com/entry/math-seigo14>

計算量

- ・分割段階数: $\log_2 N$ 回 ($N \rightarrow N/2 \rightarrow \dots \rightarrow 1$)
- ・各段階の処理: $O(N)$ 回の演算
- ・総計: $O(N \log N)$

STARKでのFFT利用場面

FFTが活躍する3つの場面

1. 多項式の評価

- trace → 多項式への変換
- 定義域拡大($H \rightarrow G$)での効率的な評価

2. 多項式の補間

- 点の集合から多項式を効率的に復元
- ラグランジュ補間を $O(N \log N)$ で実行

3. 多項式の乗算

- 制約多項式の計算
- 商多項式の計算

STARK理解の数学的基礎②

～ Reed-Solomon符号 ～



誤り訂正符号入門

誤り訂正符号とは

基本概念

- ・冗長性を追加してデータの信頼性を向上
- ・一部のデータが破損しても元データを復元可能

符号化の流れ

元データ(k 個) → 符号化 → 符号語(n 個, $n > k$)

パラメータ

- ・ **符号長**: n (符号語の長さ)
- ・ **情報長**: k (元データの長さ)

Reed-Solomon符号の定義

Reed-Solomon符号とは

定義 $\alpha_0, \alpha_1, \dots, \alpha_{n-1}$

k 個のメッセージシンボル $[m_0, m_1, \dots, m_{k-1}]$

n 個の符号語 $[c_0, c_1, \dots, c_n]$ 符号化

符号化手順

多項式化: メッセージから多項式 $f(x)$ を構成

$$f(x) = m_0 + m_1x + m_2x^2 + \dots + m_{k-1}x^{k-1}$$

評価: n 個の異なる点 $\alpha_0, \alpha_1, \dots, \alpha_{(n-1)}$ で評価

$$c_i = f(\alpha_i) \quad (i = 0, 1, \dots, n - 1)$$

重要な性質

- 任意の k 個の符号語があれば元の多項式 $f(x)$ を完全復元可能
- 最大 $(n-k)$ 個までの誤りを訂正可能

Reed-Solomon符号の具体例

\mathbb{F}_5 での符号化プロセス

1. 設定

メッセージ: [4, 2] ($k=2$)

多項式: $f(x) = 2x + 4$ (1次多項式)

2. 評価点と符号化

評価点 $H = \{1, 4\}$ ($T=2$)

拡大領域 $G = \{1, 3, 4, 2\}$ ($N=4$)

3. 符号化の実行

$$c_0 = f(1) = 2 \times 1 + 4 = 6 \equiv 1 \pmod{5}$$

$$c_1 = f(3) = 2 \times 3 + 4 = 10 \equiv 0 \pmod{5}$$

$$c_2 = f(4) = 2 \times 4 + 4 = 12 \equiv 2 \pmod{5}$$

$$c_3 = f(2) = 2 \times 2 + 4 = 8 \equiv 3 \pmod{5}$$



符号語: [1, 0, 2, 3]

Reed-Solomon符号の復元能力

誤り訂正の例

受信した符号語(一部破損)

- ・送信: [1, 0, 2, 3]
- ・受信: [1, ?, 2, ?] (2箇所が破損)

一般的性質

k 個の正しい符号語があれば
必ず復元可能

復元プロセス

1. 残りの2点 (1,1), (4,2) を使用

2. これらを通る1次多項式を求める

直線の方程式: $f(x) = 2x + 4$

2. 破損箇所を計算

$f(3) = 0, f(2) = 3$

STARKでのReed-Solomon利用①

定義域拡大による検証効率化

基本アイデア

元のトレース $H(T$ 個の点)から拡大領域 $G(8T$ 個の点)へ Reed-Solomon符号化

- ・元のトレース: H 上の T 点
- ・拡大トレース: G 上の $8T$ 点

STARKでのReed-Solomon利用①

定義域拡大による検証効率化

基本アイデア

元のトレース $H(T$ 個の点)から拡大領域 $G(8T$ 個の点)へ Reed-Solomon符号化

- ・元のトレース: H 上の T 点
- ・拡大トレース: G 上の $8T$ 点

正しい多項式

正当な計算トレースから構築された次数 $T-1$ 以下の多項式。

不正な多項式

正当でない計算から構築された多項式、または次数が T 以上の多項式。

- 高確率で $7T$ 個以上の点で異なる
- ランダムサンプリングで効率的に検出可能

数学的根拠

- ・次数 $T-1$ 以下の異なる多項式は、高々 T 個の点でしか一致しない
- ・ $|G| = 8T$ なので、7/8以上の点で異なる
- ・少数の点をチェックするだけで高い確度で検証可能

STARKでのReed-Solomon利用②

低次数テスト(Low Degree Testing)

