Succinct Boot Camp: Proof Systems and Arithmetization

Lecture Notes

September 17, 2024

1 Introduction to Proof Systems

1.1 What is a Proof?

An interactive proof is an algorithm that allows two parties, a prover $\mathcal{P}$ and a verifier $\mathcal{V}$, to engage in a dialogue where $\mathcal{P}$ attempts to convince $\mathcal{V}$ of the truth of a claim. This process is interactive, meaning the two parties can exchange messages back and forth.

Definition 1 (Claim)**.** A claim in this context is represented by a set of valid pairs (X, W), where:

- X is the common input that both the prover and verifier have access to.
- W is the witness, which is additional information that only the prover possesses.

The set of all valid (X, W) pairs is called the validity relation.

The goal of the prover is to convince the verifier that for a given X, they know a corresponding W such that (X, W) is in the validity relation.

1.2 Examples of Validity Relations

To illustrate the concept of validity relations, consider the following examples:

1. **Fibonacci Relation:** Given a number n, define the relation as (A, W) where $A =$ Fibonacci(n). The prover aims to convince the verifier that A is indeed the n-th Fibonacci number.
2. **Addition Relation:** Given three numbers a, b, and c, define the relation $((a, b, c), W)$ as valid if $a = b + c \mod 2^{32}$. This relation proves that a is the sum of b and c in 32-bit arithmetic.

3. **Secret Key Relation:** Given a public key P and a secret key α, the pair (P, α) is valid if α is the correct secret key corresponding to P. This relation is fundamental in cryptographic protocols.

4. **SP1 Relation:** Given a program P, input I, and output O, the pair $((P, I, O), W)$ is valid if running program P with input I produces output O. This relation is crucial for proving correct execution of programs.

1.3 Properties of Interactive Proofs

Interactive proof systems have two fundamental properties:

1. **Completeness:** For all honest provers $\mathcal{P}$ and valid pairs (X, W), the verifier $\mathcal{V}$ accepts. This means that if the prover is honest and the claim is true, the verifier will be convinced.

2. **Soundness:** For every non-valid pair (X, W') and any potentially malicious prover $\mathcal{P}'$, the probability that the verifier $\mathcal{V}$ accepts is negligibly small (typically less than 2^{-100}). This property ensures that it's extremely difficult for a prover to convince the verifier of a false claim.

Additionally, a protocol is called *public coin* if the verifier's messages primarily consist of random values. This property is important for certain proof constructions and security arguments.

1.4 Merkle Trees: A Blockchain Example

To illustrate a practical application of proof systems, consider the following blockchain example: Suppose we have a list of account balances $(i, b_i)_{i=1}^n$, where i is the account number and b_i is the balance of account i. We want to prove that a specific account (say, account 125) has a particular balance (say, 1000) without requiring the verifier to store the entire list of balances. We can use a Merkle tree structure:

1. Compute the hash $h(b_i)$ for each balance b_i.

2. Combine hashes pairwise: $h(h(b_i), h(b_{i+1}))$.

3. Continue this process until we reach a single root hash.

Here's an illustration of a simple Merkle tree:

The prover can then provide a Merkle proof, which consists of the necessary hashes along the path from the leaf (the balance of account 125) to the root. For example, to prove the value of b_2, the prover would provide:

- The value of b_2

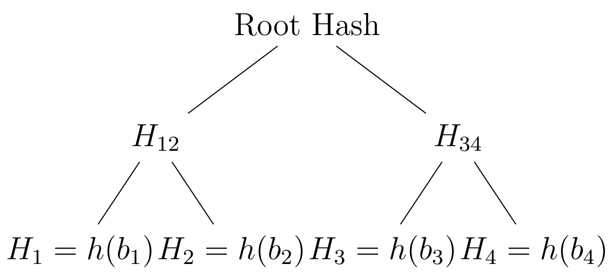

Figure 1: Simple Merkle Tree with 4 Leaf Nodes

- H_1 (the sibling hash)
- H_{34} (the hash of the other branch)

The verifier, who only needs to store the root hash, can efficiently verify the claim by recomputing the path using the provided hashes:

1. Compute $H_2 = h(b_2)$
2. Compute $H_{12} = h(H_1||H_2)$
3. Compute Root $= h(H_{12}||H_{34})$
4. Check if the computed root matches the stored root hash

This method allows for efficient verification of membership in a large set, which is crucial for blockchain and other distributed systems. The Merkle tree structure reduces the verification complexity from linear (in the number of accounts) to logarithmic, as the verifier only needs to check a number of hashes equal to the height of the tree.

