6.841/18.405J: Advanced Complexity Theory	April 08, 2002
Lecture 14	
Instructor: Madhu Sudan	*Scribe: Dah-Yoh Lim*

In this lecture, we will show that $IP \supseteq PSPACE$; together with $IP \subseteq PSPACE$ (proved last lecture), this concludes our prove that $IP = PSPACE$. Actually, we will first prove that $\#P \subseteq IP$, then introduce straight line programs of polynomials to aid us in proving $IP \supseteq PSPACE$.

1 $\#P \subseteq IP$

Suppose that we have a formula ϕ, and we wish to count the number of satisfying assignments, $\#\phi$, by setting up an interactive protocol. If the prover claims that $\#\phi = N$, how can we check it? Using the self reducibility of SAT, we can progress down the self reducibility tree, and try to verify consistency at every level. Let ϕ_0 denote the formula ϕ with its first variable set to 0, i.e. $\phi_0 = \phi(x_1 = 0)$; likewise, let $\phi_1 = \phi(x_1 = 1)$. Say that the prover tells us that $\phi_0 = N_0$ and $\phi_1 = N_1$. Therefore, the verifier should try to check that $\#\phi = N$ by checking that $N = N_0 + N_1 = (N_{00} + N_{01}) + (N_{10} + N_{11})$ and so on, until it reaches the nodes where all the varibles have been assigned, at which point it can directly check the prover's claim by evaluating the formula. The problem is that the tree is exponential, so the polynomial-timed verifier cannot walk down the whole tree.

The way out of this is to change the logical computations to arithmetic computations, so that we can somehow combine the checks. This is done by setting up an arithmetic way of looking at $\#SAT$.

1.1 Arithmetizing SAT

Consider the following correspondence:

Boolean Constructs		Arithmetic Polynomials
variables:	x_i	z_i
literals:	x_i, $\neg x_i$	z_i, $(1 - z_i)$
clauses:	$C_l = x_i \vee x_j \vee x_k$	$P_l = 1 - (1 - z_i)(1 - z_j)(1 - z_k)$
formulae:	$\phi(x_1, ..., x_n) = \wedge_{l=1}^m C_l$	$Q(\mathbf{z}) = \Pi_{l=1}^m P_l(\mathbf{z})$

Note:

1. For our purposes, we can think of the arithmetic as done over $\mathbb{Z}$, $\mathbb{Z}_p$, or $\mathbb{F}$.
2. For $\mathbf{a} \in \{0, 1\}^n$, $Q(\mathbf{a}) = 1$ iff $\mathbf{a}$ satisfies ϕ.
3. Q is a polynomial of total degree $\leq 3 \cdot m$.
4. $\#\phi = \Sigma_{\mathbf{a} \in \{0,1\}^n} Q(\mathbf{a})$

1.2 $\#SAT \in IP$

So now, instead of working on the N_αs directly (where $\alpha \in \{0,1\}$), we can work with the analogous Q_αs (where $\alpha \in \{0,...,p-1\}$); all calculations are done modulo p. We should think of p as a very large prime (we will determine its value later). This generalizes our previous self-reducibility tree, a binary one, to a p-nary tree. At the root node we have $Q_\lambda \stackrel{\text{def}}{=} \Sigma_{\mathbf{a}\in\{0,1\}^n} Q(\mathbf{a})$; at level one, we have $Q_0 \stackrel{\text{def}}{=} \#\phi = \Sigma_{a_2,...a_n\in\{0,1\}} Q(0,a_2,...,a_n)$, $Q_1 \stackrel{\text{def}}{=} \#\phi = \Sigma_{a_2,...a_n\in\{0,1\}} Q(1,a_2,...,a_n)$, and in general, $Q_\alpha \stackrel{\text{def}}{=} \#\phi = \Sigma_{a_2,...a_n\in\{0,1\}} Q(\alpha,a_2,...,a_n)$.

Suppose the prover claims that $Q_\alpha = \#\phi = N$, and it gives $Q_0 = N_0, Q_1 = N_1, ..., Q_{p-1} = N_{p-1}$ to support its claim. Consider the polynomial $p(x) = \Sigma_{a_2,...,a_n} Q(x,a_2,...,a_n)$, a univariate function of x (the other variables are being summed over); $p(x)$ is still a polynomial of degree $\leq 3\cdot m$, because it is just a sum of polynomials all of degree $\leq 3\cdot m$.

