Succinct Arguments for Circuit Sat, Merkle Trees and Polynomial Commitment Schemes

Alexander Lindenbaum

Notes are from Chapter 7 of [Tha22].

1 Arithmetic Circuit Satisfiability

In the arithmetic circuit satisfiability problem, you have a fixed circuit $\mathcal{C}(x, w)$. with inputs in $\mathbb{F}^n$. Given x and an output(s) y, the problem is to find w so that $\mathcal{C}(x, w) = y$. We care about this problem because arithmetic circuits encode time-bounded computation. The problem of, whether a TM M on input x outputs y in time $\leq T$ can be reduced the arithmetic circuit satisfiability, where $\text{size}(\mathcal{C}) \approx T$ and $\text{depth}(\mathcal{C}) \approx O(\log T)$. Arithmetic circuits also encode commonly used circuits like SHA-256, which we would want to prove we have a w such that SHA-256$(w) = y$.

[GKR08] gives an IP for evaluating a circuit $\mathcal{C}$ on an input. In GKR, the prover runs in time polynomial in the size of $\mathcal{C}$, and the verifier runs in $\text{polylog}(n + \text{depth}(\mathcal{C}) \log(\text{size}(\mathcal{C}))$.

A naive IP for circuit satisfiability: $\mathcal{P}$ sends w such that $\mathcal{C}(x, w) = y$. Then $\mathcal{P}$ and $\sqsubseteq$ simulate GKR on (x, w).

But this is not a "succinct" IP, because $\mathcal{P}$ has to send a proof of size $|w|$. We are shooting for total communication which is sublinear in $|w|$. Without going into too much detail for GKR, the way to get a shorter proof is to have $\mathcal{P}$ commit to a low-degree polynomial $\tilde{w}(x)$, and $\mathcal{V}$ only needs to ask for its evaluation on a single field element.

2 Polynomial Commitment Schemes

2.1 (String) Commitment Schemes

A sender has a message $m \in \Sigma^n$ which they want to commit to. Sender uses randomness to generate and send a commitment string com_m to receiver. When the receiver wants to "open" the commitment, they ask for the random bits and run a verification algorithm on com_m, m and the random bits. A proper commitmnent scheme holds two properties:

1. Hiding: (t, ε)-hiding: for all messages m and m' of the same size, the distributions $\text{com}(m, r)$ and $\text{com}(m', r)$ are (t, ε)-indistinguishable, where the distributions are over the random bits r.

2. Binding: (this is "perfect binding") for all messages m and random bits r, r',

$$\text{ver}(\text{com}(m, r), r')$$

is either m itself or "FAIL".

One way to obtain a commitment scheme is by using a collision-resistant hash function (CRHF) $H : \mathcal{M} \times \mathcal{R} \to \mathcal{C}$. We would have:

$$\text{com}(m, r) = H(m, r),$$
$$\text{ver}(m, \text{com}, r) = 1 \iff \text{com} = H(m, r).$$

(The form of the verification function is slightly different here from above. These forms are all equivalent.)

2.2 Functional Commitment Schemes

Fix a function family $\mathcal{F} = \{f : \mathcal{X} \to \mathcal{Y}\}$. For example, $\mathcal{F}$ could be $\mathbb{F}_p^{(\leq d)}[X]$, the set of all univariate polynomials of degree $\leq d$, over a prime field. Or $\mathcal{C}_s$, the set of all arithmetic circuits of size $\leq s$. The idea is now the sender "commits to a function" $f \in \mathcal{F}$. The reciever can query the sender on a particular input $x \in \mathcal{X}$, and can verify that the response y is indeed the output of f on input x.

[KZG10] gives a formal definition of a polynomial commitment scheme, and provides constructions based on the discrete log, and elliptic curves and pairings. I will only go over the definition informally.