問題設定

証明者が「私の多項式は次数 T 以下です」と主張

検証者が効率的にこれを検証したい

FRIプロトコル

- Reed-Solomonの性質を利用した効率的なLDT
- 証明コスト: $O(T)$
- 検証コスト: $O(\log T)$

Reed-Solomonを用いた解決法

1. 符号化: 多項式を G 上で Reed-Solomon 符号化
2. サンプリング: ランダムに数点を選択して評価値を要求
3. 検証: 次数の整合性をチェック

AIR



AIRの基本概念

AIR (Arithmetic Intermediate Representation) とは何か？

-> 計算の制約条件を多項式の関係式で表現する手法

基本的な変換の流れ

1. 実行トレースを構築する
2. 上記1を多項式 $P(X)$ に変換する
3. 制約条件を多項式に変換する

フィボナッチ数列における制約条件

計算制約: $a_{i+2} = a_{i+1} + a_i$

初期条件: $a_0 = 1, a_1 = 1$

最終条件: $a_{T-1} = A$

-> これを多項式 $P(X)$ の性質として表現する

1. 実行トレースの構築

実行トレース: 計算の各ステップでの状態を記録したもの

フィボナッチの例 ($T = 8$)

Step	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
値	1	1	2	3	5	8	13	21

遷移制約 (Transition Constraints):

- ステップ間の関係
- $a_{i+2} = a_{i+1} + a_i$

境界制約 (Boundary Constraints):

- 初期値: $a_0 = 1, a_1 = 1$
- 最終値: $a_7 = 21$

→次のステップでこれらの制約を多項式にエンコードする！

2. 多項式による制約表現

実行トレース を有限体上の多項式として表した $P(X)$ に変換する

$P(\omega^i) = a_i \quad (i = 0, 1, \dots, T-1)$ 次数 $< T$ の多項式を **ラグランジュ補間** で構成する。
 ここで、巡回群 $H = \{\omega^0, \omega^1, \dots, \omega^{T-1}\}$ を定義域とする

つまり、離散データ列 $(a_0, a_1, \dots, a_{T-1})$ を有限体上の一つの多項式に変換！

制約条件

多項式での表現：

$$P(\omega^{i+2}) = P(\omega^{i+1}) + P(\omega^i)$$

↓

$$P(\omega^2 X) = P(\omega X) + P(X) \quad (X \in \{\omega^0, \omega^1, \dots, \omega^{T-3}\})$$

境界制約

初期条件

$$\begin{aligned} P(\omega^0) &= P(1) = 1 \\ P(\omega^1) &= P(\omega) = 1 \end{aligned}$$

最終条件：

$$P(\omega^{T-1}) = 21$$

3. 遷移制約多項式の構築

遷移制約の処理

制約: $P(\omega^2 X) = P(\omega X) + P(X)$ ($X \in \{\omega^0, \dots, \omega^{T-3}\}$)

制約多項式を定義:

$$C_0(X) := P(\omega^2 X) - P(\omega X) - P(X)$$

この多項式は $X = \omega^0, \omega^1, \dots, \omega^{T-3}$ で 0 となる

ゼロ多項式の構築

$$Z_0(X) := (X - \omega^0)(X - \omega^1) \cdots (X - \omega^{T-3})$$

ここで

$$(X - \omega^0)(X - \omega^1) \cdots (X - \omega^{T-1}) = X^T - 1$$

よって: $Z_0(X) = \frac{X^T - 1}{(X - \omega^{T-2})(X - \omega^{T-1})}$

$$= \frac{X^T - 1}{X^2 - (\omega^{T-2} + \omega^{T-1})X + \omega^{T-2}\omega^{T-1}}$$

商多項式の定義

制約が満たされる $\Leftrightarrow C_0(X)$ が $Z_0(X)$ で割り切れる。
多項式の整除性 (divisibility) を使う

$$D_0(X) := \frac{C_0(X)}{Z_0(X)}$$

検証すべき関係:

$$C_0(X) = D_0(X) \times Z_0(X)$$

4. 境界制約多項式の構築

初期条件の制約多項式

制約: $P(1) = 1$

制約多項式:

$$C_1(X) := P(X) - 1$$

これは $X = 1$ で 0 になる

ゼロ多項式: $Z_1(X) = X - 1$

商多項式: $D_1(X) = \frac{C_1(X)}{X - 1}$

4. 境界制約多項式の構築

初期条件の制約多項式

制約: $P(1) = 1$

制約多項式:

$$C_1(X) := P(X) - 1$$

これは $X = 1$ で 0 になる

ゼロ多項式: $Z_1(X) = X - 1$

商多項式: $D_1(X) = \frac{C_1(X)}{X - 1}$

第二初期条件

制約: $P(\omega) = 1$

制約多項式: $C_2(X) := P(X) - 1$

ゼロ多項式: $Z_2(X) = X - \omega$

商多項式: $D_2(X) = \frac{C_2(X)}{X - \omega}$

4. 境界制約多項式の構築

初期条件の制約多項式

制約: $P(1) = 1$

制約多項式:

