A Survey to Zero-Knowledge Interactive Verifiable Computing: Utilizing Randomness in Low-Degree Polynomials

Angold Jiawei Wang
awang@weids.dev
jwwang4@comp.hkbu.edu.hk
Hong Kong Baptist University

December 2024

Abstract

A Survey to Zero-Knowledge Interactive Verifiable Computing: Utilizing Randomness in Low-Degree Polynomials

By

Jiawei Wang

This survey provides a comprehensive examination of zero-knowledge interactive verifiable computing, emphasizing the utilization of randomnes in low-degree polynomials. We begin by tracing the evolution of general-purpose verifiable computing, starting with the foundational concepts of complexity theory developed in the 1980s, including classes such as P, NP and NP-completeness. Through an exploration of the Cook-Levin Theorem and the transformation between NP problems like HAMPATH and SAT, we demonstrate the reducibility of NP problems to a unified framework, laying the groundwork for subsequent advancements.

Recognizing the limitations of NP-based proof systems in effectively verifying certain problems, we then delve into interactive proof systems (IPS) as a probabilistic extension of NP. IPS enhance verification efficiency by incorporating randomness and interaction, while accepting a small chance of error for that speed. We address the practical challenges of traditional IPS, where the assumption of a prover with unlimited computational power is unrealistic, and introduce the concept of secret knowledge. This approach allows a prover with bounded computational resources to convincingly demonstrate possession of secret knowledge to the verifier, thereby enabling high-probability verification by the verifier. We quantify this knowledge by assessing the verifier's ability to distinguish between a simulator and genuine prover, referencing seminal works such as Goldwasser et al.'s "The knowledge Complexity of Theorem Proving Procedures"

The survey further explores essential mathematical theories and cryptographic protocols, including the Schwartz-Zippel lemma and Reed-Solomon error correction, which underpin the power of low-degree polynomials in error detection and interactive proof systems. We provide a detailed, step-by-step introduction to tyhe sum-check protocol, proving its soundness and runtime characteristics.

Despite the sum-check protocol's theoretical applicability to all NP problems via SAT reduction, we highlight the sum-check protocol's limitation in requiring superpolynomial time for general-purpose computations of a honest prover. To address these limitations, we introduce the GKR protocol, a sophisticate general-purpose interactive proof system developed in the 2010s. We demonstrate how the sum-check protocol integrates into the GKR framework to achieve efficient, sound verification of computations in polynomial time. This survey not only reviews the historical and theoretical advancement in verifiable computing in the past 30 years but also offers an accessible introduction for newcomers by providing a solid foundation to understand the significant advancements in verifiable computing over the past decade, including developments such as ZK-SNARKs.

Contents

1 Preliminaries: Complexity Theory

1.1 P (Polynomial Time)

Polynomial time decidable languages [11] P is the class of languages that is decidable in polynomial time on a deterministic single-tape Turing machine. $P = U_k Time(n^k)$ where k is a constant.

1.2 NP (Nondeterministic Polynomial time)

In P, we can avoid brute-force search in many problems and obtain polynomiaIntroduction to the Theory of Computationl-time solutions. However, attempts to avoid brute force in certain other problems, including many interesting and useful ones, haven't been successful, and polynomial time algorithms that solve them aren't known to exist.
Languages View:
P = The class of languages which membership can be **decided** (solved) quickly. (polynomial time)
NP = The class of languages which membership can be **verified** quickly. (polynomial time)
TM view:
NP is the set of problems **solvable** in polynomial time by a **non-deterministic Turing machine.**
NP is the set of problems **verifiable** in polynomial time by a **deterministic Turing machine.**

1.2.1 NTM Decider

A Nondeterministic Turing Machine (NTM) decider is guaranteed to halt on all inputs. [11] An NTM decider is designed so that every possible computation branch halts, either by accepting or rejecting the input. This means that for any input the machine is given, it will always come to a conclusion (halt) within a finite number of steps. There are no branches where the machine runs forever without deciding the outcome.

1.2.2 Solving NP

The non-deterministic nature of NP gives us an abstraction to imagine a machine (NTM) that could guess a solution "in parallel" and verify it quickly. If we had such a machine, it would allow us to "solve" NP problems quickly by magically finding the right solution path. However, for real-world deterministic machines, we still don't have efficient algorithms to solve many NP problems.

1.2.3 Example: Hamiltonian Path Problem

If a directed or undirected graph, G, contains a Hamiltonian path, a path that visits every vertex in the graph exactly once. The HAMPATH problem has a feature called polynomial verifiiability that is important to understand its complexity. Verifying the existence of a Hamiltonian path may be much easier than determining its existence.
Theorem: HAMPATH belongs to NP:
On input $\langle G, s, t\rangle$ (Say G has m nodes): we non-derterministically write a sequence $v_1, v_2, ..., v_m$ of m nodes, and only accept if: a. $v_1 = s$ b. $v_m = t$ and c. each (v_i, v_{i+1}) is an edge and no v_i repeats.
We do not know whether the complement of HAMPATH (Co-HAMPATH) is NP since we do not know whether or not we can give a short certificate for a graph not have a Hamiltonian path.
If P equaled NP: Then we can test in polynomial time whether a graph has a Hamiltonian path by directly solving the problem, which yields a short certificate.
If P not equal to NP: then co-HAMPATH is not an NP problem, since it is not easily verified.

1.3 NP-Completeness

The Relationship between P and NP indicates that whether all problems can be solved in polynomial time, typically, without **searching**.

NP-completeness is a cornerstone concept in computational complexity theory, providing a framework for understanding the inherent difficulty of computational problems. Building upon the detailed exposition presented in Michael Sipser's Lecture Notes [12], this section delves into the foundational definitions, key theorems, and proof techniques that characterize NP-complete problems.

1.3.1 P $\equiv$ NP

You can always eliminate searching. If these classes were equal, any polynomially verifiable problem would be polynomially decidable.

1.3.2 P $\neq$ NP

There were cases where you need to search. [11] "Most researchers believe that the two classes are not equal because people have invested enormous effort to find polynomial time algorithms for certain problems in NP, without success. Researchers also have tried proving that the classes are unequal, but that would entail showing that no fast algorithm exists to replace brute-force search."
Defn: B is NP-complete if:

1. B is a member of NP
2. For all A in NP, A $\leq p$ B

Every language in NP has the polynominal time reduced to a complete language of NP, which means if B is NP-complete and B is in P then P = NP.

One important advance on the P versus NP question came in the early 1970s with the work of Stephen Cook and Leonid Levin.[2] which shows that SAT is NP-complete.
In general,**NP-completeness is a very important complexity property of any question**:

1. Showing NP-complete is strong evidence of computational intractability (hard).

2. Gives a good candidate for proving P $\neq$ NP.

Michael Sipser in 2020: [13] *"Back 20 years ago, I was working very hard to show the composites number problem is not in P. And then, turns out, composites was in P (proved by Agrawal, M., Kayal, N., & Saxena, [1] 2002). So it was the wrong to pick the composite number problem, but what NP-complete is guarentees is that: If you work on a problem, which is NP-complete, you can't pick the wrong problem, because if any problem is in NP and not in P, an NP-complete problem is going to be an example of that. Because if the NP-complete problems in P, everything in NP is in P."*

1.4 The 3SAT Problem

1.4.1 Conjunctive Normal Form (CNF)

A boolean formula ϕ is in Conjunctive Normal Form (CNF) if it has the form:

$$\phi = (x \vee \neg y \vee z) \wedge (\neg x \vee \neg s \vee z \vee u) \wedge ... \wedge (\neg z \vee \neg u)$$

- **Literal:** a variable $\neg x$ or a negated variable
- **Clause:** an OR of the literals.
- **CNF:** an AND of the clauses.
- **3CNF:** a CNF with exactly 3 literals in each clause.

1.4.2 SAT

Boolean satisfiability problem (SAT) is the problem of determining if there exists an interpretation that satisfies a given Boolean formula. In other words, it asks whether the variables of a given Boolean formula can be consistently replaced by the values (TRUE or FALSE) in such a way that the formula evaluates to TRUE.

1.4.3 3SAT is the satisfatory problem restrited to 3CNF formulars

$$3SAT = \{\phi \mid \phi \text{ is a satisfiable 3CNF formular}\}$$

1.4.4 Theorem: 3SAT $\leq p$ K-CLIQUE

We will show that we can reduce 3SAT to K-CLIQUE in polynominal time by *building a model on 3SAT*. And hence to show that K-CLIQUE problem is also NP-complete.