The protocol starts as follows:

1. The prover gives Q_λ, Q_0, Q_1
2. The verifier verifies that $Q_\lambda \leq 2^n$ and rejects if not. The verifier asks for the polynomial $p(x) \stackrel{\text{def}}{=} \Sigma_{a_2,...,a_n} Q(x,a_2,...,a_n)$.
3. The prover responds with $h(x)$ (by sending the coefficients).
4. The verifier verifies that $h(x)$ is of degree $\leq 3\cdot m$, $Q_0 = h(0)$, $Q_1 = h(1)$, and $Q_\lambda = Q_0 + Q_1 (mod p)$; if any one fails, it rejects. Now it picks a random $\alpha \in \mathbb{Z}_p$ and sents it to the prover, asking it to prove that the polynomial $p(\alpha)$ as defined would evaluate to Q_α.

Recursively:

2j. The verifier is trying to verity that $Q_\mathbf{b} = \Sigma_{\mathbf{s}\in\{0,1\}^{n-i}} Q(\mathbf{b},\mathbf{s})$, where the vector $\mathbf{b} = b_1 b_2 \cdots b_i$ represents the sequence of choices that the verifier made (at random) from the root to the current node. Now, the verifier asks for the polynomial $p(x) \stackrel{\text{def}}{=} \Sigma_{\mathbf{s}'\in\{0,1\}^{n-i-1}} Q(\mathbf{b},x,\mathbf{s}')$.

2j+1. The prover responds with a $h(x)$.

2j+2. The verifier verifies that $h(x)$ is of degree $\leq 3\cdot m$ and $Q_\mathbf{b} = h(0) + h(1) (mod p)$; if any one fails, it rejects. Now it picks a random $\alpha \in \mathbb{Z}_p$ and sents it to the prover, asking it to prove that the polynomial $p(\alpha)$ as defined would evaluate to $Q_{\mathbf{b}\alpha}$.

... $\vdots$

2n+4. At the end, the verifier can verify directly the claims, since $Q_\mathbf{a} = Q(\mathbf{a})$, where $\mathbf{a}$ is the vector $\mathbf{a} = a_1 a_2 \cdots a_n$.

1.2.1 Proof of Correctness of the protocol

So far, we have only outlined the protocol, without any claims- this allows us to separate the protocol from its proof.

completeness This is quite obvious as the correct prover can start with the correct value of $\#\phi$, and at all iterations, the prover can send the correct polynomial. Certainly then, every (local consistency) check that the verifier makes will work out, so it accepts with probability 1.

soundness Without loss of generality assume that the prover always responds with $h(\cdot)$ s.t. $h(0)+h(1)=Q_{\mathbf{b}}$ at the i-th level, because or else the verifier would have rejected right away. Also, we assume that $Q_\lambda \neq \Sigma_{\mathbf{a}\in\{0,1\}^n} Q(\mathbf{a})$, i.e. it is a crooked prover that is trying to prove something that is false to the verifier.

Claim 1 *If $Q_\lambda \neq \Sigma_{\mathbf{a}\in\{0,1\}^n} Q(\mathbf{a})$ then the inequality holds (mod p) with probability $\geq \frac{1}{10}$ provided that $p \in_R [n^2, 2n^2]$ or $[10mn, 20mn]$, which ever is larger.*

Proof $Q_\lambda - \Sigma_{\mathbf{a}\in\{0,1\}^n} Q(\mathbf{a})$ has at most $n \leq \frac{n^2}{10logn}$ prime factors for p in the given range. $\blacksquare$

Claim 2 *Suppose $Q_{\mathbf{b}} \neq \Sigma_{\mathbf{s}} Q(\mathbf{b},\mathbf{s})$; then, $Q_{\mathbf{b},\alpha} \neq \Sigma_{\mathbf{s}'} Q(\mathbf{b},\alpha,\mathbf{s}')$ with a certain probability over α.*

