Advanced Computational Complexity

George Voutsadakis[1]

[1]Mathematics and Computer Science
Lake Superior State University

LSSU Math 600

Subsection 1

Interactive Proofs: Some Variations

Introducing Interaction

- Interactive proofs introduce interaction into the basic NP scenario.
  - Instead of the prover sending a written proof to the verifier, the verifier conducts an interrogation of the prover, repeatedly asking questions and listening to the prover's responses.
  - At the end, the verifier decides whether or not to accept the input.
  - The message of each party at any point in the interaction can depend upon messages sent and received previously.
- The prover is assumed to be an all-powerful machine.
- However, as we will see, it suffices to assume it is a PSPACE machine.
- Additional choices to make:
  (a) Is the prover deterministic or probabilistic?
  (b) Is the verifier deterministic or probabilistic?
  (c) If we allow probabilistic machines, how do we define "accept" and "reject"?

Interactive Proofs with Deterministic Verifier and Prover

- For interactive proofs with deterministic verifier and prover, consider a trivial example of such an interactive proof for membership in 3SAT.
  - Proceeding clause by clause, the verifier asks the prover to announce the values for the literals in the clause.
  - The verifier keeps a record of these answers, and accepts at the end if:
    - All clauses were indeed satisfied;
    - The prover never announced conflicting values for a variable.

  Both verifier and prover are deterministic.
- In this case, we may well ask what the point of interaction is, as the prover could just announce values of the literals for all clauses in the very first round, and then take a nap from then on.
- We will soon see this is a subcase of a more general phenomenon.

  Interactive proofs with deterministic verifiers never need to last more than a single round.

Interaction of Deterministic Functions

- The verifier and prover may be two deterministic functions that, at each round, compute the next question/response as a function of:
  - The input;
  - The questions and responses of the previous rounds.

Definition (Interaction of Deterministic Functions)

Let $f, g : \{0,1\}^* \to \{0,1\}^*$ be functions.
Let $k \geq 0$ be an integer (allowed to depend upon the input size).
A **k-round interaction of f and g on input $x \in \{0,1\}^*$**, denoted by

$$\langle f, g \rangle(x),$$

is the sequence of strings $a_1, \ldots, a_k \in \{0,1\}^*$, defined as follows.

Interaction of Deterministic Functions (Cont'd)

Definition (Interaction of Deterministic Functions (Cont'd))

- $a_1 = f(x)$;
- $a_2 = g(x, a_1)$;

  $\vdots$
- $a_{2i+1} = f(x, a_1, \ldots, a_{2i})$, for $2i < k$;
- $a_{2i+2} = g(x, a_1, \ldots, a_{2i+1})$, for $2i + 1 < k$.

The **output** of f at the end of the interaction, denoted

$$\text{out}_f \langle f, g \rangle (x),$$

is defined to be $f(x, a_1, \ldots, a_k)$.
We assume this output is in $\{0, 1\}$.

Deterministic Proof Systems

Definition (Deterministic Proof Systems)

We say that a language L has a k-**round deterministic interactive proof system** if there is a deterministic TM V that, on input $x, a_1, \ldots, a_i$, runs in time polynomial in $|x|$, and can have a k-round interaction with any function P, such that the following implications hold:

- **Completeness**

$$x \in L \Rightarrow \exists P : \{0,1\}^* \to \{0,1\}^* \text{out}_V\langle V, P\rangle(x) = 1;$$

- **Soundness**

$$x \notin L \Rightarrow \forall P : \{0,1\}^* \to \{0,1\}^* \text{out}_V\langle V, P\rangle(x) = 0.$$

The class dIP contains all languages with a $k(n)$-round deterministic interactive proof system, where $k(n)$ is polynomial in n.

Comments

- This definition places no limits on the computational power of the prover P.
- This makes intuitive sense, since a false assertion should not be provable, no matter how clever the prover.
- Because we place no such limits, it does not matter that we allow the prover in the completeness and soundness conditions to depend on x.

dIP = NP

- dIP is actually a class we know well.

Lemma

$$\text{dIP} = \text{NP}.$$

- Every NP language has a one-round deterministic proof system.

  Thus, it is in dIP and $\text{NP} \subseteq \text{dIP}$.

  Now we prove that, if $L \in \text{dIP}$, then $L \in \text{NP}$.

  Let V be a verifier for L.

  A certificate that an input is in L is just a transcript $(a_1, a_2, \ldots, a_k)$ causing the verifier V to accept.

dIP = NP (Cont'd)

- To verify this transcript, one checks that :
  - $V(x) = a_1$;
  - $V(x, a_1, a_2) = a_3$;
  - $\vdots$
  - $V(x, a_1, \ldots, a_k) = 1$.

  If $x \in L$, then such a transcript exists.

  Conversely, suppose such a transcript $(a_1, \ldots, a_k)$ exists.
  Then we can define a prover function P to satisfy:
  - $P(x, a_1) = a_2$;
  - $P(x, a_1, a_2, a_3) = a_4$;
  - $\vdots$

  This deterministic prover satisfies

$$\text{out}_V\langle V, P\rangle(x) = 1.$$

  This implies $x \in L$.

Probabilistic Verifier

- In order for interaction to provide any benefit (decide more that NP), we need to let the verifier be probabilistic.
  - This means that the verifier's questions will be computed using a probabilistic algorithm.
  - Further, the verifier will be allowed to come to a wrong conclusion, e.g., accept a proof for a wrong statement, with some small probability.
    - As in the case of probabilistic algorithms, this probability is over the choice of the verifier's coins.
  - Moreover, we require the verifier to reject proofs for a wrong statement with good probability regardless of the strategy the prover uses.
- Allowing this combination of interaction and randomization has a huge effect.
- We will see that the set of languages that have such interactive proof systems jumps from NP to PSPACE.

Example

- Marla has one red sock and one yellow sock.
- Her friend Arthur, who is color-blind, does not believe her that the socks have different colors.
- How can she convince him that this is really the case?
- Marla gives both socks to Arthur.
- She tells him which sock is yellow and which one is red.
- Arthur holds the red sock in his right hand and the yellow sock in his left hand.
- Then Marla turns her back to Arthur and he tosses a coin.
  - If the coin comes up "heads", then Arthur keeps the socks as they are;
  - Otherwise, he switches them between his left and right hands.

Example (Cont'd)

- He then asks Marla to guess whether he switched the socks or not.
- Marla can easily do so by seeing whether the red sock is still in Arthur's right hand or not.
- If the socks were identical, then she would not be able to guess the answer with probability better than $\frac{1}{2}$.
- Thus, if Marla manages to answer correctly in all of, say, 100 repetitions of this game, then Arthur can indeed be convinced that the socks have different colors.
- In the example, the verifier, being colorblind, has less power (i.e., fewer capabilities) than the prover.
- In general interactive proofs, the verifier, being polynomial-time, also has less computational power than the prover.

Introducing Randomness in the Protocol

- To model an interaction between f and g, where f is probabilistic, we add an additional m-bit input r to the function f,
  - $a_1 = f(x, r)$;
  - $a_3 = f(x, r, a_1, a_2)$;
  - $\vdots$
- The function g is evaluated only on the a_i's and does not get r as an additional input, since the prover is not supposed to be able to "see" the verifier's coins but only his messages.
- For this reason, this is called the **private coins model** for interactive proofs, as opposed to the **public coins model**.
- The interaction $\langle f, g \rangle(x)$ is now a random variable over $r \in_R \{0,1\}^m$.
- Similarly, the output $\text{out}_f \langle f, g \rangle(x)$ is also a random variable.

The Class IP[k]

Definition (Probabilistic Verifiers and the Class IP[k])

For an integer $k \geq 1$ (that may depend on the input length), we say that a language L is in IP[k] if there is a probabilistic polynomial-time Turing machine V that can have a k-round interaction with a function $P : \{0,1\}^* \to \{0,1\}^*$, such that:

- **Completeness**

$$x \in L \;\Rightarrow\; \Pr[\text{out}_V\langle V, P\rangle(x) = 1] \geq \frac{2}{3};$$

- **Soundness**

$$x \notin L \;\Rightarrow\; \Pr[\text{out}_V\langle V, P\rangle(x) = 1] \leq \frac{1}{3}.$$

All probabilities are over the choice of r.

The Class IP

Definition (The Class IP)

We define

$$\mathsf{IP} = \bigcup_{c \geq 1} \mathsf{IP}[n^c].$$

Error Reduction

- We show, next, that the probabilities $\frac{2}{3}$ and $\frac{1}{3}$ can be made arbitrarily close to 1 and 0, respectively, by using error reduction as for BPP.

Lemma

The class IP is unchanged if we replace, for any constant $s > 0$:

- The completeness parameter $\frac{2}{3}$ by $1 - 2^{-n^s}$;
- The soundness parameter $\frac{1}{3}$ by 2^{-n^s}.

- The verifier repeats the protocol m times and accepts iff more than $\frac{1}{2}$ the runs resulted in an accept.

  Assume, first, that $x \in L$.

  Suppose a prover can make the verifier accept with probability $\frac{2}{3}$ in each repetition.

  By the Chernoff bound, at the end it succeeds with probability $1 - 2^{-\Omega(m)}$.

Error Reduction (Cont'd)

- If $x \notin L$, we show every prover strategy will fail with high probability.