The K-Clique Problem

A k-clique in a graph is a subset of k nodes all directly connected by edges, the input of k-clique problem is an undirected graph and k. The output is a clique (closed) with k vertices, if one exists.

$$\text{K-CLIQUE} = \{\langle G, k\rangle \mid \text{ graph G contains a k-clique}\}$$

The K-Clique Problem is in NP

You can easily verify that a graph has a k-clique by exhibiting the clique.

Proof: 3SAT $\leq p$ K-CLIQUE[12]

Given polynomial-time reduction f that maps ϕ to $\langle G, k\rangle$ where ϕ is satisfiable if and only if G has a k-clique. Given the structure of a CNF, a satisfying assignment to a CNF formula has ≥ 1 true literal in each clause.

We will show that we can reduce 3SAT to K-CLIQUE in polynominal time by *constructing a model based on 3SAT* (adding rules).

$$\phi = (a \vee b \vee \neg c) \wedge (\neg a \vee b \vee d) \wedge (a \vee c \vee \neg e) \wedge ... \wedge (\neg x \vee y \vee \neg z)$$

- G: Assume each literal in the formula is a node in G, where:
  - The forbidden edges:
    1. No edges within a clause.
    2. No edges that go between inconsistent labels (a and $\neg a$)
  - G has all non-forbidden edges.
    * k is the number of clauses
    * Other than those forbidden edges, all other edges are connected.

Claim: ϕ **is satisfiable if and only if (iff)** G **has k-clique**.

- $\rightarrow$ **Proof: If ϕ is satisfiable then G has a k-clique.**
  - Taking any satisfying assignment to ϕ. Pick 1 true literal in each clause.

    Assuming that we can find that satisfying assignemnt (that 3SAT is solvable).
  - Then the corresponding nodes in G are a k-clique:
    1. No forbidden edges among them

       Because based on our rules of the model, those nodes on different clauses have edges connected.
    2. No chosen nodes between inconsistent labels (e.g., a, $\neg a$)

       Because they all came from the same assignment. (a is true then $\neg a$ is false, we cannot pick them both in different clauses)
- $\leftarrow$ **Proof: If G has a k-clique then it will make ϕ satisfiable**.
  - Taking any k-clique in G. It must have 1 node in each clause.
    * It cannot have 2 nodes or 0 nodes in the same clause.

    Because when we construct 3SAT from given G, nodes cannot appear in a clique together, since there are k clauses, each clause must have exactly one node to form a k-clique graph.

Setting each corresponding literal TRUE gives a satisfying assignment to ϕ.

Since the the reduction f is computable in polynominal time. which suggest that: **If k-clique can be solved in polynominal time, then 3SAT can be solved in polynominal time.**

Conversely, a polynomial-time solution to 3SAT implies that all NP problems, including K-clique, are in P.

1.5 The Cook-Levin Theorem

Once we have one NP-complete problem, we may obtain others by polynomial time reduction from it, as we've seen in K-CLIQUE and HAMPATH. However, establishing the first NP-complete problem is more difficult. Now we do so by proving that SAT is NP-complete. In 1971, Stephen Cook states that the **Boolean satisfiability problem is NP-complete** [2]. That means any problem in NP can be reduced in polynomial time by a deterministic Turing machine to the Boolean satisfiability problem.

1.5.1 SAT is in NP

A nondeterministic polynomial time machine can guess an assignment to a given formula ϕ and accept if the assignment satisfies ϕ.

1.5.2 For each A in NP, we have A $\leq p$ SAT:

(Any language in NP is polynomial time reducible to SAT).

1. **Proof Idea:** [11]

   Let N be a nondeterministic Turing machine that decides A in n^k time for some constant k. *We are trying to proof that for any w belongs to any NP problems, there is a polynominal time reduction procedure that can transform that w to $\phi(SAT)$*

2. Key to the Proof:

   For any w belongs to any NP problems, it can be determinned in polynomial time by a nondeterministic Turing machine N, say the running time is n^k. Then we can construct a Tableau for N is an $n^k \times n^k$ table whose rows are the configurations of a branch of the computation of N on input w. Which represents the computation steps/history of that branch of NTM (N). Based on this Tableau, by carefully define each part of *Phi*:

   $$\phi = \phi_{cell} \wedge \phi_{start} \wedge \phi_{move} \wedge \phi_{accept}$$

   We can show that the construction time is is $O(n^{2k})$, the size of ϕ is polynomial in n. Therefore we may easily constract a reduction that produces *Phi* in polynomial time from the input w of any NP problem.

2 Interactive Proof Systems

2.1 The Limitation of NP Proof Systems

Let's recall Stephen Cook and Leonid Levin's influential model definition of NP: [2] The NP proof-system consists of two communicating deterministic Turing machines A and B: respectively, the *prover* and the *verifier*. Where the *prover* is **exponential-time**, the *verifier* is **polynomial-time**. They read a common input and interact in a very elementary way. On input a string x belonging to an NP language L, A computes a string y (whose length is bounded by a polynomial in the length of x) and writes y on a special tape that B can read. B then checks that $f_l(y) = x$ (where f_l is a polynomial-time computable function) and, if so, it halts and accepts.

2.1.1 $f_l(y) = x$

We can understand $f_l(y) = x$ in this way: "The output (certificate) y belongs to the input x, where $f_l()$ is a function that can check that y in poly-time."

2.1.2 Formalization vs. Intuition

Sometimes formalization cannot entirely capture the inituitive notions. In the context of **theorem-proving**: NP captures a specific form of proving a theorem, where a proof can be "written down" and verified without interacting with the prover. The certificates, which is like a formal written proof, and the verifier just passively checks it. This is like reading a proof in a book. Once you have the book, there is no back-and-forth to clarify or ask questions about the proof.

2.1.3 Example: Co-HAMPTATH Problem

As we mentioned in the previous section, we do not know whether the complement of HAMPATH (Co-HAMPATH) is in NP:

- y is easy to be verified: **HAMPATH**

  For the Hamiltonian Path (HAMPATH) problem, given a solution (i.e., a path), it's easy for a verifier to check it in polynomial time.

  The prover can just present the path (certificates, or y), and the verifier checks whether it's a valid Hamiltonian path (i.e., visits each vertex exactly once and satisfies the graph's edges).

- y is hard to be verified: **Co-HAMPATH**

  The Co-HAMPATH problem asks whether a graph does not have a Hamiltonian path. Here, proving the non-existence of something becomes far more complex.

  If you ask a prover to convince you that no Hamiltonian path exists, the proof isn't as simple as just pointing to something (like a path). Instead, you'd need to somehow verify all possible paths don't work, which could take **exponential time**.

2.1.4 Limitation of the NP Proof-System

In the NP model, some problems in NP (like HAMPATH) are easily verifiable, but NP does not capture the complexity of some other problems (often their complements i.e., Co-NP problems).

That's why problems like Co-HAMPATH are much harder to verify using the static NP model: **In our example, there isno easy way for a prover to present a simple "proof" that no Hamiltonian path exists, and for the verifier to check it efficiently.**

2.2 The Interactive Proof Systems

In 1985, Goldwasser et al. [6] introduced an interactive proof-systems to capture a more general way of communicating a proof.

Much like in computation, BPP [11, Section 10.2.1: The class BPP, pp. 336–339] (Bounded-error Probabilistic Polynomial time) algorithms provide a probailistic analog to P to enhance efficiency while accepting a small chance of error for that speed.

In verification, **IP (Interactive Proof) systems provide a way to define a probabilistic analog of the class NP**. IP includes problems not known to be in NP, demonstrating greater verification power due to *randomness* and *interaction.*

2.2.1 Interactive Pairs of Turing Machines

- **Prover (P)**

  An entity with unlimited computational power, aiming to convince the verifier the truth of a statement.

- **Verifier (V)**

  A probabilistic polynomial-time Turing machine (with a random tape) that interacts with the prover to verify the statement's validity.

2.2.2 Interactions

The interaction consists of multiple rounds where the prover and verifier exchange messages. The verifier uses randomness to generate challenges, and the prover responds accordingly. The key properties of such systems are:

- **Completeness**

  If the statement is true, an honest prover can convince the verifier with high probability.

- **Soundness**

  If the statement is false, no cheating prover can convince the verifier except a small probability.

2.2.3 Example: Quadratic Nonresidue Problem

An integer a is a quadratic residue modulo n if there exists an integer x such that:

$$x^2 \equiv a \pmod{n}$$

An integer a is a quadratic nonresidue modulo n if no such integer x exists. Suppose A (Prover) claim that a is a **quadratic nonresidue**, and the B (Verifier) wants to check that using an **Interactive Proof System**.

An Interactive Proof System to the QAP can be: The verifier B begins by choosing m random numbers $\{r_1, r_2, ..., r_m\}$. For each i, $1 \leq i \leq m$, he flips a coin:

- If it comes up heads he forms $t = r_i^2 \pmod{m}$.
- If it comes up tails he forms $t = a \times ri^2 \pmod{m}$.

