Lecture Lecture 14

Instructor: Madhu Sudan *Scribe: Dion Harmon*

In this lecture we cover

- **IP = PSPACE**
  - Interactive proof for straightline programs.
  - Straightline program for **PSPACE**.
- Other definitions of proofs (and nice applications).
  - Multi-prover interactive proofs (**MIP**) and **MIP** = **NEXP**.
  - Oracle Interactive Proofs.
  - Probabilistically Checkable Proofs (PCP's).

1 IP = PSPACE

We have left to show that **PSPACE** $\subseteq$ **IP**. We do this by using straightline programs of polynomials. We first show there is an interactive proof to evaluate any straightline program of a certain type and then show that any $L \in$ **PSPACE** has a nice straightline program of the same type.

1.1 Straightline Programs of Polynomials

A straightline program of polynomials is characterized by four parameters: n, L, w, and d.

n The number of variables.

L The number of polynomials (minus 1).

w The width of the program.

d The degree of the program.

The program consists of polynomials $P_0, \ldots, P_L$ each of degree at most d in at most n variables. Polynomial P_i is easy to evaluate in polynomial time with at most w calls to P_{i-1}. We will deal with $w = 2$ today. We are interested in verifying claims of the form $P_L(z) = a$.

1.2 Lines in $\mathbf{Z}_p^n$

Algebraically a line in $\mathbf{Z}_p^n$ is a set

$$\ell_{xy} = \{\ell_{xy}(t) \mid t \in \mathbf{Z}_p\} = \left\{ \left[\begin{array}{c} (1-t)x_1 + y_1 \\ (1-t)x_2 + y_2 \\ \vdots \\ (1-t)x_n + y_n \end{array} \right] \middle| \, t \in \mathbf{Z}_p \right\} \tag{1}$$

where x and y are elements of $\mathbf{Z}_p^n$. The definition of a function p from $\mathbf{Z}_p^n$ to $\mathbf{Z}_p^n$ restricted to a line is now straightforward:

$$\begin{array}{rcl} p & : & \mathbf{Z}_p^n \mapsto \mathbf{Z}_p \\ p|_\ell & : & \mathbf{Z}_p \mapsto \mathbf{Z}_p \quad \text{such that} \quad p|_\ell(t) = p(\ell(t)). \end{array}$$

where $\ell(t)$ is defined as in equation (1). Note that if a function P has degree d in $\mathbf{Z}_p^n$ then the restriction of P to a line ℓ has the same degree (in x, y, and t variables).

1.3 Interactive Proof for some width 2 straightline programs.

We consider straightline programs of the form

$$P_i(x) = P_{i-1}(f_i(x)) \quad \sigma_i \quad P_{i-1}(g_i(x))$$

where $\sigma_i \in \{+, \cdot\}$ and f_i and g_i are polytime computable.

We want to see if $P_L(z_L) = a_L$. For the first step we do the following:

Round 1: Both V and P compute $x_L = f_L(z_L)$ and $y_L = g_L(z_L)$.

Round 2: P sends V a polynomial $h_L(t)$ which is supposed to be $P_{L-1}(\ell_{x_L,y_L}(t))$.

Round 3: V checks to make sure that $a_L = h_L(0)\ \sigma_L\ h_L(1)$. If not, V rejects. Otherwise V picks t_L randomly in $\mathbf{Z}_p$. Let $z_{L-1} = \ell_{x_L,y_L}(t_L)$ and $a_{L-1} = h_L(t_L)$.

P and V verify recursively that $P_{L-1}(z_{L-1}) = a_{L-1}$. We end at the question, "Does $P_0(z_0) = a_0$?" and the verifier can compute this in poly time.

This is sound and complete:

Completeness The prover is honest and computes $h_i = P_i|\ell_i$ at each step and the verifier will not reject: we accept with probability 1.

Soundness If $P_i(z_i) \neq a_i$ and the verifier does not reject during step $L-i+1$ then with probability $1-d/p$ $P_{i-1}(z_{i-1}) \neq a_{i-1}$ and the inequality is preserved. The inequality is preserved with probability $(1-d/p)^L \geq 1-dL/p$ through all steps. The verifier checks for itself if $P_0(z_0) = a_0$. If $P_L(z_L) \neq a_L$ it will discover $P_0(z_0) \neq a_0$ with probability $\geq 1-dL/p$. Picking p larger than $2dL$ gives us the necessary bound.

