Chapter 10

Interactive proofs

Recall the certificate definition of NP. We can think of this characterization of NP as an interaction between a polynomial-time *verifier* and an all-powerful prover. For a language $L \in$ NP, x is in L if and only if the prover can send V a certificate (that depends on x) that convinces the verifier.

Of course, when humans try to prove something to one another, they *interact*, with the verifier asking the prover for a series of explanations. Formalizing this interaction gives rise to *interactive proofs*, a central notion of complexity theory and cryptography today. Studying this notion allows us to prove surprising results such as the fact that the famous *graph isomorphism* and *approximate shortest vector* problems are probably not NP-complete (i.e., if PH doesn't collapse). Above all, we are led to surprising new characterizations of conventional complexity classes such as PSPACE. In fact, I feel that the field of interactive proofs will continue to yield surprising results for many more more years.

10.1 Warmup: Interactive proofs with a deterministic verifier

Let us consider what happens when we introduce *interaction* into the NP scenario above. Suppose we say that the language L *has an interactive proof system* if there is a polynomial-time verifier V that on n-bit inputs asks the prover a series of $N = \text{poly}(n)$ queries and receives answers to each. In general, the $(i+1)$th query can depend upon the answers received to the first i queries. The prover is any function that, given the input x and the first i queries $q_1, q_2, \ldots, q_i$ outputs a response a_i to the ith query. For any prover P we denote by $V^P(x)$ the final decision (i.e., 1/0) of V on x after interacting with P. We require

$$x \in L \Rightarrow \exists \text{ prover } P \text{ s.t. } V^P(x) = 1 \tag{10.1}$$

$$x \notin L \Rightarrow \exists \text{ prover } P \text{ s.t. } V^P(x) = 0. \tag{10.2}$$

Unfortunately, interaction alone does not result in a new complexity class so long as the verifier is deterministic.

THEOREM 45
Language L has an interactive proof of the above type iff $L \in$ NP.

PROOF: Clearly, every NP language has a 1-round proof system. Now we prove that if a L has an interactive proof system of this type then $L \in$ NP. The certificate for membership is just the query-answer transcript $(q_1, a_1, q_2, a_2, \ldots, q_N, a_N)$ whose end-result is $V^P(x) = 1$. To verify this transcript, check that the first query of V on x is q_1, that the second query (after receiving a_1 as response to q_1) is q_2, and so on. Clearly, if such a transcript exists, it is proof that $x \in L$. □

10.2 IP

In order to realize the full potential of interaction, we let the verifier be a *probabilistic polynomial time* TM. The set of languages which have interactive proof systems now jumps from NP to PSPACE, as we will see.

DEFINITION 38 (IP) *A language L is in IP if there is probabilistic, polynomial-time TM V that on an input x of size n can compute the following $N = N(n) = n^{O(1)}$ functions $q_1, q_2, \ldots, q_N$. Function q_i computes the ith query and its arguments are x, a random string r, and an $(i-1)$-tuple of strings $(a_1, a_2, \ldots, a_{i-1})$ (which are the prover's answers to previous queries).*

A prover P for inputs of length n is any arbitrary set of N functions where the ith function maps $(x, q_1, q_2, \ldots, q_i)$ to an answer string a_i.

V and P interact by sending queries and answers alternately (V goes first, as the definition implies) and at the end of N rounds V outputs its 0/1 decision, denoted $V^P(x, r)$. The verifier has the property that:

$$x \in L \Rightarrow \exists P \quad \Pr_r[V^P(x, r) = 1] \geq 2/3 \tag{10.3}$$

$$x \notin L \Rightarrow \forall P \quad \Pr_r[V^P(x, r) = 0] \geq 2/3. \tag{10.4}$$

If k is a fixed constant (independent of input size n) we define IP[k] (for $k \geq 2$) be the set of all languages that have a k round interactive proof similar to above, but with at most k rounds of interaction, where a "round" is either a query or a response and the first round is a query.

REMARK 7 We make a series of observations and leave some of the proofs as exercises.