Then B sends $t_1, t_2, ..., t_m$ to A, the prover, who having unrestricted computing power, finds which of the t_i are quadratic residues, and uses this information this information to tell B the results of his last m coin tosses. If this information is correct, B accepts.

Why this Will Work?

- If a is really a quadratic nonresidue: According to the property of quadratic nonresidue:
  - $t = a \times r_i^2 \pmod{m}$ is a quadratic non-residue.
  - $t = r_i^2 \pmod{m}$ is a quadratic residue.

  The prover can distinguish which side of the coin by looking whether t is a quadratic nonresidue or residue.

- If a is a quadratic residue (Prover is lying): Then both $t = a \times r_i^2 \pmod{m}$ and $x = r_i^2 \pmod{m}$ are quadratic residues.

  Which means the t_i are just random quadratic residues, all the t_i looks the "same" for the prover to guess the coin side, the prover will respond correctly in the last part of the computation with probability $1/2^m$.

3 Knowledge Complexity

By assuming the prover has unlimited computing power, The theoritical model introduced in Interactive Proof Systems can describe many languages that cannot be captured using NP model. But back to practice, seems that this model will only work in theory until those kinds of unlimited computing machine comes in real life. (Can determine NP problems in Polynominal Time). (i.e., Can quickly determine NP problems like whether t is a quadratic nonresidue or residue) Is that true?

3.1 Secret Knowledge

It is true that having unlimited computing machine is infeasible in current practice. However, by assuming the prover runs in polynomial time with some "secret knowledge" that can help it communicating with verifier efficiently. It can convince the verifier that the prover has that "secret knowledge" without revealing it.

3.1.1 Eyewitness & Police Officer

Let us try to illustrate the above ideas using an informal example: Assume that a crime x has happened, B is a police officer and A is the only eyewitness. A is greedy that he tells B that in order for him to tell about what happened in x, \$100,000 must be transferred to his bank account first. For B, it is important to verify whether A has that "secret knowledge" – details of crime x before making transfer. And for obvious reasons, A cannot just prove that he has that "secret knowledge" by telling it directly to the police officer B. By using an interactive proof systems, A can convince B he has that "secret knowledge" x without revealing it.

3.1.2 Quadratic Nonresidue Problem

In the interactive proof system describe in Quadratic Nonresidue Problem, the key challenge for the Prover (A) is to determine whether each number t_i sent by the verifier (B) is a quadratic residue or a quadratic nonresidue modulo m. This determination is crucial because it allows the Prover to infer the results of B's coin tosses and respond correctly. In this case, the "secret knowledge" for efficient computation on A is the **prime factorization of the modulus** m[1]. If the prover knows the prime factors of m, they can efficiently compute the *Legendre*[2] or *Jacobi*[3] symbols to determine quadratic residuosity. This "secret knowledge" enables polynomial prover (A) to interact with B that are otherwise computationally infeasible.

$$(\text{"secret knowledge"} + \text{poly-time machine} \equiv \text{unlimited computing power})$$

[1]Note: In practice, the modulus m is often chosen such that its factorization is hard to obtain (e.g., a product of two large primes), ensuring that without the secret knowledge, determining quadratic residuosity remains difficult.

[2]Legendre, A. M. (1798). Essai sur la théorie des nombres. Paris. p. 186.

[3]Jacobi, C. G. J. (1837). "Über die Kreisteilung und ihre Anwendung auf die Zahlentheorie". Bericht Ak. Wiss: 127–136.

3.2 The Knowledge Complexity

3.2.1 The Knowledge Computable from a Communication

Which communications (interactions) convey knowledge? Informally, those that transmit the output of an unfeasible computation (we cannot perform ourselves).

3.2.2 Knowledge Complexity

How to proof & ensure that the verifier gains no additional (secret) knowledge beyond the validity of the statement being proven?

Simulator

To formalize this, Goldwasser et al.[6] introduced the idea of a **simulator**: – An algorithm that can generate transcripts of the interaction without access to the prover's secret information. The key is that the verifier cannot distinguish between transcripts generated by interacting with the actual prover and those produced by the simulator.

Quantifying Knowledge

By linking the **chance** that the verifier able to distinguish the simulator, we quantify the knowledge complexity of proofs.

$\rightarrow$ If the simulator can effectively replicate the interaction, making it indistinguishable to the verifier, **the knowledge complexity** is considered zero.

This formalization allows us to assess and prove the zero-knowledge property of certain interactive proofs. $\rightarrow$ Show that sometimes the verifier cannot gain knowledge because whatever it sees could be simulated without the prover's help.

The Simulator acts as an algorithm that can generate transcripts of the interaction between the prover (A) and verifier (B) without **knowing the prover's secret knowledge**. By producing transcripts that are indistinguishable from those of a real interaction, the simulator demonstrates that the verifier gains no additional knowledge from the interaction. Therefore, if such a simulator exists, we say that the proof is zero-knowldge because any information the verifier receives could have been simulated without the prover's secret.

3.2.3 Zero Knowledge Interactive Proof System for the Quadratic Residuosity Problem

[6, Section 4.2] introduces a zero knowledge IP system by carefully designing the protocol and demostrating the existence of a poly-time simulator.

The difficulties of the proof is that M must compute the coin tosses correctly as a real prover (A) with secrect knowledge does.

Since the simulator M simulates both sides of the interaction, it both can know/control the randomness of the coin.

4 The Power of Low-Degree Polynomials

This section of the survey builds upon the concepts presented by Justin Thaler [15] during the Proofs, Consensus, and Decentralizing Society Boot Camp in 2019. Thaler's insightful discussion on the power of low-degree polynomials in verifiable computing serves as the backbone for the detailed explanations and proofs provided herein.

4.1 Example: Equality Testing

Two parties (i.e., Alice and Bob) each have an equal-length binary string:

$$a = (a_1, a_2, ..., a_n) \in \{0,1\}^n \mid b = (b_1, b_2, ..., b_n) \in \{0,1\}^n.$$

They want to collaborate with each other (No malicious user) to determine whether $a \equiv b$, while exchanging as few bits as possible.

4.1.1 A trivial solution

Alice sends a to Bob, who checks whether $a \equiv b$. The communication cost is n, which is optimal amongst deterministic protocols.

4.1.2 A logarithmic cost randomized solution

According to [14, Section 2.3], let F be any finite field with $|F| \geq n^2$, then we interpret each a_i, b_i as elements of F: Let $p(x) = \sum_{i=1}^{n} a_i \times a^i$ and $q(x) = \sum_{i=1}^{n} b_i \times b^i$

1. Alice picks a random r in F and sends $(r, p(r))$ to Bob.
2. Bob calculates $q(r)$ and outputs EQUAL iff $p(r) \equiv q(r)$, otherwise he outputs NOT-EQUAL.
    - Total Communication Cost: $\mathcal{O}(\log n)$ bits

      Since there are at least total n^2 elements in F, to represent any of each elements, we need $\log(|F|) = \log(n^2) = 2\log(n) = \mathcal{O}(\log n)$ bits.
3. If $a \equiv b$: Then Bob outputs EQUAL with probability 1
4. If $a \neq b$: Then Bob outputs NOT-EQUAL with probability at least $(1 - 1/n)$ over the choice of r in F.

A detailed proof of this statement will be given in the next subsection: Low-degree Polynomials.

4.2 Low-degree Polynomials

4.2.1 Field

Field [3] arises from the need for a structured and versatile system in mathematics and science to perform algebraic operations in a consistent and predictable way.

A field is a set equipped with two operations, addition and multiplication, along with their respective inverses, subtraction and division (except division by zero). Operations in a field follow specific rules, such as commutativity, associativity, and distributivity, ensuring that the result of any operation between elements of the field remains within the field itself (closure). Here are some mon-field examples:

- The set of 2x3 matrix cannot perform multiplication.
- The set of 2x2 matrix, the multiplication between any two elements is not commutative.
- Some of the elements in the set of Z/6Z (integers modulo 6) do not have multiplicative inverses.

  A number a in Z/6Z has a multiplicative inverse if there exists a number b in Z/6Z such that $a \times b \equiv 1 \pmod{6}$. For example, 2, 3, 4 do not have a multiplicative inverse, in fact, integers mod p (Z/pZ) is a field when p is a prime number.