2 Straightline polynomials produce all of PSPACE

2.1 Description of polynomials used

We use only the types of polynomials described at the beginning of section 1.3. Our polynomials are $P_i(x,y)$ and $Q_{i,j}(x,y,z_1,\ldots,z_j)$, where x and y are length n vectors in $\mathbf{Z}_p^n$ and $z_i \in \mathbf{Z}_p$. If the input vectors x and y are length n bit vectors (0's and 1's) then we interpret them as configurations of some **PSPACE** machine M. The vectors x and y have no special meaning if they are not bit vectors. For state vectors x and y let

$$P_i(x,y) = \begin{cases} 1 & \text{if } M \text{ goes from } x \text{ to } y \text{ in exactly } 2^i \text{ steps} \\ 0 & \text{otherwise} \end{cases}$$

To get P_i from P_{i-1}, consider some state z that is halfway between x and y in the execution of M. We have $P_{i-1}(x,z) = P_{i-1}(z,y) = 1$. Execution of M is deterministic so there is only one such z (in $\{0,1\}^n$). Thus

$$P_i(x,y) = \sum_{z \in \{0,1\}^n} P_{i-1}(x,z) \cdot P_{i-1}(z,y).$$

It takes exponential time to sum over all such z so we use the $Q_{i,j}$ polynomials to evaluate the sum efficiently:

$$\begin{aligned} Q_{i,n}(x,y,\underbrace{z_1,\ldots,z_n}_{z \in \mathbf{Z}_p^n}) &= P_{i-1}(x,z) \cdot P_{i-1}(z,y) \\ Q_{i,j}(x,y,z_1,\ldots,z_j) &= Q_{i,j+1}(x,y,z_1,\ldots,z_j,0) + Q_{i,j+1}(x,y,z_1,\ldots,z_j,1) \end{aligned}$$

Thus

$$Q_{i,0} = \sum_{z \in \{0,1\}^n} P_{i-1}(x,z) \cdot P_{i-1}(z,y) = P_i(x,y).$$

Our polynomials (in order) for the straightline program are

$$P_0, Q_{1,n}, \ldots, Q_{1,0}, P_1, Q_{2,n}, \ldots, Q_{2,0}, P_2, \ldots, P_L.$$

Our program has width 2 and length n^2. The degree of this straightline program is discussed below.

2.2 Degree of P_0

We can give P_0 explicitly in terms of single step of the machine:

$$P_0(x,y) = \prod_{j=1}^{n} \left(\begin{array}{|ccc|} \hline x_{i-1} & x_i & x_{i+1} \\ \hline & \multicolumn{1}{|c|}{y_i} & \\ \cline{2-2} \end{array} \right)$$

where n is the total amount of space used in the computation. The polynomial represented by

$$\begin{array}{|ccc|} \hline x_{i-1} & x_i & x_{i+1} \\ \hline & \multicolumn{1}{|c|}{y_i} & \\ \cline{2-2} \end{array}$$

is one if y_i is the appropriate bit given the x bits and 0 otherwise. It is of some constant degree at most D in x_i's and y_i's. The degree of $P_0(x,y)$ in any one x variable is at most $3D$ and D in any y variable. Thus $P_0(x,y)$ is of degree at most $3nD$ which is polynomial in the input size.

2.3 Degree of other polynomials

Assume P_{i-1} is of degree at most d in any one variable (the total degree may be nd). The expression for $Q_{i,n}$ indicates it can be degree at most d in any x or y variables and $2d$ in any z. The sums to get $Q_{i,j}$'s for $j < n$ eliminate one z variable at each step but do nothing to the degree in x and y variables. Thus all P_i's are of degree at most that of P_0 and all $Q_{i,j}$'s are at most twice this.

3 Comments on PSPACE

- This reduction shows that all **PSPACE** problems are self-reducible. This is not true for higher complexity classes.

- Papadimitriu[1] interpreted **PSPACE** as the complexity of games. For example GO is **PSPACE**-complete. One player is the existential quantifier and the other is the universal quantifier: "There is some move player one can make so that for every move player two makes, ..."

  IP is the class of solitare games. The verifier is kind stupid. It generats random numbers and makes sure the rules are followed. The prover is the player and tries to convince the verifier that it can win.

[1]Papadimitriou, C. H. "Games against nature [PSPACE, new characterization]" J. Comp. and Sys. Sci. Oct. 1985; 31(2): 288–301.

The result that **IP** = **PSPACE** shows that for almost any two player game (we don't quite know about Chess for example) there exists a solitare game which is just as "intellectually stimulating" (complex). The solitare game Mah-Jongg was shown to be **PSPACE**-hard[2].

