Lecture 12

Probablistically Checkable Proofs

May 13, 2004
Lecturer: Paul Beame
Notes: Chris Re

12.1 Probablisitically Checkable Proofs Overview

We know that $\mathsf{IP} = \mathsf{PSPACE}$. This means there is an interactive protocol between a prover P and a verifier V such that

$$\begin{aligned} \Psi \in \text{TQBF} &\rightarrow (P,V) \text{ accepts } \Psi \text{ with probability } 1 \\ \Psi \notin \text{TQBF} &\rightarrow (P,V) \text{ accepts } \Psi \text{ with Probability } \leq \frac{1}{2}. \end{aligned}$$

Suppose that we wanted to express the entire strategy of the prover in this protocol for all possible interactions with a verifier. This strategy would consist of a rooted tree of the possible interactions, where each node of the tree corresponds to the state of the system between rounds of communication and the edges are labelled by the values sent during the rounds. At each round, the verifier sends a polynomial number of random bits (in the size N of the input Ψ) so each node just before a verifier move has fan-out $2^{N^{O(1)}}$. Each node just before a prover move has fan-out 1 which lists the prover's (best) response to the previous communcation. The entire tree has depth $N^{O(1)}$. Because of the depth and fan-out of the tree, it has $2^{N^{O(1)}}$ nodes overall.

Thus, the IP protocol corresponds to a table of size $2^{N^{O(1)}}$ of which the protocol accesses/queries only $N^{O(1)}$ bits determined by the choice of $N^{O(1)}$ random bits. PCP is a generalization of these proof techniques along the two mentioned axes: the number of random bits and the number of queries allowed.

12.2 PCP definition

Think of a proof as a large table and the verification has access to only a small number of places in the table. This is the main idea of probabilistically checkable proofs.

Definition 12.1. $L \in \mathsf{PCP}(r(n), q(n)) \Leftrightarrow \exists$ a polynomial time oracle TM $V^?$ with access to $O(r(n))$ random bits and that makes $O(q(n))$ queries to its oracle such that

$$\begin{aligned} x \in L &\rightarrow \exists\, \Pi \text{ such that } \Pr_r[V^{\Pi}(x) \text{ accepts}] = 1 \\ x \notin L &\rightarrow \forall\, \Pi.\ \Pr_r[V^{\Pi}(x) \text{ accepts}] \leq \frac{1}{2} \end{aligned}$$

where Π denotes an oracle which we think of as a proof for L. Π is viewed as a table listing all the values on which it can be queried.

Upper bound on $|\Pi|$ How big does the proof table need to be? Think of a large table indexed by the queries and the random strings, this case implies it is at most $O(q(n))2^{O(r(n))}$. It follows that

Lemma 12.1. $\mathsf{PCP}(r(n), q(n)) \subseteq \mathsf{NTIME}(2^{O(r(n))}q(n))$.

A more general formulation There are still two constants in our formulation 1 and $\frac{1}{2}$. We can generalize PCP further by introducing the notions of the *soundness* and *completeness* of the proof system.

Definition 12.2. $L \in PCP_{c,s}(r(n), q(n)) \Leftrightarrow \exists$ a polynomial time oracle TM $V^?$ with access to $O(r(n))$ random bits and that makes $O(q(n))$ queries to its oracle such that

$$x \in L \rightarrow \exists\, \Pi \text{ such that } \Pr_r[V^{\Pi}(x) \text{ accepts}] \geq c \qquad \text{(Completeness)}$$

$$x \notin L \rightarrow \forall\, \Pi.\ \Pr_r[V^{\Pi}(x) \text{ accepts}] \leq s \qquad \text{(Soundness)}$$

This formulation is important for some hardness of approximation results. If the parameters are not specified we assume the original formulation.

