**CS 710: Complexity Theory** 4/20/2010

# Lecture 25: Interactive Proofs

Instructor: Dieter van Melkebeek Scribe: Christopher Hopman, Seeun William Umboh and Tom Watson

Last time we introduced the notion of interactive proof systems which models computation as a result of interaction between two parties. We defined public/private coin protocols, and showed that the graph non-isomorphism problem admits an interactive proof using private coins, as well as one using public coins.

Today we will continue our discussion of interactive proof systems. We will study the power of bounded round interactive proof systems and of general interactive proof systems.

## 1 Interactive Proof Systems

**Definition 1** (Interactive Proof Systems)**.** *An interactive proof system for $L$ is a* protocol *$(P, V)$, where $P$ is called the* prover*, and $V$ is called the* verifier*. $P$ is an all-powerful(*i.e.*, no restrictions on its computational resources), randomized Turing machine, and $V$ is a randomized Turing machine running in time polynomial in the input length. In a protocol $(P, V)$, we have:*

1. *$P, V$ have read-only access to the input $x$.*
2. *$P, V$ have read-only access to separate random-bit tapes.*
3. *$P$ can write messages on tape $T_{P\to V}$ which $V$ has read access to.*
4. *$V$ can write messages to $P$ on tape $T_{V\to P}$ which $P$ has read access to.*
5. *$x \in L$ iff $V$ accepts.*
6. *At any one time, exactly one of $V$ and $P$ is active, and the protocol defines how they take turns being active.*

*The protocol $(P, V)$ also has to satisfy the following conditions:*

- *completeness(easy to prove membership of members):*

$$x \in L \implies \Pr[(V \leftrightarrow P) \text{ accepts } x] \geq c$$

- *soundness(hard to prove membership of non-members):*

$$x \notin L \implies (\forall P')\Pr[(V \leftrightarrow P') \text{ accepts } x] \leq s$$

*where the probabilities are over all coin flips of $P$ and $V$; $(V \leftrightarrow P)$ refers to an interaction between the $P$ and $V$; $c, s$ are the* completeness *and* soundness *parameters of the interactive proof system, respectively. $c$ is normally set to 2/3 and $s$ to 1/3.*

For the soundness condition, we choose to quantify over all provers since the critical issue is that the verifier itself needs to be sufficiently robust even against "malicious" provers that do not follow the protocol. Note that we can also apply the amplification technique to boost the probability of deciding correctly, and the verifier would still be poly-time. For completeness, $c = 1$ means *perfect completeness* and this means that no true statement is rejected, and is something we strive for.

We make the following observations:

1. Randomness for $P$ is not essential. For each turn it takes, since it is all-powerful, it can figure out the coin flips that will maximize the probability of the verifier accepting, and perform the computation corresponding to those coin flips.

2. On the other hand, the randomness of $V$ is essential, as otherwise, we get only NP. If $V$ is deterministic, then it accepts/rejects with probability 1. Then, using the above fact that we can assume $P$ is deterministic, and that $c > 0$, we get that $(V \leftrightarrow P)$ always accepts members of $L$. Since $V$ can only read and write a polynomial number of symbols, the length of the transcript of the interaction $(V \leftrightarrow P)$ is polynomial in the length of $x$, and is a polynomial-length certificate of $x$'s membership. In particular, the sequence of responses of $P$ convinces $V$ of $x$'s membership.

3. If we assume perfect soundness, we get NP as well. This is because $x \in L$ iff there exists a sequence of random "questions" by the verifier and a sequence of "answers" by $P$ that convinces $V$ of $x$'s membership. The converse holds by the assumption of perfect soundness. The other direction has a similar proof as above.

4. Without interaction, since the verifier does not receive any information from the prover, we get BPP.

5. By requiring the verifier's random bit tape to be separate from the prover's, the proofs we defined are actually *private coin* interactive proofs. *Public coin* interactive proofs are where the verifier tosses coins and reveals them to the prover after it's turn is up.

The corresponding complexity class is IP $= \{L | L \textit{ has an interactive proof system}\}$.

