Lecture 18: **IP = PSPACE**

Anup Rao

May 20, 2020

Computing the Permanent in **IP**

The permanent of an $n \times n$ matrix M is defined to be $\sum_\pi \prod_{i=1}^n M_{i,\pi(i)}$, where the sum is taken over all permutations $\pi : [n] \to [n]$.

The permanent is important because it is a complete function for the class **#P**:

Definition 1. *A function $f : \{0,1\}^n \to \mathbb{N}$ is in* **#P** *if there exists a polynomial p and a poly time machine M such that*

$$f(x) = |\{y \in \{0,1\}^{p(|x|)} : M(x,y) = 1\}|$$

For example, in **#P** one can count the number of satisfying assignments to a boolean formula, which is potentially much harder than just determining whether the formula is satisfiable or not. One can show that any such problem can be reduced in polynomial time to computing the permanent of a matrix with $0/1$ entries. On the other hand, the permanent itself can be computed in **#P**. Thus the permanent is **#P**-complete.

Some Math Background

A finite field is a finite set that behaves just like the real numbers, in that you can add, multiply, divide and subtract the elements, and the sets include $0, 1$. An example of such a field is $\mathbb{F}_p$, the set $\{0, 1, 2, \ldots, p\}$, where here p is a prime number. We can perform addition and multiplication by adding/multiplying the integers and taking their remainder after division by p. This leaves us back in the set. For any $0 \neq a \in \mathbb{F}_p$, one can show that there exists $a^{-1} \in \mathbb{F}_p$ such that $a \cdot a^{-1} = 1$. To see this, note that since a is relatively prime to p, Euclid's gcd algorithm shows that there exist integers b, d such that $ab + pd = 1$, so we can define $a^{-1} = b \mod p$.

We shall work with polynomials over finite fields. $\mathbb{F}_p[X]$ denotes the set of polynomials in the variable X with coefficients from $\mathbb{F}_p$.

Fact 2. *Given any set of $d+1$ distinct points $a_0, a_1, \ldots, a_d$, there is a one to one correspondence between polynomials of degree at most d, and their evaluations on the points $a_0, \ldots, a_d$.*

Proof Given any set of constraints $f(a_i) = b_i$, we can build a degree

d polynomial for f as follows:

$$f(x) = \sum_{i=0}^{d} b_i \prod_{j \neq i} (x - a_j)/(a_i - a_j)$$

Thus, for every such map, we have defined a polynomial that evaluates that map.

Since the dimension of the set of functions $f : \{a_0, \ldots, a_d\} \to \mathbb{F}$ is $d+1$, which is the same as the dimension of the space of polynomials of degree d, this relationship must be a one to one correspondence. ■

An easy consequence of the above fact is the following:

Fact 3. *Any non-zero polynomial $f(X)$ of degree d has at most d roots. (a is a root if $f(a) = 0$).*

Proof Suppose there are $d+1$ roots $a_0, \ldots, a_d$. Then there must be exactly one degree d polynomial evaluating to 0 on all these roots, and so f must be the 0 polynomial, which is a contradiction. ■

The density of primes:

Fact 4. *Let $t(n)$ denote the number of primes in the set $[n]$. Then*

$$\lim_{n \to \infty} \frac{t(n)}{n/\ln n} = 1.$$

The fact says that a random n-bit number is likely to be a prime with probability $\approx 1/n$. Thus we can sample an n-bit prime by repeatedly sampling random n-bit numbers and checking whether or not they are prime (which can be done in polynomial time). In fact, the prime we obtain in this way will be larger than $2^{n/2}$ with high probability, since with high probability all of our samples will be larger than $2^{n/2}$.

The Permanent Protocol

Suppose the verifier is given a boolean matrix M and wants to check that $\mathsf{Perm}(M) = k$. Let $M^{1,i}$ denote the matrix obtained by deleting row 1 and column i from the matrix M. Then:

$$\mathsf{Perm}(M) = \sum_{i=1}^{n} M_{1,i} \mathsf{Perm}(M^{1,i}).$$

Consider the function D that maps an index $i \in [n]$ to the matrix $D(i) = M^{1,i}$. By Fact 2, we can write $D(x)$ for $n \times n$ matrix whose entries are all polynomials of degree $n-1$ in x such that $D(i) = M^{1,i}$. In other words, $D(x)$ is a $n-1 \times n-1$ matrix in which each entry of the matrix is given by a polynomial of degree $n-1$ and for every $i \in [n]$, $D(i) = M^{1,i}$. Here is a first attempt at a protocol for the verifier:

1. If $n = 1$, the verifier checks that $M_{1,1} = k$.
2. The verifier asks the prover to send a prime $2^{2n} > p > 2^n$, and checks that it is in fact a prime larger than k.
3. If $n > 1$, verifier asks the prover to send the degree n^2 polynomial $g \in \mathbb{F}_p[X]$, $g(X) = \mathsf{Perm}(D(X))$.
4. The verifier checks that $k = \sum_{i=1}^{n} M_{1,i} \cdot \mathsf{Perm}(D(i))$.
5. The verifier picks a uniformly random $a \in \mathbb{F}_p$ and recursively checks that $\mathsf{Perm}(D(a)) = g(a)$.

Analysis of the Protocol

If $\mathsf{Perm}(M) = k$, then $k \leq 2^n$ and there is a prime p as required (we know that the primes are sufficiently dense for such a prime to exist. Then we have that $\mathsf{Perm}(M) = k \mod p$. $\mathsf{Perm}(D(X))$ is a polynomial if degree at most n^2 so the prover that responds honestly will convince the verifier to accept with probability 1. It only remains to show that a dishonest prover cannot fool the verifier except with small probability.

Suppose $\mathsf{Perm}(M) \neq k$. Then it must be that $\mathsf{Perm}(M) \neq k \mod p$, since p is larger than both $\mathsf{Perm}(M)$ and k. Note that if the prover sends $g(X) = \mathsf{Perm}(D(X)$, then the verifier will immediately conclude that $\mathsf{Perm}(M) \neq k$, thus the verifier can only be fooled if $g(X) \neq \mathsf{Perm}(D(X))$. Then we have that

$$\Pr_a[g(a) = \mathsf{Perm}(D(a))] \leq n^2/p,$$

since both $g(X), \mathsf{Perm}(D(X))$ are degree n^2 polynomials. Indeed, the only way that the prover can succeed once he has sent the wrong polynomial $g(X)$ is if $g(a) = \mathsf{Perm}(D(a))$ in some recursive call. The recursion has at most n steps, so by the union bound, the probability that the prover succeeds is at most $n^3/p \ll 1/3$, for n large enough.

A protocol for counting satisfying assignments

We continue to exhibit the power of interaction by showing how it can be used to solve any problem in **PSPACE**. Recall that the problem of computing whether a totally quantified boolean formula is true is complete for **PSPACE**, so it will be enough to give an interactive protocol that verifies that such a formula is true.

As a warmup, let us consider the case when we are given a formula of the type $\exists x_1, \ldots, x_n \phi(x_1, \ldots, x_n)$ and want to *count* the

number of satisfying assignments to this formula. Since the permanent is complete for **#P**, we can reduce this counting problem to the computation of the permanent, and then use the interactive protocol from the last lecture, but let us be more direct.

As in the protocol for the permanent, we shall leverage algebra. Since polynomials are much nicer to deal with than formulas, let us try to encode the formula ϕ using a multivariate polynomial. Here is a first attempt at building such an encoding gate by gate:

- $x \wedge y \to xy$.
- $\neg x \to 1 - x$.
- $x \vee y \to x + y - xy$.

This encoding gives us a polynomial g_ϕ that computes the same value as the formula ϕ, however it is not clear that g_ϕ can be computed in polynomial time. The problem is the encoding for $\vee$ gates, which could potentially double the size of the polynomial obtained in each step. Instead, we use the more clever encoding:

- $x \vee y \to 1 - (1 - x)(1 - y)$.

This allows us to obtain a polynomial g_ϕ which can be written down in time polynomial in the size of ϕ.

Then the task of counting the number of satisfying assignments to ϕ reduces to computing $\sum_{x \in \{0,1\}^n} g_\phi(x)$. Following the ideas used in the protocol for the permanent, here is a protocol for a verifier that checks that $\sum_{x \in \{0,1\}^n} g_\phi(x) = k$.