$$C_1(X) := P(X) - 1$$

これは $X = 1$ で 0 になる

ゼロ多項式: $Z_1(X) = X - 1$

$$\text{商多項式: } D_1(X) = \frac{C_1(X)}{X - 1}$$

第二初期条件

制約: $P(\omega) = 1$

制約多項式: $C_2(X) := P(X) - 1$

ゼロ多項式: $Z_2(X) = X - \omega$

$$\text{商多項式: } D_2(X) = \frac{C_2(X)}{X - \omega}$$

最終条件

制約: $P(\omega^{T-1}) = A$

制約多項式: $C_3(X) := P(X) - A$

ゼロ多項式: $Z_3(X) = X - \omega^{T-1}$

$$\text{商多項式: } D_3(X) = \frac{C_3(X)}{X - \omega^{T-1}}$$

検証: すべての制約が満たされる

\Leftrightarrow

$$C_0(X) = D_0(X) \times Z_0(X)$$

$$C_1(X) = D_1(X) \times Z_1(X)$$

$$C_2(X) = D_2(X) \times Z_2(X)$$

$$C_3(X) = D_3(X) \times Z_3(X)$$

コミットメント (Merkle Tree Commitment)



コミットメントの復習と選択

既習内容の復習

コミットメントの基本概念

- ・**Binding Property**: コミット後にデータを変更できない
- ・**基本フロー** : Commit → Open → Verify

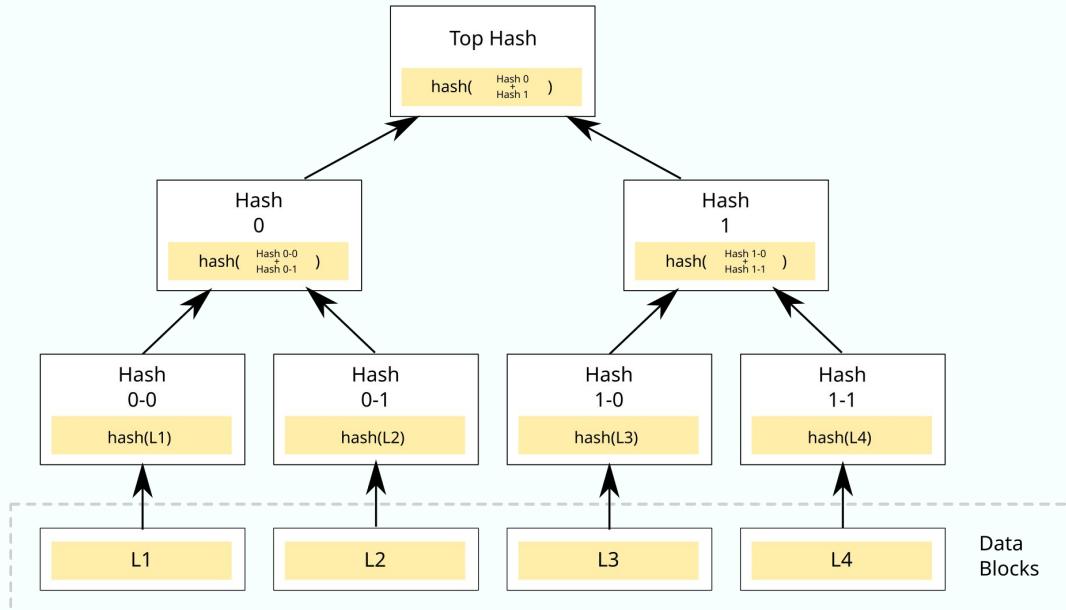
STARKではMerkle Treeを選択

方式	KZG Commitment	Merkle Tree Commitment
証明サイズ	$O(1)$ (定数)	$O(\log N)$
Trusted Setup	必要	不要 (Transparent)
量子耐性	脆弱	耐性あり
基盤技術	楕円曲線	暗号学的ハッシュ関数

Merkle Treeの構造と性質

サイズとセキュリティ

- ・葉(Leaf)の数: N
- ・木の高さ: $\log_2(N)$
- ・根(Root)のサイズ: ハッシュ値1個(通常256bit)
- ・セキュリティ: ハッシュ関数の衝突耐性に依存



https://en.wikipedia.org/wiki/Merkle_tree

Merkle Pathによる検証

Merkle Pathの定義

特定の葉 L_i が Merkle tree に含まれていることを証明するために必要な最小限のハッシュ値の集合

具体例: L3のMerkle Path

L_3 がこのMerkle treeに含まれていることを証明したい

必要なハッシュ値

1. $\text{Hash } 1-1 = \text{hash}(L_4) \leftarrow L_3$ の兄弟
2. $\text{Hash } 0 \leftarrow \text{親}(\text{Hash } 1)$ の兄弟