4.2.2 Reed-Solomon Error Correction

Since there are total n bits of a and b, the lowest-degree polynomials that for us can ensure the uniqueness of representation of each a_i and b_i is n. Which means each bit a_i (and correspondingly b_i) is uniquely represented as the coefficient of a distinct term in the polynomial and affects the polynomial differently. And that is why we need to define $p(x)$ and $q(x)$ in the following way for error detection in equality testing:

$$p(x) = \sum_{i=1}^{n} a_i \times a^i$$

$$q(x) = \sum_{i=1}^{n} b_i \times b^i$$

4.2.3 Proof: Any non-zero polynomials $d(x)$ of degree n has at most n roots

Assume the polynomials has more than n roots. Let $r_1, r_2, ..., r_{n+1}$ be distinct elements of the field F, such that $d(r_i) = 0$ for each $i = 1, 2, ..., n+1$. Then the polynomail d(x) can be written as:

$$d(x) = (x - r_1)(x - r_2)...(x - r_{n+1})q(x)$$

where $q(x)$ is some polynomial of degree $m \geq 0$ and $(x - r_i)$ are factors corresponding to the roots.

Then the product of $(x - r_1)(x - r_2)...(x - r_{n+1})$ is a polynomial of degree $n+1$. This assumption leads to a contradiction. Hence the polynomial $d(x)$ can have at most n distinct roots.

4.2.4 Proof: In Equality Testing, if $a \neq b$, then the probability of Bob is wrong is $1/n$

In equality testing, if $p(x) \neq q(x)$, then the chance that Alice picks a random r in F such that $d(r) = p(r) - q(r) \equiv 0$ is at most $(n/|F| \leq n/n^2 \leq 1/n)$.

The reason is that a n-degree polynomial $d(r) = p(r) - q(r)$ has at most n roots, a randomly picked value r in F of size n^2 will only let $d(r)$ equal to zero with a probability $(n/n^2 = 1/n)$.

4.2.5 Polynomials are Constrained by Their Degree

As we can see that, A polynomial $d(x)$ of degree n over a large field F is uniquely determined by its values on $n+1$ distinct points. (one extra constant coefficient with no variable)

If $p(x)$ is not equal to $q(x)$, then they can only agree on at most n points ($d(x) = p(x) - q(x) = 0$), meaning they **differ at most everywhere** on the field. This strong divergence is very useful and powerful for error detection.

The Power of Low-degree In practice, we aim to keep the degree of a polynomial as low as possible while ensuring that each term uniquely affects the polynomial.

A polynomial of degree n has $n+1$ terms, each with a distinct power of x and a unique coefficient, which ensures that every term influences the polynomial differently.

Low-degree polynomials are powerful because if two polynomials differ, they will diverge over many points when evaluated multiple times with efficient evaluation. This makes discrepancies clear across a larger number of evaluations.

In cryptography, this property allows for efficient detection of errors or differences, especially when random evaluation are used.

- Introducing randomness amplifies this power by preventing predictable evaluations, making it harder to cheat or hide discrepancies.

- Random evaluations allow us to verify claims efficiently and securely, ensuring robustness.

  By using nondeterministic inputs in low-degree polynomials, **we combine the precision of low-degree polynomials with the unpredictability of randomness**.

4.3 Example: Freivalds's algorithm for Verifying Matrix Products

Input are two $(n \times n)$ matrices A, B. The goal is to verify the correctness of $A \cdot B$. The time complexity of matrix multiplication is $\mathcal{O}(n^3)$, this is because each element in the resulting matrix $A \cdot B$ is computed by taking the dot product of a row of A and a column of B. For each of the n^2 elements in the resulting matrix, you perform n multiplications and additions, leading to a total of $\mathcal{O}(n^3)$ operations. The best bound of matrix multiplication algorithm for now is $\mathcal{O}(n^{2.371552})$ [16].

If a prover P claims the answer of $A \cdot B$ is a matrix C? Can V verify it in $\mathcal{O}(n^2)$ time?
The $\mathcal{O}(n^2)$ Protocol:[4]

1. V picks a random r in F and lets $x = (r, r^2, ..., r^n)$.

2. V computes $C \cdot x$ and $A \cdot (B \cdot x)$, accepting if and only if they are equal.

Runtime Analysis:

V's runtime dominated by computing 3 matrix-vector products, each of which takes O(n^2) time.

- $C \cdot x$ is one matrix $(n \times n)$ times a vector $(n \times 1)$, the time complexity is $\mathcal{O}(n^2)$.

  Because each row of B is multiplied by the vector x, requiring n multiplications and n additions per row, and there are n rows.

- $(A \cdot B) \cdot x = A \cdot (B \cdot x)$ takes two matrix-vector multiplications.

  Matrix multiplication is associative, $B \cdot x$ takes $\mathcal{O}(n^2)$ first, and produces a $(n \times 1)$ matrix M, then $A \cdot M$ will also take $\mathcal{O}(n^2)$.

Correctness Analysis:

- If $C \equiv A \cdot B$:

  Then V accepts with probability 1

- If $C \neq A \cdot B$:

  The V rejects with high probability at least $(1 - 1/n)$.

  Simplified Proof: Recall that $x = (r, r^2, ..., r^n)$. So each matrix-vector multiplication is indeed the polynomials we've seen in **Reed-Solomon Error Corretion** at r of the i-th row of C.

  So if one row of C does not equal the corresponding row of $A \cdot B$, the fingerprints for that row will differ with probability at least $(1 - 1/n)$, causing V to reject w.h.p.

4.4 Function Extensions

4.4.1 Schwarts-Zippel Lemma

The Schwarts-Zippel Lemma[9] is an extension of univariate error-detection to multivariate polynomials. If p and q are distinct l-variate polynomials of **total degree** at most d. Then the same kind of statement holds. If we evaluate them at random-chosen inputs, they agree at the probability at most $d/|F|$.

- **Total Degree**:

  The total degree of a polynomial is the maximal of the sums of all the powers of the variables in one single monomial.

  For example: $\deg(x^2yz^4 - 3y + 4xe^5 - xy^3z^2) = 7$ (first monomial).

4.4.2 Extensions

An extension polynomial bridges the gap between a function defined on a discrete set of points and a function defined over a continuous (or larger discrete) domain.

A l-variate polynomial g over F is said to extend f if and only if g agrees at all of the input where f is defined. For example: We are given a function f that maps l-bits binary strings to a field F. This means f is defined on all possible combinations of l bits ($\{0,1\}^l$). But it is only defined on a finite set of points (the binary strings), which usually cannot form a field, **where we can leveraging algegratic tools of polynomials**.

A function g is said to extend f if:

- For all x in $\{0,1\}^l$, $f(x) \equiv g(x)$.

  g agrees with f on all inputs where f is initially defined.

- g is defined in a larger field.

  Let's say $l = 1$, then f is defined on input set $0, 1$. Where $f(0) = 2$, $f(1) = 3$.

  If we want to extend f to field R (real numbers), then our objective is to find a polynomial $g(x)$ in R such that $g(0) = 2$, and $g(1) = 3$.

  We can find $g(x) = x + 2$ defined for all x in R, not just $0, 1$.

  By representing f as a polynomial g, **we can apply the rich toolbox of algebraic methods & theorems avaliable for polynomials**. For example, Schwarts-Zippel Lemma is more poweful when there is a low-degree extension represents that function.

4.4.3 Constructing Low-Degree Extensions

In this section, we present a general way to construct low-degree polynomials.

There is a vector $w = (w_0, w_1, ..., w_{k-1})$ in F^k. $W : H^m \to F$: We define a function $W : H^m \to F$ such that $W(z) = w_{a(z)}$ if $a(z) \leq k-1$, and $W(z) = 0$ otherwise. W acts as a way to represent the vector w as a function over H^m that for indices corresponding to elements of w, $W(z)$ returns the corresponding w_i, otherwise 0.

The $a(z) : H^m \to F$ is the lexicographic order of z, which means **transforming a m-element vector to an index in the original** w.

Low-Degree Extension $\widetilde{W} : F^m \to F$:

$\widetilde{W}$ is an extension of W input from H^m to F^m, such that $\widetilde{W}$ is a polynomial of degree **at most** $|H| - 1$ in each variable, which enables efficient computation and has nice algebraic properties. **The degree of** W **is at most |H|-1 can be understand in the following cases:**

- Univariate Case: $W(x) : H \to F$
  - We have $n = |H|$ distinct points in the subset H.
  - For each $h \in |H|$, we want $\widetilde{W}(x) : F \to F)$ to satisfy $\widetilde{W}(h) \equiv W(h)$. **A univariate polynomial of degree at most** $(|H| - 1)$ **can be uniquely determined by its values on those** n **points**.
- Multivariate Case: $W(x) : H^m \to F^m$
  - Similarly, we have an m-dimensional grid H^m, where $H \subseteq F$ and $|H| = n$.
  - A polynomial $\widetilde{W}(x_1, x_2, ..., x_m)$ (in m variables) that:
    1. Agrees with W on every poit in H^m (i.e., $\widetilde{W}(h) = W(h)$ for all $h \in H^m$.
    2. Has degree at most $|H|-1$ in each variable. (i.e., for each variable x_i, the highest exponent of x_i in $\widetilde{W}$ is $\leq |H| - 1$. In other words, just like in the univariate case (where we need $deg(\widetilde{W}) \leq |H| - 1$ to interpolate $|H|$ points), here each variables is similarly bounded by $|H| - 1$. This ensures $\widetilde{W}$ can uniquely "pass through" all the points specified by W on H^m.