1. Ask the prover for a prime $2^{2n} > p > 2^n$, and check that it is correct. Reject if $k < p$. All arithmetic is henceforth done modulo p.
2. If $n = 1$, check the identity by computing it.
3. If $n > 1$, ask the prover for the degree n polynomial
$$f(X) = \sum_{x_2, \ldots, x_n \in \{0,1\}^{n-1}} g_\phi(X, x_2, \ldots, x_n).$$
4. Check that $f(0) + f(1) = k \mod p$.
5. Pick a random element $a \in \mathbb{F}_p$ and recursively check that
$$f(a) = \sum_{x_2, x_3, \ldots, x_n \in \{0,1\}^{n-1}} g_\phi(a, x_2, \ldots, x_n)$$

For the analysis, note that $f(X)$ is indeed a degree n polynomial, since there are at most n gates in the formula ϕ. Thus if

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

an honest prover can convince the verifier with probability 1.

If $\sum_{x \in \{0,1\}^n} g_\phi(x) \neq k$, then the if the prover succeeds, it must be that

$$f(X) \neq \sum_{x_2,\ldots,x_n \in \{0,1\}^{n-1}} g_\phi(X, x_2, \ldots, x_n),$$

for if the prover is honest, he will be caught immediately.

Since $f(X), \sum_{x_2,\ldots,x_n \in \{0,1\}^{n-1}} g_\phi(X, x_2, \ldots, x_n)$ are both degree n polynomials, we have that

$$\Pr_a \left[f(a) = \sum_{x_2,\ldots,x_n \in \{0,1\}^{n-1}} g_\phi(a, x_2, \ldots, x_n) \right] \leq n/p,$$

so with high probability, the prover is left with trying to prove an incorrect statement in the next step. By the union bound, the probability that the prover succeeds in any step is at most $n^2/p \ll 1/3$ for large n.

A protocol for TQBF

To handle checking whether a formula of the type

$$\exists x_i \forall x_2 \exists x_3 \ldots \forall x_n \phi(x_1, \ldots, x_n)$$

is true, it is clear that this is equivalent to checking the identity that

$$\sum_{x_1} \prod_{x_2} \sum_{x_3} \cdots \prod_{x_n} g_\phi(x_1, \ldots, x_n) = k > 0.$$

This is just another polynomial identity, so a first attempt might be to use a protocol of the following type:

1. Ask the prover for a suitably large prime p, and check that it is correct. Reject if $k < p$. All arithmetic is henceforth done modulo p.
2. If $n = 1$, check the identity by computing it.
3. If $n > 1$, ask the prover for the polynomial
$$f(X) = \prod_{x_2} \sum_{x_3} \cdots \prod_{x_n} g_\phi(X, x_2, \ldots, x_n).$$
4. Check that $f(0) + f(1) = k \mod p$ (or $f(0) \cdot f(1) = k \mod p$ as appropriate.

5. Pick a random element $a \in \mathbb{F}_p$ and recursively check that

$$f(a) = \prod_{x_2} \sum_{x_3} \cdots \prod_{x_n} g_\phi(a, x_2, \ldots, x_n)$$

There are several problems with this approach. For one thing the product term can generate the product of 2^n terms giving a number k that is as large as 2^{2^n}. So the prover cannot even write down k using less than 2^n bits, which means that the verifier cannot compute with it in polynomial time. Similarly, the degree of the polynomial f can be as large as 2^n, so the verifier cannot do any computations with it.

In order to handle the first problem, we appeal to the prime number theorem and the chinese remainder theorems:

Theorem 5 (Prime Number Theorem)**.** *Let $\pi(t)$ denote the number of primes in $[t]$. Then*

$$\lim_{t \to \infty} \frac{\pi(t)}{t / \ln t} = 1.$$

The theorem says that $\Theta(1/n)$ fraction of all n bit numbers are prime.

Theorem 6 (Chinese Remainder Theorem)**.** *If k is divisible by distinct primes $p_1, \ldots, p_t$, then k must be bigger than the product $\prod_i p_i$.*

Now consider the set of primes in the interval $[2^n, 2^{10n}]$. By Theorem 5 there $\Theta(2^{10n}/n)$ primes that are less than 2^{10n}, but at most 2^n of them are less than 2^n, so this interval must contain $\Theta(2^{10n}/n)$ primes. The product of all these primes is at least $(2^n)^{\Omega(2^{10n}/n)} = 2^{\Omega(2^{10n})}$. Thus, for n large enough, the product is much larger than $\sum_{x_1} \prod_{x_2} \sum_{x_3} \cdots \prod_{x_n} g_\phi(x_1, \ldots, x_n) = k$. Recall that $k \leq 2^{2^n}$.

