

Lecture 15 $\text{IP} = \text{PSPACE}$

Lecturer: Anup Rao

Scribe:

1 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 $\exists x_1, \dots, x_n \phi(x_1, \dots, x_n)$ and want to *count* the number of satisfying assignments to this formula. Since the permanent is complete for $\#\text{P}$, 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:

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

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:

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

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

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

4. Check that $f(0) + f(1) = k \pmod p$.
5. Pick a random element $a \in \mathbb{F}_p$ and recursively check that

$$f(a) = \sum_{x_2, x_3, \dots, x_n \in \{0,1\}^{n-1}} g_\phi(a, x_2, \dots, x_n)$$

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 if the prover succeeds, it must be that

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

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

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

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

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 .

2 A protocol for TQBF

To handle checking whether a formula of the type $\exists x_i \forall x_2 \exists x_3 \dots \forall x_n \phi(x_1, \dots, x_n)$ is true, it is clear that this is equivalent to checking the identity that

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

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

$$f(X) = \prod_{x_2} \sum_{x_3} \dots \prod_{x_n} g_\phi(X, x_2, \dots, x_n).$$

4. Check that $f(0) + f(1) = k \pmod p$ (or $f(0) \cdot f(1) = k \pmod p$ as appropriate).

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

$$f(a) = \sum_{x_2, x_3, \dots, x_n \in \{0,1\}^{n-1}} g_\phi(a, x_2, \dots, x_n)$$

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

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

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

%begintheorem [Chinese Remainder Theorem] If k is divisible by distinct primes p_1, \dots, 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 1 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 $\sum_{x_1} \prod_{x_2} \sum_{x_3} \dots \prod_{x_n} g_\phi(x_1, \dots, x_n) = k$.

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

$$\sum_{x_1} \prod_{x_2} \sum_{x_3} \dots \prod_{x_n} g_\phi(x_1, \dots, x_n) = k \not\equiv 0 \pmod{p}.$$

This allows us to fix the first problem: the verifier can ask the prover to send this prime and the value of $k \pmod{p}$, 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 $g(X_1, \dots, X_n)$ define the operator L_1 as

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

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

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

we shall instead ask the prover to prove that

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

In order to describe the protocol, in general we are going to be trying to prove a statement of the form $\mathcal{O}_1 \mathcal{O}_2 \mathcal{O}_t g_\phi(x_1, \dots, x_n) = k \pmod{p}$, 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

$$\mathcal{O}_1 \mathcal{O}_2 \mathcal{O}_t g_\phi(x_1, \dots, x_n) = k \pmod{p},$$

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

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

which is a polynomial of degree at most 2.

- (b) Check that $f(0) + f(1) = k \pmod{p}$.

4. If \mathcal{O}_1 is \prod_{x_i} ,

- (a) Ask the prover for the polynomial

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

which is a polynomial of degree at most 2.

- (b) Check that $f(0) \cdot f(1) = k \pmod{p}$.

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

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

which is a polynomial of degree at most 2.

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

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

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

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 $O(t/p) \ll 1/3$.