A Primer on Probabilistic Proof Systems

Naman Kumar

February 12, 2024

Abstract

Contents

1 Introduction to Interactive Proofs

The standard idea of a mathematical proof is intuitively provided by the complexity class **NP**, which is the set of problems that can be efficiently verified by a polynomial-time verifier (a Turing Machine) given a polynomial time *witness*. Fundamentally, this witness is nothing but a proof of a statement. We recall the formal definition of the class **NP**.

Definition 1.1 (**NP**.). *A language L is in* **NP** *iff there exists a polynomial-time verifier M and a polynomial p such that*

$$\forall x \in L, \exists w : |w| = p(|x|) \implies M(x, w) = 1 \tag{1}$$

$$\forall x \notin L, \forall w : |w| = p(|x|) \implies M(x, w) = 0 \tag{2}$$

Here, the first predicate (1) may be termed *completeness*, since it certifies that a proof exists for all instances of L, and the second predicate (2) may be termed *soundness*, since it certifies that no proof exists for any $x \notin L$.

The interesting idea here is that **NP** can be taken to be a proof system between a (potentially computationally unbounded) prover and an efficient verifier.

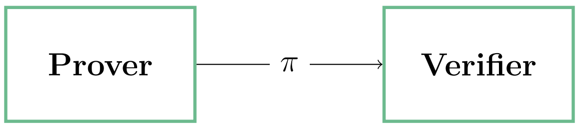

Figure 1: **NP** as a proof system.

1.1 Interactive Proofs

The idea of interactive proofs was originally formalized by Goldwasser, Micali, and Rackoff [GMR85] in 1985. The key insight into this definition is that **NP** can be equipped with two additional tools to make it more powerful, which leads to a new and groundbreaking form of computation.

Remark. We will denote the prover by $\mathcal{P}$ and the verifier by $\mathcal{V}$.

1. **Interaction.** Instead of a single prover message (consisting of the proof) from $\mathcal{P}$ to $\mathcal{V}$, $\mathcal{V}$ is also allowed to communicate messages to the prover, and multiple rounds of communication are allowed.

2. **Randomness.** The verifier is allowed access to a random string.

We formally define interactive proofs as follows.

Definition 1.2 (Interactive Proofs.). *An interactive proof for L is a pair $(\mathcal{P}, \mathcal{V})$ such that the following are satisfied:*

$$\forall x \in L, \Pr[\langle \mathcal{P}(x), \mathcal{V}(x, \cdot) \rangle = 1] = 1$$

$$\forall x \notin L, \forall \tilde{\mathcal{P}} \Pr[\langle \tilde{\mathcal{P}}(x), \mathcal{V}(x, \cdot) \rangle = 1] \leq \frac{1}{2}$$

Here, $\langle \mathcal{P}(x), \mathcal{V}(x, \cdot)\rangle$ is the output of the interaction between $\mathcal{P}$ and $\mathcal{V}$. The soundness error need not be $1/2$ and any large gap suffices.

The complexity class of languages which have an interactive proof is termed **IP**.

1.2 IP for Graph Non-Isomorphism

We know that $\mathbf{NP} \subseteq \mathbf{IP}$. Is the other way around true? In other words, do interaction and randomness give additional power to the proof system? We now see an example of a non-trivial IP for a problem not known to be in **NP**.

Graph Non-Isomorphism The problem of checking whether two graphs G_1 and G_2 are isomorphic (ie. whether there is a suitable permutation of the vertices of G_1 such it equals G_2) is in **NP**, since the permutation is a polynomial-size witness. GI is one of the problems known to be in **NP** that, as far as we know, has not yet proven to be **NP**-complete. GNI (graph non-isomorphism) is the problem of determining whether G_1 and G_2 are *not* isomorphic, which is clearly in *co***NP**, but it is still unknown whether it is *co***NP**-complete.

In the following protocol, both have access to both graphs, but the verifier does not have the computational power to check non-isomorphism.

1. $\mathcal{V}$ samples a random bit $b \leftarrow \{0, 1\}$.
2. $\mathcal{V}$ permutes the vertices of graph G_b randomly and sends the permuted graph G' to $\mathcal{P}$.
3. If $G_1 \not\cong G_2$, then $\mathcal{P}$ sends $G'' = G_b$ to $\mathcal{V}$.
4. Otherwise, $\mathcal{P}$ sends a random $G'' \leftarrow \{G_1, G_2\}$ to $\mathcal{V}$.
5. If $G'' = G_b$, the verifier outputs 1. Otherwise it outputs 0.

Figure 2: IP for Graph Non-isomorphism.

Analysis. First, it is clear that $\mathcal{P}$ will only be able to send G_b back with 100% certainty if $G_1 \not\cong G_2$. Otherwise, it won't be able to uniquely determine which graph was permuted by $\mathcal{V}$.

We now formally show that the above protocol is an interactive proof for GNI by verifying completeness and soundness.

Theorem 1.1. *The protocol in 2 is an interactive proof for* GNI.

Proof. **Completeness.** If $(G_1, G_2) \in \mathsf{GNI}$, then an honest prover will always be able to correctly find G_b by running an isomorphism check on both graphs. Thus, $\Pr[\langle \mathcal{P}(x), \mathcal{V}(x, \cdot)\rangle = 1] = 1$.

Soundness. If $(G_1, G_2) \notin \mathsf{GNI}$, then a dishonest prover cannot do better than randomly guessing some G'' and sending it. In this case, $\Pr[\langle \mathcal{P}(x), \mathcal{V}(x, \cdot)\rangle = 1] = \frac{1}{2}$, because it will output the correct G_b with probability 1/2. □

A natural question is that if $\mathcal{P}$ is so powerful, why can't it simply *simulate* $\mathcal{V}$ and determine what bit $\mathcal{V}$ has picked? The answer is that $\mathcal{V}$ has access to *private randomness* that $\mathcal{P}$ does not have access to. This is a crucial part of the protocol – if the prover had access to this randomness, the protocol fails, since even in the case of isomorphism, a dishonest prover would be able to determine which graph the $\mathcal{V}$ chose.

It turns out that private randomness is not necessary for **IP**. We will later see how proofs for all languages can be turned into public-coin proofs where $\mathcal{P}$ can convince $\mathcal{V}$ without $\mathcal{V}$ having any additional information that $\mathcal{P}$ does not.

1.3 IP = PSPACE

We now show the first part of a classical result that upper bounds the class **IP**. The full result was first shown by Adi Shamir [Sha92] in 1990 and formally published in 1992. We thus have good reason to believe that interactive proofs form a much broader class of computation than **NP**, and that randomness and interactivity both give additional amount of power to a proof system.

Theorem 1.2. IP ⊆ PSPACE.

Proof. Let $L \in \mathbf{IP}$, and let $(\mathcal{P}, \mathcal{V})$ be an IP for L. We show that $L \in \mathbf{PSPACE}$. First, we fix an instance x, and define

$$\mathbf{q}_x = \max_{\tilde{\mathcal{P}}} \Pr_r[\langle \tilde{\mathcal{P}}(x), \mathcal{V}(x, \cdot)\rangle = 1].$$

Concretely, $\mathbf{q}_x$ is the maximum probability of acceptance over all possible provers. It is clear that $\mathbf{q}_x = 1$ if $x \in L$ and $\leq 1/2$ otherwise. Thus if we manage to calculate $\mathbf{q}_x$ in polynomial space, we are done. The difficulty here is simulating all provers, since the prover can be a machine of potentially unbounded complexity.

Instead of attempting to simulate the prover, we will attempt to simulate the set of all possible transcripts (which must be polysize), and in the process simulate the *optimal prover strategy*. Let a partial transcript be $\pi_i = (a_1, b_1, a_2, b_2, \ldots a_i, b_i)$, where the a_i are the prover's messages and the b_i are the verifier's.

Definition. Let $P^*(x, \pi_i)$ be a function that outputs a^*_{i+1}, the prover message that maximizes probability of $\mathcal{V}$ accepting.

Lemma. $P^* \in \mathbf{PSPACE} \implies \mathbf{q}_x \in \mathbf{PSPACE}$.

Proof. Let $d(x, r)$ be the decision of $\mathcal{V}$ when it interacts with P^*. Then $\mathbf{q}_x$ can be written as

$$\frac{\sum_{r \in \mathcal{R}} d(x, r)}{|\mathcal{R}|}$$

Each $d(x, r)$ can be calculated using P^*. In particular, we take $P^*(x, \perp) = a_1^*, \mathcal{V}(x, a_1^*) = b_1, \ldots,$ till we get the verifier's output. Here we are simulating the verifier in polytime, and since $P^* \in$ **PSPACE**, the total computation occurs in **PSPACE**. Once we have found each $d(x, r)$, we can simply add and calculate the average, since $\mathcal{R}$ is only polynomial in size. □

Our problem now reduces to showing that P^*, the optimal prover strategy, is in **PSPACE**. This requires some amount of backward induction over the inputs, but the heuristic is as follows. If the protocol for L is k-round, then if we have the optimal $k-1$ messages, finding the optimal k^{th} message is as simple as iterating over all the possible polynomial-length output messages, running the verifier over each one (with different randomness) and choosing the one which maximizes verifier acceptance over the largest amount of random strings. If we have access to $k-2$ messages, we can do this recursively. We induct all the way back to the case in which we have no messages. k is constant, so we can be assured that that total amount of space used is polynomial. Simulating the verifier again requires polynomial space. All other considerations such as keeping a counter can be done in polynomial space or less.