1. Allowing the prover to be probabilistic (i.e., the answer function a_i depends upon some random string used by the prover) does not change the class IP. The reason is that for any language L, if a probabilistic prover P results in making verifier V accept with some probability, then averaging implies there is a deterministic prover which makes V accept with the same probability.

2. Since the prover consists of arbitrary functions, in principle it could be undecidable. However, one can show that given any verifier V, we can compute the optimum prover (which maximizes $\Pr_r[V^P(x, r) = 1]$ for each x) using poly($|x|$) space. (Exercise!) Thus IP $\subseteq$ PSPACE.

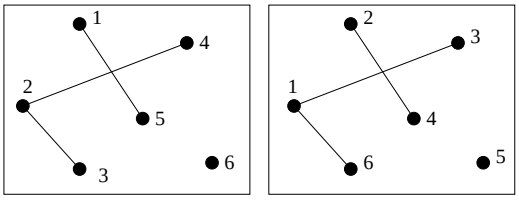

Figure 10.1: Two isomorphic graphs.

3. The probabilities of correctly classifying an input can be made arbitrarily large by using the same boosting technique we used for BPP: to replace $2/3$ by $1-\exp(-m)$, sequentially repeat the protocol m times and take the majority answer. If $x \in L$ then the same prover can be used on each repetition and in each of the independent trials it has a probability at least $2/3$ of making V accept. If $x \notin L$ then even if a new prover is used in each repetition, the definition assures us that the probability that the verifier will accept in any trial is less than $1/3$.

4. We emphasize that the prover functions do not depend upon the verifier's random string. In other words, the verifier has *private coins*.

Let us look at an example of a language in IP.

EXAMPLE 22 (GRAPH NON-ISOMORPHISM) Given two graphs G_1 and G_2 we say they are isomorphic to each other if there is a permutation π of the labels of the nodes of G_1 such that $\pi G_1 = G_2$. The graphs in figure 10.1, for example, are isomorphic with $\pi = (12)(3654)$. If G_1 and G_2 are isomorphic, we write $G_1 \equiv G_2$. The GRAPH NON-ISOMORPHISM problem is this: given two graphs, are they *not* isomorphic?

It is clear that the complement of GRAPH NON-ISOMORPHISM is in NP— a certificate is simply the permutation π that is an isomorphism. What is more surprising is that GRAPH NON-ISOMORPHISM is in IP. To show this, we give a private-coin protocol that satisfies definition 38:

Protocol: 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.

To see that definition 38 is satisfied by the above protocol, note that if $G_1 \not\equiv G_2$ then there exists a prover such that $\Pr[V \text{ accepts}] = 1$, because if the graphs are non-isomorphic, an all-powerful prover can certainly tell which one of the two is isomorphic to H. On the other hand, if $G_1 \equiv G_2$ 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; that is $\Pr[V \text{accepts}] \leq 1/2$.

10.2.1 Public coins and AM

Allowing the prover full access to the verifier's random string leads to the model of *interactive proofs with public-coins*.

DEFINITION 39 (AM) *The classes AM and AM[k] are defined similarly to IP and IP[k] respectively, except the prover's answers are allowed to depend upon the verifier's random string r.*

The interactive proof for GRAPH NON-ISOMORPHISM seemed to crucially depend upon the fact that P cannot see the random bits of V. If P knew those bits, P would know i and so could trivially always guess correctly. The following result contradicts this intuition.

THEOREM 46 (GOLDWASSER, SIPSER)
GRAPH NON-ISOMORPHISM $\in AM[2]$.

PROOF: To develop a public-coin protocol, we need to look at the problem in a different way. Consider the set $S = \{H : H \equiv G_1 \text{ or } H \equiv G_2\}$. The size of this set depends on whether G_1 is isomorphic to G_2. For a graph of n vertices, there are $n!$ possible ways to label the vertices, so we have

$$\text{if } G_1 \not\equiv G_2 \text{ then } |S| = 2n!$$
$$\text{if } G_1 \equiv G_2 \text{ then } |S| = n!$$

(Actually this intuition is not quite correct: we need the set

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

where $aut(H)$ is the group of permutations π such that $\pi(H) = H$.)