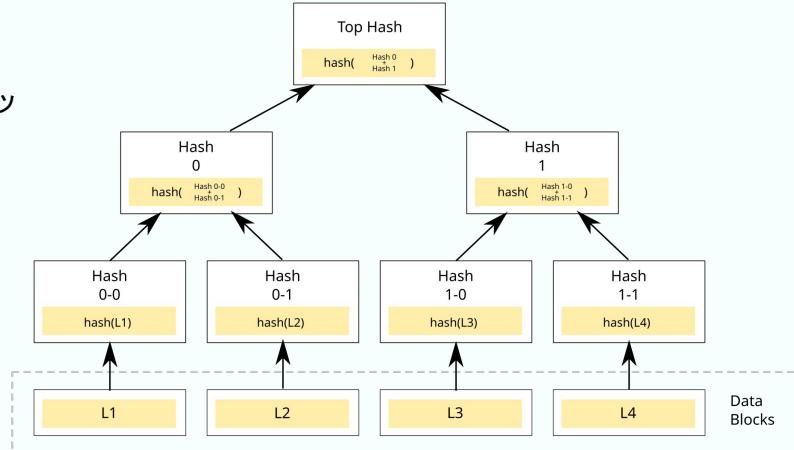
検証プロセス

ステップ1: $\text{Hash } 1-0 = \text{hash}(L_3)$ を計算

ステップ2: $\text{Hash } 1 = \text{hash}(\text{Hash } 1-0 \parallel \text{Hash } 1-1)$ を計算

ステップ3: Root Hash (右表中 Top Hashのこと) = $\text{hash}(\text{Hash } 0 \parallel \text{Hash } 1)$ を計算

ステップ4: 計算したRoot Hashと既知のRoot Hashを比較



https://en.wikipedia.org/wiki/Merkle_tree

STARKでの統合利用

コミットメント -> データが後から改ざんされていないことを保証する

■ STARKでの実際の使用例

- ・ $P(X)$ の評価値: $[P(g_0), P(g_1), \dots, (N=8)]$ 個)
- ・商多項式の評価値: $[D_i(g_0), D_i(g_1), \dots, (i=0,1,2,3)]$

→統合コミットの構築 各点 x の全評価値をまとめて 1つの葉とする

$$\text{leaf}_j = (P(g_j), D_0(g_j), D_1(g_j), D_2(g_j), D_3(g_j))$$

■ クエリ・検証プロセス

検証者のクエリ: $x \in \mathbb{F}$ ランダムに選択

証明者

→評価値: $\text{leaf}_x = (P(x), D_0(x), D_1(x), D_2(x), D_3(x))$

検証:

1. **Merkle Path検証**: leaf_x が Merkle Tree に含まれることを確認
2. **制約検証**: $C_i(x) \stackrel{?}{=} D_i(x) \times$ をチェック



FRI(Fast Reed-Solomon IOP)

AIRとMerkle Commitmentからの課題

これまでに解決したこと

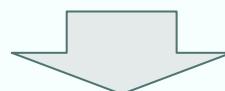
- AIR: 計算制約を多項式の関係式 $C_i(X) = D_i(X)$ に変換
- Merkle Commitment: 多項式評価値の信頼性を確保

残された問題

- 証明者が「 D の次数は T より小さい」と主張するとき、それをどう検証するか？

なぜ次数が重要なのか？

- 高次数の偽多項式:多くの点で正しい多項式と一致する可能性
- 低次数の制約:異なる多項式は高々次数個の点でしか一致しない



具体的な脅威

悪意ある証明者が高次数の $D'_i(X)$ を使って偽証明を作成
→ ランダムサンプリングでの検出が困難になる

必要なもの: 効率的な Low Degree Testing (LDT) プロトコル

Low Degree Testing (LDT) とは

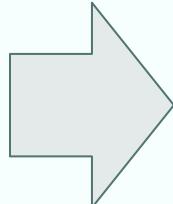
多項式の次数検証問題

問題設定

- ・証明者: 「私の多項式 $f(X)$ の次数は d 以下です」
- ・検証者: 「それを効率的に検証したい」
- ・制約条件:
 - ・ $f(X)$ の係数を全て見るのは非効率
 - ・ $f(X)$ の一部の評価値のみアクセス可能

直接的な検証方法

- ・方法1: 全ての係数をチェック $\rightarrow O(d)$ の通信量・検証時間
- ・方法2: $d+1$ 個の点をサンプル $\rightarrow O(d)$ の計算量