  Claim: The prover can succeed in each repetition of the protocol with probability only $\frac{1}{3}$, irrespective of what happened in earlier rounds.

  The prover's responses may depend arbitrarily on its responses in earlier repetitions.

  However, soundness holds for all provers, even for one that knows the questions of earlier rounds.

  Chernoff bounds imply that the probability that the prover can succeed in a majority of the repetitions only with probability $2^{-\Omega(m)}$.

  Choosing $m = \mathrm{O}\left(n^s\right)$ completes the proof.

Some Properties of IP

1. Allowing the prover to be probabilistic, that is, allowing the answer function a_i to depend upon some random string used by the prover (and unknown to the verifier), does not change the class IP.

   Suppose that, for any language L, a probabilistic prover P can make a verifier V accept with some probability.

   Then averaging implies that there is a deterministic prover that makes V accept with the same probability.

Some Properties of IP (Cont'd)

2. The prover can use an arbitrary function.

   So it can in principle use unbounded computational power or even compute undecidable functions.

   However, it can be shown that, given any verifier V, we can compute the **optimum prover** (which, given x, maximizes the verifier's acceptance probability) using $\text{poly}(|x|)$ space.

   A fortiori, this can be done using $2^{\text{poly}(|x|)}$ time.

   So we have

$$\text{IP} \subseteq \text{PSPACE}.$$

Some Properties of IP (Cont'd)

3. Replacing the constant $\frac{2}{3}$ with 1 in the completeness requirement does not change the class IP.

   This is a nontrivial fact that can be proved using the characterization of IP as being PSPACE.

4. By contrast, replacing the constant $\frac{1}{3}$ with 0 in the soundness condition is equivalent to having a deterministic verifier.

   Hence, it reduces the class IP to NP.

Some Properties of IP (Cont'd)

5. **Private Coins**: Thus far, the prover functions do not depend upon the verifier's random strings, but only on the messages/questions the verifier sends.

   In other words, the verifier's random string is private.

   Often these are called **private coin interactive proofs**.

   We will also consider the model of **public-coin proofs** (also known as **Arthur-Merlin proofs**).

   In those, all the verifier's questions are simply obtained by tossing coins and revealing them to the prover.

Some Properties of IP (Cont'd)

6. The proof of the error reduction lemma sequentially repeats the basic protocol m times and takes the majority answer.

   It can be shown that we can decrease the probability without increasing the number of rounds using parallel repetition.

   The prover and verifier will run m executions of the protocol in parallel, i.e., by asking all m questions in one go.

   The proof of this fact is easier for the case of public-coin protocols.

Graph Isomorphism

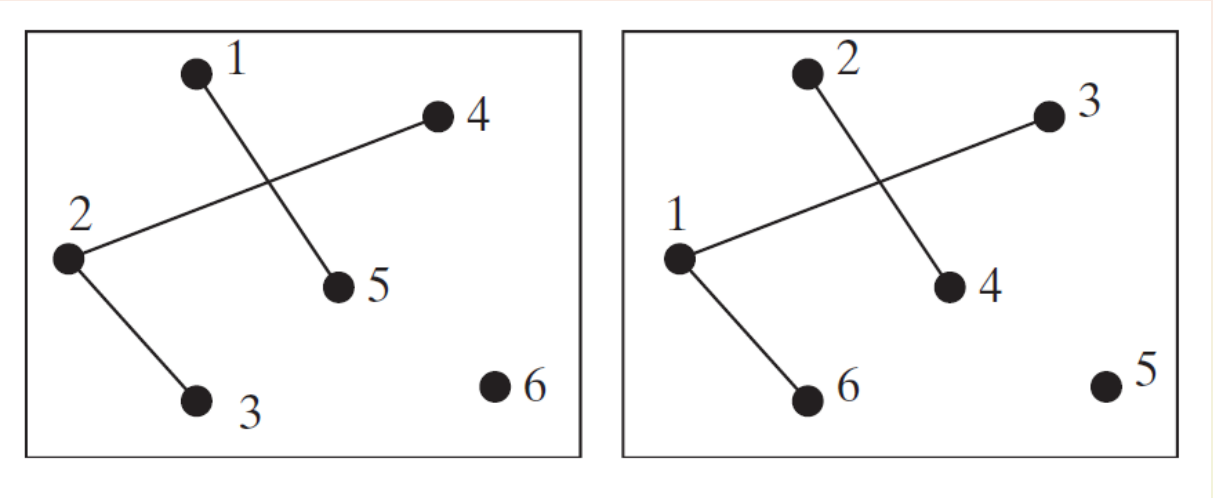
- An example of a language in IP that is not known to be in NP is graph non-isomorphism.
- The usual ways of representing graphs, i.e., adjacency lists or adjacency matrices, involve labeling each vertex with a unique number.
- Two graphs G_1 and G_2 are **isomorphic** if they are the same up to a renumbering of vertices,
- That is, G_1 and G_2 are isomorphic if there is a permutation π of the labels of the nodes of G_1, such that

$$\pi(G_1) = G_2,$$

where $\pi(G_1)$ is the labeled graph obtained by applying π on its vertex labels.

Example

- Consider the following graphs.



- The graphs are isomorphic with

$$\pi = (12)(3654).$$

Graph Isomorphism and Graph Non-Isomorphism

- If G_1 and G_2 are isomorphic, we write $G_1 \cong G_2$.
- The GI problem:

  Given two graphs G_1, G_2, decide if they are isomorphic.

- Clearly, $\mathrm{GI} \in$ NP.

  A certificate is simply the description of the permutation π.
- It is open whether GI is NP-complete.
- Along with the factoring problem, GI is the most famous NP-problem that is not known to be either in P or NP-complete.
- It holds that GI is not NP-complete unless the polynomial hierarchy collapses.
- The first step of this proof is an interactive proof for the complement of GI.

Interactive Proof for Graph Non-Isomorphism

- The complement of GI is the problem GNI:

  Decide whether two given graphs are not isomorphic.

Interactive Proof for GNI (Private-Coin Graph Non-isomorphism)

V: Pick $i \in \{1, 2\}$ uniformly randomly.

Randomly permute the vertices of G_i to get a new graph H.

Send H to P.

P: Identify which of G_1, G_2 was used to produce H.

Let G_j be that graph.

Send j to V.

V: Accept if $i = j$.

Reject otherwise.

Correctness

- If $G_1 \not\cong G_2$, then there exists a prover such that $\Pr[V \text{ accepts}] = 1$.

  Suppose the graphs are nonisomorphic.

  An all-powerful prover can certainly tell which one of the two is isomorphic to H.
- Suppose $G_1 \cong G_2$.

  The best any prover can do is to randomly guess because a random permutation of G_1 looks exactly like a random permutation of G_2.

  Thus, in this case, for every prover,

$$\Pr[V \text{ accepts}] \leq \frac{1}{2}.$$

  This probability can be reduced to $\frac{1}{3}$ by sequential or parallel repetition.

Zero-Knowledge Proofs

- We briefly touch upon zero-knowledge proofs, a topic related to interactive proofs.
- Roughly speaking, a zero-knowledge proof system for membership in a language is an interactive proof protocol, where the verifier:
  - Is convinced at the end that the input x is in the language;
  - Learns nothing else.
- To quantify that the verifier learns nothing else, we show that the verifier could have produced the transcript of the protocol in polynomial time with no help from the prover.

Comments

- The above protocol for graph non-isomorphism is zero-knowledge.
- In cryptography, the concept raises the possibility of parties being able to prove things to each other without revealing any secrets.
- An example may be someone proving to hold the password without revealing the password.

Quadratic Residues

- An example for an interactive proof for a language not known to be in NP is quadratic nonresiduosity.
- A number a is a **quadratic residue** mod p if there is another number b such that
$$a \equiv b^2 \pmod{p}.$$
- Such a b is called the **square root** of $a \pmod{p}$.
- Clearly, $-b$ is another square root.
- There are no other square roots since the equation $x^2 - a$ has at most two solutions over GF(p).

Quadratic Nonresiduosity

- Consider the language QR of pairs (a, p), where:
  - p is a prime;
  - a is a quadratic residue mod p.
- QR is in NP.
  - That p is a prime has a short membership proof (indeed primality can be tested in polynomial time).
  - A square root constitutes a membership proof.
- The language QNR of pairs (a, p) such that p is a prime and a is not a quadratic residue modulo p has no natural short membership proof.
- QNR is not known to be in NP.

Interactive Proof for Quadratic Nonresiduosity

- The language QNR consists of all pairs (a, p) such that p is a prime and a is not a quadratic residue modulo p.
- It has a simple interactive proof if the verifier is probabilistic.

  The verifier takes a random number r mod p and a random bit $b \in \{0, 1\}$ (kept secret from the prover).
  - If $b = 0$, she sends the prover r^2 mod p;
  - If $b = 1$, she sends ar^2 mod p.

  She asks the prover to guess what b was.

  She accepts iff the prover guesses correctly.

Correctness

- Suppose a is a quadratic residue.

  Then the distributions of ar^2 and r^2 are identical.

  A residue a' can be written as as^2 where s is a square root of $\frac{a}{a'}$.

  So, both ar^2 and r^2 are random elements of the group of residues modulo p.

  Thus, the prover has probability at most $\frac{1}{2}$ of guessing b.
- Suppose a is a nonresidue.

  Then the distributions ar^2 and r^2 are completely distinct.
  - The first is a random nonresidue modulo p;
  - The second is a random quadratic residue modulo p.