We can amplify the gap between the two cases. Choose an integer m such that $n! \leq 2^m$. Let S' be the Cartesian product of S with itself a sufficient number of times so that the following hold:

$$\text{if } G_1 \not\equiv G_2 \text{ then } \left|S'\right| \geq 100 \cdot 2^m \quad \text{(A)}$$
$$\text{if } G_1 \equiv G_2 \text{ then } \left|S'\right| \leq \frac{1}{10} 2^m \quad \text{(B)}$$

We can now use the size of S' to determine if $G_1 \equiv G_2$. The question that remains is: how do we design an interactive protocol that can distinguish between cases (A) and (B) above?

Let $\mathcal{H}$ be a set of 2-universal hash functions from $\mathcal{U}$ to $\{0,1\}^m$, where $\mathcal{U}$ is a superset of S'. Given $h \in \mathcal{H}$, if S' is much bigger than $\{0,1\}^m$, as it is in case (A), then for every $y \in \{0,1\}^m$ it is very likely that there is some $x \in S'$ such that $h(x) = y$. Conversely, if the size of S' is much smaller than $\{0,1\}^m$, —if (B) holds— then for most $y \in \{0,1\}^m$ there is no $x \in S'$ such that $h(x) = y$. This is the basis of our protocol: V gives P an $h \in \mathcal{H}$ and $y \in \{0,1\}^m$, and if P can find an $x \in S'$ that h maps to y, the verifier guesses that the set is large, otherwise, we guess that the set is small. See figure 10.2. More formally, we have:

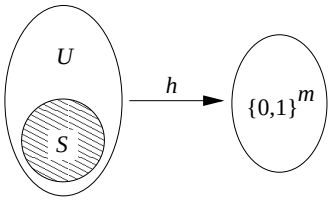

Figure 10.2: The bigger S' is, the more likely $h(S')$ will hit a given point in $\{0,1\}^m$.

Protocol: Goldwasser-Sipser Set Lowerbound

V: Randomly pick $h \in \mathcal{H}$, and $y \in_R \{0,1\}^m$. Send h, y to P.

P: Try to find an $x \in S'$ such that $h(x) = y$. Send such an x to V, or send a random element in $\mathcal{U}$ if no such x exists.

V: If $x \in S'$ and $h(x) = y$, accept; otherwise reject.

It remains to be shown that the above protocol fits Definition 38. Suppose (B) holds, then since the domain (S') is 1/10th the size of the range ($\{0,1\}^m$), we have $\Pr[V \text{ accepts if } G_1 \equiv G_2] \leq 1/10$. To find the probability of acceptance if case (A) holds we need a lemma that tells us that if S' is bigger than a certain size, any hash function we pick will likely map S' to a large fraction its range. This follows from Lemma 47 below. In the language of that lemma, here we have $\mu = 100$. Even if the hash function that we choose in the protocol is good, there is still a chance $1/\sqrt{\mu}$ that we will pick a $y \in \{0,1\}^m$ that is not mapped to by S. Hence,

$$\Pr[V \text{ accepts if } G_1 \equiv G_2] \geq 1 - \frac{1}{\sqrt{100}} - \frac{1}{\sqrt{100}} = \frac{8}{10}.$$

□

Lemma 47
Suppose $S \geq \mu 2^m$ and $\mathcal{H}$ is a pairwise-independent family of hash functions from $\mathcal{U}$ to $\{0,1\}^m$ where $S \subseteq \mathcal{U}$. Then $\Pr_{h \in \mathcal{H}}\left[|h(S)| \geq \left(1 - \frac{1}{\sqrt{\mu}}\right) 2^m\right] \geq 1 - \frac{1}{\sqrt{\mu}}$.

Proof:Left as exercise. □

10.2.2 Some properties of IP and AM

We state the following comments about IP[·], AM[·], without proof:

1. IP[k] ⊆ AM[k+2] for all constants k. This can be proved using the Goldwasser-Sipser technique.

2. Whereas BPP is a probabilistic version of P, AM[2] is as probabilistic version of NP. In fact AM[2] = $BP\cdot$ NP (see exercises) where $BP\cdot$ NP is the class in Definition 25.

