CIS 511: Theory of Computation **Apr 4, 2017**

Lecture 20: Interactive Proof

Professor Sampath Kannan *Zach Schutzman*

NB: *These notes are from CIS511 at Penn. The course followed Michael Sipser's Introduction to the Theory of Computation (3ed) text.*

Interactive Proof

NP is the class of languages for which an infinitely powerful prover can convince a polynomial time verifier of string membership by sending one message (the certificate). This is a non-interactive case.

We need to show that such proofs are both **sound** and **complete**. That is, if $x \notin L$, no message from the prover will convince the verifier, and for all $x \in L$, there is some message from the prover which will convince the verifier.

We'll denote P the prover, V the verifier, and $\tilde{P}$ a prover which may be adversarial. Completeness means one good P exists. Soundness means no $\tilde{P}$ can falsely convince V of membership.

What if we allow for interaction? Are there languages for which there are **interactive proofs** that are not in NP? Well, if the prover is all-powerful, because the verifier is deterministic, the prover can simulate the interaction, and can produce as a proof a transcript of the potential interaction. Therefore, interaction alone only lets us prove things in NP.

The problem of Graph Non-isomorphism L_{GNI} asks whether two input graphs G, H are not isomorphic , where two graphs are isomorphic if there exists a bijection f from $V(G)$ to $V(H)$ such that $(u, v) \in E(G)$ if and only if $(f(u), f(v)) \in E(H)$. Graph Isomorphism is in NP (the certificate is the bijection f). L_{GNI} is not known to be in NP.

We do have an interactive proof for L_{GNI}.

Proof:

Suppose we have two graphs G, H and we want to try to prove $G \not\simeq H$. V will pick one of G, H at random, call it K. V then permutes the adjacency matrix of K to form K' which is, by definition, isomorphic to K. V then sends K' to P, and P must figure out whether $K' \simeq G$ or $K' \simeq H$.

Suppose $G \not\simeq H$. An honest prover will try to find an isomorphism between K' and G and K' and H, and will find isomorphism for exactly one of them. If G, H are actually isomorphic, then the prover will have no idea whether K' was derived from G or H, and it is equally likely to have been generated by V from either. In this case, P will have to guess and will get it wrong half of the time. Therefore, a prover cannot convince a verifier that two truly isomorphic graphs are not isomorphic, so we have soundness and completeness.

■

Definition 20.1 *The class **interactive proof** IP is the class of languages L such that there exists a prover P and a probabilistic polynomial time verifier V and a communication protocol such that P can convince V that some $x \in L$ with probability at least $\frac{2}{3}$. For any $x \notin L$ and any prover P', V is convinced $x \in L$ with probability at most $\frac{1}{3}$.*

Claim 20.2 *As it turns out, $IP = PSPACE$.*

Proof:

First, we will show $IP \subseteq PSPACE$. Let's think about the prover as a tree, where it chooses one of 2^l messages of length l and the verifier does the same. A leaf of the tree corresponds to the verifier accepting or rejecting, with probability 0 or 1. At a prover node, we assign it a probability equal to the maximum across its children, because the prover wants to convince the verifier. At an internal verifier node, we assign it the average probability of its children. We just need to answer if the probability assigned to the root is more or less than $\frac{2}{3}$. If greater, $x \in L$, else $x \notin L$. Since the tree has polynomial breadth and depth, we can use depth-first traversal and only store polynomially many answers at a time, therefore, this is in $PSPACE$.

Now, to see $PSPACE \subseteq IP$, we will start by showing $\#SAT(\phi)$, the number of satisfying assignments of ϕ, is in IP. We want to show that P can convince V that ϕ has exactly k satisfying assignments (we'll assume ϕ is in 3-CNF).

Let $f_i(a_1, a_2 \dots a_i)$ be the number of satisfying assignments of ϕ restricted to $x_1 = a_1, x_2 = a_2 \dots$. Here is a proof that's too long, but it will be useful conceptually. Suppose the prover sends $f_0()$, and V checks that $f_0() = k$, then $f_1(0), f_1(1)$. The sum $f_1(0) + f_1(1)$ should equal $f_0()$, and V checks that this is true. We continue this procedure sending all partial assignments, and the verifier checks appropriate sums ($f_2(11) + f_2(10) = f_1(1)$). At the final round, we check all full assignments, and the verifier knows what values to expect in the last round. If the prover tries to lie it must cascade through the tree structure and the verifier can catch it in the last round.

The problem, of course, is that we send 2^m things in the last step. We can embed this structure in a richer algebraic space by arithmetizing our boolean formula. Given a boolean formula ψ, we can call its arithmetization Ψ. If the formula has one variable, its arithmetization is the single variable. If the formula is the AND of two formulas, the arithmetization is the product, NOTs become $(1 - \Psi)$, and ORs become $(1 - \Psi_1)(1 - \Psi_2)$. We can see that $\{0, 1\}$ assignments $\vec{a}$ of ψ agree with $\Psi(\vec{a})$. Once we arithmetize our formula $\phi \to \Phi$.

■