Definition 12.3. $\mathsf{PCP}(\mathsf{poly}, \mathsf{poly}) = \bigcup_k \mathsf{PCP}(n^k, n^k)$

Remark. The definition of $\mathsf{PCP}(\mathsf{poly}, \mathsf{poly})$ was originally motivated by a different way of extending the idea of IP. This idea was to allow multiple all-powerful instead of just one prover. With such Multiprover Interactive Proof systems the multiple provers can be used with the restriction that they may not collude with each other. The set of languages proved in polynomial time in such settings was/is known as MIP. Later it was shown that the languages in MIP are precisely those in $\mathsf{PCP}(\mathsf{poly}, \mathsf{poly})$ and the oracle characterization was used but results are still sometimes stated as results about MIP.

12.2.1 Results about PCP

Immediately from the motivating discussion for the definition of the PCP classes above we have $\mathsf{IP} \subseteq \mathsf{PCP}(\mathsf{poly}, \mathsf{poly})$ and thus the following lemma.

Lemma 12.2. $\mathsf{PSPACE} \subseteq \mathsf{PCP}(\mathsf{poly}, \mathsf{poly})$.

As a corollary to Lemma 12.1 we have

Lemma 12.3. $\mathsf{PCP}(\mathsf{poly}, \mathsf{poly}) \subseteq \mathsf{NEXP}$ *and for any polynomial* $q(n)$ *we have* $\mathsf{PCP}(\log n, q(n)) \subseteq \mathsf{NP}$.

The main results about PCP are the following which show strong converses of the above simple inclusions.

Theorem 12.4 (Babai-Fortnow-Lund). $\mathsf{PCP}(\mathsf{poly}, \mathsf{poly}) = \mathsf{NEXP}$.

The following result is so strong it is known as *the* PCP Theorem. It was the culimination of a sequence improvements of the above result by Babai, Levin, Fortnow, and Szegedy, by Feige, Goldwasser, Lovasz, Safra, and Szegedy and by Arora and Safra.

Theorem 12.5 (Arora-Lund-Motwani-Szegedy-Sudan). $\mathsf{NP} = \mathsf{PCP}(\log \mathsf{n}, 1)$.

It is interesting to note that the $O(1)$ in the number of queries and can actually be reduced to 3 in the strongest versions of the PCP Theorem. This is important for many of the results about approximation problems. For example in approximating CLIQUE this stronger version is used to show that it is hard to approximate CLIQUE with even an $n^{1-o(1)}$ factor.

Note that the PCP Theorem can be seens as a strict strengthening of the result of Babai, Fortnow, and Lund since it yields the following corollary.

Corollary 12.6. $\mathsf{PCP}(\mathsf{poly}, 1) = \mathsf{NEXP}$.

Over the next several lectures we will go over the proofs of these theorems. To begin with we need a convenient characterization for NEXP.

12.2.2 A Complete Problem for NEXP

3SAT worked nicely as a complete problem for NP, so it is natural to look for an analog of 3SAT for NEXP. In fact, the Cook-Levin tableau argument will again be our basis for showing completeness.

The analogous problem will be an implicitly defined version of 3SAT. First we will define a problem ORACLE-3SAT that we will directly show to be NEXP-complete. ORACLE-3SAT will be 3SAT defined on exponential size formulas in 3CNF defined on 2^n variables and 2^m clauses.

Definition 12.4. An *oracle truth assignment* is a function $A : \{0,1\}^n \to \{0,1\}$ where we interpret $A(v) = 1 \Leftrightarrow x_v = 1$.

Definition 12.5. An *oracle 3CNF* is a map $C : \{0,1\}^m \to \{0,1\}^{3n+3}$ that specifies, given the index $w \in \{0,1\}^m$ of a clause, the 3 variables in that clause and the three signs for those variables. For convenience, the output $C(w)$ be represented by a tuple $(v_1, v_2, v_3, s_1, s_2, s_3)$ where $v_1, v_2, v_3 \in \{0,1\}^n$, $s_1, s_2, s_3 \in \{0,1\}$ and the clause represented is $x_{v_1}^{s_1} \vee x_{v_2}^{s_2} \vee x_{v_3}^{s_3}$ where x^0 denotes x and x^1 denotes $\neg x$.

