Lecture 26

Daniel Apon

1 From IP=PSPACE to NP=PCP(log, 1): NEXP has multi-prover interactive protocols

If you've read the notes on the history of the PCP theorem referenced in Lecture 19 [3], you will already be familiar with the excitement that surrounded discoveries such as IP=PSPACE and NP=PCP(log, 1). Taken alone, however, these two theorems might seem at most cursorily related to one another. The earlier discovery of IP=PSPACE deals with a much stronger class, assuming the polynomial hierarchy doesn't collapse, than the NP version of the PCP theorem. The former characterizes the verification power of a Turing machine with an adaptive, all-powerful prover, while the latter studies an efficient, probabilistic notion of constant-query proof checking with applications to inapproximability of NP-complete problems, and so on.

But in fact, a series of natural generalizations lead from one to the other. An immediate step after IP=PSPACE is to ask what happens in the case of *multiple*, all-powerful provers interacting with a single PPT verifier. In particular, for what power of computational problems can a *multi-prover interactive proof* system correctly verify candidate-solutions with high probability?

As it turns out, we can (and will, in this lecture) show that MIP=NEXP, where MIP is the class of languages who have a multi-prover interactive proof and NEXP is the class of languages that can be decided in nondeterministic exponential time. The techniques involved are similar to that in the proof IP=PSPACE; however, the fact that we want to reliably verify exponential-sized objects using a PPT machine requires new ideas. We will show that the fact that the verifier interacts with multiple provers (who can speak with the verifier but may not speak with one another during the course of the protocol) gives us the desired power, because the verifier has access to an additional consistency test – namely, re-query new provers on a random subset of the verifier's queries made during the course of some protocol.

Just beyond the scope of this lecture, the ideas used in the proof of MIP=NEXP can be used to prove NEXP=PCP(poly, poly). A series of attempts to "scale down" this result to NP eventually produced the PCP theorem of Arora, et al. (See [3] for more on this.)

In the sequel, we prove MIP=NEXP following the original paper by Babai, Fortnow, and Lund [1] and Lund's PhD thesis [4].

2 Preliminaries

Formally, we define the class of decision problems MIP as follows. Let the provers $P_1, ..., P_k$ be computationally unbounded Turing machines. Let the verifier V be a PPT machine. We allow each P_i to communicate with V and vice versa during the course of the protocol, but

we do not allow the P_i to communicate with one another. Then we say $L \in \mathsf{MIP}$ if

$$x \in L \Longrightarrow \Pr[P_1, ..., P_k \text{ convince V to accept x}] = 1$$
$$x \notin L \Longrightarrow \forall P'_1, ..., P'_k, \Pr[P'_1, ..., P'_k \text{ convince V to accept x}] \leq 1/2$$

where the P' notation denotes a dishonest prover and where the constants are arbitrary.

For completeness, we recall the definition of NEXP here as well; namely,

$$\mathsf{NEXP} \stackrel{\text{def}}{=} \bigcup_{c \in \mathbb{N}} \mathsf{NTIME}\left(2^{n^c}\right).$$

3 $\mathsf{MIP} \subseteq \mathsf{NEXP}$

The first direction of MIP=NEXP is, by comparison, rather simple.

Theorem 1. $\mathsf{MIP} \subseteq \mathsf{NEXP}$.

Proof Sketch. We need to show that every multi-prover interactive proof system can be faithfully, locally simulated by a NEXP machine. Crucially, observe that since the verifier is a PPT machine, the length of the message history between the $P_1, ..., P_k$ and V is bounded by some polynomial in the input size.

Let $L \in \mathsf{MIP}$. Construct a NEXP machine M that behaves as follows. On input x, M nondeterministically guesses a strategy for each of the P_i. We view the strategy of each prover P_i as a function $\mathcal{S}_i : \{0,1\}^{n^c} \to \{0,1\}^{n^{c'}}$ that determines the messages sent by each P_i in response to its interaction with V. Since M is a NEXP machine, M can guess and write down the entire (exponentially large) function table of each $\mathcal{S}_i$.