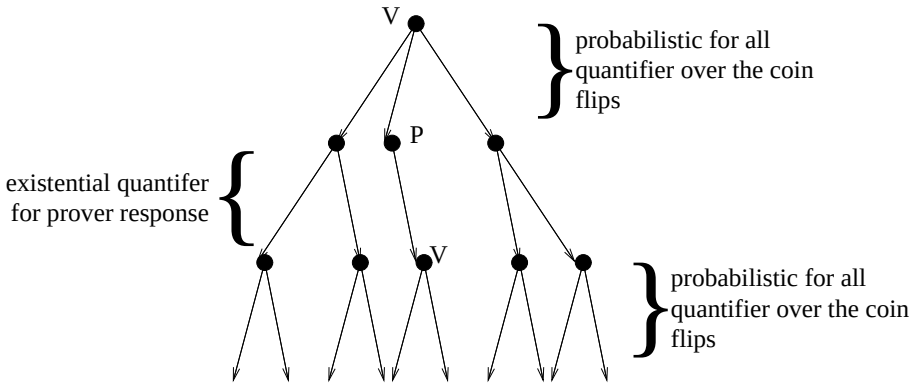

Figure 10.3: AM[k] looks like $\prod_k^p$

3. For constants $k \geq 2$ we have AM[k] = AM[2]. This is surprising because AM[k] seems similar to PH with the $\forall$ quantifiers changed to "probabilistic $\forall$" quantifiers, where *most* of the branches lead to acceptance. See figure 10.3.

   It is open whether there is any nice characterization of AM[$\sigma(n)$], where $\sigma(n)$ is a suitably slow growing function of n, such as $\log \log n$.

4. Changing 2/3 to 1 in the definition of IP does not change the class. That is, defining the IP in a manner similar to coRP is equivalent to defining it like BPP.

10.3 IP = PSPACE

We now turn to our main theorem.

THEOREM 48 (LFKN, SHAMIR, 1990)
IP = PSPACE.

By our earlier Remark 7 we need only show that PSPACE $\subseteq$ IP[$poly(n)$]. To do so, we'll show that TQBF $\in$ IP[$poly(n)$]. This is sufficient because every $L \in$ PSPACE is polytime reducible to TQBF.

10.3.1 Arithmetization

Again, we change the representation of the problem. For any Boolean formula of n variables $\phi(b_1, b_2, \ldots, b_n)$ there is a polynomial $P_\phi(x_1, x_2, \ldots, x_n)$ that is 1 if ϕ is true and 0 if ϕ is false. To see that this is true, consider the following correspondence between formulas and polynomials:

$$\begin{aligned} x \wedge y &\longleftrightarrow X \cdot Y \\ x \vee y &\longleftrightarrow 1 - (1 - X)(1 - Y) \\ \neg x &\longleftrightarrow 1 - X \end{aligned}$$

For example, $\phi = x \vee y \vee \neg z \longleftrightarrow 1 - (1 - X)(1 - Y)Z$. If ϕ is a 3CNF formula with n variables and m clauses then we can write such a polynomial for each clause and multiply those polynomials to get a polynomial P_ϕ in n variables, with degree at most m in each variable. This conversion of ϕ to P_ϕ is called *arithmetization* and will be useful in our protocol.

10.3.2 Interactive protocol for #SAT$_L$

Rather than tackle the job of finding a protocol for TQBF right away, we first present a protocol for #SAT$_L$, where

$$\#\text{SAT}_L = \{\langle \phi, K \rangle : K \text{ is the number of sat. assignments of } \phi\}.$$

and ϕ is a 3CNF formula of n variables and m clauses.

THEOREM 49 (LFKN)
#SAT$_L \in$ IP.

PROOF: Given ϕ, we construct, by arithmetization, P_ϕ. The number of satisfying assignments $\#\phi$ of ϕ is:

$$\#\phi = \sum_{b_1 \in \{0,1\}} \sum_{b_2 \in \{0,1\}} \cdots \sum_{b_n \in \{0,1\}} P_\phi(b_1, \ldots, b_n) \tag{10.5}$$

There is a general protocol, *Sumcheck*, for verifying equations such as (10.5).

