Lecture 9: **IP** $=$ **PSPACE**, *Balancing Arithmetic Circuits*

Anup Rao

November 27, 2023

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.

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 1 (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 2 (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 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 $\sum_{x_1} \prod_{x_2} \sum_{x_3} \cdots \prod_{x_n} g_\phi(x_1, \ldots, x_n) = k$. Recall that $k \leq 2^{2^n}$.

Thus by Theorem 2, if $k > 0$, there must be some prime $p \in [2^n, 2^{10n}]$ such that
$$\sum_{x_1} \prod_{x_2} \sum_{x_3} \cdots \prod_{x_n} g_\phi(x_1, \ldots, x_n) = k \neq 0 \mod p.$$

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

Balancing Arithmetic Circuits

IN THIS SECTION, WE FINALLY PROVE something that I mentioned in my very first lecture: it is possible to balance every arithmetic circuit.

Homogenization

FIRST, WE NEED THE CONCEPT of a *homogenous* polynomial/circuit. A polynomial is homogenous if all of its monomials have the same degree. An arithmetic circuit is homogenous if every gate computes a homogenous polynomial. Given a polynomial f of degree d, we write f_i to denote its i'th homogenous part. So, $f = f_0 + \ldots + f_d$.

A useful fact is that every circuit can be made homogenous in the following sense:

Theorem 3. *If f is a degree d polynomial that can be computed by a circuit of size s, then $f_0, \ldots, f_d$ can all be computed by a homogenous arithmetic circuit of size $O(sd^2)$.*

Proof The idea of the proof is to compute $g_0, \ldots, g_d$ for every gate g in the circuit of size s. If $g = u + v$, then $g_i = u_i + v_i$, so the homogenous parts of g can be computed from the homogenous parts of u, v. If $g = u \cdot v$, then $g_i = u_0 \cdot v_i + u_1 \cdot v_{i-1} + \ldots + u_i \cdot v_i$, so once again the homogenous parts of g can be computed. All of these operations may increase the size of the circuit by a factor of $O(d^2)$. ■

The key claim

The key claim we shall make is the following:

Theorem 4. *Suppose $f(X_1, \ldots, X_n)$ is a degree d homogenous polynomial computed by a homogenous arithmetic circuit of size s. Then we can express*

$$f = \sum_{i=1}^{s} u_i v_i,$$

where for every i, u_i and v_i both have degree at least $d/3$ and at most $2d/3$, u_i occurs as a gate in the original circuit, and v_i can be computed by the same circuit after replacing some of the gates with the constants 0 or 1.

Balancing

THEOREM 4 IS extremely powerful. In particular, it implies that one can compute f using a circuit of depth at most $O((\log s)(\log d))$. To see this, generate a circuit of depth $O(\log s)$ that computes f from inputs u_i, v_i as above. Then, since each of u_i, v_i can be computed by circuits of size s, we can recursively apply the Theorem to these polynomials and continue. In each step, the degree of the polynomials we are working with drops by a constant factor, so there can be at most $O(\log d)$ steps.

Even if f is not homogenous, we can use Theorem 3 to make a homogenous circuit computing the homogenous parts of f in size $O(sd^2)$. Then, applying Theorem 4, we obtain a circuit of depth $O((\log sd^2) + \log d) \leq O((\log s + \log d) \log d)$ computing the homogenous parts of f. We can then sum up these parts adding another $O(\log d)$ to the depth to recover f. As a consequence, we obtain:

Theorem 5. *If f is a polynomial of degree d that can be computed using an arithmetic circuit of size s, then f can be computed by an arithmetic circuit of depth $O((\log s + \log d) \log d)$.*

Proving the theorem

FINALLY, LET US TURN to proving the theorem. The given circuit is assumed to be homogenous. In fact, it is no loss of generality to assume that every gate of the circuit computes a polynomial of degree at most d. This is because if the circuit contains a $+$ gate that computes the polynomial 0, then we can eliminate that gate. Once all such gates have been eliminated, we see that every gate computes a polynomial whose degree is larger than the degrees of its inputs. Thus, any gate computing a polynomial of degree larger than d cannot be connected to the output gate, and it can be dropped.

Next we run a process similar to what we have seen when found a way to balance Boolean formulas. Let $a_1, a_2, \ldots$ be a sequence of gates, where a_1 the output gate, and given a_i, a_{i+1} is the gate that feeds into a_i of larger degree (breaking ties arbitrarily). Since the product of two gates adds the degrees, the degree of the polynomial computed by a_{i+1} must be at least $1/2$ of the degree of a_i. Let a_{i+1} be the first gate in this sequence with

$$d/3 \leq \mathsf{deg}(a_{i+1}) \leq 2d/3.$$

By construction, we must have $a_i = a_{i+1} \cdot b$, and the degree of a_i must be greater than $2d/3$. Now, imagine replacing the gate a_i with a new variable Y. Let $g(X_1, \ldots, X_n, Y)$ denote the output of the circuit after making this change, so $f(X_1, \ldots, X_n) = g(X_1, \ldots, X_n, a_i)$, where here a_i denotes the polynomial computed by the gate a_i.

We claim:

Claim 6. *If a gate r in the circuit computing g computes a polynomial containing the monomial $Y \cdot h$, then the degree of r in the circuit for f must be $\mathsf{deg}(a_i) + \mathsf{deg}(h)$.*

The claim holds by induction. It is true for the gate a_i, and given that the claim holds for the inputs of r, it must hold for r, since we have eliminated all gates of the circuit for f that compute the 0 polynomial.

Next, we claim that the degree of Y in g is at most 1. Indeed, if the circuit ever multiplies a polynomial containing Y with another polynomial containing Y, then the degree of this gate in the original circuit has to be at least $4d/3$, but there are no such gates, since we got rid of them in the first step of the proof. Thus, we must have

$$g = h \cdot Y + q,$$

for some polynomials $h(X_1, \ldots, X_n), q(X_1, \ldots, X_n)$.

Now, set $u_1 = a_{i+1}$, $v_1 = h \cdot b$. Then we have

$$f = u_1 \cdot v_1 + q.$$

v_1 can be computed by considering the path from b to the output gate, replacing the gate a_i by 1, and replacing every polynomial that is added to this path by 0.

Moreover, q can be computed by substituting $Y = 0$ in the circuit computing g. Thus, q must be homogenous and have the same degree as f (or be 0). Since q can be computed by a circuit of size at most $s - 1$, the proof is completed by induction.