**Theorem 1.** *For any interactive proof system for a language $L$ with $r(n)$ rounds, there exists an equivalent interactive proof system with:*

1. *Perfect completeness using $r(n)$ rounds.*

2. *Perfect completeness and public coins using $r(n) + 2$ rounds.*

We will next look at the complexity of these proof systems measured by the number of rounds of interaction.

## 2 Bounded-round interactive proof systems

These public coin systems are often referred to as Arthur-Merlin games. If there are no rounds, then Arthur just decides by himself. This can be denoted by A and is equal to BPP. Just Merlin, M, is NP, where no randomness is involved in the verification. These protocols become more interesting as we add more rounds, denoted for example, MA, AM, AMA, AMAM...

We start with the formal definitions of AM and MA. In addition to requiring public coins, we limit the number of rounds of interaction between the verifier and prover.

**Definition 2.** *AM is a public coin system in which Arthur moves first. He flips some coins to determine what to ask Merlin. Arthur then runs a deterministic predicate on the coins, Merlin's reply and the input.*

*Formally, a language $L$ is in AM if for some $V \in \mathrm{P}$:*

$$x \in L \implies \Pr_{|z|=|x|^c}[(\exists y \in \Sigma^{|x|^c})V(x,y,z)] \geq 2/3$$

$$x \notin L \implies \Pr_{|z|=|x|^c}[(\exists y \in \Sigma^{|x|^c})V(x,y,z)] \leq 1/3$$

**Definition 3.** *MA is a public coin system in which Merlin moves first, and then Arthur flips some coins and runs a deterministic predicate on the coins, Merlin's message and the input.*

*Formally, a language $L$ is in MA if:*

$$x \in L \implies (\exists y \in \Sigma^{|x|^c}) \Pr_z[V(x,y,z)] \geq 2/3$$

$$x \notin L \implies (\forall y \in \Sigma^{|x|^c}) \Pr_z[V(x,y,z)] \leq 1/3$$

We can extend AM and MA in a similar way that $\Sigma_2^p$ extends NP: AMA, MAM, .... This gives a hierarchy of classes based on the number of rounds allowed, and whether Arthur or Merlin goes first.

However, unlike the polynomial hierarchy, this hierarchy collapses. It may seem surprising since Merlin behaves like an existential quantifier, and Arthur behaves like a universal quantifier with restricted power. We now prove the key ingredient for the collapse:

**Theorem 2.** MA $\subseteq$ AM

One of the main ideas behind the proof is that MA and AM are somewhat similar to a randomized NP machine. For example, in MA, the prover provides a proof and the verifier verifies it probabilistically instead of deterministically like in NP.

*Proof.* For $L \in$ MA, we have:

$$x \in L \implies (\exists y \in \Sigma^{|x|^c}) \Pr_{|z|=|x|^c}[V(x,y,z)=1] \geq 2/3$$

$$x \notin L \implies (\forall y \in \Sigma^{|x|^c}) \Pr_{|z|=|x|^c}[V(x,y,z)=1] \leq 1/3$$

where $V$ is the underlying deterministic predicate that the verifier evaluates.

So, to get MA $\subseteq$ AM, we would like to swap the choice of the purported proof and the choice of the random string. For the case where $x \in L$, an examination of the condition on the RHS reveals this swap is not a problem. But for the case when $x \notin L$, the swap of quantifiers does not work out. We solve this problem by using amplification and the union bound to our advantage. In particular, for a fixed $y$, we amplify $V$ such that when $x \notin L$, $Pr_z[V(x,y,z)] \leq \frac{1}{3} \cdot 2^{-|x|^c}$. Given this, we can now swap the quantifiers, to get

$$x \in L \implies \Pr_{|z|=n^d}[(\exists y \in \Sigma^{|x|^c})V(x,y,z)=1] \geq 2/3$$

$$x \notin L \implies \Pr_{|z|=n^d}[(\exists y \in \Sigma^{|x|^c})V(x,y,z)=1] \leq \frac{1}{3} \cdot 2^{-|x|^c} \cdot 2^{|x|^c} = \frac{1}{3}$$

where $n^d$ is some polynomial of $|x|^c$.