  An all-powerful prover can tell them apart.

  Thus, it can guess b with probability 1.

Subsection 2

Public Coins and AM

Introducing Public Coins

- Our proof system for graph non-isormorphism and nonresiduosity seemed to crucially rely on the verifier's access to a source of private random coins that are not seen by the prover.
- Allowing the prover full access to the verifier's random string leads to the model of interactive proofs with public coins.

The Class AM

Definition (AM)

For every k, the complexity class AM[k] is defined as the subset of IP[k] obtained when:

- We restrict the verifier's messages to be random bits;
- We are not allowing the verifier to use any other random bits that are not contained in these messages.

An interactive proof where the verifier has this form is called a **public coin proof**, sometimes also known as an **Arthur-Merlin proof**.

- We define AM $=$ AM[2].
- Note that AM $=$ AM[2], while IP $=$ IP[poly].
- Although this is indeed somewhat inconsistent, it is the standard notation used in the literature.

The Class MA

- AM is, thus, the class of languages with an interactive proof that consist of:
  - The verifier sending a random string;
  - The prover responding with a message;
  - The verifier's decision is obtained by applying a deterministic polynomial-time function to the transcript.
- The class MA denotes the class of languages with a two-round public-coin interactive proof with the prover sending the first message.
- So $L \in$ MA if there is a proof system for L that consists of:
  - The prover first sending a message;
  - The verifier, then, tossing coins and computing its decision by doing a deterministic polynomial time computation involving the input, the prover's message and the coins.

Properties of the Class AM[k]

1. Even in a public-coins proof, the prover does not get to see immediately all of the verifier's random coins.

   They are revealed to the prover iteratively message by message.

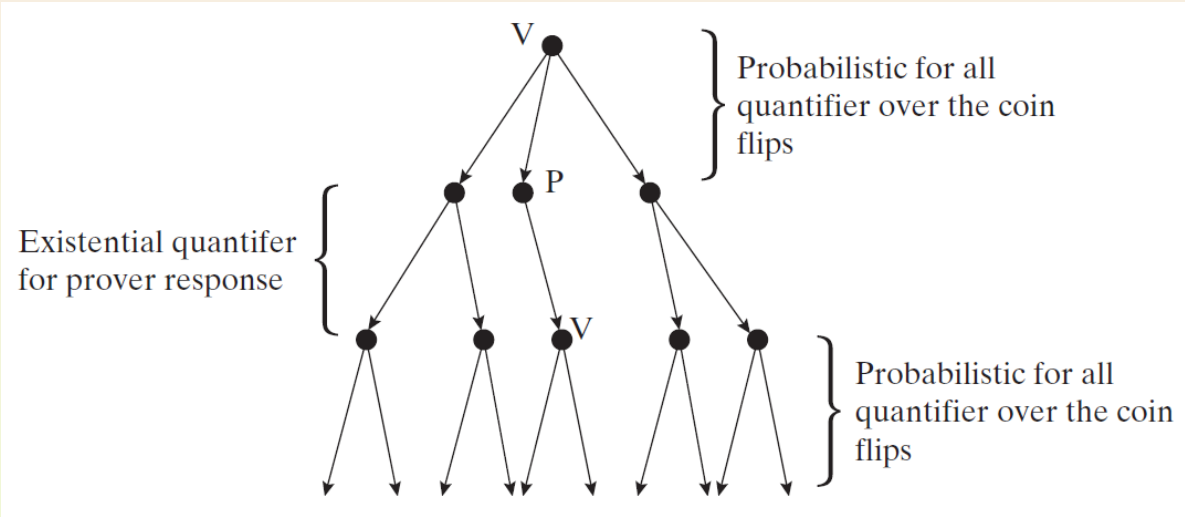
   I.e., an AM[k]-proof is an IP[k]-proof where:
   - The verifier's random tape r consists of $\lceil \frac{k}{2} \rceil$ strings $r_1, \ldots, r_{\lceil k/2 \rceil}$;
   - His i-th message is simply the string r_i;
   - The decision whether to accept or reject is obtained by applying a deterministic polynomial-time computable function to the transcript.
2. AM[2] $=$ BP $\cdot$ NP.

   In particular, it follows that AM[2] $\subseteq \Sigma_3^p$.

Properties of the Class AM[k] (Cont'd)

3. For constants $k \geq 2$, we have AM[k] = AM[2].

   This "collapse" is somewhat surprising because AM[k] at first glance seems similar to PH, with the $\forall$ quantifiers changed to "probabilistic $\forall$" quantifiers, where most of the branches lead to acceptance.



4. There is no known "nice" characterization of AM[$\sigma(n)$], where $\sigma(n)$ is a suitably slowly growing function of n, such as $\log \log n$.

Simulating Private Coins

- For every k, $\mathsf{AM}[k] \subseteq \mathsf{IP}[k]$.
- It may seem that allowing the verifier to keep its coins private adds significant power to interactive proofs, but this is not the case.

Theorem (Goldwasser-Sipser)

For all $k : \mathbb{N} \to \mathbb{N}$, with $k(n)$ computable in $\mathrm{poly}(n)$,

$$\mathsf{IP}[k] \subseteq \mathsf{AM}[k+2].$$

- We sketch the proof of the theorem after proving the next Theorem, which concerns the subcase of GNI.

GNI $\in$ AM[2]

Theorem

GNI $\in$ AM[2].

- We look at graph non-isomorphism in a more quantitative way.

  We consider the set of labeled graphs

  $$S = \{H : H \cong G_1 \text{ or } H \cong G_2\}.$$

  It is easy to certify that a graph H is a member of S, by providing the permutation mapping either G_1 or G_2 to H.

  An n vertex graph G has at most $n!$ equivalent graphs.

  Assume, first, that G_1 and G_2 have each exactly $n!$ equivalent graphs.

  The size of S differs by a factor 2, depending upon whether or not G_1 is isomorphic to G_2.
  - If $G_1 \ncong G_2$, then $|S| = 2n!$;
  - If $G_1 \cong G_2$, then $|S| = n!$.

GNI $\in$ AM[2] (Cont'd)

- Assume, next, that G_1 or G_2 may have less than $n!$ equivalent graphs.
  An n-vertex graph G has less than $n!$ equivalent graphs iff it has a nontrivial automorphism.
  Such an automorphism is a permutation π, that is not the identity permutation, such that $\pi(G) = G$.
  Let $\mathrm{aut}(G)$ denote the set of automorphisms of graph G.
  We change the definition of S to

$$S = \{(H, \pi) : H \cong G_1 \text{ or } H \cong G_2 \text{ and } \pi \in \mathrm{aut}(H)\}.$$

  Using the fact that $\mathrm{aut}(G)$ is a subgroup, one can verify that S satisfies the previous relations.
  Also, membership in this set is easy to certify.
  Thus to convince the verifier that $G_1 \ncong G_2$, the prover has to convince the verifier that $|S| = 2n!$ rather than $n!$.
  This is done by using a set lower bound protocol.

Set Lower Bound Protocol

- Suppose there is a set S known to both prover and verifier, such that membership in S is easily certifiable.
- This means that, given some string x that happens to be in S, the prover, using its superior computational power, can provide the verifier with a certificate.
- More formally, S is in $\mathrm{BP} \cdot \mathrm{NP}$.
- The set lower bound protocol is a public-coins protocol that allows the prover to certify the approximate size of S:
  - The prover can compute and announce $|S|$.
  - The question is how to convince the verifier that this answer is correct, or even approximately correct.

Set Lower Bound Protocol (Cont'd)

- Suppose the prover's claimed value for $|S|$ is K.
- Then the protocol satisfies:
  - Suppose the true value of $|S|$ is indeed at least K.
    Then the prover can cause the verifier to accept with high probability.
  - Suppose the true value of $|S|$ is at most $\frac{K}{2}$.
    That is, the prover's answer is grossly on the high side.
    Then the verifier will reject with high probability, no matter what the prover does.
- This protocol is called the set lower bound protocol.
- The protocol suffices to complete the proof of $\mathrm{GNI} \in \mathrm{AM}[2]$.

Pairwise Independent Hash Functions

- The main tool in the set lower bound protocol is a pairwise independent hash function collection.

Definition (Pairwise Independent Hash Function)

Let $\mathcal{H}_{n,k}$ be a collection of functions from $\{0,1\}^n$ to $\{0,1\}^k$.
We say that $\mathcal{H}_{n,k}$ is **pairwise independent** if, for every $x, x' \in \{0,1\}^n$, with $x \neq x'$, and for every $y, y' \in \{0,1\}^k$,

$$\Pr_{h \in_R \mathcal{H}_{n,k}}[h(x) = y \ \wedge \ h(x') = y'] = 2^{-2k}.$$

- An equivalent formulation is that, for every two distinct but fixed strings $x, x' \in \{0,1\}^n$, when we choose h at random from $\mathcal{H}_{n,k}$, then the random variable $\langle h(x), h(x') \rangle$ is distributed according to the uniform distribution on $\{0,1\}^k \times \{0,1\}^k$.

Relations with GF(2^n)

- We can identify the elements of $\{0,1\}^n$ with the finite field GF(2^n) containing 2^n elements.
- Addition ($+$) and multiplication ($\cdot$) operations in this field are efficiently computable.
- Moreover, they satisfy several well-known algebraic properties:
  - The usual commutative and distributive laws;
  - Every element x has an additive inverse $-x$;
  - Every nonzero element x has a multiplicative inverse x^{-1}.
- We may use these properties to construct a family of efficiently computable pairwise independent hash functions.