FRIの革新

- ・証明時間: $O(d)$
- ・検証時間: $O(\log d)$
- ・通信量: $O(\log d)$

基本アイデア: 「次数 d の検証問題」を「次数 $d/2$ の検証問題」に再帰的に変換

FRIの基本アイデア - 次数半減

再帰的次数削減の仕組み

Step 1: 多項式の分解

→ 次数 d の多項式 $f_0(X)$ を偶数次と奇数次に分解

$$f_0(X) = g_0(\text{ただし } Y = X^2)$$

$$\text{偶数次項: } E(Y) = a_0 + a_2 Y + a_4 Y^2 + \dots \quad (Y = X^2)$$

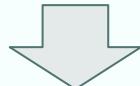
$$\text{奇数次項: } O(Y) = a_1 + a_3 Y + a_5 Y^2 + \dots \quad (Y = X^2)$$

$$\Rightarrow f(X) = E(X^2) + X \cdot O(X^2)$$

Step 2: 線形結合による次数削減

→ 検証者がランダムな β_0 を送信し、証明者が計算

$$f_1(Y) := g_0(\beta_0, Y) = E(Y) + \beta_0 \cdot O(Y)$$



再帰的適用

$$f_0(\text{次数 } d) \rightarrow f_1(\text{次数 } d/2) \rightarrow f_2(\text{次数 } d/4) \rightarrow \dots \rightarrow (\text{底数})$$

具体例

$$f_0(X) = 7 + 5X + 3X^3 \quad (\text{次数 } 3)$$

分解:

$$E(Y) = 7 + (\text{偶数次項, } Y=X^2)$$

$$O(Y) = 5 + (\text{奇数次項, } Y=X^2)$$

$$\Rightarrow f_0(X) = (7 + 3X^2) + X(5 + 2X^2) = E(X^2) + X \cdot O(X^2)$$

検証者の乱数: $\beta_0 = 4$

$$f_1(Y) = E(Y) + 4 \cdot O(Y) = (7 + 3Y) + 4(5 + 2Y) = 27 + 11Y$$

重要な性質

$$f_0(X) \text{ の次数} \leq d \Leftrightarrow f_1(Y) \text{ の次数} \leq d/2$$

FRI Commit Phase

証明者の証明生成過程

Phase 1: 多項式系列の生成

※以下の手順を、多項式が定数になるまで繰り返す

1. $f_i(X) = E_i(X^2) + X$ に分解⁽²⁾
2. $f_i(X)$ の評価値を $G^{(i)}$ 上でコミット
3. 検証者から β_i を受信
4. $f_{i+1}(Y) = E_i(Y) + \beta_i$ を計算^(Y)
5. 定義域を半分に: $G^{(i+1)}$ (サイズは \mathcal{O} (半分))

Phase 2: Merkle Commitment

各段階の多項式評価値を Merkle Tree にコミット

- $f_0(G^{(0)}) \rightarrow \text{Merkle Tree}_0 \rightarrow \text{Root}_0$
- $f_1(G^{(1)}) \rightarrow \text{Merkle Tree}_1 \rightarrow \text{Root}_1$
- ...
- $f_{n-1}(G^{(n-1)}) \rightarrow \text{Merkle Tree}_{n-1} \rightarrow \text{Root}_{n-1}$

Phase 3: 最終値の送信

$f_n(X) \neq$ 定数 C を直接送信 (Commit Phase完了)

初期設定

- ・目標: $f_0(X)$ の次数が $d = T$ 以下であることを証明
- ・定義域: $G^{(0)}$ (サイズは一般には $\rho \cdot T$ で、 ρ は「拡大率」と呼ばれる)

※ ρ はエラー検出能力と計算量のトレードオフを考慮して設定され、通常 4~16 の範囲が用いられる。

FRI Query Phase

検証者の検証過程

Phase 1: 一貫性検証の準備

検証者: $x \in G$ をランダムに選択

証明者: 以下を計算

- $s_0 = x$
- $s_1 = s_0^2 (= x^2)$
- $s_2 = s_1^2 (= x^4)$
- ...
- $s_{i+1} = s_i^2$

Phase 2: 評価値とパスの送信

各段階 $i = 0, \dots, n-1$ について証明者が送信

1. $f_i(s)$ とその Merkle Path

2. $f_i(s'_i)$ とその Merkle Path -> ここで s'_i は s_{i+1} を満たすもう一つの値 (つまり $s'_i = -s_i$)

Phase 3: 一貫性の検証

検証者は各段階で以下をチェック

1. Merkle Path 検証: 送信された値が正しくコミットされているか

2. 線形関係検証: $f_{i+1}(s_{i+1}) = E_i(s_{i+1}) + \beta$ が成立するか

一貫性検証の詳細

線形関係の確認方法

問題設定 $f_i(X) = E_i(X^2) + X \text{か} O_i(X^2)$ $f_{i+1}(Y) = \text{O} \text{べき} + \beta_i \cdot O_i \text{が正しく計算されているか？}$