Thus we can calculate $\mathbf{q}_x$ in polyspace. The value of $\mathbf{q}_x$ determines whether $x \in L$ or not. We are done. □

2 Containments of IP

Before we prove the other side of **IP** = **PSPACE**, we turn our attention to understanding more containments of **IP**. The techniques developed in this section will be integral in proving that **PSPACE** $\subseteq$ **IP** and in showing how IPs are developed for problems which don't have the nice properties of GNI. In particular, we will be proving the following two theorems.

Theorem 2.1. UNSAT $\in$ **IP**. *Since* UNSAT *is co*$\mathbf{NP}$*-complete, co*$\mathbf{NP} \subseteq \mathbf{IP}$.

Theorem 2.2. #SAT $\in$ **IP**. *Since* #SAT *is* $\mathbf{P}^{\#\mathbf{P}}$*-complete,* $\mathbf{P}^{\#\mathbf{P}} \subseteq \mathbf{IP}$.

2.1 Arithmetization

We consider *boolean formulae* of the form $\phi(x_1, \ldots, x_n)$, which are trees in which

- Every leaf node is labelled with a variable x_i, and
- Every internal node is a logical operator $(\wedge, \vee, \neg)$.

The goal of the arithmetization procedure is to reduce each boolean formula to a corresponding polynomial, such that $\phi(x_1, \ldots, x_n) = p(x_1, \ldots, x_n)$. Furthermore, we require $\mathsf{deg}(p) \leq |\phi|$, and that any evaluation of p at a point can be done in $|\phi|$ operations.

This can be accomplished through replacing each variable with the following.

$$\begin{aligned} \neg x &\Leftrightarrow 1 - x \\ x_1 \wedge x_2 &\Leftrightarrow x_1 \cdot x_2 \\ x_1 \vee x_2 &\Leftrightarrow x_1 + x_2 \end{aligned}$$

2.1.1 Application to UNSAT

Let ϕ be a 3CNF. Then we have

$$\begin{aligned} \phi \in \mathsf{UNSAT} &\implies \sum_{a_1, a_2, \ldots, a_n \in \{0,1\}} p(a_1, \ldots, a_n) = 0 \\ \phi \notin \mathsf{UNSAT} &\implies \sum_{a_1, a_2, \ldots, a_n \in \{0,1\}} p(a_1, \ldots, a_n) > 0 \end{aligned}$$

In other words, the sum of the corresponding polynomial over the boolean hypercube is zero if $\phi \in$ UNSAT, since on every input ϕ evaluates to 0, and strictly greater than 0 if $\phi \notin$ UNSAT, since ϕ must have at least one satisfying assignment. Furthermore, this fact remains true over all finite fields of sufficiently large size (in particular, size greater than $2^n 3^m$, where 3^m is the number of formulae, and 2^n is the number of satisfying assignments).

We will now use this fact to obtain an IP for UNSAT.

2.2 The Sumcheck Protocol

Sumcheck is a powerful and well-studied protocol first introduced in 1992 [LFKN92]. Many improvements to sumcheck have been made over the years, and today it is regularly used in prototype cryptographic systems as an interactive proof system for verifying polynomial addition.

Definition 2.1 (Sumcheck.). *The sumcheck protocol is an IP for verifying that, given* $(p, n, H, \gamma, \mathbb{F})$*, where* p *is a polynomial of degree* n *that takes* m *variables,* H *is any subset of* $\mathbb{F}$*, and* $\gamma \in \mathbb{F}$*,*

$$\sum_{\mathbf{x} \in H^m} p(\mathbf{x}) = \gamma.$$

The protocol for sumcheck is given in 3.

1. $\mathcal{P}$ calculates the polynomial
$$p_1(x) = \sum_{a_2,\ldots,a_n \in H} p(x, a_2, \ldots, a_n)$$
and sends it to $\mathcal{V}$.
2. $\mathcal{V}$ checks whether $\sum_{a_1 \in H} p_1(a_1) = \gamma$. If the check passes, it samples a challenge $w_1 \in \mathbb{F}$ and sends it to $\mathcal{P}$.
3. $\mathcal{P}$ calculates
$$p_2(x) = \sum_{a_3,\ldots,a_n \in H} p(w_1, x, a_3, \ldots, a_n)$$
and sends it to $\mathcal{V}$.
4. $\mathcal{V}$ checks whether $\sum_{\mathbf{x} \in H^m} p_2(x) = p_1(w_1)$. If the check passes, it samples another challenge $w_2 \in \mathbb{F}$ and sends it to the prover.
5. The protocol proceeds as in the previous two rounds. In the i^{th} round, $\mathcal{P}$ calculates $p_i(x)$ and sends it to the verifier, and the verifier checks whether $\sum_{\mathbf{x} \in H^m} p_i(x) = p_{i-1}(w_{i-1})$.
6. In the last round, $\mathcal{P}$ sends the polynomial $p_n(x) = p(w_1, \ldots, x)$. $\mathcal{V}$ samples w_n and checks whether $p_n(w_n) = p(w_1, \ldots, w_n)$, and outputs the result.

Figure 3: The Sumcheck Protocol.

Note. It is interesting to see that the verifier's messages in this protocol are all completely *random*. This fact is of independent cryptographic interest, and can be used to make the protocol non-interactive.

Theorem 2.3. *Sumcheck is a correct IP for the sum of polynomial evaluations over a subset.*

Proof. We begin by checking completeness. Given that $\sum_{\mathbf{x} \in H^m} p(\mathbf{x}) = \gamma$, an honest prover can always form the correct polynomials p_i such that any challenge by the verifier is always answered in the affirmative.

The more difficult check is that of soundness. We will show that given $\sum_{\mathbf{x}\in H^m} p(\mathbf{x}) \neq \gamma$, $\mathcal{V}$ outputs 1 with probability

$$\frac{n\deg(p)}{|\mathbb{F}|}$$

Define W be the event of $\mathcal{V}$ accepting, and let E_i be the event that $\tilde{p}_i = p_i$ (where $\tilde{p}_i$ is the polynomial output by a potentially dishonest prover.) We need to place a bound on $\Pr[W]$.

We will begin by showing that for $j \in [n]$,

$$\Pr[W] \leq \frac{(n-j+1)\deg(p)}{|\mathbb{F}|} + \Pr[W \mid \wedge_{i=j}^{n} E_i]$$

To show this, we use the Schwartz-Zippel Lemma.

Lemma 2.1 (Schwartz-Zippel)**.** *Let $f \in \mathbb{F}[x_1,\ldots,x_n]$ be a nonzero multivariate polynomial of degree d. Then for any set $S \subseteq \mathbb{F}$, we have*

$$\Pr_{\alpha_i \leftarrow S}[f(\alpha_1,\ldots,\alpha_n) = 0] = \frac{d}{|S|}. \quad \square$$

We proceed by induction. For the base case $j = n, \Pr[W] = \Pr[W \mid E_n] + \Pr[W \mid \overline{E_n}]$.

$$\Pr[W \mid \overline{E_n}] = \Pr[W \mid \tilde{p}_n(w_n) = p_n(w_n) \mid \tilde{p_n} \neq p_n] \leq \frac{\deg p_n - \tilde{p}_n}{|\mathbb{F}|} \leq \frac{\deg(p)}{|\mathbb{F}|}$$

Where the inequality comes from 2.1. For the inductive step,

$$\begin{aligned}
\Pr[W] &\leq \frac{(n-j+1)\deg(p)}{|\mathbb{F}|} + \Pr[W \mid \wedge_{i=j}^{n} E_i] \\
&\leq \frac{(n-j+1)\deg(p)}{|\mathbb{F}|} + \Pr[W \mid E_{j-1} \wedge \wedge_{i=j}^{n} E_i] + \Pr[W \mid \overline{E_{j-1}} \wedge \wedge_{i=j}^{n} E_i] \\
&\leq \frac{(n-j+1)\deg(p)}{|\mathbb{F}|} + \frac{\deg(p)}{|\mathbb{F}|} + \Pr[W \mid \wedge_{i=j-1}^{n} E_i] \\
&\leq \frac{(n-(j-1)+1)\deg(p)}{|\mathbb{F}|} + \Pr[W \mid \wedge_{i=j-1}^{n} E_i]
\end{aligned}$$

It follows that for $j = 1$, we have

$$\Pr[W] \leq \frac{n\deg(p)}{|\mathbb{F}|} + \Pr[W \mid \wedge_{i=1}^{n} E_i]$$

But $\Pr[W \mid E_1] = 0$, since $\sum_{\alpha \in H} p_1(\alpha) \neq \gamma$. Thus, we can conclude that

$$\Pr[W] \leq \frac{n\deg(p)}{|\mathbb{F}|}.$$

$\square$

2.3 An IP for UNSAT

We can obtain an IP for UNSAT from sumcheck, thereby proving 2.1. The IP proceeds as follows. Both the prover and the verifier pick a prime $q > 2^n 3^m$ and work over $\mathbb{F}_q$ (to do this, the $\mathcal{P}$ sends $\mathcal{V}$ a *claimed* prime, which $\mathcal{V}$ can verify in polytime since PRIMES $\in \mathbf{P}$). They then arithmetize $\phi \mapsto p$. Finally, the prover sends the verifier a claimed value of $\sum_{\mathbf{x}\in\{0,1\}} p(\mathbf{x})$, and they run the sumcheck protocol. If $\mathcal{V}$ accepts on $\mathbf{x} = \mathbf{0}$, then it outputs 1, otherwise it outputs 0.