Efficient Pairwise Independent Hash Functions

Theorem (Efficient Pairwise Independent Hash Functions)

For every n, define the collection $\mathcal{H}_{n,n}$ by

$$\mathcal{H}_{n,n} = \{h_{a,b}\}_{a,b \in \mathrm{GF}(2^n)},$$

where, for $a, b \in \mathrm{GF}(2^n)$, the function $h_{a,b} : \mathrm{GF}(2^n) \to \mathrm{GF}(2^n)$ is defined by

$$x \mapsto ax + b.$$

Then, $\mathcal{H}_{n,n}$ is a collection of pairwise independent hash functions.

- The theorem implies the existence of an efficiently computable family of pairwise independent hash functions $\mathcal{H}_{n,k}$, for every n, k:
  - If $k > n$ we can use the collection $\mathcal{H}_{k,k}$ and extend n bit inputs to k bits by padding with zeros.
  - If $k < n$, then we can use the collection $\mathcal{H}_{n,n}$ and reduce n bit outputs to k bits by truncating the last $n - k$ bits.

Efficient Pairwise Independent Hash Functions (Cont'd)

- For every $x \neq x' \in \mathrm{GF}(2^n)$ and $y, y' \in \mathrm{GF}(2^n)$, we have

$$\left\{ \begin{array}{rcl} h_{a,b}(x) & = & y \\ h_{a,b}(x') & = & y' \end{array} \right\} \quad \text{iff} \quad \left\{ \begin{array}{rcl} a \cdot x + b & = & y \\ a \cdot x' + b & = & y' \end{array} \right\}$$

  Then

$$a = (y - y')(x - x')^{-1}.$$

  This expression is well defined, since $x - x' \neq 0$.

  We have $b = y - a \cdot x$.

  So the pair $\langle a, b \rangle$ is completely determined by these equations.

  The probability that this happens over the choice of a, b is 1.

  The number of possible pairs are 2^{2n}.

The Set Lower-Bound Protocol

The Set Lower Bound Protocol (Goldwasser-Sipser)

Conditions:

$S \subseteq \{0,1\}^m$ is a set, such that membership in S can be certified.

Both parties know a number K.

The prover's goal is to convince the verifier that $|S| \geq K$.

The verifier should reject with good probability if $|S| \leq \frac{K}{2}$.

Let K be an integer such that

$$2^{k-2} < K \leq 2^{k-1}.$$

The Set Lower-Bound Protocol (Cont'd)

The Set Lower Bound Protocol (Cont'd)

V: Randomly pick a function

$$h : \{0,1\}^m \to \{0,1\}^k$$

from a pairwise independent hash function collection $\mathcal{H}_{m,k}$.

Pick $y \in_R \{0,1\}^k$.

Send h, y to prover.

P: Try to find an $x \in S$, such that $h(x) = y$.

Send such an x to V, together with a certificate that $x \in S$.

V Output:

If $h(x) = y$ and the certificate validates that $x \in S$ then accept.

Otherwise reject.

Correctness of The Set Lower-Bound Protocol

- The prover (being all powerful) can make the verifier accept iff h, y happen to be such that an $x \in S$ exists satisfying $h(x) = y$.

  Let $p^* = \frac{K}{2^k}$.

  There is a gap of $\frac{3}{4}p^*$ versus $\frac{p^*}{2}$ in the probability of the existence of $x \in S$, with $h(x) = y$, depending on whether $|S| \geq K$ or $|S| < \frac{K}{2}$.

  Claim: Let $S \subseteq \{0,1\}^m$ satisfy $|S| \leq \frac{2^k}{2}$.

  Then, for $p = \frac{|S|}{2^k}$,

$$p \geq \Pr_{h \in_R \mathcal{H}_{m,k}, y \in \{0,1\}^k}[\exists x \in S : h(x) = y] \geq \frac{3p}{4} - \frac{p}{2^k}.$$

  Note that the set $h(S)$ of y's with preimages in S has size $\leq |S|$.

  This establishes the upper bound.

Correctness of The Set Lower-Bound Protocol (Cont'd)

- We next show that, for every $y \in \{0,1\}^k$,

$$\Pr_{h \in_R \mathcal{H}_{m,k}}[\exists x \in S \ h(x) = y] \geq \frac{3}{4}p.$$

Indeed, for every $x \in S$, define E_x as the event that $h(x) = y$.

Then,

$$\Pr[\exists x \in S : h(x) = y] = \Pr[\vee_{x \in S} E_x].$$

By Inclusion-Exclusion, this is

$$\geq \sum_{x \in S} \Pr[E_x] - \frac{1}{2} \sum_{x \neq x' \in S} \Pr[E_x \cap E_{x'}].$$

Correctness of The Set Lower-Bound Protocol (Cont'd)

- By pairwise independence, if $x \neq x'$, then

$$\begin{aligned} \Pr[E_x] &= 2^{-k}, \\ \Pr[E_x \cap E_{x'}] &= 2^{-2k}. \end{aligned}$$

So up to the low order term $\frac{|S|}{2^{2k}}$ this probability is at least

$$\frac{|S|}{2^k} - \frac{1}{2}\frac{|S|^2}{2^{2k}} = \frac{|S|}{2^k}\left(1 - \frac{|S|}{2^{k+1}}\right) \geq \frac{3}{4}p.$$

Proof of $\mathrm{GNI} \in \mathrm{AM}[2]$

- The public-coin interactive proof system for GNI consists of the verifier and prover running several iterations of the set lower bound protocol.

  The verifier accepts iff the fraction of accepting iterations is

  $$\geq \frac{5p^*}{8}$$

  ($p^* = \frac{K}{2^k}$ can be easily computed by the verifier).

  The Chernoff bound shows that a constant number of iterations will suffice to ensure:

  - Completeness probability at least $\frac{2}{3}$;
  - Soundness error at most $\frac{1}{3}$.

Proof of $\mathrm{GNI} \in \mathrm{AM}[2]$ (Cont'd)

- The number of rounds stays at 2 because the verifier can do all iterations in parallel.
  - Pick several choices of h, y and send them all to the prover at once.
  - The preceding analysis of the probability of the prover's success is unaffected even if the prover is asked many questions in parallel.
- Unlike the private-coins protocol for GNI, the public-coins protocol does not have perfect completeness.
- The set lower bound protocol does not satisfy this property.
- A public-coins set lower bound protocol with completeness parameter 1 is possible.

Sketch of Proof of the Goldwasser-Sipser Theorem

- The public-coin prover demonstrates to the public-coin verifier an approximate lower bound on the size of the set of random strings which would have made the private-coin verifier accept.
- The set S in the public-coin protocol roughly corresponds to the set of possible messages sent by the verifier in the private protocol, where the verifier's message is a random element in S.
  - If the two graphs are isomorphic, then the verifier's message completely hides its choice of a random $i \in_R \{1, 2\}$;
  - If they are not, then the message distribution completely reveals it (at least to a prover that has unbounded computation time).

Sketch of Proof of the Goldwasser-Sipser Theorem

- Thus, roughly speaking:
  - If the two graphs are isomorphic, then the mapping from the verifier's coins to the message is 2-to-1;
  - If they are not, then the mapping from the verifier's coins to the message is 1-to-1.
    This results in a set that is twice as large.
- We can think of the public-coin prover as convincing the verifier that the private coin verifier would have accepted with large probability.
- A similar idea underlies the proof of

$$\mathrm{IP}[k] \subseteq \mathrm{AM}[k+2].$$

- However, one has to proceed in a round-by-round fashion.

NP-Completeness of GI Implies Collapsing of PH

Theorem

If GI is NP-complete, then $\Sigma_2 = \Pi_2$.

- We show that under this condition $\Sigma_2 \subseteq \Pi_2$.

  This will imply $\Sigma_2 = \Pi_2$ because $\Sigma_2 = \mathrm{co}\Pi_2$.

  If GI is NP-complete, then GNI is coNP-complete.

  So, there exists a function f, such that, for every n-variable formula φ,

$$\forall y \varphi(y) \text{ holds} \quad \text{iff} \quad f(\varphi) \in \mathrm{GNI}.$$

  Consider an arbitrary $\Sigma_2\mathrm{SAT}$ formula

$$\psi = \exists x \in \{0,1\}^n \forall y \in \{0,1\}^n \varphi(x,y).$$

NP-Completeness of GI and PH (Cont'd)

- The formula ψ is equivalent to

$$\exists x \in \{0,1\}^n g(x) \in \mathrm{GNI},$$

where:

- $g(x) = f(\varphi\restriction_x)$;
- $\varphi\restriction_x$ is the formula obtained from $\varphi(x,y)$ by fixing x.

GNI has a two-round AM proof with perfect completeness and (after appropriate amplification) soundness error less than 2^{-n}.

Let V be the verifier algorithm for this proof system.

Denote by m the length of the verifier's random tape.

Denote by m' the length of the prover's message.

Claim: ψ is true if and only if

$$\forall r \in \{0,1\}^m \exists x \in \{0,1\}^n \exists a \in \{0,1\}^{m'} (V(g(x), r, a) = 1).$$

If ψ is true, then perfect completeness implies the given condition.