Then, M computes the probability that V accepts by enumerating over all possible coin flips of V and simulating its computation on each. From this, M decides (precisely) whether V accepts with high probability or not. Therefore, $\mathsf{MIP} \subseteq \mathsf{NEXP}$. □

4 $\mathsf{NEXP} \subseteq \mathsf{MIP}$

The other direction of MIP=NEXP proceeds in three parts.

1. First we show a NEXP version of the Cook-Levin theorem (recall the NP version proved SAT to be NP-complete). This allow us to reduce questions of the form "$x \stackrel{?}{\in} L$" for $L \in \mathsf{NEXP}$ to the satisfiability of an (exponentially large) Boolean formula Φ.

2. Then, similar to IP=PSPACE, we convert Φ into an arithmetic formula P using arithmetization. In particular, to enable the verifier to participate, we ensure that can do this step both in polynomial time in x (only given access to x, but not Φ) and in a way that allows evaluation of any desired clause of Φ in polynomial time (given an assignment to the variables of the clause) by evaluating P.

3. Using P, we design an MIP protocol for any $L \in \mathsf{NEXP}$ that is similar to the IP protocol for PSPACE with a few modifications. We first will need an efficient, probabilistic *multilinearity* test so that the verifier can reliably constrain the space of legal messages from the provers. Then, we proceed as with the IP-PSPACE protocol, except that the verifier asks the provers for assignments to P.

4.1 NEXP Cook-Levin

We state the NEXP version of the Cook-Levin theorem without proof.

Theorem 2. (NEXP Cook-Levin) *Fix $L \in \mathsf{NEXP}$. Then there is a constant c such that*

1. *for every input $x \in \{0,1\}^n$, there is a 3-CNF formula on an exponential number of variables and clauses; that is,*

$$\Phi_x(X(0), X(1), ..., X(2^{n^c}-1)) = \bigwedge_{i=0}^{2^{n^c}-1} C_i$$

where

   (a) $C_i = t_1^i X(b_1^i) \vee t_2^i X(b_2^i) \vee t_3^i X(b_3^i)$,
   (b) $t_j^i \in \{0,1\}$ *(that is, "0X" = "$\overline{X}$" and "1X" = "X"), and*
   (c) $b_j^i \in \{0,1\}^{n^c}$,

2. *and there is a polynomial-time algorithm that takes $x \in \{0,1\}^n$ as input and outputs a circuit $g_x : \{0,1\}^{n^c} \to \{0,1\}^{3n^c+3}$ with the following properties:*

   (a) *on input $0 \leq i < 2^{n^c}$, g_x outputs $t_1^i, t_2^i, t_3^i, b_1^i, b_2^i, b_3^i$,*
   (b) *$x \in L \iff$ there is an assignment to the variables $X(0), ...,$ such that every C_i is satisfied.*

4.2 Arithmetization of NEXP

Now we are interested in finding a polynomial-time computable arithmetization of Φ_x. To begin, define the function f_x as

$$f_x(i, t_1, t_2, t_3, b_1, b_2, b_3) = 1 \iff t_1^i A(b_1^i) \vee t_2^i A(b_2^i) \vee t_3^i A(b_3^i) = 1.$$

Then we will rewrite Theorem 1 as:

Lemma 1. *Let $L \in \mathsf{NEXP}$ and $x \in \{0,1\}^n$. Then there is a constant c and a function $f_x : \{0,1\}^{4n^c+3} \to \{0,1\}$ computable in time polynomial in $|x|$ such that*

$$x \in L \iff \exists A : \{0,1\}^{n^c} \to \{0,1\} : \forall i, t_1, t_2, t_3, b_1, b_2, b_3 :$$

$$[f_x(i, t_1, t_2, t_3, b_1, b_2, b_3) = 1] \iff [t_1^i A(b_1^i) \vee t_2^i A(b_2^i) \vee t_3^i A(b_3^i) = 1].$$