2.4 An IP for #SAT

The IP for #SAT is similar, but unlike UNSAT, we require that we exactly count the number of satisfying assignments. This means that the arithmetization we described in 2.1 does not immediately work since $x_1 + x_2 \geq 1$, and in particular, $\phi(x_1, \ldots, x_n) = 0$ implies that $p(x_1, \ldots, x_n) = 0$, but $\phi(x_1, \ldots, x_n) = 1$ only implies that $p(x_1, \ldots, x_n) > 0$. Fortunately, there is a simple fix:

$$\begin{aligned} \neg x &\Leftrightarrow 1 - x \\ x_1 \wedge x_2 &\Leftrightarrow x_1 \cdot x_2 \\ x_1 \vee x_2 &\Leftrightarrow x_1 + x_2 - xy \end{aligned}$$

The $-xy$ term in $x_1 \vee x_2$ ensures that $x_1 \vee x_2 \leq 1$, and we can thus adapt the UNSAT protocol for #SAT. Note that $q > 2^n$ suffices, since p is boolean on boolean inputs.

3 PSPACE $\subseteq$ IP

Using the tools we have developed in the previous section, we are now ready to show the other side of **IP** = **PSPACE**. Our approach will be nearly identical to our IPs for UNSAT and #SAT – first we identify a **PSPACE**-complete problem that deals with boolean formulae, then we arithmetize it, and finally we run a counting IP to decide the arithmetization.

3.1 Quantized Boolean Formulae

Definition 3.1 (Fully Quantized Boolean Formula). *A QBF is a logical expression of the form*

$$\sim_1 x_1, \sim_2 x_2, \ldots, \sim_n x_n \phi(x_1, x_2, \ldots, x_n)$$

where $\phi(x_1, \ldots, x_n)$ is a CNF, and $\sim_i$ is either $\forall$ or $\exists$ operator.

Each QBF evaluates to either true or false. In fact, we can view **NP**, *co***NP** and various other complexity through the lens of QBFs:

$$\mathbf{NP} := \{\phi \mid \exists x_1, \ldots, \exists x_2 \phi(x_1, \ldots, x_n = 1)\}$$
$$co\mathbf{NP} := \{\phi \mid \forall x_1, \ldots, \forall x_2 \phi(x_1, \ldots, x_n = 1)\}$$

We can further recall that **PH** is defined by QBFs in which the quantifiers alternate. The fact that **PH** $\subseteq$ **PSPACE** can be easily seen once derive the following result.

Definition 3.2 (TQBF). TQBF *is the language consisting of 3-CNFs with alternating quantifiers, ie.*

$$\mathsf{TQBF} = \{\phi(x_1, x_2, \ldots, x_n) \mid \forall x_1, \exists x_2, \ldots \phi(x_1, x_2, \ldots, x_n) = 1\}$$

Theorem 3.1. TQBF *is* **PSPACE***-complete.*

We will look at the proof of 3.1 in a later section.

3.2 Arithmetization of TQBF

Before we can start running an interactive proof over **TQBF**, we arithmetize it to a polynomial. The inner 3-CNF can simply be arithmetized as the polynomial for #SAT. The transformation for the outer quantifers is more involved, but comes intuitively from the definition.

- $\forall$ acts as a conjunction, since $\forall x_i \phi = \phi(x_1, \ldots, 1, \ldots, x_n) \wedge \phi(x_1, \ldots, 0, \ldots, x_n)$.
- $\exists$ acts as a disjunction, since $\forall x_i \phi = \phi(x_1, \ldots, 1, \ldots, x_n) \vee \phi(x_1, \ldots, 0, \ldots, x_n)$.

Thus, we can arithmetize the quantifiers as

$$\forall x_i \phi(x_1, \ldots, x_i, \ldots, x_n) \Leftrightarrow p_\phi(x_1, \ldots, 0, \ldots, x_n) p_\phi(x_1, \ldots, 1, \ldots, x_n)$$
$$\exists x_i \phi(x_1, \ldots, x_i, \ldots, x_n) \Leftrightarrow 1 - ((1 - p_\phi(x_1, \ldots, 0, \ldots, x_n))(1 - p_\phi(x_1, \ldots, 1, \ldots, x_n)))$$

We assign symbols to these operations, namely $\prod_{x_i}$ and $\coprod_{x_i}$.

There is a slight problem, however. Since we are multiplying polynomials a total of 2^n times (each variable has a quantifier) our final polynomial will contain variables of the form x^{2^n}, a calculation the verifier cannot do in polynomial time. However, we notice that $\phi(\mathbf{x}) = p(\mathbf{x}) \leq 1$. If this is the case, then x_i^k can be replaced by x_i in all fields (including $\mathbb{F}_2$). Thus, we have to add an additional degree quantifier, Δ, which performs the *degree reduction* operation[1].

Our final arithmetization becomes

$$\forall x_1, \exists x_2, \ldots, \phi(\mathbf{x}) \Leftrightarrow \prod_{x_1} \Delta_{x_1} \coprod_{x_2} \Delta_{x_1} \Delta_{x_2} \ldots p_\phi(\mathbf{x})$$

3.3 Shamir's Protocol

The protocol we now describe was given by Adi Shamir in his original [Sha92] paper. It is a variation of the sumcheck protocol that also allows evaluation of our new quantifier arithmetizations. The protocol is described in 4. It proceeds for a total of $O(n^2)$ rounds.

Theorem 3.2. *Shamir's Protocol is a correct IP for* TQBF.

Proof. The full proof can get a bit tedious, but the high-level overview follows from a good understanding of the sumcheck protocol. Completeness is guaranteed from the fact that an honest prover will correctly send the polynomials p_i and finally $p(w_1, \ldots, w_n) = \gamma_k$ follows from correctness and definition of the p_i.

For soundness, we use similar properties as that of sumcheck, but separate our analysis into two cases:

- Case 1: The round is a $\prod / \coprod$ round. In this case the prover can't send the correct polynomial p_i since the arithmetized polynomial does not evaluate to a nonzero value, and so $p_i(0)p_i(1) \neq \gamma_{i-1}$. Hence it must send an incorrect polynomial $\tilde{p}_i$ which passes the check. The probability that $\tilde{p}$ passes the verifier's check is $\Pr[\tilde{p}_i(w_i) = p_i(w_i)] \leq \frac{1}{\mathbb{F}}$.
- Case 2: The round is a Δ round. In this case the same applies, but the degree of the polynomial is higher, so the error is at most $\frac{3m}{\mathbb{F}}$ or $\frac{2}{\mathbb{F}}$.

The total soundness error we get is $\frac{1}{\mathbb{F}}(n + \sum_{i=1}^{n-1} 2i + 3mn) = \frac{3mn+n^2}{\mathbb{F}}$, and if we take $\mathbb{F}$ large enough, the error can get as small as we like. $\square$

[1]It should be clear why $\mathcal{V}$ cannot multiply polynomials in polynomial amount of time. It is less apparent, however, why the verifier cannot simply do degree reduction by itself by a simple sift through the monomial expansion. The reason is that the monomial expansion is a particularly inefficient representation of polynomials which takes more than polynomial amount of space in the number of variables. Thus the obvious algorithm is not actually a polytime operation. The 'correct' representation of polynomials is in the form of an arithmetic circuit, and degree reduction is not a polynomial-time operation under this representation.

1. In the i^{th} step, $\mathcal{P}$ sends the polynomial p_i to the verifier, which represents the polynomial in the i^{th} variable with all the later quantifiers applied and all the previous variables substituted by the verifier's random challenges w_i.
   (a) If the quantifier is $\prod / \coprod$, the polynomial is a one-degree polynomial.
   (b) If the quantifier is Δ, the polynomial is a $3m$- or 2- degree polynomial, the polynomial *before* it became the degree-reduced version of the previous round. In the first $n-1$ variable, this is a 2 degree polynomial since it is the product of two different 1-degree polynomials. In the last variable, it is the arithmetization itself, which could have up to a $3m$-degree.
2. On receiving this polynomial, the verifier does one of the following:
   (a) If the i^{th} quantifier is $\prod$, it checks if $p_i(0)p_i(1) = \gamma_{i-1} = p_{i-1}(w_{i-1})$, the evaluation of the previous step.
   (b) If the quantifier is $\coprod$, it checks if $1-(1-p_i(0))(1-p_i(1)) = \gamma_{i-1}$.
   (c) If the quantifier is Δ, it checks if $p_i(\mathbf{w}_{i-1}) = \gamma_{i-1}$, where $\mathbf{w}_i = (w_1, \dots, w_i)$, since the degree-reduced polynomial should have the same evaluation.
3. The verifier sends a challenge $w_i \leftarrow \mathbb{F}$.
4. After $O(n^2)$ rounds, the verifier checks if $p(w_1, \dots, w_n) = \gamma_k$.