Finishing the Proof

- Suppose, on the other hand, that ψ is false.

  Then $\forall x \in \{0,1\}^n \; g(x) \notin \mathrm{GNI}$.

  Now the soundness error of the interactive proof is less than 2^{-n}.

  Moreover, the number of x's is 2^n.

  We conclude (by the "probabilistic method basic principle") that there exists a single string $r \in \{0,1\}^m$, such that, for every $x \in \{0,1\}^n$, the prover in the AM proof for GNI has no response a that will cause the verifier to accept $g(x)$ if the verifier's first message is r.

  In other words, we have

$$\exists r \in \{0,1\}^m \forall x \in \{0,1\}^n \forall a \in \{0,1\}^{m'} (V(g(x), r, a) = 0).$$

  This is exactly the negation of the condition.

  Deciding the truth of the condition is in Π_2.

  We have, thus, shown $\Sigma_2 \subseteq \Pi_2$.

Subsection 3

IP=PSPACE

Intuition on IP vs PSPACE

- The inclusions NP $\subseteq$ IP $\subseteq$ PSPACE are clear.
- There was evidence (e.g., the protocols for quadratic nonresiduosity and GNI) that the first containment is proper.
- Most researchers felt that the second would also be proper.
- This feeling was based on the following points.
  - First, interaction alone does not give us any languages outside NP.
  - We also suspect that randomization alone does not add significant power to computation.
    Researchers even suspect that BPP $=$ P.
- So the question is how much additional power is provided by the combination of randomization and interaction.

Intuition on IP vs PSPACE (Cont'd)

- The evidence up to 1990 suggested that the answer was "not much".
  - For any fixed k, IP[k] collapses to the class AM = AM[2].
  - The latter equals BP $\cdot$ NP.
  - BP $\cdot$ NP seems not "much different" from NP.
- Finally, there were simply no protocols known that required k to be superconstant.
- So IP = IP[poly(n)] did not seem much bigger than IP[O (1)].
- However, this intuition was drastically wrong.

IP = PSPACE

Theorem

$$\text{IP} = \text{PSPACE}.$$

- By earlier remarks, it suffices to show PSPACE $\subseteq$ IP.

  Recall that every $L \in$ PSPACE is polytime reducible to TQBF.

  So it suffices to show $\mathrm{TQBF} \in$ IP[poly(n)].

  We describe a protocol for TQBF that:
  - Uses public coins.
  - Has the property that, if the input is in TQBF, then there is a prover that makes the verifier accept with probability 1.

The Language $\#\text{SAT}_D$

- We first think about how to design such a protocol for $\overline{3\text{SAT}}$.
- A prover must convince the verifier than a given 3CNF formula has no satisfying assignment.
- We show how to prove something more general.
- The prover can prove to the verifier that the number of satisfying assignments is exactly K, for some number K.
- We give an interactive proof for membership in the language

Definition ($\#\text{SAT}_D$)

$$\#\text{SAT}_D = \{\langle \varphi, K \rangle : \varphi \text{ is a 3CNF formula and it has exactly } K \text{ satisfying assignments}\}.$$

- $\#\text{SAT}_D$ clearly contains $\overline{\text{SAT}}$ as a special case.

Introducing Arithmetization

- The set lower bound protocol can tackle an approximation version of this problem.
- Namely, it can prove the value of K within a factor 2 (or any other constant factor).
- The protocol takes only two rounds.
- The protocol for $\#\mathrm{SAT}_D$ will use n rounds.
- The idea of arithmetization will also prove useful in our protocol for TQBF.
- The key idea is to represent Boolean formulas as polynomials.
- Note that 0,1 can be thought of both as truth values and as elements of some finite field $\mathbb{F}$.
  - $x_i \wedge x_j$ is true iff $x_i \cdot x_j = 1$ in the field;
  - $\neg x_i$ is true iff $1 - x_i = 1$.

Arithmetization Components

- Given any 3CNF formula $\varphi(x_1, x_2, \ldots, x_n)$ with m clauses and n variables, we introduce field variables $X_1, X_2, \ldots, X_n$.
- For any clause of size 3, we can write an equivalent degree 3 polynomial.

  Example: $x_i \vee \overline{x_j} \vee x_k \leftrightarrow 1 - (1 - X_i)X_j(1 - X_k)$.
- We denote the polynomial for the j-th clause by

$$p_j(X_1, X_2, \ldots, X_n).$$

- The notation allows the polynomial to depend on all n variables even though each p_j only depends upon at most three variables.
- For every 0,1 assignment to $X_1, X_2, \ldots, X_n$:
  - $p_j(X_1, X_2, \ldots, X_n) = 1$ if the assignment satisfies the clause;
  - $p_j(X_1, X_2, \ldots, X_n) = 0$ otherwise.

Arithmetization

- Multiplying the polynomials p_j, we obtain a multivariate polynomial

$$P_\varphi(X_1, X_2, \ldots, X_n) = \prod_{j \leq m} p_j(X_1, \ldots, X_n).$$

  - P_φ evaluates to 1 on satisfying assignments;
  - P_φ evaluates to 0 for unsatisfying assignments.
- This polynomial has degree at most $3m$.
- We represent such a polynomial as a product of all the above degree 3 polynomials without opening up the parenthesis.
- So $P_\varphi(X_1, X_2, \ldots, X_n)$ has a representation of size $O(m)$.
- This conversion of φ to P_φ is called **arithmetization**.
- Once we have written such a polynomial, we are able to:
  - Substitute arbitrary values from the field $\mathbb{F}$ instead of just 0, 1;
  - Evaluate the polynomial.
- This gives the verifier unexpected power over the prover.

Interactive protocol for $\#\mathrm{SAT}_D$

Theorem

$\#\mathrm{SAT}_D \subseteq \mathrm{IP}$.

- Consider an input

$$\langle \varphi, K \rangle,$$

where:

- φ is a 3CNF formula, with n variables and m clauses;
- K is a nonnegative integer.

We construct P_φ by arithmetization.

The number of satisfying assignments $\#\varphi$ of φ satisfies

$$\#\varphi = \sum_{b_1 \in \{0,1\}} \sum_{b_2 \in \{0,1\}} \cdots \sum_{b_n \in \{0,1\}} P_\varphi(b_1, \ldots, b_n).$$

The prover's claim is that this sum is exactly K.

Interactive protocol for $\#\mathrm{SAT}_D$

- We forget about φ and concentrate only on this claim about P_φ.
  - To start, the prover sends to the verifier a prime p in $(2^n, 2^{2n}]$.
  - The verifier can check that p is prime using a probabilistic or deterministic primality testing algorithm.
    All computations are done in the field $\mathbb{F} = \mathbb{F}_p$ of integers modulo p.

  The sum above is between 0 and 2^n.

  So the equation is true over the integers iff it is true modulo p.

Notation for the Sumcheck Protocol

- The prover is given:
  - A polynomial $g(X_1, \ldots, X_n)$ of degree d;
  - An integer K;
  - A prime p.
- Then it can provide an interactive proof for the claim

$$K = \sum_{b_1 \in \{0,1\}} \sum_{b_2 \in \{0,1\}} \cdots \sum_{b_n \in \{0,1\}} g(X_1, \ldots, X_n),$$

  where all computations are modulo p.
- To execute the protocol, the only property of g that the verifier needs is that it has a poly(n) size representation.
- If the property holds, then, for any assignment of values for the variables from the field GF(p), say

$$X_1 = b_1, X_2 = b_2, \ldots, X_n = b_n,$$

  the verifier can evaluate $g(b_1, b_2, \ldots, b_n)$ in polynomial time.

Notation for the Sumcheck Protocol (Cont'd)

- Note that the required condition is satisfied by $g = P_\varphi$.
- For each sequence of values $b_2, b_3, \ldots, b_n$ for $X_2, X_3, \ldots, X_n$,

$$g(X_1, b_2, b_3, \ldots, b_n)$$

  is a univariate degree d polynomial in the variable X_1.
- Thus the following is also a univariate degree d polynomial:

$$h(X_1) = \sum_{b_2 \in \{0,1\}} \cdots \sum_{b_n \in \{0,1\}} g(X_1, b_2, \ldots, b_n).$$

- If the claim is true, then we must have

$$h(0) + h(1) = K.$$

The Sumcheck Protocol

Sumcheck Protocol

V: If $n = 1$, check that $g(1) + g(0) = K$.

If so accept.

Otherwise reject.

If $n \geq 2$, ask P to send $h(X_1)$.

P: Sends some polynomial $s(X_1)$.

If the prover is not "cheating", then we will have $s(X_1) = h(X_1)$.

V: Reject if $s(0) + s(1) \neq K$.

Otherwise, pick random a in GF(p).

Recursively use the protocol to check that

$$s(a) = \sum_{b_2 \in \{0,1\}} \cdots \sum_{b_n \in \{0,1\}} g(a, b_2, \ldots, b_n).$$

Bounding the Probability of Rejection

Claim: Suppose that

$$K \neq \sum_{b_1 \in \{0,1\}} \sum_{b_2 \in \{0,1\}} \cdots \sum_{b_n \in \{0,1\}} g(X_1, \ldots, X_n).$$

Then V rejects with probability at least $(1 - \frac{d}{p})^n$.

- The claim implies the theorem.