Definition 12.6. An oracle 3CNF C is *satisfiable* if and only if there exists an oracle truth assignment A such that for all $w \in \{0,1\}^m$ there exists an $i \in \{1,2,3\}$ such that if $C(w) = (v_1, v_2, v_3, s_1, s_2, s_3)$ then $A(v_i) = s_i$.

Definition 12.7. We will represent oracle 3CNF formulas using multi-output combinational circuits C computing functions $C : \{0,1\}^m \to \{0,1\}^{3n+3}$. Therefore we define

$$\text{ORACLE-3SAT} = \{\langle C \rangle \mid C \text{ is a Boolean circuit representing a satisfiable oracle 3CNF}\}.$$

Lemma 12.7. ORACLE-3SAT *is* NEXP*-Complete*

Proof. Given a circuit C let m be the number of inputs to C and $3n + 3$ be the number of outputs of C. Given C, a NEXP machine can clearly guess and write down a truth assignment S for all 2^n choices of $v \in \{0,1\}^n$ and then verify that for all 2^m values of w, clause $C(w)$ is satisfied by A. Therefore ORACLE-3SAT $\in$ NEXP.

Let L be an arbitrary language in NEXP. Now consider the Cook-Levin tableau for L and how this converts first to a CIRCUIT-SAT problem and then to a 3SAT problem. For some polynomial p, The tableau has width $2^{p(|x|)}$ and height $2^{p(|x|)}$. The first $|x|$ entries in the first row depend on x; the remainder of

the first row are based on generic nondeterministic guesses y and in the entire rest of the tableau the tableau is very generic with each entry based on the 3 entries immediately above it in the tableau. Except for the first entries in this table the complexity of each local window depends only on the complexity of the Turing machine for L which is constant size and the only difference is the dependence of the first $|x|$ entries on x.

Now consider the 3CNF formula that results from the reduction of the CIRCUIT-SAT formula which simulates this tableau. Given the indices (i, j) in binary of a cell in the tableau it is easy to describe in polynomial time what the connections of the pieces of the circuit are that simulates this tableau and therefore what the pieces of the 3CNF formula are that will be produced as the 3CNF formula in the Cook-Levin proof. Thus we have a polynomial-time algorithm C' that, based on the index of a clause, will produce a 3-clause that is in an (exponential-size) 3CNF formula φ such that $x \in L$ if and only if φ is satisfiable. Since C' is a polynomial-time algorithm there is a polynomial size circuit C that simulates C'; moreover it is very easy to produce C given the input x and the description of the Turing machine for L. The reduction maps x to $\langle C \rangle$. Clearly $x \in L$ if and only if $\langle C \rangle \in$ ORACLE-3SAT. □

ORACLE-3SAT is slightly inconvenient to use for our purposes so we will use a variant of the idea that includes as a single Boolean function both the output of the circuit and the verification of the 3-CNF clause that it outputs. Suppose that C is a boolean circuit with g gates. Consider a function B which takes as input: the original input w to C, a triple of 3 variables v_1, v_2, v_3 output by C and purported values of each of the gates of C and 3 binary values a_1, a_2, a_3 and states that *if* on input w, the variables appearing in the output clause of C are correctly represented in the input description for B and the values of the gates of C are correctly represented in the input description to B *then* the signs of the clause output by C on input w will evaluate to true if variable $x_{v_1} = a_1$, $x_{v_2} = a_2$ and $x_{v_3} = a_3$. Because we have included the values of the internal gates of C we can easily express B as a Boolean formula based on C.

More formally: $B : \{0,1\}^{m+g+3n+3} \to \{0,1\}$ is defined as follows where $|w| = m$, $|z| = g$, $|v_i| = n$, and $|a_i| = 1$.

$$B(w, z, v_1, v_2, v_3, a_1, a_2, a_3)$$
$$= \bigvee_{s_1,s_2,s_3} (C(w) = (v_1, v_2, v_3, s_1, s_2, s_3) \text{ and } z \text{ represents values of the gates of } C) \to \exists i.\ (s_i = a_i)$$

The fact that B can represented as a Boolean formula simply mirrors the usual argument converting CIRCUIT-SAT to SAT.