Figure 4: Shamir's Protocol.

We have thus shown an IP for TQBF. To complete our proof, we show that TQBF is **PSPACE**-complete.

3.4 PSPACE-completeness of TQBF

We will now provide a proof of 3.1.

Proof. First, we see that TQBF is actually in **PSPACE**. To show this, it is enough to notice that evaluating the QBF over all possible inputs along with keeping a polysize tab on whether each quantifier is satisfied is enough. The evaluation of the QBF over all inputs can be easily done in polyspace, since each evaluation can be done in polytime as well.

We now show that TQBF is **PSPACE**-hard. Consider a language $L \in \mathbf{PSPACE}$. We wish to construct in polytime a polysize QBF ϕ such that $x \in L \iff \phi \in \mathsf{TQBF}$. We will use a Turing Machine reduction for this.

First, consider the *configuration graph* $\mathcal{C}_x$ of the computation for L on x, which is the graph with nodes corresponding to valid configurations of the TM that computes L and directed edges between valid 1-step transitions during the evaluation on x. Let C_s be the universal start state and C_a be the universal accept state. It follows that

$$x \in L \iff P(C_s, C_a) \in \mathcal{C}_x$$

ie, there exists a path between C_s and C_a in $\mathcal{C}_x$. Furthermore, the path length is of less than $2^{O(S(n))}$, since the total number of configurations of that amount. We will recursively define our QBF $\Phi = \Phi_{S(n)}$. We begin by defining

$$\Phi_i(C, C') = 1 \iff \exists P(C, C') \in \mathcal{C}_x, |P| \leq 2^i$$

First, Φ_1 is simply the Cook-Levin formula corresponding to a 1-step transition between C and C'. This is polysize and contains no quantifiers. Recursively we can then define

$$\Phi_i(C, C') = \exists C'' \Phi_{i-1}(C, C'') \wedge \Phi_{i-1}(C'', C)$$

The problem is that this effectively doubles the size of the formula, which we can't afford (as it will take more than polynomial the amount of space). To offset this, we introduce new variables D_1 and D_2 such that

$$\Phi_i(C, C') = \exists C'' \forall D_1, D_2 (D_1 = C \wedge D_2 = C'') \vee (D_1 = C'' \wedge D_2 = C) \implies \Phi_{i-1}(D_1, D_2)$$

It is worth looking at what is actually happening here. The two new variables we have introduced sift over all the vertices and check whether there exists some C'' for which this works using just a single evaluation of Φ_{i-1}. Parsed, this means that for whatever D_1 and D_2 that are taken, if even one of the two possible valuations is satisfied, it will follow that $\Phi_{i-1}(D_1, D_2)$ will be true. The 'implies' function can be encoded as a formula with just a polynomial overhead. Thus there is only a polynomial size jump at each step, and we have

$$\Phi_i = \Phi_{i-1} + \mathsf{poly}(S(n))$$

It follows that $\Phi_{S(n)}$ is polysize. Thus, TQBF is **PSPACE**-complete. □

Theorems 3.1 and 3.2 together imply that **PSPACE** $\subseteq$ **IP**. The other direction was previously shown. It follows that the class of languages decidable by **IP** is exactly equal to the class decidable by **PSPACE**, which, according to conventional belief, gives a considerable amount of power over base **NP**.

Theorem 3.3. IP = PSPACE.

Proof. Follows from 1.2, 3.1 and 3.2. □

4 Public-Coin Interactive Protocols

In the IP for GNI, we saw that a crucial component of correctness was the fact that $\mathcal{P}$ did not have access to the private randomness of $\mathcal{V}$. This was in contrast to sumcheck, in which all randomness of the verifier was communicated to $\mathcal{P}$. We now turn the question of encoding these different kinds of ramdomness, and see what can be said about the complexity classes in which they belong.

4.1 AM and MA

Definition 4.1 (Public-Coin Verifier)**.** *A verifier $\mathcal{V}$ is public-coin iff every message communicated by $\mathcal{V}$ is a freshly sampled random string, otherwise it is private-coin.*

Definition 4.2 ($\mathbf{AM}$ and $\mathbf{MA}$)**.** *The complexity classes $\mathbf{AM}[k]$ and $\mathbf{MA}[k]$ are languages decidable by a public-coin verifier through a k-round IP in which the verifier and prover send the first message, respectively.*

$\mathbf{AM}$ and $\mathbf{MA}$ were introduced by László Babai [Bab85] in 1985, around the time interactive proofs were introduced. The names $\mathbf{AM}$ and $\mathbf{MA}$ stand for Arthur-Merlin and Merlin-Arthur respectively, with Merlin corresponding to the prover and Arthur to the verifier. It is interesting to note that $\mathbf{MA}[1]$ is the probabilistic analog of $\mathbf{NP}$, and thus the question of $\mathbf{MA}[1] = \mathbf{NP}$ is closely tied to derandomization, with the equality holding if PRGs exist. Conventionally, $\mathbf{AM}$ and $\mathbf{MA}$ are used to refer to $\mathbf{AM}[2]$ and $\mathbf{MA}[1]$, respectively. The following lemma is trivial:

Lemma 4.1. *$\forall k \in \mathbb{N}, \mathbf{AM}[k]$ and $\mathbf{MA}[k] \subseteq \mathbf{IP}[k]$.*

In 1986, Shafi Goldwasser and Michael Sipser [GS86] showed that private coin protocols can be simulated by public coin protocols with an extra round of communication.

Theorem 4.1. *$\mathbf{IP}[k] \subseteq \mathbf{AM}[k+1]$.*

Note that this does not imply that $\mathbf{IP} \subseteq \bigcup_{i=1}^{\infty} \mathbf{AM}[i]$, since $\mathbf{IP}$ also contains protocols where the number of rounds is not constant, ie. polynomial in the length of the input. However, if $\mathbf{AM}$ is extended to support polynomial size public-coin proofs, then $\mathbf{IP} \subseteq \mathbf{AM}[\mathsf{poly}]$. We will not prove the above theorem, but we will look at a weaker case.

Theorem 4.2. *$\mathsf{GNI} \in \mathbf{AM}[1]$.*

In other words, we will show that there is a public-coin IP for GNI which involves only one round of communication between the prover and the verifier ($\mathbf{AM}[1]$ is one-*round*, so that means that $\mathcal{V}$ sends a message and $\mathcal{P}$ responds).

4.2 The Set Lower Bound Protocol

To find a public-coin protocol for GNI we will have to find a way to look at GNI quantitatively. Once we have accomplished this, we will run the set lower-bound protocol that differentiates between sets of different sizes with high probability, assuming that there is an efficient technique to verify membership in the set. Before we look at the application of set lower-bound to GNI we will study the protocol in its full generality.

Definition 4.3. *Let $S \in \{0,1\}^m$ be a set such that membership is verifiable in polynomial time. The Set Lower Bound Protocol is an IP for the promise problem "1 if $|S| > B$, 0 if $|S| \leq \frac{B}{2}$."*

The protocol is illustrated in 5.

1. $\mathcal{V}$ sets ℓ such that $2^{\ell-2} \leq B \leq 2^{\ell-1}$. It samples $h \leftarrow \mathcal{H}$ and $y \leftarrow \{0,1\}^\ell$ and sends them to $\mathcal{P}$.
2. $\mathcal{P}$ determines an x such that $h(x) = y$ and $x \in S$. It then sends (x, π) to the verifier where π is a proof certifying membership of x in S.
3. If π is a valid proof and $h(x) = y$, $\mathcal{V}$ accepts. Otherwise it rejects.

Figure 5: The Set Lower Bound Protocol.

Definition 4.4 (Pairwise-independent Hash Functions)**.** *A family $\mathcal{H}_{m,\ell} = \{h : \{0,1\}^m \to \{0,1\}^\ell\}$ is pairwise independent if, for all distinct x, x' and for all (not necessarily distinct) y, y' we have*

$$\Pr_{h \in \mathcal{H}_{m,\ell}}[h(x) = y \wedge h(x') = y'] \leq \frac{1}{2^{2\ell}}$$

We can construct a pairwise-independent hash function family from a set of random affine functions, where $\mathcal{H}_{m,m} = \{h_{a,b}(x) = ax + b\}_{a,b \in \mathbb{F}_{2^m}}$. We have

$$\begin{aligned}
&\Pr_{h \in \mathcal{H}_{m,\ell}}[h(x) = y \wedge h(x') = y'] \\
=&\Pr_{h \in \mathcal{H}_{m,m}}[ax + b = y \wedge ax' + b = y'] \\
=&\frac{1}{2^m} \times \frac{1}{2^m} \\
=&\frac{1}{2^{2m}}
\end{aligned}$$

We can reduce the size of the domain by considering $\mathcal{H}_{m,\ell} = \{h_{a,b}(x) = ax+b \pmod{2^\ell}\}_{a,b \in \mathbb{F}_{2^m}}$. This does not affect pairwise independence.