Suppose

$$K = \sum_{b_1 \in \{0,1\}} \sum_{b_2 \in \{0,1\}} \cdots \sum_{b_n \in \{0,1\}} g(X_1, \ldots, X_n).$$

Then the prover can make the V accept with probability 1.

With our choice of p, $(1 - \frac{d}{p})^n$ is roughly $1 - \frac{dn}{p}$, very close to 1.

Proof of Correctness Claim

- Assume that

$$K \neq \sum_{b_1 \in \{0,1\}} \sum_{b_2 \in \{0,1\}} \cdots \sum_{b_n \in \{0,1\}} g(X_1, \ldots, X_n).$$

We prove the claim by induction on n.

Suppose, first, $n = 1$.

Then V simply evaluates $g(0), g(1)$.

It rejects with probability 1 if their sum is not K.

Assume the hypothesis is true for degree d polynomials in $n - 1$ variables.

In the first round, the prover is supposed to return the polynomial h.

Suppose, first, it indeed returns h.

By assumption, $h(0) + h(1) \neq K$.

So V will immediately reject (with probability 1).

Proof of Correctness Claim (Cont'd)

- Suppose, next, the prover returns some $s(X_1)$ different from $h(X_1)$.
  The degree d nonzero polynomial $s(X_1) - h(X_1)$ has at most d roots.
  So there are at most d values a, such that $s(a) = h(a)$.
  Thus, when V picks a random a,

$$\Pr_a[s(a) \neq h(a)] \geq 1 - \frac{d}{p}.$$

  Suppose $s(a) \neq h(a)$.
  The prover is left with an incorrect claim to prove in the next step.
  By the induction hypothesis, it fails with probability $\geq (1 - \frac{d}{p})^{n-1}$.
  Thus,

$$\Pr[V \text{ rejects}] \geq \left(1 - \frac{d}{p}\right)\left(1 - \frac{d}{p}\right)^{n-1} = \left(1 - \frac{d}{p}\right)^{n}.$$

  This completes the proof of the claim.

Protocol for TQBF: A First Approach

- We are given a quantified Boolean formula

$$\Psi = \forall x_1 \exists x_2 \forall x_3 \cdots \exists x_n \varphi(x_1, \ldots, x_n).$$

- We use arithmetization to construct the polynomial P_φ.
- Then we have $\Psi \in \mathrm{TQBF}$ if and only if

$$\prod_{b_1 \in \{0,1\}} \sum_{b_2 \in \{0,1\}} \prod_{b_3 \in \{0,1\}} \cdots \sum_{b_n \in \{0,1\}} P_\varphi(b_1, \ldots, b_n) \neq 0.$$

- A first thought is that we could use the same protocol as in the $\#\mathrm{SAT}_D$ case, with some modifications.
  - Since x_1 is quantified with $\forall$, we check that $s(0) \cdot s(1) = K$.
  - The do something analogous for all other variables that are quantified with $\forall$.

Protocol for TQBF: Exponential Blowup

- The problem that arises is the running time.
- Multiplying polynomials, unlike addition, increases the degree.
- If we define $h(X_1)$ analogously as before by making X_1 a free variable, then its degree may be as high as 2^n.
- This polynomial may have 2^n coefficients.
- So it cannot be transmitted to a polynomial-time verifier.
- To solve the issue, we observe that:
  - The condition

$$\prod_{b_1 \in \{0,1\}} \sum_{b_2 \in \{0,1\}} \prod_{b_3 \in \{0,1\}} \cdots \sum_{b_n \in \{0,1\}} P_\varphi(b_1, \ldots, b_n) \neq 0$$

    only uses $\{0, 1\}$ values;
  - For $x \in \{0, 1\}$

$$x^k = x, \quad \text{for all } k \geq 1.$$

Avoiding the Exponential Blowup

- These observations allow us to convert any polynomial $p(X_1, \ldots, X_n)$ into a multilinear polynomial (i.e., the degree of q in any variable X_i is at most one) $q(X_1, \ldots, X_n)$, such that, for all $X_1, \ldots, X_n \in \{0, 1\}$

$$p(X_1, \ldots, X_n) = q(X_1, \ldots, X_n).$$

- Specifically, we define a **linearization operator**.
- For any polynomial p, let $L_{X_i}(p)$ (or $L_i(p)$ for short) be the polynomial

$$\begin{aligned} L_{X_i}(p)(X_1, \ldots, X_n) &= X_i \cdot p(X_1, \ldots, X_{i-1}, 1, X_{i+1}, \ldots, X_n) \\ &\quad + (1 - X_i) \cdot p(X_1, \ldots, X_{i-1}, 0, X_{i+1}, \ldots, X_n). \end{aligned}$$

- Thus, $L_i(p)$ is linear in X_i and agrees with p, for $X_i \in \{0, 1\}$.
- So $L_1(L_2(\cdots(L_n(p))\cdots))$ is a multilinear polynomial agreeing with p on all values in $\{0, 1\}$.

Avoiding the Exponential Blowup (Cont'd)

- We will also think of $\forall x_i$ and $\exists x_i$ as operators on polynomials where

$$\begin{aligned}
\forall X_i p(X_1, X_2, \ldots, X_n) &= p(X_1, \ldots, X_{i-1}, 0, X_{i+1}, \cdots, X_n) \\
&\quad \cdot p(X_1, \ldots, X_{i-1}, 1, X_{i+1}, \cdots, X_n); \\
\exists X_i p(X_1, X_2, \ldots, X_n) &= p(X_1, \ldots, X_{i-1}, 0, X_{i+1}, \ldots, X_n) \\
&\quad + p(X_1, \ldots, X_{i-1}, 1, X_{i+1}, \ldots, X_n).
\end{aligned}$$

- Thus, the claim takes the following form.
- If we apply the sequence of operators $\forall X_1 \exists X_2 \forall X_3 \cdots \exists X_n$ (where $\exists X_n$ is applied first and $\forall X_1$ is applied last) on $P_\varphi(X_1, \ldots, X_n)$, then we get a nonzero value K.

Avoiding the Exponential Blowup (Cont'd)

- This claim only concerns values taken when variables are in $\{0,1\}$.
- Thus, it is unaffected if we sprinkle in any arbitrary sequence of the linearization operators in between.
- We do so, so that the intermediate polynomials arising in our sum check protocol all have low degree.
- In this way we get

$$\forall X_1 L_1 \exists X_2 L_1 L_2 \forall X_3 L_1 L_2 L_3 \cdots \exists X_n L_1 L_2 L_3 \cdots L_n \; P_\varphi(X_1, \ldots, X_n).$$

- The size of the expression is

$$\mathrm{O}\left(1 + 2 + 3 + \cdots + n\right) = \mathrm{O}\left(n^2\right).$$

Hypotheses on the Protocol

- Suppose for some polynomial $g(X_1, \ldots, X_k)$ the prover has the ability to convince the verifier that

$$g(a_1, a_2, \ldots, a_k) = C$$

with:

  - Probability 1, for any $a_1, a_2, \ldots, a_k, C$, when this is true;
  - Probability less than ϵ, when it is false.

- Let $U(X_1, X_2, \ldots, X_\ell)$ be any polynomial on ℓ variables obtained as

$$U(X_1, X_2, \ldots, X_\ell) = \mathcal{O}g(X_1, \ldots, X_k),$$

where $\mathcal{O}$ is $\exists X_i$ or $\forall X_i$ or L_{X_i}, for some variable.

- Note ℓ is $k-1$ in the first two cases and k in the third.

Goal of the Protocol

- Let d be an upper bound (known to the verifier) on the degree of U with respect to x_i.
- In our case, $d \leq 3m$.
- We show how the prover can convince the verifier that

$$U(a_1, a_2, \ldots, a_\ell) = C',$$

with:

- Probability 1, for any $a_1, a_2, \ldots, a_k, C'$ for which it is true;
- Probability at most $\epsilon + \frac{d}{p}$, when it is false.

Description of the Protocol

- By renaming variables if necessary, assume $i = 1$.
- The verifier's check is as follows.

  Case 1: $\mathcal{O} = \exists X_1$.

  The prover provides a degree d polynomial $s(X_1)$, supposed to be

  $$g(X_1, a_2, \ldots, a_k);$$

  Verifier checks if $s(0) + s(1) = C'$.
  - If not, it rejects.
  - If yes, it picks a random value $a \in \mathbb{F}_p$.
    It then asks the prover to prove $s(a) = g(a, a_2, \ldots, a_k)$.

  Case 2: $\mathcal{O} = \forall X_1$.

  The prover provides a degree d polynomial $s(X_1)$, supposed to be

  $$g(X_1, a_2, \ldots, a_k);$$

  Verifier checks if $s(0) \cdot s(1) = C'$.
  - If not, it rejects.
  - If yes, it picks a random value $a \in \mathbb{F}_p$.
    It then asks the prover to prove $s(a) = g(a, a_2, \ldots, a_k)$.

Description (Cont'd) and Correctness

- Case 3: $\mathcal{O} = L_{X_1}$.
  The prover wishes to prove that $U(a_1, a_2, \ldots, a_k) = C'$.
  The prover provides a degree d polynomial $s(X_1)$, supposed to be

  $$g(X_1, a_2, \ldots, a_k).$$

  Verifier checks if $a_1 s(0) + (1 - a_1)s(1) = C'$.
  - If not, it rejects.
  - If yes, it picks random $a \in \mathbb{F}_p$.
    It then asks prover to prove $s(a) = g(a, a_2, \ldots, a_k)$.
- The proof of correctness follows as in case of $\#\text{SAT}_D$.
- It uses the observation that, if $s(X_1)$ is not the right polynomial, then with probability $1 - \frac{d}{p}$ the prover is still stuck with proving an incorrect statement at the next round.