2 Arithmetization

Arithmetization is the process of encoding computations as polynomial equations, which allows us to create efficient proofs for complex computations.

2.1 Finite Fields

We work over a finite field $\mathbb{F}_p$, where p is a prime. The field consists of elements $\{0, 1, ..., p-1\}$ with addition and multiplication defined modulo p.

Proposition 1. *For any non-zero $a \in \mathbb{F}_p$, there exists a unique $c \in \mathbb{F}_p$ such that $a \cdot c \equiv 1 \pmod{p}$.*

This c is called the multiplicative inverse of a and can be computed as $a^{p-2} \pmod{p}$.

2.2 Types of Arithmetization

We consider two main types of arithmetization:

1. **Arithmetic Circuits:** These are computational graphs where the operations are limited to addition and multiplication over the field.

2. **Algebraic Intermediate Representation (AIR):** This consists of:

   - A table T of size $n \times m$
   - Public outputs
   - Constraints:
     - First row constraint: $P_f(T_{1,1}, ..., T_{1,m}) = 0$
     - Last row constraint: $P_l(T_{n,1}, ..., T_{n,m}) = 0$
     - Transition constraints: $P_t(T_{i,1}, ..., T_{i,m}, T_{i+1,1}, ..., T_{i+1,m}) = 0$ for $1 \leq i < n$

   All constraint polynomials have degree less than 3 in SP1.

2.3 Fibonacci AIR Example

To illustrate AIR, let's encode the Fibonacci sequence computation:

- First row: $a = 0$, $b = 1$
- Transition rows: $a' = b$, $b' = a + b$
- Last row: $a = c$ (public output)

2.4 Reducing Multiple Equations to One

To simplify verification, we can use randomness to combine multiple equations into one:

Theorem 1. *Given equations $a = 0$ and $b = 1$, these are equivalent to proving $a + r(b - 1) = 0$ for a random $r \in \mathbb{F}_p$, with soundness error $\frac{1}{p}$.*

This technique allows us to combine multiple constraints into a single polynomial equation, significantly reducing the verification complexity.

2.5 Polynomial Encoding

We can encode the entire computation table as a set of polynomials:

Theorem 2. *Let $n = 2^d$ and $g \in \mathbb{F}_p$ such that $g^{2^d} = 1$. For each column j, there exists a unique polynomial $\tilde{T}_j(x)$ of degree $2^d - 1$ such that $\tilde{T}_j(g^i) = T_{ij}$ for all $0 \leq i < n$.*

This allows us to represent the entire computation as a polynomial equation:

$$\tilde{P}(x) = P(\tilde{T}_1(x), ..., \tilde{T}_m(x)) = 0 \quad \forall x \in \{1, g, ..., g^{2^d - 1}\} \tag{1}$$

2.6 Final Polynomial Equation

The AIR claim can be reduced to proving the existence of a polynomial $Q(x)$ such that:

$$\tilde{P}(x) = Q(x)(x^{2^d} - 1) \tag{2}$$

This equation can be verified probabilistically by checking it at a random point $r \in \mathbb{F}_p$:

$$\tilde{P}(r) = Q(r)(r^{2^d} - 1) \tag{3}$$

This verification has a soundness error of approximately $\frac{\deg(P)\cdot 2^d}{p}$.

3 Polynomial Commitment Schemes

To complete the proof system, we need a way for the prover to commit to polynomials and prove evaluations. This is achieved through polynomial commitment schemes.

3.1 FRI (Fast Reed-Solomon Interactive Oracle Proof of Proximity)

FRI is one example of a polynomial commitment scheme. It allows the prover to commit to a polynomial $T(x)$ and later prove its evaluation at specific points.

The basic idea of FRI involves:

1. Evaluating the polynomial on a larger domain (typically twice the size of the degree).
2. Creating a Merkle tree of these evaluations.
3. Using interactive protocols to prove consistency and degree bounds.

FRI achieves logarithmic-time verification while maintaining soundness, making it particularly suitable for zk-SNARKs and other succinct proof systems.

4 Conclusion

Arithmetization and polynomial commitment schemes form the backbone of modern succinct proof systems. By encoding computations as polynomial equations and leveraging the properties of finite fields, we can create highly efficient and verifiable proofs for complex computations. These techniques have far-reaching applications in cryptography, blockchain technology, and verifiable computation.