Theorem 4.3. *The protocol of 5 is a correct IP for set lower bound.*

Proof. Both soundness and completeness can be verified using simple probability.

- Soundness: Let $|S| \leq \frac{B}{2}$. Then

$$\Pr[\exists x \in S \mid h(x) = y] \leq \sum_{x \in S} \Pr[h(x) = y] \leq \frac{1}{2^\ell} \times \frac{B}{2} = \frac{B}{2^{\ell+1}}.$$

- Completeness: Let $|S| > B$. We fix y and take randomness only over x. Using the inclusion-exclusion principle, we get

$$\begin{aligned}\Pr[\exists x \in S \mid h(x) = y] &\geq \sum_{x\in S} \Pr[h(x) = y] - \sum_{x_0,x_1\in S} \Pr[h(x_i) = y] \\ &\geq \frac{B}{2^\ell} - \frac{B}{2^{2\ell}} \\ &\geq \frac{3B}{4\cdot 2^\ell}.\end{aligned}$$

□

Note that the analysis of this protocol did not have *perfect completness*, ie. there was a completeness error. We can use standard methods in order to reduce this error. In particular, if we repeat the protocol a large number of times, then the number of successes will tend to the fraction of correct executions (assuming that $|S| \geq B$). The probability that even when $|S| \geq B$, the executions turn out to be low enough that we can be convinced of $|S| \leq \frac{B}{2}$ can be found by the Chernoff bound since the output of each trial is an independent binary random variable. This turns out to be very small even for a low number of repititions.

However, we can obtain a completeness error of 0 simply by adding another round of interaction. We will see how to do this in the next section.

4.2.1 A Public-Coin Protocol for GNI

Our public-coin protocol for GNI will make heavy use of the machinery given to us by the set lower bound protocol. Consider this: if $G_1 \cong G_2$, then the total number of graphs which are isomorphic to either G_1 or G_2 is $|V_1|!$, since there is an isomorphic graph for each permuation. In the case where $G_1 \ncong G_2$, the total number of graphs is $2 \cdot |V_1|!$. This gives us the opportunity to use the set lower bound protocol on the set

$$S = \{H \in \{0,1\}^{n^2} \mid H \cong G_1 \vee H \cong G_2\}.$$

Checking membership in S is a matter of determining whether $H \cong G_1$ or $H \cong G_2$, which is an instance of graph isomorphism. But graph isomorphism is in **NP**, since a polysize certificate for it is simply the permutation corresponding to the node labels of G_1. Thus we can efficiently check membership in S.

We are done! Running the set lower bound protocol on this set with $B = 2 \cdot |V_1|!$ is a correct public-coin protocol for GNI.

4.3 Perfect Completeness

The set lower bound protocol of 5 is a correct protocol for the problem, but it does not strictly satisfy the definition of an IP which we have laid out in section 1. This is because there is some amount of completeness error. Fortunately, this error can be mitigated simply by adding a single extra round to the protocol.

Theorem 4.4. *If L has a k-round interactive proof then it has a $k+1$-round interactive proof with perfect completeness.*

The proof of this theorem is highly involved and has been omitted. The basic idea is that from the Sipser-Gács-Lautemann [Lau83] proof that $\mathbf{BPP} \subseteq \Sigma_2^P$. We use the probabilistic method to find a set of strings s_i such that $s_i \oplus r$ for any randomness r, when used as randomness, outputs 1 if $x \in L$, and outputs 0 if $x \notin L$. This procedure is carried out over k rounds with the set of s_i being sent to the verifier in the first round.

5 IPs with Bounded Resources

Until now we have looked at the class **IP** on a global level and considered its relationships with other well-known complexity classes. We have also looked at the details of the randomness and interaction that the prover and verifier can use to prove a statement. We now turn our attention to understanding the kind of problems that the prover and the verifier can solve when we place bounds on the amount of information they can communicate.

Definition 5.1. $\mathbf{IP}[\mathsf{pc} = 1]$ *The class* $\mathbf{IP}[\mathsf{pc} = 1]$ *consists of IPs in which the prover only communicates* 1 *bit of information.*

This class contains GNI! Thus we can reasonably believe that even 1 bit of prover communication allows problems outside of **P** to be decided. We will attempt to create a rigorous theory for IPs with bounded resources, first described by Goldreich and Håstad in [GH98].

5.1 Languages Decidable with Limited Resources

Definition 5.2 (IPs with bounded resources)**.** *The class* $\mathbf{IP}[\mathsf{pc}, \mathsf{vc}, \mathsf{vr}]$ *is the class of languages decidable by an IP in which the prover sends* pc *bits, the verifier sends* vc *bits, and the amount of verifier randomness is* vr *bits.*

The definition of $\mathbf{AM}[\mathsf{pc}, \mathsf{vc}, \mathsf{vr}]$ is analogous. It turns out that there is a large number of theorems that determine what languages are decidable by such IPs.

Theorem 5.1. *The following theorems give a relationship between the communication complexity of IPs and the time complexity of the languages decidable by them.*

$$\begin{aligned}
\mathbf{IP}[\mathsf{pc}, \mathsf{vc}, \mathsf{vr}] &\subseteq \mathbf{DTIME}(2^{O(\mathsf{pc}+\mathsf{vc}+\mathsf{vr})}\mathsf{poly}(n)) \\
\mathbf{IP}[\mathsf{pc}, \mathsf{vc}, *] &\subseteq \mathbf{BPTIME}(2^{O(\mathsf{pc}+\mathsf{vc})}\mathsf{poly}(n)) \\
\mathbf{AM}[\mathsf{pc}, *, *] &\subseteq \mathbf{BPTIME}(2^{O(\mathsf{pc}\log\mathsf{pc})}\mathsf{poly}(n)) \\
\mathbf{IP}[\mathsf{pc}, *, *] &\subseteq \mathbf{BPTIME}(2^{O(\mathsf{pc}\log\mathsf{pc})}\mathsf{poly}(n))^{\mathbf{NP}}
\end{aligned}$$

Here, n is the number of messages exchanged.

5.2 Game Trees

Definition 5.3 (Transcript)**.** *A transcript of an interaction is the tuple $(a_1, b_1, \ldots, a_n, b_n)$ of messages exchanged. An augmented transcript is the tuple $(a_1, b_1, \ldots, a_n, b_n, r)$, where r is the verifier randomness.*

Fixing a verifier $\mathcal{V}$ and an instance x, we can define the game tree $T = T(\mathcal{V}, x)$ of $\mathcal{V}(x)$ as the tree of all possible augmented transcripts. The game tree is of height k, with each layer labeled $0, \ldots, k-1$. The prover communicates at each $2i$ layer and the verifier communicates at each $2i+1$ layer. In this setting,

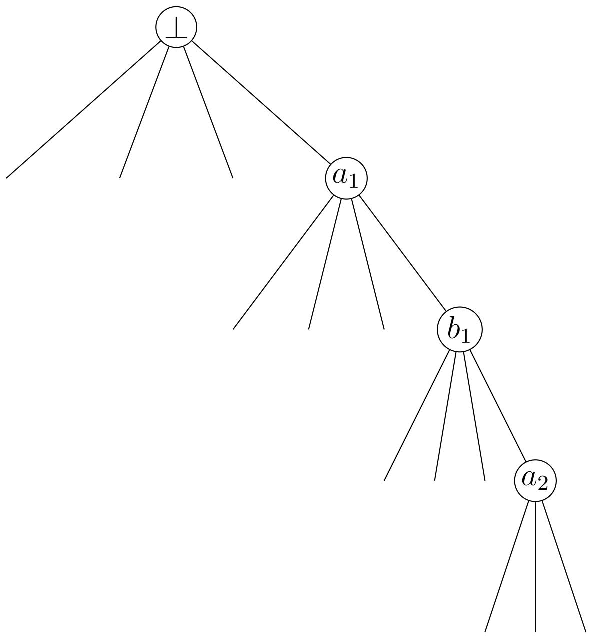

Figure 6: A Game Tree.

- edges from $2i$ to $2i+1$ are each of the possible messages communicated by the prover when given the previous node,
- edges from $2i+1$ to $2(i+1)$ are each of the possible messages communicated by the verifier when given the previous node,
- the last set of edges is all the verifier random strings compatible with the transcript.

We can use the game tree in order to determine the complexity of the languages decidable by it, since it forms a complete representation of some interactive proof. To do this, we will approximate the *value* of a tree.

Definition 5.4 (Value)**.** *The value* $\mathsf{val}(T)$ *is the value of the root, where values of nodes are defined as follows:*

1. *the value of a leaf* $(a_1, b_1, \ldots, a_n, b_n, r)$ *is* $\mathcal{V}(x, a_1, b_1, \ldots, a_n, b_n, r) \in \{0, 1\}$*,*
2. *the value of a node at* $2i$ *is the maximum of the values of its children,*
3. *the value of a node at* $2i+1$ *is the average of the values of the children weighted by the probability that the child is selected.*

We notice that if $x \in L$ then $\mathsf{val}(T) \geq 2/3$, since a high proportion of paths will lead to an accepting leaf, while $\mathsf{val}(T) \leq 1/3$ if $x \notin L$ since even a cheating prover that attempts to maximize acceptance will not be able to convince the verifier in more than 1/3rd of the choices taken by the verifier.