In particular, we will arithmetize the following sentence:

$$\mathcal{S}_1 \stackrel{\text{def}}{=} \forall i, t_1, t_2, t_3, b_1, b_2, b_3 : [f_x(i, t_1, t_2, t_3, b_1, b_2, b_3) = 1] \iff [t_1^i A(b_1^i) \vee t_2^i A(b_2^i) \vee t_3^i A(b_3^i) = 1]$$

which is equivalent to:

$$\mathcal{S}_2 \stackrel{\text{def}}{=} \neg\exists i, t_1, t_2, t_3, b_1, b_2, b_3 : [f_x(i, t_1, t_2, t_3, b_1, b_2, b_3) = 1] \wedge \neg[t_1^i A(b_1^i) \vee t_2^i A(b_2^i) \vee t_3^i A(b_3^i) = 1]$$

We will begin with a subexpression, and let true be 1 and false be 0:

$$\mathcal{S}_3 \stackrel{\text{def}}{=} [f_x(i, t_1, t_2, t_3, b_1, b_2, b_3) = 1] \wedge \neg[t_1^i A(b_1^i) \vee t_2^i A(b_2^i) \vee t_3^i A(b_3^i) = 1]$$

For completeness, we briefly review polynomial interpolation before Lemma 2 and its proof.

In general, *polynomial interpolation* is the task of "fitting" polynomials to a set of discrete data points. The following fact is derived from the Fundamental Theorem of Algebra.

Fact 1. *Let $x_0, x_1, ..., x_n$ be $n+1$ distinct points in $[a, b]$. Then there exists a* unique *polynomial p of degree $\leq n$ that interpolates $f(x)$ at the points $\{x_i\}$; that is, $p(x_i) = f(x_i)$, for $0 \leq i \leq n$.*

So regardless of which interpolation technique we use, the difference is only in factors such as the computational complexity of the algorithm, degree of error in the interpolation, stability of the solution, and so forth.

A full treatment is outside the scope of this lecture; however in general, the interpolation polynomial p of a set of $n+1$ data points is a linear combination

$$p(x) = \sum_{i=0}^{n} y_i \ell_i(x)$$

of Lagrange basis polynomials

$$\ell_i(x) = \prod_{\substack{0 \leq m \leq n \\ m \neq i}} \frac{(x - x_m)}{(x_i - x_m)} = \frac{(x - x_0)}{(x_i - x_0)} \cdots \frac{(x - x_{i-1})}{(x_i - x_{i-1})} \frac{(x - x_{i+1})}{(x_i - x_{i+1})} \cdots \frac{(x - x_n)}{(x_i - x_n)}.$$

Lemma 2. *There is a polynomial-time computable polynomial $P_x(i, b_1, b_2, b_3, t_1, t_2, t_3, z, a_1, a_2, a_3)$ with degree at most $n^{O(1)}$ such that for all $i, b_1, b_2, b_3 \in \{0,1\}^{n^c}$ and for all $t_1, t_2, t_3 \in \{0, 1\}$,*

$$\sum_{z \in \{0,1\}^{n^{O(1)}}} P_x(i, b_1, b_2, b_3, t_1, t_2, t_3, z, A(b_1), A(b_2), A(b_3)) = \begin{cases} 1, & \text{if } \mathcal{S}_3 \text{ is true} \\ 0, & \text{otherwise} \end{cases}$$

Proof. To arithmetize

$$[t_1^i A(b_1^i) \vee t_2^i A(b_2^i) \vee t_3^i A(b_3^i) = 1]$$

define a predicate

$$p(t_1, t_2, t_3, a_1, a_2, a_3) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{if } (a_1, a_2, a_3) \text{ satisfies a clause of the form } (t_1, t_2, t_3) \\ 0, & \text{otherwise} \end{cases}$$

Let $q(t_1, t_2, t_3, a_1, a_2, a_3)$ be a polynomial that interpolates p.