Sumcheck protocol. Given a degree d polynomial $g(x_1, \ldots, x_n)$ and an integer K, we present an interactive proof for the claim

$$K = \sum_{b_1 \in \{0,1\}} \sum_{b_2 \in \{0,1\}} \cdots \sum_{b_n \in \{0,1\}} g(x_1, \ldots, x_n). \tag{10.6}$$

V simply needs to be able to arbitrarily evaluate g. Define

$$h(x_1) = \sum_{b_2 \in \{0,1\}} \cdots \sum_{b_n \in \{0,1\}} g(x_1, b_2 \ldots, b_n).$$

If (10.6) is true, it must be the case that $h(0) + h(1) = K$.

To start, V randomly picks a prime p in the interval $[n^3, n^4]$ and instructs the prover to reduce all numbers modulo p in the remaining protocol. (For instance, if the prover wants to send a polynomial, it only needs to send the coefficients modulo p.) All computations described below are also modulo p. Consider the following protocol:

Protocol: Sumcheck protocol to check claim (10.6)

V: If $n = 1$ check that $g(1) + g(0) = K$. If so accept, otherwise reject. If $n \geq 2$, ask P to send $h(x_1)$ as defined above.

P: Send $h(x_1)$.

V: Let $s(x_1)$ be the polynomial received. Reject if $s(0)+s(1) \neq K$; otherwise pick a random a. Recursively use this protocol to check that

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

If Claim (10.6) is true, the prover that always returns the correct polynomial will always convince V. We prove by induction on n that if (10.6) is false, V rejects with high probability; we prove

$$\Pr[V \text{ rejects } \langle K, g \rangle] \geq \left(1 - \frac{d}{p}\right)^n. \tag{10.7}$$

With our choice of p, the right hand side is about $1 - dn/p$, which is very close to 1 since $p \geq n^3$.

We prove (10.7) by induction on n. The statement is true for $n = 1$ since V simply evaluates $g(0), g(1)$ and 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 P is supposed to return the polynomial h. Suppose that P returns $s \neq h$. If $s(0) + s(1) \neq K$, then V rejects with probability 1. So assume that $s(0) + s(1) = K$. In this case, V picks a random a. If s and h are two different degree d polynomials, then there are at most d values of x_1 such that $s(x_1) = h(x_1)$. Thus,

$$\Pr_a[s(a) \neq h(a)] \geq 1 - \frac{d}{p}. \tag{10.8}$$

If $s(a) \neq h(a)$ then the prover is left with an incorrect claim to prove in the recursive step. By the induction hypothesis, with probability $\geq \left(1 - \frac{d}{p}\right)^{n-1}$, P cannot prove this false claim. Thus we have

$$\Pr[V \text{ rejects}] \geq \left(1 - \frac{d}{p}\right) \cdot \left(1 - \frac{d}{p}\right)^{n-1} = \left(1 - \frac{d}{p}\right)^n \tag{10.9}$$

This finishes the induction.

We still have to justify why it is OK to perform all operations modulo p. The fear, of course, is that equation (10.6) might be false over the integers, but true over p, which happens if p divides the difference of the two sides of (10.6). The following lemma implies that the chance that this happens for a random choice of p is small, since an n-bit integer (which is what K is) has at most n prime divisors.

Lemma 50 (Fingerprinting)
Suppose x, y, $x \neq y$ are n-bit integers. Then there are at most n primes that divide $|x - y|$.

An interactive proof for $\#\text{SAT}_L$ is obtained by letting $g = P_\phi$. □

10.3.3 Protocol for TQBF

PROOF:(Theorem 48) We use a very similar idea to obtain a protocol for TQBF. Given a true, fully qualified Boolean formula $\exists x_1 \forall x_2 \exists x_3 \cdots \forall x_n \phi(x_1, \ldots, x_n)$, we use arithmetization to construct the polynomial P_ϕ. We have that $\phi \in$ TQBF if and only if