Note. The game tree is unique to a particular IP and not to the set of all possible IPs. To better understand this, consider that $\mathsf{val}(T) = 1$ in case of IPs with perfect completeness.

This is because the value of each leaf node is 1 – given any particular verifier random string, regardless of what message $\mathcal{V}$ sends, the prover will always be able to convince the verifier.

We will now attempt to place a bound on the complexity required to approximate $\mathsf{val}(T)$.

5.2.1 IPs with All Parameters Bounded

We would like to approxiate $\mathsf{val}(T)$ up to an error of $\pm 1/6$. Let $c = \mathsf{pc} + \mathsf{vc} + \mathsf{vr}$ be the total communication complexity and randomness.

Proof of theorem 5.1 (a). We note that the number of possible transcripts is $\leq 2^c$, and the number of augmentations is also $\leq 2^c$. Then it follows that the total number of internal nodes is $2^{O(c)}$.

We can thus write out the entire tree in $2^{O(c)}\mathsf{poly}(n)$ time, and furthermore compute the value in the same amount by recursion. To calculate the value, we associate each node with its consistent random strings, and then at every verifier step we iterate over the random strings and take the weighted average. Thus, at each verifier node, the step partitions the randomness among its children (this is not the case with the prover, since the prover cannot be controlled using the verifier randomness). □

5.3 IPs with Unlimited Randomness

We will now consider the class $\mathbf{IP}[\mathsf{pc}, \mathsf{vc}, *]$ of interactive proofs where the verifier randomness is unlimited. Intuitively this allows for a somewhat greater class of languages to be accepted, dependent on $\mathbf{DTIME} \overset{?}{=} \mathbf{BPTIME}$. We let $c = \mathsf{pc} + \mathsf{vc}$.

The main addition to the game trees of $\mathbf{IP}[\mathsf{pc}, \mathsf{vc}, *]$ is that the number of augmentations is now $2^{\mathsf{poly}(n)}$, since a polynomial time verifier can make use of as many random bits as it desires up to some $\mathsf{poly}(n)$. We can thus not construct the game tree in the allowed time. However, since we are working in **BPTIME**, we will use a randomized algorithm to approximate $\mathsf{val}(T)$.

vr is still defined, albeit as a function of n.

1. Sample $R = \{r_1, r_2, \ldots, r_n\} \in \{0,1\}^{\mathsf{vr}}$, with $m = \Theta(2^c \cdot c)$.
2. Compute $\mathsf{val}(T[R])$, where $T[R]$ is the residual game tree obtained by omitting the randomness not in R.

Figure 7: Probabilistic Algorithm for $\mathbf{IP}[\mathsf{pc}, \mathsf{vc}, *]$.

The algorithm runs in time $2^{O(c)}\mathsf{poly}(n)$, since $|T[R]| = 2^{O(c)}|R| = 2^{O(c)}$.

Lemma 5.1. $\Pr_R\left[|\mathsf{val}(T[R]) - \mathsf{val}(T)| \leq \frac{1}{10}\right] \geq \frac{99}{100}$.

Proof. We use a concentration argument. Begin by defining $\mathcal{V}^R$ to be the verifier when it remains constrained to the randomness in R versus $\mathcal{V}$ to be the verifier when it chooses vr bits of randomness. It follows that

$$\mathsf{val}(T) = \big[\text{maximum acceptance probability of } \mathcal{V}^R \text{ with any prover strategy}\big]$$

For any given prover strategy $\tilde{\mathcal{P}}$, we define

$$\begin{aligned}\Delta(\tilde{\mathcal{P}}, R) &= \Pr_{r\in R}\left[\left\langle \tilde{\mathcal{P}}, \mathcal{V}(x;r) = 1\right\rangle\right] - \Pr_{r\in\{0,1\in \mathsf{vr}\}}\left[\left\langle \tilde{\mathcal{P}}, \mathcal{V}(x;r) = 1\right\rangle\right] \\ &= \Pr\left[\left\langle \tilde{\mathcal{P}}, \mathcal{V}^R(x) = 1\right\rangle\right] - \Pr\left[\left\langle \tilde{\mathcal{P}}, \mathcal{V}(x) = 1\right\rangle\right]\end{aligned}$$

We will now show that $|\Delta(\tilde{\mathcal{P}}, R)|$ is small over the choice of R. To do this, define the variables $z_i = \langle \tilde{\mathcal{P}}, \mathcal{V}(x; r_i) = 1\rangle$ where r_i is the i^{th} string in R. Then the variables z_i are independently distributed, since r_i are independently distributed.

Furthermore, we have $\mathbb{E}[z_i] = \Pr\left[\langle \tilde{\mathcal{P}}, \mathcal{V}(x) = 1\rangle\right]$, and that

$$\frac{z_1 + \cdots + z_m}{m} = \Pr\left[\left\langle \tilde{\mathcal{P}}, \mathcal{V}^R(x) = 1\right\rangle\right]$$

Then we can bound the probability immmediately by using a Chernoff bound to get

$$\Pr\left[\left|\Delta(\tilde{\mathcal{P}}, R)\right| \geq \frac{1}{10}\right] \leq \Pr\left[\left|\frac{z_1 + \cdots + z_m}{m} - \mathbb{E}[z_i]\right| \geq \frac{1}{10}\right] \leq 2e^{-2m(1/10)^2}$$

Thus for any *particular* prover, the probability that $|\Delta(\tilde{\mathcal{P}}, R)|$ is large is quite low. We consider all possible provers, the number of which is 2^{c2^c}, since each prover is a function from a partial transcript to the next prover string, and thus less than $(2^c)^{2^c}$. Taking a union bound over all provers, we get

$$\Pr_R\left[\exists \tilde{\mathcal{P}} : \left|\Delta(\tilde{\mathcal{P}}, R)\right| \geq \frac{1}{10}\right] \leq 2^{c2^c} \cdot 2e^{-2m(1/10)^2} \leq \frac{1}{100}$$

if we choose the right value of m, which is $\Theta(c2^c)$. □

The above serves as a proof for Theorem 5.1 (b).

6 Delegating Computation and the GKR Protocol (Incomplete)

Until now we have considered interactive proofs with a polynomial time verifier. While the formal definition of IPs allows the prover to have unbounded computational power, we often consider situations in which the honest prover also works efficiently.

Suppose we are given a boolean formula $\phi = \phi(x_1, \ldots, x_n)$. Then,

- in the sumcheck protocol for $\#\mathsf{SAT}$, the prover takes time $\Omega(2^n|\phi|)$, since it has to sum over all possible values of $(x_1, \ldots, x_n)$.
- in Shamir's protocol for TQBF, the prover similarly has to take time $\Omega(2^n|\phi|)$.

These times are too high to be practically useful for computation. Suppose we have a machine M that runs in time T and in space S. Then, we can reduce the language $L_m = \{x : M(x) = 1\}$ to a TQBF instance $\phi(x_1, \ldots, x_n)$, where

$$n \geq \log T \cdot S$$

Even if both $T, S \in \mathsf{poly}(n)$, it follows that the prover time is $\Omega(2^n) = \Omega(T^S) = n^{\omega(1)}$.

6.1 Doubly-Efficient Interactive Proofs

Definition 6.1 (Doubly-Efficient Interactive Proof)**.** *A deIP is an interactive proof in which the prover is additionally constrained to run in polynomial time.*

Intuitively, deIPs are interactive proofs which are 'computationally useful,' ie. can be used to delegate computation. These are particularly useful when the verifier requires less resources to run than calculating the proof itself – for instance, instead of performing a heavy $O(n^6)$ algorithm, the verifier can delegate the task to the prover and simply check the solution using only $O(n)$ or $O(n \log n)$ time. In special cases, the prover may even need to run for time sublinear in the problem length.

We now give a first characterization of **deIP**.

Theorem 6.1. deIP $\subseteq$ BPP*.*

Proof. The probabilistic algorithm simply simulates the interaction between prover and verifier, which it can do in polynomial time and using the verifier's randomness. □

6.2 Delegation for Bounded-Depth Circuits

Definition 6.2 (S-Space Uniform Circuit Family)**.** *A circuit family $\mathcal{C}_i = \{C\}_{i\in\mathbb{N}}$ is S-space uniform if there exists a machine M such that $M(1^n) = C_n$ and M runs in space $O(S(n))$.*

We now see a theorem that relates **IP** and languages decidable by circuits.

Theorem 6.2. *Suppose that L is decidable by an $O(\log S)$-space uniform circuit family of size S and depth D. Then L has a public coin IP such that*

- *the prover time is* $\mathsf{poly}(S)$,
- *the verifier time is* $(n + D) \cdot \mathsf{polylog}(S)$ *and space is* $O(\log S)$,
- *the total communication is* $D \cdot \mathsf{polylog}(S)$.

The proof of this theorem is long and technical. We will see only the bare-bones protocol, in which the verifier has query access to the circuit's topology. We will thus not need to discuss uniformity.

The protocol we will discuss is called the GKR protocol and is highly efficient in practice. [GKR15] was introduced by Goldwasser, Kalai and Rothblum in 2008 and uses ideas of arithmetization and sumcheck. Furthermore, GKR can be modified so that the prover runs in linear time, and the verifier can in fact verify the computation using time sublinear in the size of the circuit.

