| | |
|---|---|
| **18.405J/6.841J Advanced Complexity Theory** | 24 April 2001 |

Lecture 19

*Lecturer: Daniel Spielman* *Scribe: Ken McCracken*

In this lecture we begin the proof of the following theorem:

**Theorem 1** **NEXP** $\subseteq$ **PCP** (poly,poly)

The lecture is divided into three parts:

- Section 1 presents a review of $\epsilon$-reductions through arithmetization techniques from last lecture's proof that PSPACE is contained in IP.
- Section 2 provides an introduction to the EXP-complete language Implicit-Circuit-Sat as well as a proof outline for using Implicit-Circuit-Sat to show **NEXP** $\subseteq$ **PCP** (*poly,poly*).
- Section 3 contains an introduction to multilinear polynomials and applications to the Implicit-Circuit-Sat proof which contains the satisfying assignments of the circuit.
- Section 4 introduces the multilinearity test to be used in arithmetization of said proof.

# 1 Last Class

Last class we proved that **PSPACE** $\subseteq$ **IP**. Recall the proof strategy. Namely, for a **PSPACE** $TM$ $M$ with input $w$ consider the graph (e.g. tableau) of configurations of $M$ on $w$. We presented the $\{0,1\}$-function $FT_k(q_1, q_2) = FromTo(q_1, q_2, k) \equiv 1 \Leftrightarrow \exists\ path\ p\ from\ state\ q_1\ to\ q_2\ of\ length\ at\ most\ k$. The proof merely asked if there's a path $FromTo\left(q_s tart, q_a ccept, 2^{|w|}\right)$.

Recall the main parts of an $\epsilon$-reduction:

- The two-for-one lemma.
- The $\Sigma$ protocol, where: $\Sigma_{(x_1,\ldots,x_n)\in\{0,1\}^n} f(x_1, \ldots, x_n) = s$ was solved by checking, for random $c_i \in \{1, \ldots, \lceil \frac{d}{\epsilon} \rceil\}$, if $f(c_1, \ldots, c_n) = r$.

Arithmetization was used with $\epsilon$-reductions from from $FT_k(x, y) = s \rightarrow FT_{\left(\frac{k}{2}\right)}(u, v) = r$. This divide-and-conquer technique brought us to statements of the form $FT_1(u, v) = r$, which could evaluate in poly time.

We will use a similar technique in approaching the proof of the statement **NEXP** $\subseteq$ **PCP** (*poly,poly*).

# 2 Implicit Circuit Sat and The Proof Outline

We use the definition $L \in$ **PCP** (*poly,poly*) $if\ \exists\ a\ probabilistic\ poly\ time\ OTM\ V^?\ s.t.$

$w \in L \Rightarrow \exists\ \Pi\ s.t.\ Pr[V^{\Pi}(w)\, accepts] = 1$

$w \notin L \Rightarrow \forall\ \Pi\ Pr[V^{\Pi}(w)\, accepts] < \frac{1}{2}$

It is easy now to show

**Lemma 2** **PSPACE** $\subseteq$ **PCP** (poly,poly)

**Sketch of Proof** Consider an **IP** problem in which a prover $P$ is a function from the dialog history thus far to the next statement. The act of parsing $\Pi$ turns it into a prover. The verifier $V$ replaces its interactions with the prover with queries to the oracle. The result is that $\mathbf{IP} \subseteq \mathbf{PCP}$ (*poly,poly*). Since $\mathbf{PSPACE} \subseteq \mathbf{IP}$ the lemma holds. ■

Now we return to the more interesting case of proving Theorem 1. Note first that for **PCP** (*poly,poly*) the proof $\Pi$ may be at most exponential in length, since we need to be able to ask for a specific bit of the proof in poly time.

**Theorem 3** *Implicit-Circuit-Sat is* **NEXP**-*complete*

**Proof Idea** Follow the same reasoning as in proofs that SAT is **NP**-complete, using **NEXP** machinery to solve the exponentially large problem. ■

If $w$ is the input to Implicit-Circuit-Sat, $C(w)$ describes an exponentially large circuit. Moreover, if $\Pi$ is a proof containing a satisfying assignment $A$ then $A$ has exponentially many variables. If the variables to $C$ are $\{x_1, \ldots, x_{2^n}\}$ then $A$ defines the mapping $x_i \to 0, 1$ for all $i$.

The input to Implicit Circuit Sat is a circuit computing the function $C(x_1, \ldots, x_{2^n}) \Rightarrow \{0,1\}^{3n+3}$. Such a circuit describes an instance of SAT with $2^n$ clauses.

We are asking if the 3cnf of expenential length that $C$ describes is satisfiable. But a poly time machine can't even read the satisfying assignment within its bounds.

**Idea:** Let $\phi(C)$ be the 3cnf instance described by $C$

**Lemma 4** $\exists$ *a* **PSPACE** *OTM* $M^?$ *s.t.* $M^A(C)$ *accepts* $\Leftrightarrow$ $A$ *is a satisfying assignment of the variables in* $\phi(C)$.

**Sketch of Proof** We consider the special case of Implicit-3Sat, which is also **NEXP**-complete. This problem takes input $(x_1, \ldots, x_n) \in \{1, \ldots, 2^n\}$ and ouputs a description of a clause of the 3cnf formula it represents. That is, it outputs the three variables in the clause including any negations. A **PSPACE** Turing machine can, with access to the oracle $A$ containing the satisfying assignments, solve $C(w)$. It can iterate through every one of the clauses and verify that the corresponding assignments from $A$ satisfy the clause. If any clause isn't satisfied, reject. If no iteration has rejected we can accept. ■

**Outline of Proof** We return to Lemma 1. We now apply the prover-verifier ideas from the proof that $\mathbf{PSPACE} \subseteq \mathbf{PCP}$ (*poly,poly*). In our case, the proof $\Pi$ should contain:

- $A$, the satisfying assignment of $\phi(C)$.
- The table for the prover in **IP** that shows $M^A(C)$ accepts.