This gives us an AM protocol and we are done. □

Equipped with this theorem, intuitively we can switch the order of the first 2 turns in MAM to get AMM and then merge the 2 Merlin turns into 1, and thus MAM = AMM = AM. Similarly, AMA = AAM = AM. We omit the formal proof.

In fact, we can prove something stronger for bounded-round protocols. By the above process, we can halve the number of rounds needed:

**Theorem 3.** *For any poly-time $r(n)$ with $r(n) \geq 2$,* $\mathrm{AM}(2r(n)) = \mathrm{AM}(r(n))$.

*Proof.* Omitted. □

**Corollary 1.** $\mathrm{AM}(k) = \mathrm{AM}$ *for all constant $k$.*

Thus, this corollary and Theorem 1 give us that private coin systems with a constant number of rounds are equivalent to AM. Note that this does not imply that IP collapses to AM because we would need a super-constant number of applications of the above. Intuitively, this is due to a polynomial factor blow-up in the verification procedure when we apply the above. So, when we apply it only a constant number of times, the verification still runs in polynomial time, but not if we require super-constant number of applications.

## 2.1 Relationship with other complexity classes

We have the following facts:

1. $\mathrm{MA} \subseteq \Sigma_2^p$
2. $\mathrm{AM} \subseteq \Pi_2^p$
3. $\mathrm{AM} \subseteq \mathrm{NP/poly}$
4. Under reasonable derandomization hypotheses, AM = NP.

(1) follows from the fact that Arthur's computation is a BPP computation on Merlin's proof and the input, and we can simulate BPP using approximate counting with a $\Sigma_2^p$ predicate $P$ on $(x,y)$. Then, since we can simulate Merlin's computation with an existential quantifier, we get $(\exists y \in \Sigma^{|x|^c})[P(x,y)=1]$. Merging the existential quantifier with the one in $P$, we get a $\Sigma_2^p$ predicate. (2) follows similarly, but we use the fact that $\mathrm{BPP} \subseteq \Pi_2^p$ instead.

(3) follows using a proof similar to the one for $\mathrm{BPP} \subseteq \mathrm{P/poly}$. We use amplification to obtain a random string that will work for all the inputs of a fixed length, and then use that string as an advice. We leave the details as an exercise.

The intuition behind (4) is that AM is somewhat similar to NP. The proof is by relativizing the proof that E has $2^{\Omega(n)}$-size circuits implies that P = BPP. (4) also suggests that $GI$ is not NP-complete, since otherwise $GNI$ would be coNP-complete and since $GNI \in \mathrm{AM} = \mathrm{NP}$ under reasonable hypotheses, that would imply that NP = coNP. In fact, we get a collapse of $PH$ if GI is NP-complete even without a derandomization assumption.

**Theorem 4.** *If* coNP $\subseteq AM$*, then* $\Sigma_2^p = \Pi_2^p$.

*Proof.* Let $L$ be a $\Sigma_2^p$ language. Then, we have that

$$x \in L \iff (\exists y \in \Sigma^{|x|^c})(\forall z \in \Sigma^{|x|^c})P(x, y, z)$$

for some constant $c$ and a polynomial-time predicate $P$. Since coNP $\subseteq$ AM, we can reduce the $(\forall z \in \Sigma^{|x|^c})P(x, y, z)$ coNP predicate as an AM protocol. Since we can express the existential quantifier as a Merlin phase, we have that $\Sigma_2^p \subseteq MAM$. But because all constant-round interactive proof systems collapse to AM, $\Sigma_2^p \subseteq$ AM $\subseteq \Pi_2^p$. Thus, $\Sigma_2^p = \Pi_2^p$. □

**Corollary 2.** *Graph Isomorphism is not NP-complete, unless* $\Sigma_2^p = \Pi_2^p$.

# 3 coNP is in IP

One of the main reasons we have for studying the class IP is to see if we can obtain short proofs using interactive proof systems in settings where short standard proofs are unlikely, such as tautologies. While it is unlikely that coNP $\subseteq$ AM, since the above proof would then show that $\Sigma_2^p = \Pi_2^p$, we do have that coNP $\subseteq$ IP. Note that this is one of the few non-trivial results in complexity theory that does not relativize.