Definition 1. A *polynomial commitment scheme* is six algorithms: Setup, Commit, Open, VerifyPoly, CreateWitness, and VerifyEval:

- Setup($1^\lambda, t$) generates an appropriate algebraic structure $\mathcal{G}$ and a public key-private key pair (pk, sk) to commit to a polynomial of degree $\leq d$.
- Commit(pk, $\phi(x)$) belongs to $\mathcal{P}$. Commit outputs a commitment $\mathcal{C}$ to a polynomial $\phi(x)$ for public key pk, and also outputs decommitment information d.
- Open(pk, $\mathcal{C}$, $\phi(x)$, d) belongs to $\mathcal{P}$. At request of $\mathcal{V}$, opens the commitment to $\phi(x)$ by simply sending $\phi(x)$ (in case we study the adversarial situation, Open reveals some other polynomial).
- VerifyPoly(pk, $\mathcal{C}$, $\phi(x)$, d) belongs to $\mathcal{V}$. Verifies that $\mathcal{C}$ is indeed a commitment to $\phi(x)$.
- CreateWitness(pk, $\phi(x)$, i, d) belongs to $\mathcal{P}$. Outputs $\langle i, \phi(i), w_i \rangle$, where w_i is a witness for the evaluation of $\phi(i)$.
- VerifyEval(pk, $\mathcal{C}$, i, $\phi(i)$, w_i) belongs to $\mathcal{V}$. Verifies that $\phi(i)$ is indeed the evaluation of i on the polynomial commited to by $\mathcal{C}$.

2.3 An impractical polynomial commitment scheme

Given in [Tha22]. It is "impractical" because the prover has large runtime, but still worth studying. Uses Merkle Trees and a low-degree test, which we will cover.

Merkle Trees [Mer79] Merkle Trees (or hash trees) are immediately used for string commitment schemes. The idea is to construct a perfect binary tree bottom-up. Say $\mathcal{P}$ wants to commit to a string $s \in \Sigma^n$:

- Leaves of the tree: symbols s_i in s.
- Internal nodes: the hash of its two children.
- The final commitment string: the root of the tree.

Suppose $\mathcal{P}$ is asked to reveal s_i. $\mathcal{P}$ would send the value of s_i, every node along the root-to-leaf path for s_i, and all nodes which are siblings to nodes along that path. That way, $\mathcal{V}$ can verify the commited string by hashing themselves the correct pairs to go from s_i to the root. The depth of the tree is $O(\log n)$, meaning that the sender sends $O(\log n)$ hash values for each symbol revealed.

So how do we get a polynomial commitment scheme from a Merkle Tree? If $\mathcal{P}$ commits to a polynomial p, they commit to the string which prints all evaluations of p, $p(\ell_1), \cdots, p(\ell_N)$, where $\ell_1, \ldots, \ell_N$ form the domain of p. When $\mathcal{V}$ asks for $p(\ell_i)$, $\mathcal{P}$ the root-to-leaf path for $p(\ell_i)$ and all sibling nodes.

But this is not a correct polynomial commitment scheme, because $\mathcal{P}$ could simply commit to a polynomial which does not have degree $\leq d$, or any arbitrary function. The only guarantee that $\mathcal{V}$ will have is that $\mathcal{P}$ is commiting to the function they originally chose. For this, $\mathcal{V}$ needs to test that the function is of low-degree

Low-degree tests In this model, either the receiver has black-box query access to a function, or they are reading off a string s of concatenated evaluations. The function is m-variate, inputs over field $\mathbb{F}$. There are $|\mathbb{F}|^m$ possible inputs, so s contains a list of $|\mathbb{F}|^m$ elements of $\mathbb{F}$.

The problem: is s consistent with a polynomial of degree $\leq d$? We want to answer this problem without looking at too many bits in s. [Tha22] confirms that we have strong randomized tests for this. The procedures tend to be simple, but analyses are tricky (I suspect one of these methods could use Shwartz-Zippel Lemma).

Thus, we have a protocol for commiting to a low-degree polynomial: $\mathcal{P}$ merkle-hashes evaluations of p, and when $\mathcal{V}$ asks to reveal evaluations of p, they run a low-degree polynomial test.

References

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In *STOC'08*, pages 113–122. ACM, New York, 2008.

[KZG10] Aniket Kate, Gregory M Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *Advances in Cryptology-ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings 16*, pages 177–194. Springer, 2010.

[Mer79] Ralph Charles Merkle. *Secrecy, authentication, and public key systems.* Stanford university, 1979.

[Tha22] Justin Thaler. Proofs, arguments, and zero-knowledge. *Foundations and Trends in Privacy and Security*, 4(2–4):117–660, 2022.