Thus by Theorem 6, if $k > 0$, there must be some prime $p \in [2^n, 2^{10n}]$ such that

$$\sum_{x_1} \prod_{x_2} \sum_{x_3} \cdots \prod_{x_n} g_\phi(x_1, \ldots, x_n) = k \neq 0 \mod p.$$

This allows us to fix the first problem: the verifier can ask the prover to send this prime and the value of $k \mod p$, and perform all arithmetic modulo p.

Next we turn to the second issue. While it is true that the polynomials generated in the above proof can have high degree, note that since we are only interested in evaluating the polynomials we are working with over inputs that are bits, it never makes sense to raise a variable to degree more than 1: $x^2 = x$ for $x \in \{0, 1\}$. Thus, we could ask the prover to work with the polynomial that is obtained from g_ϕ by replacing all high degree terms with terms that have degree 1 in each variable. However, we cannot trust that the prover will

be honest, so we shall have to check that the prover does this part correctly.

Given any polynomial $g(X_1, \ldots, X_n)$ define the operator L_1 as

$$L_1 g(X_1, \ldots, X_n) = X_1 \cdot g(1, X_2, \ldots, X_n) + (1 - X_1) \cdot g(0, X_2, \ldots, X_n).$$

Then note that $L_1 g$ takes on the same value as g when $X_1 \in \{0, 1\}$. Similarly, we can define L_i for each $i \in [n]$.

Our final protocol is then as follows. In order to prove that

$$\sum_{x_1} \prod_{x_2} \cdots \prod_{x_n} g_\phi(x_1, \ldots, x_n) \neq 0,$$

we shall instead ask the prover to prove that

$$\sum_{x_1} L_1 \prod_{x_2} L_1 L_2 \sum_{x_3} L_1 L_2 L_3 \prod_{x_4} \cdots L_{n-1} L_n \prod_{x_n} g_\phi(x_1, \ldots, x_n) = k \neq 0 \mod p.$$

In order to describe the protocol, in general we are going to be trying to prove a statement of the form $\mathcal{O}_1 \mathcal{O}_2 \mathcal{O}_t g_\phi(x_1, \ldots, x_n) = k \mod p$, where $\mathcal{O}_i$ is either $\sum_{x_i}$, $\prod_{x_i}$ or L_i for some i. Some of the variables x_i may be set to constants a_i during this process, but this will not change the protocol.

The verifier proceeds as follows:

1. Ask the prover for a prime $p \in [2^n, 2^{10n}]$ and $k \in [p-1]$ such that
$$\mathcal{O}_1 \mathcal{O}_2 \mathcal{O}_t g_\phi(x_1, \ldots, x_n) = k \mod p,$$
2. If $t = 1$, check the identity by computing it and terminate the protocol.
3. If $\mathcal{O}_1$ is $\sum_{x_i}$,
   (a) Ask the prover for the polynomial
   $$f(X) = \mathcal{O}_2 \mathcal{O}_3 \ldots g_\phi(x_1, \ldots, x_{i-1}, X, x_{i+1}, x_n),$$
   which is a polynomial of degree at most 2.
   (b) Check that $f(0) + f(1) = k \mod p$.
4. If $\mathcal{O}_1$ is $\prod_{x_i}$,
   (a) Ask the prover for the polynomial
   $$f(X) = \mathcal{O}_2 \mathcal{O}_3 \ldots g_\phi(x_1, \ldots, x_{i-1}, X, x_{i+1}, x_n),$$
   which is a polynomial of degree at most 2.
   (b) Check that $f(0) \cdot f(1) = k \mod p$.
5. If $\mathcal{O}_1$ is L_i, then $x_i = a_i$ has been set to be a constant.

(a) Ask the prover for the polynomial

$$f(X) = \mathcal{O}_2\mathcal{O}_3 \ldots g_\phi(x_1, \ldots, x_{i-1}, X, x_{i+1}, x_n),$$

which is a polynomial of degree at most 2.

(b) Check that $a_i f(0) + (1 - a_i) f(1) = k \mod p$.

6. Pick a random element $a \in \mathbb{F}_p$ and recursively check that

$$f(a) = \mathcal{O}_2\mathcal{O}_3 \ldots g_\phi(x_1, \ldots, x_{i-1}, a, x_{i+1}, x_n)$$

As before, an honest prover can convince the verifier with probability 1. On the other hand, a dishonest prover can succeed only by sending an incorrect polynomial f, and then such a prover will manage to convince the verifier with probability at most $O(t/p) \ll 1/3$.