Proof $p(x) \stackrel{\text{def}}{=} \Sigma_{\mathbf{s}'} Q(\mathbf{b},\alpha,\mathbf{s}')$; we know that $p(0)+p(1) = \Sigma_{\mathbf{s}} Q(\mathbf{b},\mathbf{s}) \neq Q_{\mathbf{b}} = h(0)+h(1)$, i.e. $p(0)+p(1) \neq h(0)+h(1)$, so $p(\cdot) \neq h(\cdot)$. Since they are polynomials of degree $\leq 3\cdot m$, by the Schwartz lemma we have that for random α, $p(\alpha) \neq h(\alpha)$ with probability 1-$\frac{3m}{p}$. In each iteration, the prover has probability $\frac{3m}{p}$ of getting away with lying; by the union bound the probability that the prover successfully cheats the verifier is $\leq \frac{3mn}{p} \leq \frac{3}{10}$. $\blacksquare$

Combining the above two claims, we see that the soundness is $\frac{4}{10}$.

2 Abstracting the Proof

Note that what helped us in the above proof is not some specific features of $\#P$. What we used is essentially the downward self-reducibility of the language to reduce a complicated property to a collection of progressively less complicated properties. So we could apply the same idea to all languages in $PSPACE$ (which is exactly the collection of self-reducible languages). The arithmetization allows us to effectively compress questions down to one question, without requiring any structure on the questions. Below we look at how we can extend the compression.

2.1 Extending Compression: low-degree curves

Suppose that computing $Q_{\mathbf{b}}(\mathbf{x})$ involves computing $Q'_{\mathbf{b}}(\mathbf{x_0})$ and $Q'_{\mathbf{b}}(\mathbf{x_1})$, where $\mathbf{x_0}$ and $\mathbf{x_1}$ are not related.

Consider lines in an n-dimensonal integer grid: $l : \mathbb{F} \to \mathbb{F}^n$ Geometrically a line is... a line. Algebraically, it is a collection of n functions, each of which is a degree 1 polynomial; the i-th function gives the i-th coordinate. For any two points $\mathbf{x_0}$ and $\mathbf{x_1}$, there is a line through the two points. In other words, $\exists l$ s.t. $l(0) = \mathbf{x_0}$ and $l_1 = \mathbf{x_1}$; specifically, $l(t) = (1-t)\mathbf{x_0} + t\mathbf{x_1}$.

Since Q' maps $\mathbb{F}^n$ to $\mathbb{F}$, the composition of Q' and l, $Q' \circ l : \mathbb{F} \to \mathbb{F}$, is a univariate function. What is nice about this composition is that it preserves degrees: if Q' is of degree d, $Q' \circ l$ is of degree $\leq d$.

Now, to extend our previous protocol's capabilities, we change the protocol to:

i. The verifier wants to verify that $Q(\mathbf{x}) = a$. To do so, it generates $\mathbf{x_0}$, $\mathbf{x_1}$, and a line l through the two points. Now it asks the prover for $Q' \circ l$.

i+1. The prover responds with a degree d univariate polynomial h.

i+2. The verifier checks that h is a univariate polynomial of degree d and rejects if not. It checks the local consistency by checking that $a = h(0) + h(1)$. Iff it is locally consistent, the verifier goes on to verify recursively that $h(\alpha)$ is correct for random α.

Note that as we progress down the tree, the polynomials involved gets simpler- eventually at the leave nodes the verifier will be powerful enough to check the much simpler condition by itself.

2.2 Straight line programs of polynomials

The above extension motivates the following definitions.