検証に必要な情報

・既知:

- ・確認した $f_{i+1}(s_i)$ が $f_i(s'_i)$, β_i から正しく導出されているか
 $f_{i+1}(s_{i+1}) = f_i(s_i), f_i(s'_i)$

数学的関係

より、 (通常の場合)
 $s_i^2 = s_{i+1} = (s'_i)^2$ $s'_i = -s_i$

$$f_i(s_i) = E_i(s_{i+1}) + s_i \cdot O_i(s_{i+1})$$

$$f_i(s'_i) = E_i(s_{i+1}) + s'_i \cdot O_i(s_{i+1}) = E_i(s_{i+1}) - s_i \cdot O_i(s_{i+1})$$

$E_i(s_{i+1}), O_i(s_{i+1})$ の計算

$$E_i(s_{i+1}) = \frac{f_i(s_i) + f_i(s'_i)}{2}$$

$$O_i(s_{i+1}) = \frac{f_i(s_i) - f_i(s'_i)}{2s_i}$$



最終検証

- ・期待値: $f_{i+1}(s_{i+1}) = E_i(s_{i+1}) + \beta_i \cdot O_i(s_{i+1})$
- ・実際値: 証明者が送信した
 $=>$ 一致するかチェック $f_{i+1}(s_{i+1})$

具体例での理解

$d = 4, n = 2$ の場合

初期設定

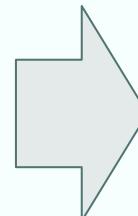
- $f_0(X) = 1 + 2X + 3X^2 + 4X^3$ (次数 $3 \leq 4$)
- $G^{(0)} = \{1, g, g^2, g^3, g^4, g^5, g^6, g^7\}$ (8個)

Commit Phase: Round 0

- 分解: $f_0(X) = (1 + 3X^2) + X(2 + 4X^2) = E_0(X^2) + X \cdot O_0(X^2)$
- 検証者: $\beta_0 = 5$
- 計算: $f_1(Y) = E_0(Y) + 5 \cdot O_0(Y) = (1 + 3Y) + 5(2 + 4Y) = 11 + 23Y$
- 定義域: $G^{(1)} = \{1, g^2, g^4, g^6\}$ (4個)

Commit Phase: Round 1

- 分解: $f_1(Y) = 11 + 23Y = E_1(Y^2) + Y \cdot O_1(Y^2) = 11 + Y \cdot 23$
- 検証者: $\beta_1 = 7$
- 計算: $f_2(Z) = E_1(Z) + 7 \cdot O_1(Z) = 11 + 7 \cdot 23 = 172$ (定数)
- 定義域: $G^{(2)} = \{1, g^4\}$ (2個)



Query Phase

- クエリ: $x = g^3 \in G^{(0)}$
 計算: $s_0 = g^3, s_1 = g^6, s_2 = g^4 \pmod{|G|}$
 対応点: $s'_0 = g^5$ ($\because (g^3)^2 = g^6 = (g^5)^2$)

検証

1. $f_0(g^3), f_0(g^5)$ から $E_0(g^6), O_0(g^6)$ 計算
2. $f_1(g^6) = E_0(g^6) + \beta_0 \cdot O_0(g^6)$ を確認
3. 同様に $f_2(g^4) = 172$ まで検証

FRIの健全性 (Soundness)

なぜFRIで偽証明が困難なのか

=> 高次数多項式は線形結合によって「ランダム化」される

1: 次数保存性

$f(X)$ の次数 > $d \Rightarrow P[f_i(Y) \text{の次数} > d/2] \geq 1/2$ (β がランダムの場合)

2: 累積効果

n 回の変換後、偽の高次数多項式が全ての段階で 低次数に見える確率 $\leq (1/2)^n$

複数クエリでの強化

m 回のクエリで偽証明の成功確率 $\leq (2^{-k})^m$

例: 20回のクエリで成功確率 $\leq 2^{-200}$ (極めて小さい)

セキュリティレベル

$T = 2^{128}$ の場合、 $n = 128$ 回の変換
偽証明の成功確率 $\leq 2^{-128}$

暗号学的セキュリティレベル

- 80ビット: 2^{-80}
- 128ビット: 2^{-128}
- 256ビット: 2^{-256}

※ STARKでは通常 128ビット以上のセキュリティレベルを使用

FRIの効率性

証明者の計算量

Commit Phase

- 各段階で $O(|G^{(i)}|) = O(T/2^i)$ 回の多項式評価
- 総計: $O(T) + O(T/2) + \dots + O(1) = O(T)$
- Merkle Tree構築: 各段階で $O(T/2^i)$
- 総計: $O(T)$ (全段階合計)

合計: $O(T)$

検証者の計算量

Query Phase

- m 回のクエリ各々で $n = \log T$ 段階の検証
- 各段階で定数時間の計算 + Merkle Path検証
- 総計: $O(m \log T)$

