Lecture 9: **IP** = **PSPACE**

Anup Rao

March 5, 2022

Interactive proofs

One way to define **NP** is via the idea of a proof system. **NP** is the set of functions f for which there is a polynomial time verifier algorithm V such that given any x with $f(x) = 1$, there exists a prover P that can prove to the verifier that $f(x) = 1$ by providing a polynomial sized witness w for which $V(x, w) = 1$, yet if $f(x) = 0$, no such prover exists.

What happens if we allow the verifier to have a longer *interactive* conversation? Presumably, giving the verifier the ability to adaptively ask the prover questions based on his previous responses should give the verifier more power, and so allow the verifier to verify the correctness of the value for a larger set of functions. In fact, this does *not* give the verifier additional power: for if there is such an interactive verifier V^I for verifying that $f(x) = 1$, we can design a non-interactive verifier that does the same job. The new verifier will demand that the prover provide the entire transcript of interactions between V^I and a convincing prover. The new verifier can then verify that the transcript is correct, and would have convinced V^I. Thus, if f has an interactive verifier, then $f \in$ **NP**.

The story is more interesting if we allow the verifier to be randomized. We say that $f \in$ **IP** if there is a polynomial time randomized verifier V such that

Completeness For all x, if $f(x) = 1$, there is an oracle P such that $\Pr_r[V^P(x, r) = 1] \geq 2/3$.

Soundness For all x, if $f(x) = 0$, for every oracle P, $\Pr_r[V^P(x, r) = 1] \leq 1/3$.

Since any prover can be simulated in polynomial space, if $f \in$ **IP**, then $f \in$ **PSPACE**. The algorithm for f can just try all possible sequences of messages from the prover until it finds a sequence of messages that convinces the verifier, if such a sequence exists.

Theorem 1. **IP** $\subseteq$ **PSPACE**.

It is easy to check that allowing the prover to be randomized does not change the model.

We shall eventually prove that **IP** = **PSPACE** (and so **IP** is potentially much more powerful than **NP**).

Example: Graph non-Isomorphism

Two graphs on n vertices are said to be *isomorphic* if the vertices of one of the graphs can be permuted to make the two equal.

Consider the problem of testing whether two graphs are *not* isomorphic: the boolean function f such that $f(G_1, G_2)$ is 1 if and only if G_1 is not isomorphic to G_2. $f \in co\mathbf{NP}$, since the prover can just send the verifier the permutation that proves that they are isomorphic. We do not know if $f \in \mathbf{NP}$, but it is easy to prove that $f \in \mathbf{IP}$.

Here is the simple interactive protocol:

1. The verifier picks a random $i \in \{1, 2\}$.
2. The verifier randomly permutes the vertices of G_i and sends the resulting graph to the prover.
3. The prover responds with $b \in \{1, 2\}$.
4. The verifier accepts if $i = b$.

If G_1, G_2 are not isomorphic, then any permutation of G_i determines i, so the prover can determine i and send it back. However, if G_1, G_2 are isomorphic, then the graph that the prover receives has the same distribution whether $i = 1$ or $i = 2$, thus the prover can guess the value of i with probability at most $1/2$. Repeating the protocol several times, the verifier can make the probability of being duped by a lying prover exponentially small.

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 2 (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 3 (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 2 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 $k \le 2^{2^n}$.

Thus by Theorem 3, 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

$$\sum_{x_1} L_1 \prod_{x_2} L_1L_2 \sum_{x_3} L_1L_2L_3 \prod_{x_4} \cdots L_{n-1}L_n \prod_{x_n} g_\phi(x_1,\ldots,x_n) = k \neq 0 \mod p.$$

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