4 Other Models of Proofs

4.1 Multiple Interactive Proofs: MIP

Use more than one prover. Provers do not have knowledge of the discussions between the verifier and the other provers, but they may collaborate before interactions begin (even based on the input). Clearly **IP** $\subseteq$ **MIP**: we ignore the other provers. The other direction is not so clear.

It was known for a long time that **MIP** $\subseteq$ **NEXP**. For the inclusion **NEXP** $\subseteq$ **MIP**, the basic idea is that **NEXP**-complete problems may be phrased as: "Does there exist a P_0 such that $P_L(z) = a$ in a straightline program of polynomials?" With multiple provers, we go through the usual interaction to get a question of the form: "Does $P_0(z_0) = a_0$?" Then we ask this of the second prover.

4.2 Oracle Interactive Proof

In this case, the prover is an oracle with no memory of past interactions. It was shown that **MIP** $\subseteq$ **OIP** $\subseteq$ 2**IP**. The power of having many provers may be simulated by having only two.

4.3 Probabilistically Checkable Proofs (PCP's)

4.3.1 Definintion of PCP$[r, q]$

We use an oracle to examine some string y that could be a certificate for the statement $x \in L$. We can only check a certain number of bits of the certificate y and only flip a certain number of coins. $L \in \mathbf{PCP}[r, q]$ if there exists a verifier v of $x \in L$ such that

- v tosses at most $r(|x|)$ coins
- v queries the certificate oracle in at most $q(n)$ bits
- v runs in polynomial time and accepts with probability
  - 1 if $x \in L$
  - $\leq 1/2$ if $x \notin L$

4.3.2 Examples

Let $|x| = n$.

- $\mathbf{PCP}[0, \text{poly}(n)] = \mathbf{NP}$
- $\mathbf{PCP}[\text{poly}(n), 0] = \textbf{co-RP}$
- $\mathbf{PCP}[\text{poly}(n), \text{poly}(n)] = \mathbf{NEXP}$ (non-trivial to prove)[3]

[2] Condon, A.; Feigenbaum, J.; Lund, C.; Shor, P. "Random Debaters and the Hardness of approximating stochastic functions" SIAM Journal on Computing. April 1997; 26(2): 369–400.

[3] Babai, L.; Fortnow, L.; Lund, C. "Non-determinstic exponential time has two-prover interactive protocols." Computational Complexity. 1991; 1(1): 3–40.

- $\mathbf{PCP}[O(\log(n)), O(1)] = \mathbf{NP}$[4]
- $\mathbf{PCP}[O(\log(n)), 3] = \mathbf{NP}$ This is hard to show[5].

4.3.3 Begin discussion of a $\mathbf{PCP}[O(\log(n)), 3]$ verifier for NP.

We can make a table of random bits and questions we want to ask about the proof y when the corresponding random bits come up.

Bits	Rule
$000\cdots00$	$y_1 = 1 \Rightarrow y_2 = y_{20}$
$000\cdots01$	$y_1 = 0 \Rightarrow y_{32} \vee \bar{y}_{100}$
$111\cdots11$	if $y_2 = 1$ then $y_3 = y_4$ else $y_{10} = y_1$

Each question may be written as an eight clause CNF formula in at most seven variables (we can only check three for any particular evaluation). We can arrange it so that seven of the eight clauses are satisfied for any assignment of the variables. Combine the CNF clauses from all the rules into one forumal with $8M$ clauses. By definition of a **PCP**, at least half the tests will not be satisfied if the proof y is bogus (not valid). Thus $\leq (8M/2) + (7M/2) = 15M/2 = (15/16)8M$ clauses are satisfied for the certficate y of any $x \notin L$. We go over this discussion in more detail next lecture and use the result to prove that no $(16/15 + \epsilon)$-approximation (see definition in next lecture) exists for SAT.

[4]Arora, S.; Safra, S. 1992. "Probabilistic checking of proofs: A new characterization of NP." *Proceedings of the 33rd IEEE Symposium on Foundations of Computer Science.* IEEE, New York, pp. 2–12.

[5]Guruswami, V. et al. "A tight characterization of NP with 3 query PCPs." *Proceedings 39th Annual Symposium on Foundations of Computer Science.* 1998: 8–17.

Hastad, J. "Some optimal inapproximability results." In Proceedings of the 29th ACM Symposium on the Theory of Computation, 1997: 1–10.