GKR is also interesting in terms of providing an IP for the circuit complexity class **NC**. Using GKR yields a polytime prover and a linear time verifier.

6.2.1 Layered Arithmetic Circuits

Definition 6.3 (Layered Arithmetic Circuit)**.** *An arithmetic circuit $C : \mathbb{F}^n \to \mathbb{F}$ of size S, depth D and fan-in 2 arranged in $D + 1$ layers.*

Layer 0	$V_0 \in \mathbb{F}$	
Layer 1	$V_1 : S \to \mathbb{F}$	$\mathsf{add}_1 : [S]^3 \to \{0,1\}$
		$\mathsf{mul}_1 : [S]^3 \to \{0,1\}$
Layer 2	$V_2 : S \to \mathbb{F}$	$\mathsf{add}_2 : [S]^3 \to \{0,1\}$
		$\mathsf{mul}_2 : [S]^3 \to \{0,1\}$
$\vdots$	$\vdots$	$\vdots$
Layer $D-1$	$V_{D-1} : S \to \mathbb{F}$	$\mathsf{add}_{D-1} : [S]^3 \to \{0,1\}$
		$\mathsf{mul}_{D-1} : [S]^3 \to \{0,1\}$
Layer D	$V_D : S \to \mathbb{F}$	$\mathsf{add}_D : [S] \times [n]^2 \to \{0,1\}$
		$\mathsf{mul}_D : [S] \times [n]^2 \to \{0,1\}$

Figure 8: A Layered Circuit.

Figure 8 gives a functional representation of a layered circuit. At each level are gates, which contain either $+$ or $\times$ and represent the sum or product of the input wires into the gate. We can additionally define two functions add_i and mul_i.

$$\mathsf{add}_i(g_j, w_k, w_l) = \begin{cases} 1 & \text{if } g_i \text{ is a } + \text{ gate with input wires from nodes } w_k \text{ and } w_l, \\ 0 & \text{otherwise.} \end{cases}$$

$$\mathsf{mul}_i(g_j, w_k, w_l) = \begin{cases} 1 & \text{if } g_i \text{ is a } \times \text{ gate with input wires from nodes } w_k \text{ and } w_l, \\ 0 & \text{otherwise.} \end{cases}$$

add and mul are called *wiring predicates.* For notational simplicity, we assume that C has only one kind of gate, $g : \mathbb{F}^n \to F$.

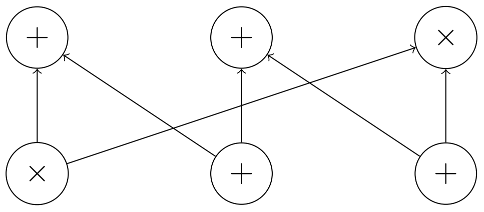

Figure 9: Two Layers of a Circuit.

6.2.2 Low-Degree Extensions

Definition 6.4 (Polynomial Extension). *Let $H \subseteq \mathbb{F}$ be a domain and $f : H \to \mathbb{F}$ be a function. Then a polynomial $p \in \mathbb{F}[x]$ is called an extension of f if $p|_H = f$.*

We are interested in low-degree extensions, in which the polynomial has a low degree. The lowest degree (or minimal) extension is the extension obtained through Lagrange Interpolation: the degree is $< |H|$, and can be calculated as

$$p(x) = \sum_{\alpha \in H} f(\alpha) \left[\prod_{\beta \in H \setminus \{\alpha\}} \left(\frac{x - \beta}{\alpha - \beta} \right) \right]$$

The term in the square braces is called the Lagrange polynomial, and we represent it as $L_{\alpha,H}(x)$. We can easily extend this to the multivariate case. It follows that

- $p \in \mathbb{F}[x_1, \ldots, x_n]$ extends $f : H^n \to \mathbb{F}$ if $p|_{H^n} = f$.
- the extension of minimum degree has individual degree $< |H|$ and equals

  $$p(x_1, \ldots, x_n) = \sum_{\alpha_1, \ldots, \alpha_n \in H} f(\alpha_1, \ldots, \alpha_n) \left(\prod_{\alpha_i} L_{\alpha_i,H} \right)$$

  where the product of the lagrange polynomials is the multivariate lagrange polynomial $L_{\alpha_1 \ldots, \alpha_n, H^n}$.

6.2.3 Layer Arithmetization

7 Zero-Knowledge Proof Systems

Until now we have studied the benefits of interaction and randomness in proof systems, and seen how **IP** can capture several classes beyond **NP** such as $\mathbf{P}^{\#\mathbf{P}}$, *co***NP** and **PSPACE**. We have also seen how IPs can help in delegating classes of computation such as **NC**. We will now turn our attention to an additional property conferred to interactive proofs, one of high cryptographic importance: zero-knowledge, introduced by Goldwasser, Micali and Rackoff in the same paper that they introduced IP [GMR85].

Informally, zero-knowledge is a property that protects the privacy of the prover – in particular, zero-knowledge IPs do not 'give away' *why* a statement is true. To begin understanding how and why zero-knowledge works, we will consider a first example of a zero-knowledge IP.

7.1 Zero-Knowledge IP for Graph Isomorphism

The problem of graph isomorphism, GI, is in **NP** since for any instance $\{G_0, G_1\}$, a polysize proof consisting only of the permutation φ such that $\varphi(G_0) = G_1$ is enough. However, this proof does not have the property of being zero-knowledge since the verifier gains additional information apart from simply the bit indicating whether the instance is true or not – it also gains the permutation.

The challenge is to define an IP that correctly verifies GI while also maintaining the prover's privacy, ie. the permutation remains a secret. We consider the following protocol, in which we assume that if $G_0 \cong G_1$, then $\mathcal{P}$ possesses the witness σ.

1. $\mathcal{P}$ determines the permutation σ such that $\sigma(G_1) = G_0$. After this, it samples a random permutation $\phi : [n] \to [n]$, determines $H = \phi(G_0)$ and sends H to $\mathcal{V}$.
2. On receiving H, $\mathcal{V}$ samples a bit $b \leftarrow \{0, 1\}$ and sends it to $\mathcal{P}$.
3. $\mathcal{P}$ calculates $\psi = \phi \circ \sigma^b$ and sends ψ to $\mathcal{V}$.
4. $\mathcal{V}$ checks if $\psi(G_b) = H$. If yes, it accepts, otherwise it rejects.

Figure 10: Zero-Knowledge Protocol for GI.

Theorem 7.1. *The IP in 10 is a correct IP for* GI.

Proof. As usual, we argue both correctness and soundness.

- Correctness: If $G_0 \cong G_1$, then whatever bit $\mathcal{V}$ sends, it follows that $\psi(G_b) = \phi(\sigma^b(G_b)) = \phi(G_0) = H$. Thus, an honest prover can always convince the verifier, regardless of the verifier's challenge.
- Soundness: If $G_0 \ncong G_1$, then with $1/2$ probability the verifier rejects, since it can only be isomorphic either to G_0 or G_1.

□

It should be clear why the proof is zero-knowledge and why $\mathcal{V}$ does not learn any information about the permutation – the intuitive reason is that the permutation is composed with a randomly picked ϕ, which 'shields' the true witness. However, one can argue that $\mathcal{V}$ does learn some information about the graph H. Thus before we provide a full proof of zero-knowledge, we will formalize the concept through some definitions.

7.2 Honest Verifier Zero-Knowledge

Definition 7.1 (Honest-Verifier Zero-Knowledge)**.** *An interactive proof $(\mathcal{P}, \mathcal{V})$ is said to be honest-verifier zero-knowlege if there exists a probabilistic polynomial time simulator $\mathcal{S}$ such that*

$$\forall x \in L, \mathcal{S}(x) \equiv \mathsf{View}_v \left(\langle \mathcal{P}, \mathcal{V} \rangle (x)\right)$$

where $\mathsf{View}_v \left(\langle \mathcal{P}, \mathcal{V} \rangle (x)\right) := (r, x, a_1, \ldots, a_n)$ is all the information seen by the verifier during the protocol, including its randomness, the problem instance, and all the prover messages.

Since there is an element of randomness involved here, we claim that the distributions of the simulator outputs and the actual protocol views are equal. The interpretation is that the honest verifier could have simulated the entire transcript without any interaction from the honest prover at all, and thus could not have gained any information during the protocol execution which it would not have otherwise gained.

Note the importance of *honesty* in this definition. We only deal with honest provers since malicious provers have no incentive to safeguard their own privacy, and can thus simply send the witness with no consequence, making the definition meaningless. Furthermore, the definition also does not protect against a malicious verifier – if the verifier deviates from the protocol (in a covert way, of course, such as sending pre-chosen, non-random bits to the prover) then it can still potentially gain some knowledge.

Thus, HVZK is a joint property of the honest prover and honest verifier. We define the complexity class **HVZK** to be the class of all languages such that there exists an HVZK IP that decides them.