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

A first thought is that we could use the same protocol as in the $\#\text{SAT}_L$ case, except check that $s(0) \cdot s(1) = K$ when you have a $\prod$. But, alas, multiplication, unlike addition, increases the degree of the polynomial — after n steps when V must evaluate g directly, the degree could be 2^n. There could be exponentially many coefficients. The solution is to observe that we are only interested in $\{0, 1\}$ values. You can always approximate a polynomial with a multi-linear function if you only evaluate it at $\{0, 1\}^n$. Let Rx_i be a linearization operator defined as

$$Rx_1[p(x_1, \ldots, x_n)] = (1 - x_1)p(0, x_2, \ldots, x_n) + (x_1)p(1, x_2, \ldots, x_n). \tag{10.10}$$

Now, instead of working with $\exists x_1 \forall x_2 \exists x_3 \cdots \forall x_n \phi(x_1, \ldots, x_n)$, work with

$$\exists x_1 Rx_1 \forall x_2 Rx_1 Rx_2 \exists x_3 Rx_1 Rx_2 R_x 3 \cdots \phi(x_1, \ldots, x_n) \tag{10.11}$$

The size of the expression is $1 + 2 + 3 + \cdots + n \approx n^2$. The protocol for $\#\text{SAT}_L$ can be used suitably modified, where in rounds involving the linearization operator, the verifier uses (10.10). The use of the linearization operator ensures that the polynomial which the prover needs to send at every round is linear, and hence the blowup in degrees is avoided.

10.4 Interactive proof for the Permanent

Although the existence of an interactive proof for the Permanent follows from that for #SAT and TQBF, we describe a specialized protocol as well. This is both for historical context (this protocol was discovered before the other two protocols) and also because this protocol may be helpful for further research. (One example will appear in a later chapter.)

DEFINITION 40 *Let $A \in F^{n \times n}$ be a matrix over the field F. The permanent of A is:*

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

The problem of calculating the determinant is #P-complete (notice the similarity to the determinant which is polynomial time computable). Recall from previous lectures that

$PH \subseteq \mathrm{P}^{PERM}$ (Toda's theorem).
Observation:

$$f(x_1, x_2, ..., x_n) := perm \begin{bmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,n} \\ x_{2,1} & \ddots & ... & x_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \dots & x_{n,n} \end{bmatrix}$$

is a degree n polynomial since

$$f(x_1, x_2, \dots, x_n) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} x_{i,\sigma(i)}.$$

We now show two properties of the permanent problem. The first is called ransom self reducibility:

Theorem 51 (Lipton 88')
There is a randomized algorithm that, given an oracle that can compute the permanent on $1 - \frac{1}{3n}$ fraction of the inputs in $F^{n \times n}$ (where the finite field F has size $> 3n$), can compute the permanent on all inputs correctly with high probability.

Proof: Let A be some input matrix. Pick a random matrix $R \in_R F^{n \times n}$ and let $B(x) := A + x \cdot R$ for a variable x. Notice that:

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

Now the algorithm for computing the permanent of A is straightforward: query oracle on all matrices $\{B(i) | 1 \leq i \leq n+1\}$. According to the union bound, with probability of at least $1 - \frac{n+1}{n} \approx \frac{2}{3}$ the oracle will compute the permanent correctly on all matrices.
Recall the fact from numerical analysis that given $n+1$ values $\{(a_i, b_i) | i \in [n+1]\}$, there exists a unique a degree n polynomial p that satisfies $\forall i \ . \ p(a_i) = b_i$.
Therefore, given that the values $B(i)$ are correct, the algorithm can interpolate the polynomial $B(x)$ and compute $B(0) = A$. □

Note: The above theorem can be strengthened to be based on the assumption that the oracle can compute the permanent on a fraction of $\frac{1}{2} + \varepsilon$ for any constant $\varepsilon > 0$ of the inputs. The observation is that not all values of the polynomial must be correct for unique interpolation. There are techniques which require only a fraction slightly higher then half of correct values for successfully interpolating a polynomial. This was not done in class but we include notes from a previous course that describe how; see the last section of this lecture's notes.

Another property of the permanent problem, called downward self reducibility:

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

Where $A_{1,i}$ is a $(n-1) \times (n-1)$ sub-matrix of A obtained by removing the 1'st row and i'th column of A (recall the analogous formula for the determinant uses alternating signs).