$m = O(\lambda)$ (セキュリティパラメータ) の場合: $O(\lambda \log T)$

通信量

- Commit Phase: n 個のMerkle Root (固定サイズ)
- Query Phase: $m \times n$ 個のMerkle Path + 評価値
- 各Merkle Pathのサイズ: $O(\log T)$

総通信量: $O(m \log T)$

STARKでのFRI統合

AIRとの連携

AIRで構築した統合多項式

$$f(X) = \alpha_0 D_0(X) + \alpha_1 D_1(X) + \alpha_2 D_2(X) + \alpha_3 D_3(X)$$

FRIプロトコルの適用

1. 検証者が $\alpha_0, \alpha_1, \alpha_2, \alpha_3$ をランダムに選択
2. 証明者が $f(X)$ を計算
3. $f(X)$ に対して FRI を実行 (次数 $< T$ の検証)
4. 並行して $f(X)$ の 正当性も検証

STARKでのFRI統合

AIRとの連携

AIRで構築した統合多項式

$$f(X) = \alpha_0 D_0(X) + \alpha_1 D_1(X) + \alpha_2 D_2(X) + \alpha_3 D_3(X)$$

FRIプロトコルの適用

1. 検証者が $\alpha_0, \alpha_1, \alpha_2, \alpha_3$ をランダムに選択
2. 証明者が $f(X)$ を計算
3. $f(X)$ に対して FRI を実行 (次数 $< T$ の検証)
4. 並行して $f(X)$ の 正当性も検証

各クエリ点 x で

$$\rightarrow f(x) =? \alpha_0 D_0(x) + \alpha_1 D_1(x) + \alpha_2 D_2(x) + \alpha_3 D_3(x)$$

左辺: FRI で送信された $f(x)$

右辺: Merkle commitment から取得した $D_0(x), D_1(x), D_2(x), D_3(x)$

STARKプロトコル



Fiat-Shamir変換との組み合わせ

非対話式証明への変換: Interactive → Non-interactive STARK

Fiat-Shamir変換の適用

各ランダム値は、それまでの全 transcript(履歴)を含めてハッシュ値を計算する

- $\alpha_0, \alpha_1, \alpha_2, \alpha_3 = \text{Hash}(\text{公開入力} \parallel \text{Root}_{P,D})$

※ Root_{P,D}: AIRフェーズで作成した P(X) と D_0 ~ D_3 の Merkle root

- $\beta_0 = \text{Hash}(\text{公開入力} \parallel \text{Root}_{P,D} \parallel \alpha_0 \parallel \alpha_1 \parallel \alpha_2 \parallel \alpha_3 \parallel \text{Root}_{t_0})$

※ Root_{t_0}: FRI の最初の段階 f_0 の Merkle root

- $\beta_1 = \text{Hash}(\text{公開入力} \parallel \text{Root}_{P,D} \parallel \alpha_0 \parallel \alpha_1 \parallel \alpha_2 \parallel \alpha_3 \parallel \text{Root}_{t_0} \parallel \beta_0 \parallel \text{Root}_{t_1})$

※ 前の β_0 を含めることで連鎖構造を作る

クエリ点 $x = \text{Hash}(\text{全transcript})$

非対話式証明の構成

証明者が単一の証明データを生成することで、検証者が独立的に検証できる

STARKプロトコル まとめ

証明生成プロセス

Phase 1: AIR + Merkle Commitment

1. 計算トレースから多項式 $P(X)$ を構築
2. 制約条件から商多項式 $D_0(X), D_1(X), D_2(X), D_3(X)$ を計算
3. 全評価値を Merkle Treeにコミット → Root_main

STARKプロトコル まとめ

証明生成プロセス

Phase 1: AIR + Merkle Commitment

1. 計算トレースから多項式 $P(X)$ を構築
2. 制約条件から商多項式 $D_0(X), D_1(X), D_2(X), D_3(X)$ を計算
3. 全評価値を Merkle Treeにコミット → Root_main

Phase 2: FRI Proof Generation

1. ランダム係数 $\alpha_0, \alpha_1, \alpha_2, \alpha_3 = \text{Hash}(\text{公開入力 } || \text{Root_main})$
2. 統合多項式 $f_0(X) = \alpha_0 D_0(X) + \alpha_1 D_1(X) + \alpha_2 D_2(X) + \alpha_3 D_3(X)$
3. FRI段階的次数削減: $f_0 \rightarrow f_1 \rightarrow f_2 \rightarrow \dots \rightarrow C$ (定数)
4. 各段階でMerkle Root生成: $\text{Root}_0, \text{Root}_1, \dots, \text{Root}_{\{n-1\}}$