  The low-degree extension is the **simplest** polynomial that fits all the given points in H^m, The size of H determines how "complex" the polynomial needs to be (i.e., degree) in order to pass through all those points without ambiguity.

Here is the full definition of $\widetilde{W} : F^m \to F$**:**

$$\widetilde{W}(t_1, ..., t_m) = \sum_{i=0}^{k-1} \widetilde{B_i}(t_1, ..., t_m) \cdot w_i.$$

Where:

$\widetilde{B_i} : F^m \to F$ **Indicator Functions**:

The polynomials $\widetilde{B_i}$ act as an indicator functions on H^m, on that field, $\widetilde{B_i}(z) = 1$ if and only if $i \equiv a(z) = a(t_1, ..., t_m)$. Otherwise $\widetilde{B_i}(z) = 0$.

Outside H^m (in F^m/H^m), $\widetilde{B_i}$ takes on values determined by its polynomial extension, which means when input is outside H^m, $\widetilde{W}$ can have sum of multiple terms.

$\widetilde{W} : F^m \to F$ **Sum Selection**:

$\widetilde{W}$ is a low-degree polynomial, summing all possible k indexes i from 0 to $k-1$, in the field of H^m. According to the definition of $\widetilde{B_i}$ there will be only one "selected" corresponding $w(w_i)$ which is equal to $\widetilde{W}$.

$\widetilde{B}(z,p) : F^m \to F$ **Lagrange Basis Polynomial**:

Also, we can express $\widetilde{W}(t_1, ..., t_m)$ as:

$$\widetilde{W}(z) = \sum_{p \in H^m} \widetilde{B}(z,p) \cdot W(p)$$

Which constructs the polynomial $\widetilde{W}$ by summing the contributions from all basis polynomials $\widetilde{B}(z,p)$, each weighted by $W(p)$.

- $\widetilde{B}(p,p) = 1$
- $\widetilde{B}(z,p) = 0$ for all $z \in H^m$ where $z \neq p$
- When z outside H^m, $\widetilde{B}(z,p)$ can be other values

This means, when z is in F^m/H^m, each $\widetilde{B}(z,p)$ is a polynomial in z and can be evaluated at any z in F^m. And To compute $\widetilde{W}(z)$ at that time, which means by summing over all $W(p) \in H^m$ that has valid $\widetilde{B}(z,p)$.

Since W is only defined in the field H^m. And the low-degree polynomial property still holds.

By enlarging the input field (F^m) while keeping the degree of $\widetilde{W}$ low, this way of constructing LDE $\widetilde{W}$ helps us effectively using the power of randomless to detect the potencial error.

4.4.4 Multilinear Extensions

A multilinear extension of a function $f : \{0,1\}^n \to F$ (where F is a finite field) is a polynomial $\bar{f} : F^n -> F$ that agrees with f on $\{0,1\}^n$ and is **multilinear**.

Multilinear means each variable x_i in $\bar{f}$ has a degree at most 1, which make them highly effectie to evaluate. (a univariate f by make other variates constants will becomes a linear function). And since multilinear polynomials have minimal degree, the error detection probability is maximized for a given field size.

5 The Sum-Check Protocol

Suppose given a l-variate polynomial g defined over a finite field F. The purpose of the **sum-check protocol** [8] is to compute the sum:

$$H := \sum_{b_1 \in \{0,1\}} \sum_{b_2 \in \{0,1\}} ... \sum_{b_l \in \{0,1\}} g(b_1, b_2, ..., b_l)$$

In applications, this sum will often be over a large number of terms, so the verifier (V) may not have the resources to compute the sum without help. Instead, she uses the **sum-check protocol** to force the prover (P) to compute the sum for her.

The verifier(V) wants to verify that the sum is correctly computed by the prover(P), where g is a known multivariate polynomial over a finite field F.

5.1 Initialization

P claims that the total sum equals a specific value H_0.

5.2 First Round

1. P sends a univariate polynomial $s_1(x_1)$ to V, which is claimed to equal:

$$s_1(x_1) := \sum_{b_2 \in \{0,1\}} \sum_{b_3 \in \{0,1\}} \dots \sum_{b_l \in \{0,1\}} g(x_1, b_2, ..., b_l)$$

2. V calculates $s_1(0) + s_1(1)$ and checks whether that value is equal to H_0.

   Since s_1 is a univariate polynomial, V can compute $s_1(0) + s_1(1)$ directly (not using structure of H_0) in relatively short amount of time.

3. V picks a random element r_1 from F and sends it to P.

4. V sets $H_1 := s_1(r_1)$ for use in the next iteration.

$$H_1 = s_1(r_1) := \sum_{b_2 \in \{0,1\}} \sum_{b_3 \in \{0,1\}} \dots \sum_{b_l \in \{0,1\}} g(r_1, b_2, ..., b_l)$$

5.3 Interative Rounds ($i = 2$ to l)

For each round i:

1. P sends a univariate polynomial $s_i(x_i)$ to V, claimed to equal:

$$s_i(x_i) := \sum_{b_{i+1} \in \{0,1\}} \dots \sum_{b_l \in \{0,1\}} g(r_1, ..., r_{i-1}, x_i, b_{i+1}, ..., b_l)$$

   Which representing the partial sum over variables b_{i+1} to b_l, with $b1$ to b_{i-1} fixed to random values chosen by the verifier in previous rounds.

2. V calculates $s_i(0) + s_i(1)$ and checks whether that value is equal to H_{i-1}. H_{i-1} is the sum in the previous iteration: $s_{i-1}(r_{i-1})$:

$$H_{i-1} := s_{i-1}(r_{i-1}) := \sum_{b_i \in \{0,1\}} \dots \sum_{b_l \in \{0,1\}} g(r_1, ..., r_{i-1}, b_i, ..., b_l)$$

3. V picks a random element r_i from F and sends it to P.

4. V sets $H_i = s_i(r_i)$ for use in the next iteration.

$$H_i = s_i(r_i) := \sum_{b_{i+1} \in \{0,1\}} \dots \sum_{b_l \in \{0,1\}} g(r_1, ..., r_i, b_{i+1}, ..., b_l)$$

5.4 Final Check

In the final iteration:

$$H_l = s_l(r_l) := g(r_1, r_2, ..., r_l)$$

All the b_i in the original equation of $g(b_1, ..., b_l)$ has being fixed to the random number r_i chosen by V in previous l rounds.

V then check whether $s_l(r_l)$ is equal to $g(r_1, r_2, ..., r_l)$ **by calculating it** herself. **If** $s_l(r_l)$ **is equal to** $g(r_1, ..., r_l)$**, V accepts**.

5.5 Soundness of the Sum-Check Protocol

Completeness holds by design: If P sends the prescribed messages, then all of V's check will pass. Let's analysis the **soundness** of the protocol:

5.5.1 Base Case: Final Check

The Final Check (*key) is the most crucial part, and plays a key role in understanding this protocol.

In the last step, V directly computes $g(r_1, r_2, ..., r_l)$ and $s_l(r_l)$ to check whether they are equal, note that this is the only time for V to actually compute l-variate polynomial g by herself.

This direct comparison is critical because it anchors the entire verification process to the actual function g. According to Schwarts-Zippel Lemma, in the final check, if $s_l \neq g$, then the probability $g(r_1, ..., r_l)$ equal to $s_l(r_l)$ is less than $d/|F|$. (d is the Total Degree of polynomial g and s_l).

5.5.2 Backward Reasoning

By working backwards from the last iteration, We can understand the correctness of each step based on the validity of the final check:

In the last step, if the equality of $g(r_1, ..., r_l)$ and $s_l(r_l)$ holds, **with high probability,** $s_l(x_l)$ **must be the correct polynomial of** $g(r_1, ..., x_l)$, because a dishonest prover would need to guess r_l to fake $s_l(x_l)$ in order to pass the test, and the chance to success is less than $d/|F|$, where d is the totdal degree of polynomial g over field F..
So if V can confirm:

$$s_l(x_l) := g(r_1, ..., r_{l-1}, x_l)$$

is correct formed w.h.p in Final Check, then in iteration $(l-1)$, $s_{l-1}(r_{l-1})$ can be written as:

$$\begin{aligned} s_{l-1}(r_{l-1}) &= \sum_{b_l \in \{0,1\}} g(r_1, ..., r_{l-2}, r_{l-1}, b_l) \\ &= g(r_1, ..., r_{l-1}, 0) + g(r_1, ..., r_{l-1}, 1) \\ &= s_l(1) + s_l(2) \end{aligned}$$

So $s_{l-1}(x_{l-1})$ must be correctly formed w.h.p based on correct $s_l(x_l)$.