Now to arithmetize

$$[f_x(i, t_1, t_2, t_3, b_1, b_2, b_3) = 1]$$

observe that f_x is a deterministic polynomial time computable function, so there will be exactly one valid computation z. Then, via the standard NP version of Cook-Levin and standard linear-time arithmetization, we get a polynomial time computable arithmetic formula F_x such that

$$\sum_{z \in \{0,1\}^{n^{O(1)}}} F_x(i, t_1, t_2, t_3, b_1, b_2, b_3, z) = \begin{cases} 1, & \text{if } f_x(i, t_1, t_2, t_3, b_1, b_2, b_3) \text{ is true} \\ 0, & \text{otherwise} \end{cases}$$

where the constant in the summation depends on L.

Combining the above (along with the original $\mathcal{S}_3$ expression), we get

$$\sum_{z\in\{0,1\}^{n^{O(1)}}} P_x(i,b_1,b_2,b_3,t_1,t_2,t_3,z,A(b_1),A(b_2),A(b_3))$$

$$= \sum_{z\in\{0,1\}^{n^{O(1)}}} F_x(i,t_1,t_2,t_3,b_1,b_2,b_3,z)(1-q(t_1,t_2,t_3,a_1,a_2,a_3))$$

$$= \begin{cases} 1, & \text{if } \mathcal{S}_3 \text{ is true} \\ 0, & \text{otherwise} \end{cases}$$

□

Recall the definitions of $\mathcal{S}_1$ and $\mathcal{S}_2$; that is, that

$\mathcal{S}_1 = \forall i,t_1,t_2,t_3,b_1,b_2,b_3 : [f_x(i,t_1,t_2,t_3,b_1,b_2,b_3)=1] \Rightarrow [t_1^i A(b_1^i) \vee t_2^i A(b_2^i) \vee t_3^i A(b_3^i) = 1]$ and

$\mathcal{S}_2 = \neg\exists i,t_1,t_2,t_3,b_1,b_2,b_3 : [f_x(i,t_1,t_2,t_3,b_1,b_2,b_3)=1] \wedge \neg[t_1^i A(b_1^i) \vee t_2^i A(b_2^i) \vee t_3^i A(b_3^i) = 1]$.

To arithmetize these, now let true be 0 and false be >0.

Lemma 3. *For all x, there is a polynomial-time computable polynomial $P_x(i,b_1,b_2,b_3,t_1,t_2,t_3,z,a_1,a_2,a_3)$ with degree at most $n^{O(1)}$ such that*

$$\sum_{i,b_1,b_2,b_3,t_1,t_2,t_3,z} P_x(i,b_1,b_2,b_3,t_1,t_2,t_3,z,A(b_1),A(b_2),A(b_3)) = \begin{cases} 0, & \textit{if } \mathcal{S}_1 \textit{ is true} \\ >0, & \textit{otherwise} \end{cases}$$

where $i,b_1,b_2,b_3,t_1,t_2,t_3,z \in \{0,1\}^{n^{O(1)}}$.

Proof. Beginning with

$$\sum_{i,b_1,b_2,b_3,t_1,t_2,t_3,z} P_x(i,b_1,b_2,b_3,t_1,t_2,t_3,z,A(b_1),A(b_2),A(b_3))$$

$$= \sum_{i,b_1,b_2,b_3,t_1,t_2,t_3} \sum_z P_x(i,b_1,b_2,b_3,t_1,t_2,t_3,z,A(b_1),A(b_2),A(b_3))$$

Then from Lemma 2,

$$= \sum_{i,b_1,b_2,b_3,t_1,t_2,t_3} \begin{cases} 1, & \text{if } \mathcal{S}_3 \text{ is true} \\ 0, & \text{otherwise} \end{cases}$$

$$= \begin{cases} 0, & \text{if } \mathcal{S}_2 \text{ is true} \\ >0, & \text{otherwise} \end{cases}$$