7.2.1 HVZK for the GI Protocol

Theorem 7.2. *The IP in 10 is HVZK.*

Proof. We construct a simulator which outputs $\mathsf{View}_v \left(\langle \mathcal{P}, \mathcal{V} \rangle (x)\right)$ given no extra information other than $x \in L$. Let $\mathcal{S}$ proceed as follows:

1. First, $\mathcal{S}$ samples a bit $b \leftarrow \{0, 1\}$.
2. It samples a permutation $\psi : [n] \to [n]$.
3. It determines a permuted graph $H = \psi(G_b)$.
4. It outputs the tuple (G_0, G_1, H, b, ψ).

This is precisely the verifier's view, which consists of (G_0, G_1, H, b, ψ) with b random, ψ random (since ϕ is random) and ψ such that $\psi(H) = G_0$ or G_1 with $1/2$ probability. The two distributions are equal, since the probability that any given element is picked either in the honest protocol execution or by the simulator is the same. □

7.3 Malicious Verifier Zero-Knowledge

Definition 7.2 ((Malicious Verifier/Perfect) Zero-Knowledge)**.** *An interactive proof* $(\mathcal{P}, \mathcal{V})$ *is said to be (malicious-verifier/perfect) zero-knowlege if there exists a probabilistic polynomial time (in expectation) simulator* $\mathcal{S}$ *such that*

$$\forall x \in L, \forall \, ppt \, \tilde{\mathcal{V}}, \;\; \mathcal{S}(\tilde{\mathcal{V}}, x) \equiv \mathsf{View}_v \left(\left\langle \mathcal{P}, \tilde{\mathcal{V}} \right\rangle (x) \right).$$

The idea is that regardless of the strategy that the verifier uses, it cannot gain any information besides the problem statement and any internal information it already possesses. The simulator is given access to the verifier to produce a convincing transcript, so that it understands the ways in which the verifier might deviate from the protocol and generate an equal distribution regardless.

The complexity class of problems with Malicious Verifier Zero-Knowledge proof systems is called **PZK** or simply **ZK**. We can immediately see two lemmas:

Lemma 7.1. PZK $\subseteq$ HVZK $\subseteq$ IP.

Lemma 7.2. BPP $\subseteq$ PZK.

The latter follows from the fact that the prover doesn't need to send anything at all, and the verifier can just find whatever it wants out by itself.

7.3.1 ZK for the GI Protocol

Theorem 7.3. *The IP in 10 is in* **ZK***.*

Proof. We use the same idea in the proof for HVZK. Our simulator is constructed as follows:

1. $\mathcal{S}$ samples the bit $b \to \{0, 1\}$.
2. It samples a permutation $\psi : [n] \to [n]$.
3. It determines the permuted graph $H = \psi(G_b)$.
4. $\mathcal{S}$ queries the verifier for its random bit $\tilde{b}$.
5. If $\tilde{b} = b$, then proceed. Else revert to step 1.
6. Output the tuple (G_0, G_1, H, b, ψ).

Since $G_0 \cong G_1$, H is independent of b, and furthermore b is independent of $\tilde{b}$. We thus have that $\Pr[b = \tilde{b}] = 1/2$, and it follows that the expected number of trials before $b = \tilde{b}$ is 2. Thus the simulator runs in probabilistic polynomial time.

It is clear the simulator is correct, since it only ouptuts the actual view of $\tilde{\mathcal{V}}$ (it can ensure this by querying $\tilde{\mathcal{V}}$ every time). Furthermore, the distribution of H and ψ is random, and the distribution of b is exactly the distribution of the b in the protocol, since

$$\Pr\left[b = 0 \mid b = \tilde{b}\right] = \frac{\Pr\left[b = 0\right] \Pr\left[b = \tilde{b}\right]}{\Pr\left[b = \tilde{b}\right]} = \Pr\left[b = 0\right].$$

□

7.4 Limitations of Zero-Knowledge

Theorem 7.4 ([For87]). **HVZK**=**AM** $\cap$ *co***AM**.

The previous theorem gives a somewhat unfortunate characterization of zero-knowledge proofs: the class of languages containing even an honest-verifier ZK proof is small enough that even **NP** is unlikely to be a subset. Furthermore this limitation holds even if we allow the simulator to be statistically indistinguishable from the view of the honest verifier.

However, we are still interested in finding zero knowledge proofs for problems in **NP** and beyond. To do this, we will look at ways we can bypass these limitations.

7.4.1 Relaxing Indistinguishability

The first technique that we can employ to extend the class of zero-knowledge proofs is to consider proofs in which the simulator and the view are *computationally* indistinguishable in contrast to statistically indistinguishable. We can thus define the classes **CZK** and **HVCZK**, representing computational zero-knowledge and honest-verifier computational zero knowledge, respectively.

Definition 7.3 (Computational Indistinguishability). *Let $\{X_n\}_n$ and $\{Y_n\}_n$ be a sequence of distribitions, where X_n and Y_n are distributions over $\{0,1\}^{\ell(n)}$ for some polynomial $\ell(n)$. We say that $\{X_n\}_n$ and $\{Y_n\}_n$ are computationally indistinguishable if for all non-uniform PPT deciders $\mathcal{D}$, there exists a negligible function ϵ such that*

$$|\Pr[t \leftarrow X_n : \mathcal{D}(t) = 1] - \Pr[t \leftarrow Y_n : \mathcal{D}(t) = 1]| \leq \epsilon(n).$$

It turns out that relaxing our definition to computational indistinguishability gives us a great deal more benefit over equality or statistical closeness.

Theorem 7.5 ([IY87]). **CZK** $=$ **IP**.

7.4.2 Changing the Model

We can gain additional power by relaxing the model of computation we are working in. For instance, consider the class of interactive proofs **MIP**, where the verifier is allowed to interact with multiple provers. The following result characterizes ZK in multiprover interctive proofs:

Theorem 7.6. **PZKMIP** = **MIP**.

The theorem reduces the use of cryptography to a physical condition, simply ensuring that the provers can't communicate.

7.5 HVZK to IP

We now consider the following lemma.

Lemma 7.3. *If $L \in$ **HVZK**$[k]$, then $\overline{L} \in$ **IP**$[O(k)]$.*

We can thus use zero-knowledge proofs to find interactive proofs for the corresponding complement language. We will see this technique applied to the case of GI, using which we will obtain another IP for GI.

Proof. We heavily use the properties of the simulator $\mathcal{S}$. It is clear that in the case of $x \in L$, we have that $\mathcal{S}$ outputs a distribution that is equal to the distribution $\mathsf{View}_v\left(\langle \mathcal{P}, \mathcal{V} \rangle (x)\right)$. If, however, $x \notin L$, then $\mathcal{S}$ must output something: in particular, it can output any of the following three.

- Complete gibberish.
- A non-accepting view.
- An accepting view.

We notice that it is possible to run the simulator and check whether $x \in L$ if the outputs are the first two possibilites. Thus for $L \notin$ **BPP**, it is likely that the output will be the third possibility. However, in that case the distributions of the view and that of the simulator will be unequal, and it is possible to exploit this to gain an IP for $\overline{L}$. In particular, the IP picks random instances of $\mathcal{S}$'s output and sends it to $\mathcal{P}$, asking it to prove that the sample is not from the actual distribution of views. □

7.5.1 An IP for GNI using GI $\in$ HVZK

We recall the simulator for GI from Theorem 7.2. The verifier runs this simulator to obtain a random bit b, a random permutation ψ, and a graph H. It then sends H to $\mathcal{P}$ and asks for a bit $\tilde{b}$ such that $H = \psi(G_{\tilde{b}})$. If $b = \tilde{b}$ we accept, otherwise we reject.

The proof of correctness and soundness are trivial. However, notice that if an instance was not in GNI, then $\mathcal{S}$ is indeed equal to the view and $\mathcal{P}$ could only choose the correct $\tilde{b}$ with half probability, exactly the same as the original choice.

8 Probabilistically Checkable Proofs

References

[Bab85] L Babai. Trading group theory for randomness. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC '85, page 421–429, New York, NY, USA, 1985. Association for Computing Machinery.

[For87] L. Fortnow. The complexity of perfect zero-knowledge. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, page 204–209, New York, NY, USA, 1987. Association for Computing Machinery.

[GH98] Oded Goldreich and Johan Håstad. On the complexity of interactive proofs with bounded communication. *Inf. Process. Lett.*, 67(4):205–214, aug 1998.

[GKR15] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for muggles. *J. ACM*, 62(4), sep 2015.

[GMR85] S Goldwasser, S Micali, and C Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC '85, page 291–304, New York, NY, USA, 1985. Association for Computing Machinery.

[GS86] S Goldwasser and M Sipser. Private coins versus public coins in interactive proof systems. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, STOC '86, page 59–68, New York, NY, USA, 1986. Association for Computing Machinery.

[IY87] Russell Impagliazzo and Moti Yung. Direct minimum-knowledge computations. In *A Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology*, CRYPTO '87, page 40–51, Berlin, Heidelberg, 1987. Springer-Verlag.

[Lau83] Clemens Lautemann. Bpp and the polynomial hierarchy. *Information Processing Letters*, 17(4):215–217, 1983.

[LFKN92] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, oct 1992.

[Sha92] Adi Shamir. Ip = pspace. *J. ACM*, 39(4):869–877, oct 1992.