Notice that the definition of B implies that for an oracle assignment A,

$$A \text{ satisfies } C \Leftrightarrow \forall w, z, v_1, v_2, v_3\ B(w, z, v_1, v_2, v_3, A(v_1), A(v_2), A(v_3)).$$

Definition 12.8. A Boolean formula B in $h + 3n + 3$ variables is a *satisfiable implicit 3CNF formula* iff there is an oracle truth assignment $A : \{0,1\}^n \to \{0,1\}$ such that $\forall w' \in \{0,1\}^h$, $\forall v_1, v_2, v_3 \in \{0,1\}^n$, $B(w', v_1, v_2, v_3, A(v_1), A(v_2), A(v_3))$ is true.

Definition 12.9. IMPLICIT-3SAT $= \{\langle B \rangle \mid B \text{ is a satisfiable implicit 3CNF formula } \}$.

Clearly the conversion from $\langle C \rangle$ to $\langle B \rangle$ as defined above is polynomial time so we have that ORACLE-3SAT is polynomial time reducible to IMPLICIT-3SAT and thus:

Theorem 12.8. IMPLICIT-3SAT *is* NEXP*-Complete.*

Now that we have a complete problem for NEXP, we need to show how to convince ourselves in the required time bounds to achieve our goal of showing that NEXP = PCP(poly, poly).

12.2.3 Verification procedure for an oracle truth assignment

1st idea The definition of the satisfiability of the implicit 3CNF B can easily be seen to be computation that can be done in coNP^A since there are only universal quantifiers other than the oracle calls and the evaluation of B. Since $\mathsf{coNP}^A \subseteq PSPACE^A$ we could try to modify the IP=PSPACE to use an oracle.

This IP procotol relied on our ability to arithmetize formulas, like quantified versions of B that give values over $\{0,1\}$, to polynomials over a field $\mathbb{F}$ of moderate size. It then relied on the ability to query such functions on randomly chosen field elements $r \in \mathbb{F}$. In trying to apply the protocol here there is no problem with arithmetizing B. However, so far, we only have an oracle A that gives Boolean outputs given an input string $v \in \{0,1\}^n$; we would need to be able to evaluate A on elements of $\mathbb{F}^n$ instead.

Fortunately, we can do this by using a *multilinear* extension of A.

Definition 12.10. A function in n variables is multilinear if and only if it can be expressed as a multivariate polynomial which has degree at most 1 in each variable.

Lemma 12.9. *For any $A : \{0,1\}^n \to \mathbb{F}$ there is a (unique) multilinear polynomial that extends A. That is there exists a multilinear $\widehat{A} : \mathbb{F}^n \to \mathbb{F}$ such that $\widehat{A}(v) = A(v)$ for $v \in \{0,1\}^n$.*

Proof. Let $\widehat{A}(x) = \sum_{v\in\{0,1\}^n} A(v) \prod x_i \prod(1-x_i)$. This is the desired multilinear polynomial. Uniqueness is left as an exercise. □

So we have two parts for the proof table so far: A table of the multilinear oracle $\widehat{A}$ and the full interaction tree for the $\mathsf{IP}^{\widehat{A}}$ protocol for the $\mathsf{coNP}^{\widehat{A}}$ problem of verifying that $\widehat{A}$ satisfies B.

However, this is not enough. The prover might not produce a correct table of $\widehat{A}$! Thus, in addition to $\widehat{A}$, the proof table must include part that allows the verifier to check with some certainty that $\widehat{A}$ really is a multilinear extension of a truth assignment.

Actually, because the prover can only check a small part of the table there will be no way to convince the verifier that the table $\widehat{A}$ really is multilinear. However, as we will see in the next lecture, we just need it to be close to such an extension which is something the verifier will be able check.