5.5.3 Inductive Case:

If $s_i(x_i)$ is correctly formed in round i For any round i, if we can make sure $s_i(x_i)$ is correctly formed w.h.p that:

$$s_i(x_i) = \sum_{b_{i+1} \in 0,1} ... \sum_{b_l \in 0,1} g(r_1, ..., r_{i-1}, x_i, b_{i+1}, ..., b_l)$$

Backwards to its previous round:

$$\begin{aligned} s_{i-1}(r_{i-1}) &= \sum_{b_i \in 0,1} ... \sum_{b_l \in 0,1} g(r_1, ..., r_{i-1}, b_i, ..., b_l) \\ &= \sum_{b_{i+1} \in 0,1} ... \sum_{b_l \in 0,1} g(r_1, ..., r_{i-1}, 0, b_{i+1}, ..., b_l) \\ &+ \sum_{b_{i+1} \in 0,1} ... \sum_{b_l \in 0,1} g(r_1, ..., r_{i-1}, 1, b_{i+1}, ..., b_l) \\ &= s_i(1) + s_i(2) \end{aligned}$$

Then, with high probability, $s_{i-1}(r_{i-1})$ should be correct, which indicate $s_{i-1}(x_{i-1})$ should also be correctly formed.

By induction, in the first round, since $s_2(x_2)$ is correctly formed, then w.h.p $s_1(x_1) = s_2(0) + s_2(1)$ should be formed correctly. And thus w.h.p, $H_0 = s_1(0) + s_1(1)$ should be the correct answer. So far, we've proved the soundness of the protocol.

5.6 Analyzing the Sum-Check Protocol

We've proved the soundness of the sum-check protocol in the previous subsection by showing that: **According to Schwarts-Zippel Lemma, with high probability a dishonest P cannot initially fake a incorrect H_0 that won't break the consistent in every round of the protocol and causes a correct $s_l(r_l)$ equals to $g(r_1, ..., r_l)$ in the final round.**

5.6.1 A Scenario where V cannot Compute $g(r_1, ..., rl)$

The soundness is relying on the final check, by validating the final step, V effectively validates all prior steps due to their interdependence in the chain of validations. For example, here we show a specific cheating strategy when V cannot compute $g(r_1, ..., r_l)$ by herself:

Imagine a dishonest P consistently uses different function $j(x)$ instead of the correct function $g(x)$ throughout the protocol. In this scenario, P computes all the partial sums and polynomials correctly with respect to $j(x)$, ensuring consistency at each step, P only deviates from the correct computation at the final check when V computes $g(r_1, ..., r_l)$.

During the protocol, V only computes $g(x)$ in the final check, the only point of detection is the final check, and the probability of detection is $d/|F|$. But if there is no final check, which means the protocol ends at round l, **V cannot have any guess of what function P used to compute H_0.**

5.6.2 Probability of Successful Cheating by P in Sum-check Protocol

In this part, we are going to quantify the probability for a dishonest P can succesfully convince V for the wrong computation H_0.

Deviation at Round i:

For any round i, let's assume what a dishonest P sends to V ($s_i(x_i)$) is formed incorrectly (i.e., $s_i(0) + s_i(1) \neq s_{i-1}(r_{i-1})$):

- **The probability V does not detect the deviations is $d/|F|$:**

  Since in round i, the $s_i(r_i)$ is set to H_i by V (in round l, H_i becomes to $g(r_1, ..., r_l)$). The probability for P to provide s_i to satisfy polynomial $H_i - s_i(r_i) = 0$ with randomly selected r_i by V from field F is $d/|F|$, where d is the total degree of polynomial g and s, according to Schwarts-Zippel Lemma.

- **The probability of V does not detect the deviations in future iterations is $(l - i)d/|F|$:**

  If $s_i(r_i) \neq H_i$, P is left to prove a false claim in the recursive call:

  The prover must construct $s_{i+1}(x_{i+1})$ such that $s_{i+1}(0) + s_{i+1}(1) = s_i(r_i)$. This means $s_{i+1}(r_{i+1})$ must deviate from true H_{i+1}, leading to a new error polynomial in subsequent rounds.

  Thus, we can get the the cumulative probability of acceptance when prover deviates at round i and the verifier does not detect in rounds $i + 1$ to l and end up accecpting: $(l - i)d/|F|$.

  The reason we sum the probabilities is due to the events of failing to detect the P in each round are over V's independent random choices r_i. And a cheating P can adapt their messages based on the V's previous random choices and messages exchanged so far. That means P's actions can be dependent on prior interactions.

- **The possibility of acceptance when prover deviates at round i is $(l-i+1)d/|F|$:**

  We have:

$$\begin{aligned} P[\text{V accepts}] &\leq P[\text{V not detect at i}] + P[\text{V not detect in i+1, l}] \\ &\leq d/|F| + (l-i)d/|F| \\ &= (l-i+1)d/|F| \end{aligned}$$

- **Upper Bound:** $ld/|F|$

  The worse-case scenario is that if P deviates in the first iteration ($i = 1$), the total probability of acceptance is:

$$d/|F| + (l-1)d/|F| = ld/|F|$$

5.6.3 Example: $l = 2$

Let's consider a minimum sum-check protocol with $l = 2$ by working backward from the last iteration to understand the soundness:

1. Final Check:
   (a) V computes $g(r_1, r_2)$
   (b) Comparison with P's $s_2(r_2)$
   If the equality holds, with high probability, $s_2(x_2)$ must be correct polynomial $g(r_1, x_2)$, because a dishonest prover would need to guess r_2 to fake $s_2(x_2)$.

2. Round 2: Since $s_2(x_2)$ that P sends to V is confirmed to be correct, the sum: $s_2(0) + s_2(1) = g(r_1, 0) + g(r_1, 1) = H_1$ must be satisfied. This confirms that H_1 is correctly computed based on $s_2(x_2)$.

3. Round 1: At the end of this round, P sets $H_1 = s_1(r_1)$. Since H_1 is now confirmed to be $g(r_1, 0) + g(r_1, 1)$, it implies $s_1(r_1) = g(r_1, 0) + g(r_1, 1)$, thus $s_1(x_1)$ is correct polynomial $\sum_{b_2 \in 0,1} g(x_1, b_2)$ w.h.p, any deviation would be detected with high probability due to the random r_1.

4. Initialization: V check $s_1(0) + s_1(1) = H_0$ Since $s_1(x_1)$ is correct w.h.p, then H_0 should be correctly formed based on $s_1(x_1)$.

   The verifier only needs to perform a few evaluations of univariate-polynomials and checks (except the final check g), making the protocol practical even for large computations.

6 General Purpose Interactive Proof Protocol

6.1 Example of the Limitations of Sum-Check Protocol: Sharp-SAT

The Sharp Satisfiability Problem (#-SAT) [10], is the problem of counting the number of interpretations that satisfy a given Boolean formula. Let ϕ be a Boolean formula of size S over n variables, (i.e., $\{0,1\}^n \to \{0,1\}$) the #-SAT ask us to count the number of satisfying assignments of ϕ.

6.1.1 Arithmetization

An intuitive way to solve the #-SAT problem is by using The Sum-Check Protocol, which means compute:

$$\sum_{x \in 0,1^n} \phi(x)$$

The final answer of all possible evaluation of ϕ is the answer we want, but as we all know that, the sum-check protocol requires the the function (g) to be polynomial, and to control communication and P and V's runtime, we need g to be "low-degree". By using Arithmetization, we can construct the extention polynomial g of the Boolean formula ϕ, which means replace ϕ with an arithmetic circuit:

By going gate-by-gate through ϕ, we can replace each gate with the gate's multilinear extension:

$$NOT(x) \to 1 - x$$

$$AND(x, y) \to x \times y$$

$$OR(x, y) \to x + y - x \times y$$

6.1.2 Costs of Sharp-SAT Using Sum-Check Protocol

The degree of g is less or equal to S (The size of ϕ), and the runtime of g is of $\mathcal{O}(S)$, let's analyzing the costs when applying sum-check protocol to this problem in total n rounds:

Communication Cost: P sends a polynomial of degree at most S in each round, V sends one field element r in each round. The total communication cost is $\mathcal{O}(S \times n)$.