DEFINITION 41 *Define a $(n-1) \times (n-1)$ matrix $D_A(x)$, such that each entry contains a degree n polynomial. This polynomial is uniquely defined by the values of the matrices $\{A_{1,i} | i \in [n]\}$. That is:*

$$\forall i \in [n] \ . \ D_A(i) = A_{1,i}$$

Where $D_A(i)$ is the matrix $D_A(x)$ with i substituted for x. (notice that these equalities force n points and values on them for each polynomial at a certain entry of $D_A(x)$, and hence according to the previously mentioned fact determine this polynomial uniquely)

Observation: $Perm(D_A(x))$ is a degree $n(n-1)$ polynomial in x.
We now show an interactive proof for the permanent (the decision problem is whether $perm(A) = k$ for some value k):

- Round 1: Prover sends to verifier a polynomial $g(x)$ of degree $n(n-1)$, which is supposedly $perm(D_A(x))$.
- Round 2: Verifier checks whether:

  $$k = \sum_{i=1}^{m} a_{1,i} g(i)$$

  If not, rejects at once. Otherwise, verifier picks a random element of the field $b_1 \in_R F$ and asks the prover to prove that $g(b_1) = perm(D_A(b_1))$. This reduces the matrix dimension to $(n-2) \times (n-2)$.

  $$\vdots$$

- Round $2(n-1)-1$: Prover sends to verifier a polynomial of degree 2, which is supposedly the permanent of a 2×2 matrix.
- Round $2(n-1)$: Verifier is left with a 2×2 matrix and calculates the permanent of this matrix and decides appropriately.

CLAIM 52
This is indeed an interactive protocol.

PROOF: If $perm(A) = k$, then there exists a prover that makes the verifier accept with probability 1, this prover just returns the correct values of the polynomials according to definition.
On the other hand, suppose that $perm(A) \neq k$. If on the first round, the polynomial $g(x)$ sent is the correct polynomial $D_A(x)$, then:

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

And the verifier would reject. Hence $g(x) \neq D_A(x)$. According to the fact on polynomials stated above, these polynomials can agree on at most $n(n-1)$ points. Hence, the probability

that they would agree on the randomly chosen point b_1 is at most $\frac{n(n-1)}{|F|}$. The same considerations apply to all subsequent rounds if exist, and the overall probability that the verifier will not accepts is thus (assuming $|F| \geq 10n^3$ and sufficiently large n):

$$\begin{aligned} Pr &\geq \left(1 - \frac{n(n-1)}{|F|}\right) \cdot \left(1 - \frac{(n-1)(n-2)}{|F|}\right) \cdot ... \left(1 - \frac{3 \cdot 2}{|F|}\right) \\ &\geq \left(1 - \frac{n(n-1)}{|F|}\right)^{n-1} \geq (1 - o(1))e^{-2\frac{n(n-1)^2}{|F|}} \geq \frac{1}{2} \end{aligned}$$

□

Open Problem: The current technique to obtain an interactive protocol for $\overline{SAT}$ (the complement of the SAT language) is to reduce $\overline{SAT}$ to permanent and then obtain the IP for permanent we've seen. The power of the prover is, hence, $\#P$ (can compute a $\#P$ complete language). An open problem is whether there exists an IP for $\overline{SAT}$ with less power, specifically can the prover be in FP^{SAT} ?

10.5 Program Checking

The first interactive proof for permanent (outlined an email Noam Nisan sent to many researchers in 1989) was motivated by a field called *program checking,* initiated by Blum and Kannan. A *program checker* C for a program P is itself another program. For any instance on which P is run, C is run subsequently and certifies that P is correct or declares it to be incorrect. Formally:

Definition 42 *Let P be a claimed program for computational task T. A* ***checker*** *for T is a probabilistic polynomial time TM, C, such that:*

1. *If $\forall x$. $P(x) = T(x)$, then $P[C^P$ accepts $P(x)] \geq \frac{2}{3}$*
2. *If $P(x) \neq T(x)$ then $P[C^P$ accepts $P(x)] < \frac{1}{3}$*

Example 23 (Graph Non-Isomorphism) The input for the problem of Graph Non-Isomorphism is a pair of labelled graphs $< G_1, G_2 >$, and the problem is to decide whether a permutation $\pi : V(G_2) \mapsto V(G_2)$ such that renaming the vertices according to the permutation results in a graph identical to G_1 (and in that case the graphs are isomorphic, denoted $G_1 \equiv G_2$).