At the very end of whatever analysis we do, the verifier evaluates some polynomial at some point. Womehow that involves checking A in just one place. Recall that in the PSPACE proof, we evaluate $FT_1(x, y)$ where x and y are states. We evaluated this polynomial for $x, y \in F^n$, instead of over all $\{0,1\}^n$. The field F was helpful because it placed less constraints on the computational complexity of the problem.

Our problem in this case comes from dealing with the proof A:

Problem 1. A enters into the transition function. The transition function is no longer a short description. Before we had 6 cells from the tableau from state x to state y to check if the move was valid.

Problem 2. We need some way to arithmetize A as we arithmetized our function FT.

Problem 3. Philosophy: If you change 1 bit of A, it can switch from a satisfying assignment to an unsatisfying one. You will never see this in your poly time machine. To overcome this predicament we will introduce error correcting code.

# 3 Multilinear Polynomials

We introduce multilinear polynomials as an approximation to use to arithmetize our exponential-length proof A. Once we can arithmetize it, it becomes tractable to reduce the problem of Implicit-Circuit-Sat using $\epsilon$-reductions that a **PCP** (*poly,poly*) machine can handle within its time and oracle constraints.

We can view A as a function $\{0,1\}^n \rightarrow \{0,1\}$. We represent this function by a multilinear polynomial $\hat{A}$, which we call the *multilinear extension* of A.

In a multilinear polynomial P, if we look at the degree in each variable, it is at most 1. We define P as follows

$$P \equiv \Sigma_{(d_1,\ldots,d_n)\in\{0,1\}} \alpha_{d_1,\ldots,d_n} x_1^{d_1} x_2^{d_2} \ldots x_n^{d_n}$$

Note this is the sum of $2^n$ monomials. We have that $\exists$ a unique multilinear polynomial $P\left(\right) \mid P\left(\bar{x}\right) = A\left(\bar{x}\right) \forall \bar{x} \in \{0,1\}^n$. Hence our proof $\Pi$ should contain a table of values of P at all $\bar{x} \in F^n$. We denote this table by $\hat{A}$, the multilinear extension of A. We can now ask for $\hat{A}$ beyond $\{0,1\}$.

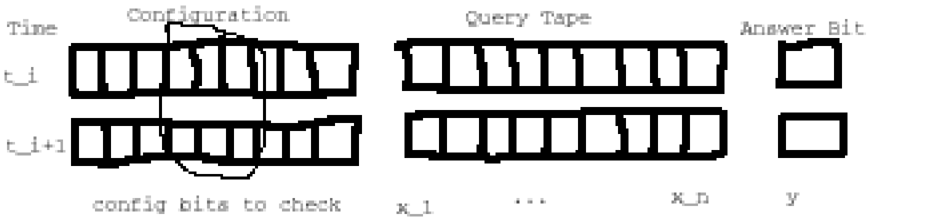

Figure 1: Examine the config bits, use query tape to ask oracle for answer

In figure 1 we now have our normal configurations to check from time step 1 to step 2 but we also have a query tape to our function $\hat{A}$.

Hence we are interested in the value of $1 - \left(y - \hat{A}\left(x_1, \ldots, x_n\right)\right)^2$. In the 0,1-case, this equals 1 if $y = \hat{A}\left(x_1, \ldots, x_n\right)$ and 0 otherwise. We can be convinced that $\hat{A}$ is mostly a polynomial. This will handle Problems 1 and 2 mentioned earlier. The goal will therefore be to build $\hat{A}$ into the transition function. We will be able to evaluate $\hat{A}$ in poly time if it is multilinear. Once we have our table, we can evaluate $FT_1$ at any point by table lookup into $\hat{A}$.

# 4 The Multilinearity Test

We now present the idea of using a multilinearity test to determine whether or not $\hat{A}$ is, in fact, multilinear. Once we know this we can use it in our reductions to $FT_1$. We will come back to multilinearity tests next lecture and apply it to our proof of Lemma 1.

It is important that we evaluate $\hat{A}$ at a random place to determine its multilinearity. The test procedure is as follows. Given an input table $\hat{A}$, query it in a poly number of places using a poly number of random bits. Then we have

$$\begin{array}{l} if\ Pr[testaccepts] > \frac{1}{2} \Rightarrow \exists!\ multilinear\ polynomial\ P \ \mid Pr_{x\in F^n}[\hat{A}\left(x\right) \neq P\left(x\right)] < \frac{1}{n^k} \\ if\ \hat{A}\ is\ multilinear\ \Rightarrow\ Pr[accepts] = 1 \end{array}$$

**Example** of a multilinearity test in one variable. Given $A : F \rightarrow F$ on which to test for multilinearity.

For $i = 1$ to $n^k$:
  Query $A(0)$, $A(1)$, $A(r)$ where r is randomly chosen from F.
  Verify that it looks linear here. That is, test if $A(r) = r \cdot (A(1) - A(0)) + A(0)$.
  Reject if it ever fails.
If A always passes the test, then A is close to linear. Accept.

We will describe in more detail how this test is used in the next lecture.