- Why degree d of $s(x)$ is at most S?

  The degree of $s(x)$ that P sents to V in each round should be equal to g, which is the arithmetic version of circuit ϕ.

  Each polynomial g can be prepresented by its coefficients, requiring $\mathcal{O}(S)$ field elements, where S represents the size of ϕ. In Arithmetization when composing gates in ϕ, each gates can affect the overall degree at most 2 (AND gate $x \times y$ is in degree 2).

  In the worst-case scenario (where the gates are composed in a way that maximizes degree growth 2), the degree of the polynomial representing ϕ can be up to 2^D, where D is the depth of the formula. For a balanced formula, the depth D is $\mathcal{O}(\log S)$, therefore, the degree of g in each variable is at most:

  $$2^{\mathcal{O}(\log S)} = \mathcal{O}(S)$$

V Time Complexity: $\mathcal{O}(S)$ time to process each of the n messages of P, and $\mathcal{O}(S)$ time to evaluate $g(r)$. The total cost is $\mathcal{O}(S \times n)$

P Time Complexity: P needs to compute the univariate polynomial g_i by summing over all possible assignments of the remaining $n - i$ variables, which involves 2^{n-i} evaluations of g per round. And P's total time is dominated by the need to consider all 2^n possible assignments to the variables, leading to $\mathcal{O}(S \times n \times 2^n)$.

6.1.3 Limitations

According to [15], Sharp-SAT is a Sharp-P-complete problem, hence, the protocol we just saw implies every problem in Sharp-P has an **interactive proof protocol** with a polynomial time verifier.

Since in #-SAT, the time complexity of P is $\mathcal{O}(S \times n \times 2^n)$, which means even to very simple problems, the honest prover would require superpolynomial time by using sum-check protocol. And this seems unavoidable for Sharp-SAT[14, Section 4.2], since we don't know how to even solve the problem in less than 2^n time. We can hope to solve/prove easier problems without turing those problems into #-SAT instances since it is not practical.

6.2 Introduction to the GKR Protocol

6.2.1 Recall: The Notion of Interactive Proofs in 1980s

Recall Interactive Proof Systems in 80s[6], when the IP model first came out, it was only a theoretical model, which means no one cares about the runtime of the all powerful P. At that time, the idea[7] was people want to see how expressive, which computation can P prove to a polynomial time verifier by using interactive proofs.

6.2.2 Delegating Computation: Interactive Proofs for Muggles

In the paper "Delegating Computation: Interactive Proofs for Muggles" [5]. The author introduced a protocol that can be used to **effectively** for both P and V to prove/verify the correctness of **general purpose computation**.

More formally: For any question/language computable by a log-space uniform boolean circuit with depth d and input length n, the protocol can ensure:

- **The costs to V grow linearly with the depth d and input size n of the circuit, and only logarithmically with size S of the circuit.**

  V runs in time $(n + d) \times polylog(S)$, where $polylog(S)$ means a polynomial function of $logS$ (e.g., $(logS)^k$).

  The space complexity of V is $\mathcal{O}(logS)$.

- **P's running time is polynomial to the input size n.**

  P's runtime is not much more than perform the computation, which is in time $poly(n)$. Meaning it is efficient for practical purpose.

6.2.3 Blueprint of the Protocol

1. **Layered Circuit**

   The protocol divide circuit C into D phases, since for each phase/layer i, if its gates value is wrong, then some gates' value in phase/layer $(i+1)$ must be wrong. More specifically, some gates that connected to the error gates in layer $i+1$ must be incorrect.

2. **Local Correctness**

   With this in mind, we run a local **sum-check protocol** at each phases/layers to ensure **local correctness**.

   That is, we define a function $V_i : F^{s_i} \to F$ (where s_i is $logSi$, means the bits of # gates in layer d) And for any gate g_i in that layer i, running $V_i(g_i)$ will give us the corresponding gate value of g_i. By running d-subprotocols of each layers, where each protocol show connections between layer i and its layer before $(i+1)$. (More specifically, if V_i is not correct, then w.h.p V_{i+1} is not correct). **Eventually, it will be reduced to a claim about the input values (layer d), which are known to the verifier.**

6.3 Detailed Descriptions of the Protocol

Fix boolean circuit C: $\{0,1\}^n - > \{0,1\}$ of size (number of gates) S and depth d. The GKR protocol is an interactive proof protocol to prove that $C(x) = 1$.

Assume without loss of generality that C is layered, which means that each gate belongs to a layer, and each gate in layer i is connected by neighbors (determined) only in layer i+1. In a nutshell, the goal is to reduce V's runtime to be proportional to the depth d of the circuit C being computed, rather than its size, without increasing P's runtime by too much.

6.3.1 Arithmetize C

Convert C to a layered arithmetic circuit with fan-in 2 with layer d, and only consists of gate of the form **ADD** and **MULT**. fan-in 2 means each gates in the i-th layer takes inputs from two gates in the (i+1)-th layer. layer 0 denotes the output layer and d denotes the input layer.

We denote the number of gates in layer i as S_i, and let s_i to be the number of input elements of the final layer d. (F^{s_i}) As we mentioned in the blueprint. We define function $V_i(z)$ at each layer i to return the value of that gate with index z.

$$V_i : H_i^s \to F$$

V_0 corresponds to the output of the circuit, and V_d corresponds to the input layer. Here is a detailed Definition of V_i: To define V_i, let's recall the function W in Connstructing LDE: Imagine layer i of the circuit C to be a vector of S_i gates: $g = (g_1, g_2, ..., g_{S_i})$ where each g_j refers to the value of that gate. Then a function $V_i : H^m \to F$ can be defined such that $V_i(z) = g_a(z)$ if $a(z)$ is a valid gate in the vector g and $V_i(z) = 0$ otherwise.
Note that for every $p \in H^{s_i}$:

$$V_i(p) = \sum_{w_1, w_2 \in H^{s_i}} \widetilde{add}_i(p, w_1, w_2) \times (\widetilde{V_{i+1}}(w_1) + \widetilde{V_{i+1}}(w_2)) + \widetilde{multi}_i(p, w_1, w_2)\widetilde{V_{i+1}}(w_1) \times \widetilde{V_{i+1}}(w_2))$$

Where $\widetilde{add_i}, \widetilde{multi_i}$ refer to the low-degree extensions of wiring predicates add_i and $mult_i$ of layer i: $add_i(mult_i)$ takes one gate label $p \in H^{s_i}$ of layer i and two gate labels w_1, $w_2 \in H^{s_i}$ in layer i+1, and outputs 1 if and only if gate p is an addition (multiplication) gate that takes the output of gate w_1, w_2 as input.

6.3.2 Low Degree Extension of V_i at Layer i

In the i-th phase ($1 \leq i \leq d$): P runs a local protocol with V to argue the correctness of V_i. To do this, a **sum-check protocol** of layer i will be applied to let **P reduce the task of proving:**

$$\widetilde{V_i}(z_i) = r_i$$

to the task of proving:

$$\widetilde{V_{i+1}}(z_{i+1}) = r_{i+1}$$

where $z_i \in F^{s_i}$ is a random value determined by the verifier.
As we discussed in previous sections, $\widetilde{V_i}(z)$ can be expressed as:

$$\widetilde{V_i}(z) = \sum_{p \in H^{s_i}} \widetilde{B}(z,p) \times V_i(p)$$

By replacing $\widetilde{V_i}(p)$ of that formula, we can get for every z in F^{s_i}:

$$\widetilde{V_i}(z) = \sum_{p,w_1,w_2 \in H^{s_i}} \widetilde{B}(z,p) \times ((\widetilde{add_i}(p,w_1,w_2) \times (\widetilde{V_{i+1}}(w_1) + \widetilde{V_{i+1}}(w_2))) + (\widetilde{mult_i}(p,w_1,w_2) \times \widetilde{V_{i+1}}(w_1) \times \widetilde{V_{i+1}}(w_2)))$$

For every z_i in F^{s_i}, let $f_{i,z_i} : F^{3s_i} \to F$ to be the function defined by:

$$f_{i,z_i}(p,w_1,w_2) = \widetilde{B}(z_i,p) \times ((\widetilde{add_i}(p,w_1,w_2) * (\widetilde{V_{i+1}}(w_1) + \widetilde{V_{i+1}}(w_2))) + (\widetilde{mult_i}(p,w_1,w_2) \times \widetilde{V_{i+1}}(w_1) \times \widetilde{V_{i+1}}(w_2)))$$

Then $\widetilde{V_i}(z_i)$ can be expressed as:

$$\widetilde{V_i}(z_i) = \sum_{p,w_1,w_2 \in H^{s_i}} f_{i,z_i}(p,w_1,w_2)$$

6.3.3 Sum-Check Protocol at Layer 0

At each layer i, P wants to convince V that $V_i(z) \equiv r_i$ (At the output layer 0: $V_0(z) = r_0$, and r_0 is the output value of the entire circuit C).