Subsection 4

The Power of the Prover

Powerful Provers

- A curious feature of some interactive proof systems is that, to prove membership in language L, the prover needs to do more powerful computation than just deciding membership in L.
- Consider, e.g., the public-coin system for graph nonisomorphism.
- The prover is given:
  - A randomly chosen hash function h;
  - A random element y in the range of h.
- It is required to produce a graph H, such that $h(H)$ is isomorphic to either G_1 or G_2 and $h(x) = y$.
- This seems harder than just solving graph nonisomorphism (but we do not know of any proof that it is).

Powerful Provers (Cont'd)

- Consider, also, the interactive proof for 3SAT, a language in coNP.
- It requires the prover to at the very least be able to compute $\#\text{SAT}_D$.
- This is not known to be computable in polynomial time, even if we have an oracle for 3SAT.
- It is true that the ability to compute $\#\text{SAT}_D$ is #P-complete.
- This implies PH $\subseteq \text{P}^{\#\text{SAT}_D}$.
- In both the case of graph non-isomorphism and 3SAT, it is an open problem whether the protocol can be redesigned to use a weaker prover.
- By contrast, in the protocol for TQBF, the prover requires no more computational power than the ability to compute TQBF.

A Karp-Lipton Type Theorem

- The Karp-Lipton Theorem states that, if $\mathrm{NP} \subseteq \mathrm{P}_{/\mathrm{poly}}$, then $\mathrm{PH} = \Sigma_2^p$.
- Meyer's Theorem states that if $\mathrm{EXP} \subseteq \mathrm{P}_{/\mathrm{poly}}$, then $\mathrm{EXP} = \Sigma_2^p$.
- The following result is in the same spirit as the Karp-Lipton results.

Theorem

If $\mathrm{PSPACE} \subseteq \mathrm{P}_{/\mathrm{poly}}$, then $\mathrm{PSPACE} = \mathrm{MA}$.

- Its conclusion is stronger, since MA is contained in Σ_2.
- In fact, an MA-proof system for L, with perfect completeness, implies $L \in \Sigma_2$.

A Karp-Lipton Type Theorem

- Suppose PSPACE $\subseteq \mathrm{P}_{/\mathrm{poly}}$.

  The prover in our TQBF protocol can be replaced by a circuit of polynomial size.
  - Merlin (the prover) can just give this circuit to Arthur (the verifier) in Round 1;
  - Arthur then runs the interactive proof using this "prover".

  No more interaction is needed.

  There is no need for Arthur to put blind trust in Merlin's circuit.

  By the correctness proof of the TQBF protocol, if the formula is not true, then no prover can make Arthur accept with high probability.

Subsection 5

Multi-Prover Interactive Proofs: MIP

Multiple Provers

- We can define interactive proofs that involve more than one prover.
  - The important assumption is that the provers do not communicate with each other during the protocol.
  - They may communicate before the protocol starts and, in particular, agree upon a shared strategy for answering questions.
- Think of the police interrogating two suspects in separate rooms.
  - The suspects may be accomplices who have decided upon a common story to tell the police.
  - However, since they are interrogated separately, they may inadvertently reveal an inconsistency in the story.

The Class MIP

- The set of languages with multiprover interactive provers is called MIP.
- In the formal definition, we assume:
  - There are two provers (allowing polynomially many provers does not change the class).
  - In each round, the verifier sends a query to each of them (the two queries need not be the same).
  - Each prover sends a response in each round.
- IP $\subseteq$ MIP, since the verifier can always simply ignore one prover.
- However, it turns out that MIP is probably strictly larger than IP (unless PSPACE $=$ NEXP).

Theorem

NEXP $=$ MIP.

Nonadaptivity

- Intuitively, one reason why two provers are more useful than one is that the second prover can be used to force nonadaptivity.
- Consider the interactive proof as an "interrogation" where:
  - The verifier asks questions;
  - The verifier gets back answers from the prover.
- Suppose the verifier wants to ensure that the answer of a prover to the question q is a function only of q and does not depend on the previous questions the prover heard.
- Then the prover can ask the second prover the question q.
- Then the verifier accepts only if both answers agree with one another.

Probabilistically Checkable Proofs

- Nonadaptivity checking was used to show that multiprover interactive proofs can be used to implement (and, in fact, are equivalent to) a model of a "probabilistically checkable proof in the sky".
- In this model, we go back to an NP-like notion of a proof as a static string.
- Since this string may be huge, it is best thought of as a huge table, consisting of the prover's answers to all possible verifier's questions.
- The verifier checks the proof by looking at only a few entries in this table that are chosen randomly from some distribution.

The Classes PCP$[r, q]$

- Let the class PCP$[r, q]$ be the set of languages that can be proven using:
  - A table of size 2^r;
  - q queries to this table.
- Then the preceding theorem can be restated as follows.

Theorem

$$\text{NEXP} = \text{PCP}[\text{poly}, \text{poly}] = \bigcup_c \text{PCP}[n^c, n^c].$$

- The theorem can be scaled down to NP $=$ PCP[polylog, polylog].
- And, with a lot of work, we can even prove

Theorem (The PCP Theorem)

$$\text{NP} = \text{PCP}[\text{O}(\log n), \text{O}(1)].$$

Applications of the PCP Theorem

- The PCP Theorem has had many applications in complexity.
- In particular, it establishes that, for many NP complete optimization problems, obtaining an approximately optimal solution is as hard as coming up with the optimal solution itself.
- Thus, it seems that complexity theory has come full circle with interactive proofs.
  - By starting with NP and adding interaction, randomization, and multiple provers to it, we get to classes as high as NEXP;
  - Then, we obtain new and fundamental insights about NP itself.

Subsection 6

Program Checking

Program Checking

- The discovery of the interactive protocol for $\#\text{SAT}_D$ was triggered by a research area called **program checking** or, sometimes, **instance checking**.
- Program verification is the task of deciding whether or not a given program solves a certain computational task on all inputs.
- Program verification is undecidable.
- The basic idea underlying program checking is that, in many situations, it suffices to have a guarantee of the program's "correctness" on an input-by-input basis.

Introducing Program Checkers

- This idea is encapsulated in the notion of a **program checker**.
- A checker for a program P is another program C that may run P as a subroutine.
- Whenever P is run on an input, C's job is to detect if P's answer is incorrect ("buggy") on that particular input.
- To do this, C may also compute P's answer on some other inputs.
- Formally, the checker C is a TM that expects to have the code of another program, which it uses as a black box.
- We denote by C^P the result of running C when it is provided P as a subroutine.

Program Checkers

Definition (Program Checker)

Let T be a computational task. A **checker** for T is a probabilistic polynomial time TM C that, given any program P, which is a claimed program for T, and any input x, has the following behavior:

1. If P is a correct program for T, i.e., $\forall y P(y) = T(y)$, then

$$\Pr[C^P \text{ accepts } P(x)] \geq \frac{2}{3}.$$

2. If $P(x) \neq T(x)$, then

$$\Pr[C^P \text{ accepts } P(x)] < \frac{1}{3}.$$

Comments

- Checkers do not certify the correctness of a program.
- Even in the case that P is correct on x, i.e., $P(x) = C(x)$ but the program P is not correct on inputs other than x, the output of the checker is allowed to be arbitrary.
- Surprisingly, for many problems, checking seems easier than actually computing the problem.
- So one could build such checkers into the software for these problems, with negligible overhead, while endowing the program with the ability to automatically check its work.

Checker for Graph Nonisomorphism

- Recall the problem of graph nonisomorphism.
  - The input is a pair of labeled graphs $\langle G_1, G_2 \rangle$.
  - The problem is to decide whether $G_1 \cong G_2$.
- We do not know of an efficient algorithm for this problem.
- However, the problem has an efficient checker.
- There are two types of inputs depending upon whether or not the program claims $G_1 \cong G_2$.
- Suppose, first, the probram claims that $G_1 \cong G_2$.

  Then one can change the graph little by little and use the program to actually obtain a permutation π mapping G_1 to G_2.

  If this fails, it finds a bug in the program.

Checker for Graph Nonisomorphism (Cont'd)

- Suppose, next, the claim is $G_1 \ncong G_2$.

  We use our earlier interactive proof of graph nonisomorphism.
  - In case the prover admits $G_1 \ncong G_2$, repeat k times:
  - Choose $i \in_R \{1, 2\}$.
    Permute G_i randomly into H.
  - Ask the prover whether G_1, H are isomorphic.
    Check to see if the answer is consistent with the earlier answer.

Correctness of the Checker for Graph Nonisomorphism

- Given a computer program P that supposedly computes graph isomorphism, to check its correctness, we use an IP, while regarding the program as the prover.
- Let C be a program that performs the above protocol using as prover the claimed program P.

Theorem

- If P is a correct program for graph nonisomorphism, then C outputs "correct" always.
- Otherwise, if $P(G_1, G_2)$ is incorrect, then

$$\Pr[C \text{ outputs "correct"}] \leq 2^{-k}.$$

Finally, C runs in polynomial time.