□

Then rewriting Lemma 1 using Lemmas 2 and 3 gives:

Theorem 3. (Arithmetization of NEXP) *Let $L \in \mathsf{NEXP}$. Then there is a constant c such that for all n and for all $x \in \{0,1\}^n$ there is a polynomial $P_x(i, b_1, b_2, b_3, t_1, t_2, t_3, z, a_1, a_2, a_3)$ such that*

$$x \in L \iff \exists A : \{0,1\}^{n^c} \to \{0,1\} \text{ such that}$$

$$\sum_{i,b_1,b_2,b_3,t_1,t_2,t_3,z} P_x(i, b_1, b_2, b_3, t_1, t_2, t_3, z, A(b_1), A(b_2), A(b_3)) = 0$$

where $i, b_1, b_2, b_3, t_1, t_2, t_3, z \in \{0,1\}^{n^{O(1)}}$ for some constant exponent depending on c and L. Moreover, P_x is computable in time polynomial in $|x|$ and has degree at most $n^{O(1)}$.

4.3 The protocol

To set-up the protocol, we state the following theorem due to Fortnow, Rompel, and Sipser, which helps characterize the type of protocol we will need.

Let M be a probabilistic polynomial-time machine with access to an oracle O (i.e. a PPTO machine). We say a language L is accepted by a PPTO machine M *iff*

1. For every $x \in L$, there is an oracle O such that M^O accepts x with probability $> 1 - 2^{-n}$.
2. For every $x \notin L$ and for all oracles O', $M^{O'}$ accepts with probability $< 2^{-n}$.

Theorem 4. *([2]) L is accepted by a PPTO machine* iff *L is accepted by a multi-prover interactive protocol.*

As mentioned before, FRS's proof of this fact uses a procedure that, given some protocol $\mathcal{P}$, runs $\mathcal{P}$ then has the verifier randomly re-query new provers on some subset of the interaction in $\mathcal{P}$'s transcript to ensure that the provers abide by a strategy *fixed beforehand*. We, however, will use this theorem to abstractly treat the set of provers $P_1, ..., P_k$ as a single, non-adaptive oracle – namely, an oracle that holds a fixed, exponentially-long proof.

Next, we extend the domain of the assignment function A from $\{0,1\}^{n^c}$ to $\mathcal{I}^{n^c}$ for some interval $\mathcal{I} = \{0, ..., N-1\}$ of a field $\mathcal{F}$, for some N sufficiently large. We will say $A : \mathcal{I}^{n^c} \to \mathcal{F}$ satisfies Φ_x *iff* $A|_{\{0,1\}^{n^c}}$ satisfies Φ_x, where $A|_{\{0,1\}^{n^c}}$ is the restriction of A to a domain of $\{0,1\}^{n^c}$. This allows us to provide a very clear structure to the possible responses of the provers.

In particular, we say that a function is *multilinear* if it is linear in each of its variables. We say that a function $f : \mathcal{I}^{n^c} \to \mathcal{F}$ is *ϵ-approximately multilinear* if the probability that f disagrees with a multilinear function is bounded from above by ϵ. Then we have the following:

Theorem 5. *([1]) Let A be an arbitrary function from $\mathcal{I}^{n^c}$ to $\mathcal{F}$ and let $\epsilon > 0$ be arbitrarily small. Then there exists a probabilistic* multilinearity test *running in time polynomial in $|x|$ and $1/\epsilon$ such that given access to A as an oracle, if A is not ϵ-approximately multilinear, then the test rejects with high probability.*

Define a *line in $\mathcal{F}^m$* as a set of the form $\{\vec{x} + t \cdot \vec{y} \mid t \in \mathcal{F}\}, \vec{x}, \vec{y} \in \mathcal{F}^m, \vec{y} \neq \vec{0}$. Then the idea of the multilinearity test is to select m_1 random lines, and then m_2 random points of each line. The verifier asks the prover-oracle for assignments to the given vectors of variables and checks that A restricted to these points is linear. For m_1, m_2 sufficiently large, a (careful) analysis

shows that the test distinguishes ϵ-approximately multilinear functions from functions far from multilinear with high probability.