First P will give the claim of the SUM over the gate in layer 0: $\widetilde{V_0}(z) = r_0$ ($F^{3s_i} \to F$), which should be the final result of the entire circuit:

$$r_0 = \widetilde{V_0}(z) = \sum_{p,w_1,w_2 \in H^{s_i}} f_0(p,w_1,w_2)$$

To verify this, P running an interactive **Sum-check** protocol with V on the output layer:

$$r_0 = \widetilde{V_0(z)} = \sum_{p,w_1,w_2 \in H^{s_i}} \widetilde{B}(z,p) \times ((\widetilde{add_0}(p,w_1,w_2) \times (\widetilde{V_1}(w_1) + \widetilde{V_1}(w_2))) + (\widetilde{mult_0}(p,w_1,w_2) \times \widetilde{V_1}(w_1) \times \widetilde{V_1}(w_2)))$$

As we described in 4. Final Check (*key), in the final step of the sum-check protocol, V needs to compute on her own to evaluate function $f_0(p, w_1, w_2)$ at random inputs p, w_1, w_2 chosen by herself.

Which means V needs to compute:

$$((\widetilde{add_0}(p, w_1, w_2) \times (\widetilde{V_1}(w_1) + \widetilde{V_1}(w_2))) + (\widetilde{mult_0}(p, w_1, w_2) \times \widetilde{V_1}(w_1) \times \widetilde{V_1}(w_2)))$$

and compare with $s_0(p, g)$ that P sends to her to check the correctness of the function $s_0()$.

Plus, since P now give the $s_0()$, V can now evaluate $\widetilde{add_0}, \widetilde{mult_0}$ on her own, using the structure of the circuit.

But things are getting different here, while some part of the $f_0(p, w_1, w_2)$ can be evaluated by V since she has the knowledge of the circuit. $(\widetilde{add_0}, \widetilde{mult_0})$. The main computational burden in this verificational task is computing $\widetilde{V_1}(w_1)$ and $\widetilde{V_1}(w_2)$, which requires time $poly(S)$ since it is related to the value of gates in layer $2, 3, ...d$.

So instead, in this protocol, P sends both these values $r_{1,1} = \widetilde{V_1}(w_1)$ and $r_{1,2} = \widetilde{V_1}(w_2)$ to V, and claim they are true. And then using the following interactive reduction protocol **to "reduce" to a single claim** used in the next layer:

So far, we reduced the task of proving that $\widetilde{V_0}(z) = r_0$ to the task of proving both $\widetilde{V_1}(w_1) = r_{1,1}$ and $\widetilde{V_1}(w_2) = r_{1,2}$.

However, recall our goal was to **reduce the task of proving** $V_0(z_0) = r_0$ **to the task of proving a single equality of the form** $V_1(z_1) = r_1$. What remains is to reduce the task of proving two equalities of the form $\widetilde{V_1}(w_1) = r_{1,1}$ and $\widetilde{V_1}(w_2) = r_{1,2}$ to the task of proving a single equality of the form $\widetilde{V_1}(z_1) = r_1$. This is done via the following (standard) process: [17]

1. V constructs line $\gamma : F \to F^{s_i}$ with two values w_1 and w_2, such that $\gamma(0) = w_1$ and $\gamma(1) = w_2$. And then sends γ to P.

2. P encodes the composition of $\widetilde{V_1}$ and γ to a univariate polynomial $f : F \to F$ such that $(f(x) = \widetilde{V_1}(\gamma(x)))$, and then sends to V.

3. V checks that $f(0) == \widetilde{V_1}(w_1) = r_{1,1}$ and $f(1) == \widetilde{V_2}(w_2) = r_{1,2}$. If check pass, V chooses a random element r from F and computes $f(r)$ produce a new claim that: $f(r) \equiv \widetilde{V_1}(\gamma(r))$

4. V then define $l := \gamma(r) \in F^{S_0}$ and sends (l, r) to P. Thus, in the next round of the sum-check protocol, P is left to prove a single claim:

$$\widetilde{V_1}(l) = f(r)$$

6.3.4 Sum-check Protocol at Layer d and the Final Check

In the iteration d, which is very similar to previous phases. P wants to convince V that $r_d = \widetilde{V_d}(z)$, and at the end of the protocol in this layer, P will sent $s_d(z)$ **which refer to the low-degree polynomial of the input**. Since this layer is the input layer, V can verify on her own. This amounts to computing a single point in the low-degree extension of the input x.

If all the input matches, this means function $s_d$is correctly formed, thus $\widetilde{V_d}$ is also correctly formed, and in the previous layer, $\widetilde{V_{d-1}}$ is also valid, all w.h.p etc.

Thus according to the Soundness of the Sum-Check Protocol, especially Backward Reasoning, we can get w.h.p that V_0 is correctly formed which implies r_0 which is $C(x)$ should should be correct w.h.p.

References

[1] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. “PRIMES is in P”. In: *Annals of Mathematics* 160 (2004), pp. 781–793. DOI: https://doi.org/10.4007/annals.2004.160.781.

[2] Stephen A. Cook. “The complexity of theorem-proving procedures”. In: *Proceedings of the third annual ACM symposium on Theory of computing* (1971), pp. 151–158. DOI: https://dl.acm.org/doi/10.1145/800157.805047.

[3] http://en.wikipedia.org/w/index.php?title=Field%20(mathematics)&oldid=1262361185. [Online; accessed 31-December-2024].

[4] Rusins Freivalds. “Probabilistic Machines Can Use Less Running Time”. In: *IFIP Congress 1977* (1977), pp. 839–842.

[5] Shafi Goldwasser, Yael Tauan Kalai, and Guy N. Rothblum. “Delegating Computation: Interactive Proofs for Muggles”. In: *Journal of the ACM Vol.62, Issue 4, Article No: 27* (2015), pp. 1–67. DOI: https://doi.org/10.1145/2699436.

[6] Shafi Goldwasser, Silvio M Micali, and Charles Rackoff. “The knowledge complexity of interactive proof-systems”. In: *Proceedings of the seventeenth annual ACM symposium on Theory of computing* (1985), pp. 291–304. DOI: https://dl.acm.org/doi/10.1145/22145.22178.

[7] Yael Kalai. *GKR based Zero-Knowledge Proofs*. URL: https://www.youtube.com/watch?v=x8pUxFptfb0.

[8] Carsten Lund et al. “Algebraic methods for interactive proof systems”. In: *Journal of the ACM Vol.39, No.4* (1992), pp. 859–868. DOI: https://doi.org/10.1145/146585.146605.

[9] J. T. Schwartz. “Fast Probabilistic Algorithms for Verification of Polynomial Identities”. In: *Journal of the ACM Vol.27, No.4* (1980), pp. 707–717. DOI: https://doi.org/10.1145/322217.322225.

[10] https://en.wikipedia.org/wiki/Sharp-SAT. [Online; accessed 31-December-2024].

[11] Michael Sipser. *Introduction to the Theory of Computation*. PWS PUblishing, 1997. ISBN: 0-534-94728-X.

[12] Michael Sipser. *NP-Completeness*. MIT 6.840, Lecture 15, NP-Completeness, 2020; accessed 31-December-2024. URL: https://ocw.mit.edu/courses/18-404j-theory-of-computation-fall-2020/0d814f5aba35f291f0253b7b34af2868_MIT18_404f20_lec15.pptx.

[13] Michael Fredic Sipser. *NP-Completeness*. URL: https://youtu.be/iZPzBHGDsWI.

[14] Justin Thaler. “Proofs, arguments, and zero-knowledge”. In: *Foundations and Trends® in Privacy and Security* (2022), pp. 117–660. DOI: http://dx.doi.org/10.1561/3300000030.

[15] Justing Thaler. *Interactive Proofs*. Presented at the Proofs, Consensus, and Decentralizing Society Boot Camp, Simons Institute for the Theory of Computing, August 2019; accessed 31-December-2024. URL: https://people.cs.georgetown.edu/jthaler/JTBootCamp.pdf.

[16] Virginia Vassilevska Williams et al. “New Bounds for Matrix Multiplication: from Alpha to Omega”. In: *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)* (2024), pp. 3792–3895. DOI: https://doi.org/10.1137/1.9781611977912.134.

[17] Tiacheng Xie et al. “Libra: Succinct Zero-Knowledge Proofs with Optimal Prover Computation”. In: *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference* (2019), pp. 733–764. DOI: https://eprint.iacr.org/2019/317.pdf.