STARKプロトコル まとめ

証明生成プロセス

Phase 1: AIR + Merkle Commitment

1. 計算トレースから多項式 $P(X)$ を構築
2. 制約条件から商多項式 $D_0(X), D_1(X), D_2(X), D_3(X)$ を計算
3. 全評価値を Merkle Treeにコミット → Root_main

Phase 2: FRI Proof Generation

1. ランダム係数 $\alpha_0, \alpha_1, \alpha_2, \alpha_3 = \text{Hash}(\text{公開入力} \parallel \text{Root_main})$
2. 統合多項式 $f_0(X) = \alpha_0 D_0(X) + \alpha_1 D_1(X) + \alpha_2 D_2(X) + \alpha_3 D_3(X)$
3. FRI段階的次数削減: $f_0 \rightarrow f_1 \rightarrow f_2 \rightarrow \dots \rightarrow C$ (定数)
4. 各段階で Merkle Root生成: $\text{Root}_0, \text{Root}_1, \dots, \text{Root}_{\{n-1\}}$

Phase 3: Query Response Generation

1. クエリ点 $x = \text{Hash}(\text{全transcript})$ を決定
2. 必要な評価値と Merkle Pathを計算
3. 証明データとして出力

STARKプロトコル まとめ

証明生成プロセス

Phase 1: AIR + Merkle Commitment

- 計算トレースから多項式 $P(X)$ を構築
- 制約条件から商多項式 $D_0(X), D_1(X), D_2(X), D_3(X)$ を計算
- 全評価値を Merkle Treeにコミット → Root_main

Phase 2: FRI Proof Generation

- ランダム係数 $\alpha_0, \alpha_1, \alpha_2, \alpha_3 = \text{Hash}(\text{公開入力} \parallel \text{Root_main})$
- 統合多項式 $f_0(X) = \alpha_0 D_0(X) + \alpha_1 D_1(X) + \alpha_2 D_2(X) + \alpha_3 D_3(X)$
- FRI段階的次数削減: $f_0 \rightarrow f_1 \rightarrow f_2 \rightarrow \dots \rightarrow C$ (定数)
- 各段階で Merkle Root生成: $\text{Root}_0, \text{Root}_1, \dots, \text{Root}_{\{n-1\}}$

Phase 3: Query Response Generation

- クエリ点 $x = \text{Hash}(\text{全transcript})$ を決定
- 必要な評価値と Merkle Pathを計算
- 証明データとして出力

検証プロセス

Step 1: 公開検証の実行

- 証明データから各段階の Merkle Rootを抽出
- 同じハッシュ処理でランダム値 $\alpha_0, \alpha_1, \alpha_2, \alpha_3, \beta_i$, クエリ点 x を再生成

Step 2: 制約検証

- FRIの一貫性検証: 各段階で $f_{i+1}(s_{i+1}) = E_i(s_{i+1}) + \beta_i \cdot O_i(s_{i+1})$
- 統合多項式の正当性:

$$f_0(x) = \alpha_0 D_0(x) + \alpha_1 D_1(x) + \alpha_2 D_2(x) + \alpha_3 D_3(x)$$
- 元制約の成立確認: $C_i(x) = D_i(x) \times Z_i(x)$

演習問題

個人課題

- 1) STARKにおいては、多項式に関する低次元テスト(LDT, p.39)が必要ですが、これまでみてきたSNARKでは不要でした。**なぜSTARKではLDTが必要で、Groth16とPLONKではそれぞれ不要なのか**を数式をまじえて説明してみましょう。
- 2) スライドで説明されている構築方法では、STARKにゼロ知識性が付与されていません。**STARKをzkSTARKに変換する方法**を1つ取り上げ、**なぜその方法がゼロ知識性を付与するのか**を数式ベースで説明してみましょう。

演習問題

グループ課題

- 3) zkSNARK・zkSTARK(ゼロ知識性のないSNARK・STARKはNG)が利用されている具体的なプロトコル・サービスを一つ取り上げ、以下の項目に沿って分かりやすく説明してみましょう(公式ドキュメント等に記載がなければ推測しましょう)。
- a) プロトコル・サービスの目的と大まかな仕組み
 - b) aにおいて、誰が誰に対して何に関する証明を生成しているのかとそれをゼロ知識証明で行う必要性
 - c) 利用しているゼロ知識証明種別(*i.e.*, zkS{N||T}ARK)と具体的なプロトコル(*e.g.*, Groth16)、その採択背景・理由
 - d) cに関する特筆すべき最適化手法の詳細(数式レベルで解説)

參考資料

- <https://zenn.dev/qope/articles/8d60f77e3a7630>
- https://en.wikipedia.org/wiki/Merkle_tree
- <https://www.momoyama-usagi.com/entry/math-seigyo14>
- <https://www.youtube.com/watch?v=XDx4YnqodU0>

Thank you!