Now there are two cases, depending upon whether or not the program claims $G_1 \equiv G_2$. If it claims that $G_1 \equiv G_2$ then the checking the claim is left as an exercise. We show how to check the claim that $G_1 \not\equiv G_2$ using our earlier interactive proof of Graph non-isomorphism.

Recall the IP for Graph Non-Isomorphism:

- In case prover admits $G_1 \not\equiv G_2$ repeat k times:
- Choose $i \in_R \{1, 2\}$. Permute G_i randomly into H
- Ask the prover $< G_1, H >; < G_2, H >$ and check to see if the prover's first answer is consistent.

Given a computer program that supposedly computes graph isomorphism, P, how would we check its correctness? The program checking approach suggests to use an IP while regarding the program as the prover. Let C be a program that performs the above protocol with P as the prover, then:

THEOREM 53
If P is a correct program for Graph Non-Isomorphism then C outputs "correct" always. Otherwise, if $P(G_1, G_2)$ is incorrect then $P[C$ outputs "correct" $] \leq 2^{-k}$. Moreover, C runs in polynomial time.

Program checking is sometimes also called *instance checking*, perhaps a more accurate name since checking a program would mean that it computes T on *all inputs*, which is of course undecidable. Blum and Kannan's definition manages to evade this undecidability result by concentrating on the correctness of the program on a single instance.

10.5.1 Program Checking Properties

The following theorem are a direct consequence of the IP we have seen:

THEOREM 54
The problems Graph Non-Isomorphism, Quadratic Residue, Permanent have checkers.

The following theorem shows that program checking is a robust notion:

THEOREM 55 (BLUM,KANNAN,BEIGEL)
Let $\mathcal{F}$ be a time complexity class which is closed under multiplication, composition and addition of its time complexity functions (such as P and $TIME(\log^{O(1)} n)$). Let π_1 and π_2 two problems reducible to each other in $\mathcal{F}$, and C_1 a checker for π_1 in $\mathcal{F}$. Then there is a checker C_2 for π_2 in $\mathcal{F}$.

PROOF:(sketch) Basically one can produce the new checker using the checker for the first problem composed with a program that computes the reduction function. Correctness follows directly from the definition of reductions and the fact that the complexity class is closed under basic operations. □

Using this theorem, and the fact that P-complete languages are NC-reducible to each other, it suffices to show a checker in NC for one P-complete language (as was shown by Blum & Kannan) to obtain the following interesting fact:

THEOREM 56
For any P-complete language there exists a program checker in NC

Since we believe that P-complete languages can not be computed in NC, this provides strong evidence that checking is easier than actual computation.
The relation between program checking and IP is more precisely stated by the following theorem:

DEFINITION 43 *Let **function restricted IP** the set of all decision problems for which there exists an IP, where the prover is further restricted to be a function from the program instances to $\{YES/NO\}$.*

THEOREM 57
A language L has a checker $\Leftrightarrow$ $L \in$ function restricted IP $\cap$ co-function restricted IP.

PROOF: $\Leftarrow$ Suppose that L has an IP of one of the sorts above. A checker for the language would simulate the IP, using the supposed machine for the language (or co-language) as provers.
$\Rightarrow$ Suppose the language L has a checker, denoted M. Then a (co-) function restricted IP for this language is just the simulation of the machine, such that each activation of M of the oracle is a question for the prover. Correctness follows simply from definitions. □

Exercises

§1 Prove the assertions in Remark 7.

§2 We say integer y is a *quadratic residue modulo* m if there is an integer x such that $y \equiv x^2 \pmod{m}$. Show that the following language is in IP[2]:

QUADRATIC-NONRESIDUOSITY $= \{(y, m) : \ y$ is not a quadratic residue modulo $m\}$.

§3 Show that AM[2] $= BP \cdot$ NP

§4 Prove Lemma 47. (Hint: use pairwise independence.)

□