Languages that Have Checkers

- Suppose L has an interactive proof system in which the prover can be implemented using oracle access to L.
- Then L has a checker.
- So the interactive proofs we have seen imply

Theorem

The following problems have checkers:

- Graph Isomorphism (GI);
- $\#\mathrm{SAT}_D$;
- True Quantified Boolean Formulas (TQBF).

- It can also be shown that problems that are random self-reducible and downward self-reducible also have checkers.

P-Complete Languages and Checkers

- We know that P-complete languages are reducible to each other via NC-reductions (in fact, even via the weaker logspace reductions).
- Consequently, it suffices to show a checker in NC for one P-complete language (proved by Blum and Kannan) to obtain the following

Theorem

For any P-complete language, there exists a program checker in NC.

- We believe that P-complete languages cannot be computed in NC.
- So the theorem suggests that checking is easier than actual computation.

Random Self-Reducibility

- Most checkers are designed by observing that the output of the program at x should be related to its output at some other points.
- The simplest such relationship is random self-reducibility.
- Roughly speaking, a problem is **random-self-reducible** if solving the problem on any input x can be reduced to solving the problem on a sequence of random inputs

$$y_1, y_2, \ldots,$$

where each y_i is uniformly distributed among all inputs.

- This property is important in understanding the average-case complexity of problems.

Example

- Consider a linear function

$$f : \mathrm{GF}(2)^n \to \mathrm{GF}(2).$$

- By definition, there exist coefficients $a_1, a_2, \ldots, a_n$, such that

$$f(x_1, x_2, \ldots, x_n) = \sum_i a_i x_i.$$

- Then for any $\boldsymbol{x}, \boldsymbol{y} \in \mathrm{GF}(2)^n$, we have

$$f(\boldsymbol{x}) + f(\boldsymbol{y}) = f(\boldsymbol{x} + \boldsymbol{y}).$$

- Using this, one shows that computing f is random-self-reducible.
- Suppose we want to compute $f(\boldsymbol{x})$, where $\boldsymbol{x}$ is arbitrary.
- It suffices to pick a random $\boldsymbol{y}$ and compute $f(\boldsymbol{y})$ and $f(\boldsymbol{x} + \boldsymbol{y})$.
- Moreover, both $\boldsymbol{y}$ and $\boldsymbol{x} + \boldsymbol{y}$ are random vectors in $\mathrm{GF}(2)^n$.

The Permanent

Definition (Permanent of a Matrix)

Let $A \in \mathbb{F}^{n \times n}$ be a matrix over the field $\mathbb{F}$. The **permanent** of A is

$$\text{perm}(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} a_{i,\sigma(i)}.$$

- The problem of calculating the permanent is in PSPACE.
- It is, in fact, #P-complete, i.e., essentially equivalent to $\#\text{SAT}_D$.
- Thus, if the permanent can be computed in polynomial time, P = NP.
- We show that the permanent is random-self- reducible.
- The key is that perm(A) as a function of n^2 variables is a polynomial of degree n.

Lipton's Permanent Theorem

Theorem (Lipton)

Suppose the finite field $\mathbb{F}$ has size $> 3n$.
There is a randomized algorithm that, given an oracle that can compute the permanent on $1 - \frac{1}{3n}$ fraction of the inputs in $\mathbb{F}^{n \times n}$, can compute the permanent on all inputs correctly with high probability.

- Let A be some input matrix.

  Pick a random matrix $R \in_R \mathbb{F}^{n \times n}$.

  Let, for a variable x,

  $$B(x) = A + x \cdot R.$$

  Notice that:

  - $\text{perm}(B(x))$ is a degree n univariate polynomial.
  - For any fixed $a \neq 0$, $B(a)$ is a random matrix.
    Hence, by hypothesis, the probability that the oracle computes $\text{perm}(B(a))$ correctly is at least $1 - \frac{1}{3n}$.

Lipton's Permanent Theorem (Cont'd)

- The algorithm for computing the permanent of A:
  - Fix any $n+1$ distinct points $a_1, a_2, \ldots, a_{n+1}$ in the field.
  - Query the oracle on all matrices $\{B(a_i) : 1 \leq i \leq n+1\}$.
  - According to the union bound, with probability of at least

$$1 - \frac{n+1}{3n} \approx \frac{2}{3},$$

    the oracle will compute the permanent correctly on all matrices.
  - Given $n+1$ (point, value) pairs $\{(a_i, b_i) : i \in [n+1]\}$, there exists a unique degree n polynomial p that satisfies

$$\forall i \ p(a_i) = b_i.$$

  - Therefore, given that the values $B(a_i)$ are correct, the algorithm can interpolate the polynomial $B(x)$ and compute $B(0) = \text{perm}(A)$.
- The hypothesis can be weakened so that the oracle only needs to compute the permanent correctly on $\frac{1}{2} + \varepsilon$, $\varepsilon > 0$, of the inputs.

Subsection 7

Interactive Proof for the Permanent

Random and Downward Self-Reducibility for Permanent

- The protocol for the interactive proof for the Permanent uses:
  - The random-self-reducibility of the permanent;
  - Downward self-reducibility, i.e., the observation that

$$\text{perm}(A) = \sum_{i=1}^{n} a_{1i}\text{perm}(A_{1,i}),$$

    where $A_{1,i}$ is an $(n-1) \times (n-1)$ submatrix of A obtained by removing the first row and i-th column of A.
- Recall that the corresponding formula for the determinant uses alternating signs.
- Computing the $n \times n$ permanent reduces to computing n permanents of $(n-1) \times (n-1)$ matrices.

Notation

- We assume the field $\mathbb{F}$ is equal to GF(p), for some prime $p > n$.
- So we have$1, 2, \ldots, n \in \mathbb{F}$.
- We reserve a_{ij} for the (i, j)-th element of the matrix.
- For every $n \times n$ matrix A and $i \in [n]$, we define $D_A(i)$ to be the $(n-1) \times (n-1)$ matrix $A_{1,i}$.
- If $x \in \mathbb{F} \backslash [n]$, then we define $D_A(x)$ in the unique way such that for every $j, k \in [n-1]$, the function $(D_A(x))_{j,k}$ is a univariate polynomial of degree at most n.
- Now the permanent of an $(n-1) \times (n-1)$ matrix is a degree $(n-1)$ polynomial in the entries of the matrix.
- So perm$(D_A(x))$ is a univariate polynomial of degree $\leq (n-1)n < n^2$.

The Problem L_{perm}

- Define L_{perm} to contain all tuples $\langle A, p, k \rangle$, such that:
  - $p > n^4$ is prime;
  - A is an $n \times n$ matrix over GF(p);
  - $\text{perm}(A) = k$.

Theorem

$L_{\text{perm}} \in \text{IP}$.

- The proof is by induction.

  We assume inductively that, for each $(n-1) \times (n-1)$ matrix B, the prover can make the verifier accept the claim $\text{perm}(B) = k'$:
  - With probability 1, if it is true;
  - With probability at most ϵ, if it is false.

Proof of the Permanent Theorem

- We show that, for every $n \times n$ matrix A, the prover can make the verifier accept the claim $\text{perm}(A) = k$:
  - With probability 1, if it is true;
  - With probability at most $\epsilon + \frac{(n-1)^2}{p}$, if it is false.

  The following simple exchange shows this:

  Round 1: Prover sends to verifier a polynomial $g(x)$ of degree $(n-1)^2$, which is supposedly $\text{perm}(D_A(x))$.
  Round 2: Verifier checks whether: $k = \sum_{i=1}^{m} a_{1i} g(i)$.
  - If not, it rejects at once.
  - Otherwise, the verifier picks a random element of the field $b \in_R \mathbb{F}_p$. Then, asks the prover to prove that $g(b) = \text{perm}(D_A(b))$.

  Notice, $D_A(b)$ is an $(n-2) \times (n-2)$ matrix over $\mathbb{F}_p$, and so now use the inductive hypothesis to design a protocol for this verification.

Correctness of the Permanent Protocol

- Suppose, first, that $\text{perm}(A) = k$.

  Then an all-powerful prover can provide $\text{perm}(D_A(x))$.

  Thus, by the inductive hypothesis, it can make the verifier accept with probability 1.

  Suppose, on the other hand, that $\text{perm}(A) \neq k$.

  Assume that the polynomial $g(x)$ sent in the first round is the correct polynomial $\text{perm}(D_A(x))$.

  Then

$$\sum_{i=1}^{m} a_{1,i} g(i) = \text{perm}(A) \neq k.$$

  So the verifier would immediately reject.

Correctness of the Permanent Protocol (Cont'd)

- Assume, on the other hand, that a prover sends $g(x) \neq \text{perm}(D_A(x))$.
  But two polynomials of degree $(n-1)^2$ can only agree for less than $(n-1)^2$ values of x.
  So the chance that the randomly chosen $b \in \mathbb{F}_p$ is one of them is at most $\frac{(n-1)^2}{p}$.
  If b is not one of these values, then the prover is stuck with proving an incorrect claim.
  By the inductive hypothesis, he can prove it with conditional probability at most ϵ.
  The probability that the prover can convince this verifier about an incorrect value of the permanent is

$$\leq \frac{(n-1)^2}{p} + \frac{(n-2)^2}{p} + \cdots + \frac{1}{p} \leq \frac{n^3}{p}.$$

  This is much smaller than $\frac{1}{3}$ for our choice of p.