Moreover, we note that A *can* be represented as a multilinear function by an honestly-constructed oracle.

But why do we care that A is multilinear? The Scwartz-Zippel lemma states:

Lemma 4. (Schwartz-Zippel) *If f, g are two different, degree d polynomials, then $f(\vec{x}) = g(\vec{x})$ holds for an at most $\frac{d}{|\mathcal{F}|}$ fraction of all $\vec{x} \in \mathcal{F}^m$.*

Intuitively, all multilinear functions arithmetize to low-degree polynomials, and so we are guaranteed that two different low-degree polynomials differ in some large constant fraction of their coordinates.

Finally, we describe the protocol in the context of a verifier interacting with a prover-oracle:

> On input x, the verifier computes the polynomial P_x as above. The provers generate a fixed oracle for the assignment function A for P_x. Then the verifier runs the multilinearity test against the oracle's assignment. If it rejects, the verifier aborts the protocol and rejects; otherwise, the verifier continues to the next phase.
>
> In the second phase, the verifier runs the IP=PSPACE protocol with the oracle for A. However, whenever the verifier needs to evaluate a polynomial expression, it uses self-correction from the proof of the PCP theorem to account for lack of access to A. (We note in passing that the explicit notion of "self-correction" was not developed until after MIP=NEXP but that the exact techniques are moral equivalents.) In particular, in each round, the verifier takes some variable v (where v is one of the i, b's, t's, or z) out of the expression, checks consistency of the expression summed over v, then replaces v with a random value $r \in \mathcal{F}$, and sends r to the provers.
>
> Finally, at the end of the protocol, the verifier asks the oracle for a final assignment to the remaining (constant-sized) arithmetic expression. If and only if all of the evaluations including the final test are consistent and correct, the verifier accepts.

Correctness Sketch. Observe that any honest prover P can always succeed in convincing the verifier to accept. However, any dishonest prover P' must continue cheating throughout the course of the protocol once P' begins cheating, by the same consistency test as in IP=PSPACE. After some polynomial number of rounds, the expression being manipulated reaches a polynomial in some constant number of variables, and the verifier can test correctness with a final substitution.

Since the arithmetic formula P_x is a degree d polynomial, if it differs from a valid assignment, then it differs in many locations by the Schwartz-Zippel lemma, and with high probability, the verifier will catch such a cheating prover. Alternatively, if the prover tries to use an arithmetic formula that is not a degree d polynomial to fool the verifier in the second phase, then the verifier will catch this deviation with the multilinearity test and reject with high probability. Finally, if the prover attempts to use an adaptive assignment (rather than a single, fixed assignment to P_x), the verifier will catch this deviation with high probability

by Theorem 4 and the definition of PPTO machines. □

Therefore, we have:

Theorem 6. NEXP $\subseteq$ MIP.

Then with Theorem 1, we have:

Corollary 1. MIP = NEXP.

References

[1] L. Babai, L. Fortnow, and C. Lund. Nondeterministic exponential time has two-prover interactive protocols. In SFCS (FOCS) '90.

[2] L. Fortnow, J. Rompel, and M. Sipser. On the power of multi-prover interactive protocols. In *Theoretical Computer Science* '94. Available at `http://people.cs.uchicago.edu/~fortnow/papers/mip.pdf`

[3] V. Guruswami and R. O'Donnell. A History of the PCP Theorem. Available at `http://www.cs.washington.edu/education/courses/533/05au/pcp-history.pdf`

[4] C. Lund. On the power of interaction. ACM Distinguished Dissertation, '91.