Definition 3 *$p_0, ..., p_L$ is an (n,d,L,w)-straight line program of polynomials if:*

1. *Every p_i is on at most n variables.*
2. *Every p_i is of degree at most d.*
3. *p_0 is easy to evaluate (i.e. computable in polynomial time)*
4. *p_i is easy to evaluate given oracle access to p_{i-1} (i.e. there is a polynomial time algorithm A that, given $i, \mathbf{x}$ and an oracle for p_{i-1} can compute $p_i(\mathbf{x})$ making at most w non-adaptive queries to p_i; w is the "width" of the straigt line program).*

Definition 4 *Polynomial straight line program satisfaction is the language whose members are $(< p_0, ..., p_L >, \mathbf{x}, \alpha)$ s.t. $p_L(\mathbf{x}) = \alpha$, where $\mathbf{x} \in \mathbb{Z}^n$, $a \in \mathbb{Z}$, and $< p_0, ..., p_L >$ is an (n, d, L, w)-straigt line program of polynomials.*

Lemma 5 *Polynomial straight line program satisfaction $\in IP$ for $w = 2$.*

Proof The rough idea is as follows:

- The verifier picks a random prime $p \approx poly(n, d, L, log||x||)$ and sends it to the prover. Set $a_L \leftarrow a$ and $x_L \leftarrow x$.
- For $i = L - 1$ downto 0 do:
  - Let $(\mathbf{x_0})_i$ and $(\mathbf{x_1})_i$ be queries of A on input $i + 1$, $(\mathbf{x})_{i+1}$. Let l_i b ethe line through $(\mathbf{x_0})_i$ and $(\mathbf{x_1})_i$. The verifier asks the prover for $p_i \circ l_i$.
  - The prover responds with h_i.
  - The verifier verifies that A's answer on oracle values $h(0)$ and $h(1)$ is a_{i+1} and rejects if not. It picks a random $\alpha \in \mathbb{Z}_p$ and sets $\mathbf{x_i} \leftarrow l_i(\alpha)$ and $a_i \leftarrow h_i(\alpha)$.

At the end the verifier verifies that $h_0(\alpha) = p_0(l_0(\alpha))$. ∎

Lemma 6 *Polynomial straight line program satisfaction is $PSPACE$ complete.*

Proof The basic idea is as follows:

- We fix a PSPACE machine M taking s space with input $\mathbf{x}$. Let $f_i(\mathbf{a}, \mathbf{b})$ be polynomials that have configurations of M (namely $\mathbf{a}$ and $\mathbf{b}$, from $\{0, 1\}^s$) as its inputs, s.t. $f_i(\mathbf{a}, \mathbf{b}) = 1$ if configuration $\mathbf{a}$ yields configuration $\mathbf{b}$ in (exactly) 2^i steps; $f_i(\mathbf{a}, \mathbf{b}) = 0$ otherwise. Notice that f_0 is a constant degree polynomial of degree $C = O(1)$ in each variable.
- $f_{i+1}(\mathbf{a}, \mathbf{b}) = \Sigma_{\mathbf{c} \in \{0,1\}^s} f_i(\mathbf{a}, \mathbf{c}) \cdot f_i(\mathbf{c}, \mathbf{b})$ is also a polynomial of degree C in each variable.

Unfortunately in the above, $w \neq 2$; but we can fix that by doing summation "slowly"- we define a longer sequence:

- $g_i = g_{is} = f_i$.
- $g_{i0}(\mathbf{a}, \mathbf{b}, \mathbf{c}) = g_{i-1,s}(\mathbf{a}, \mathbf{c}) \cdot g_{i-1,s}(\mathbf{c}, \mathbf{b})$.
- $g_{ij}(\mathbf{a}, \mathbf{b}, \mathbf{c}) = g_{i,j-1}(\mathbf{a}, \mathbf{b}, \mathbf{c}0) + g_{i,j-1}(\mathbf{a}, \mathbf{b}, \mathbf{c}1)$, where $\mathbf{c} \in \mathbb{Z}^{s-j}$.
- g has degree at most C in the variables of $\mathbf{a}, \mathbf{b}$, and at most $2C$ in the variables of $\mathbf{c}$.

Then, we have a sequence of width $w = 2$, as required: $g_0, g_{10}, g_{11}, ..., g_{1s}, g_{20}, ..., g_{ss}$. Therefore $PSPACE$ completeness follows. ∎

We have shown that a $PSPACE$-complete problem, (Polynomial straight line program satisfaction), has an IP, implying that $PSPACE \subseteq IP$. Actually, we can further generalize the line arguments even "wider", for $w > 2$. We will leave this as an exercise- this will give a direct proof that the permanent has an interactive proof, where the prover only needs to be able to compute the permanent.