**Theorem 5.** *The language* $L = \{(\varphi, \#SAT(\varphi)) | \varphi \in \{0,1\}^*\}$ *is in* IP*, where* $\#SAT(\varphi)$ *denotes the number of assignments satisfying a SAT clause* $\varphi$.

Note that $L$ is the decision variant of the counting problem for $SAT$.

*Proof(first half).* For $L$, we are given $(\varphi, k)$ and we would like to verify using an interactive proof system that $\#SAT(\varphi) = k$. One of the key ingredients is arithmetization, which we looked at briefly in the last lecture.

Assume that $\varphi$ has $n$ variables and $m$ clauses, and is 3-CNF. So, we would like to transform $\varphi(x_1, \ldots, x_n)$ into a polynomial over the integers, $\tilde{\varphi}(x_1, \ldots, x_n)$ such that $\tilde{\varphi}$ agrees with $\varphi$ on $\{0,1\}^n$ and $\tilde{\varphi}$ has degree at most $m$ in every variable and easy to evaluate as well.

Now we show how to transform a clause $C$. For example, if $C = (x_1 \vee \bar{x_2} \vee x_3)$, then after transformation we have $\tilde{C} = 1 - (1 - x_1)x_2(1 - x_3)$. Then, to construct $\tilde{\varphi}$, we use $\tilde{\varphi} = \prod_{j=1}^m \tilde{C_j}$. Since each variable occurs at most once in each clause, the construction gives a polynomial with degree at most $m$ in every variable, and also $\tilde{C_j} = 1$ if $C_j$ is satisfied and $\varphi = 1$ if and only if every $\tilde{C_j} = 1$. Thus, $\tilde{\varphi}$ agrees with $\varphi$ on $\{0,1\}^n$. Also, $\tilde{\varphi}$ is easy to evaluate since we just have to go through all the transformed clauses $\tilde{C_j}$ and make sure they all evaluate to 1.

Then, to count the number of satisfying assignments, we use $\sum_{x_1=0}^1 \cdots \sum_{x_n=0}^1 \tilde{\varphi}(x_1, \ldots, x_n)$. □

## 3.1 Sumcheck Protocol

Our goal is to design a verifier to decide whether the number of satisfying assignments to $\phi$ is equal to some number $k_0$. This is equivalent to checking the following identity:

$$\sum_{x_1=0}^{1} \sum_{x_2=0}^{1} \cdots \sum_{x_n=0}^{1} \tilde{\phi}(x_1, x_2, \ldots, x_n) = k_0. \tag{1}$$

We don't have time to evaluate $\tilde{\phi}$ for all $2^n$ settings of $x_1, \ldots, x_n$, so we enlist the help of the prover. The sumcheck protocol is a method for doing this that is secure against cheating provers.

The key idea is to consider the univariate polynomial $g(x)$ defined by

$$g(x) = \sum_{x_2=0}^{1} \cdots \sum_{x_n=0}^{1} \tilde{\phi}(x, x_2, \ldots, x_n).$$

This is the same as the expression in Equation (1) except that we don't sum over $x_1$. Since $x_1$ has degree at most $m$ in $\tilde{\phi}$, $g(x)$ has degree at most $m$. If we knew $g(x)$, then we would be able to check Equation (1) by checking $g(0) + g(1) = k_0$, using the observation that

$$g(0) + g(1) = \sum_{x_1=0}^{1} \sum_{x_2=0}^{1} \cdots \sum_{x_n=0}^{1} \tilde{\phi}(x_1, x_2, \ldots, x_n).$$

We don't have $g(x)$, but we can ask the prover to send it to us. We know that $g(x)$ has at most $m+1$ coefficients, which is manageable, but it's conceivable that these coefficients are prohibitively huge numbers. This turns out not to be an issue, but we postpone the discussion of this.

Now the prover sends us some polynomial $g'(x)$, which may not be equal to $g(x)$ if the prover is cheating. We check that $g'(x)$ has degree at most $m$ and that $g'(0) + g'(1) = k_0$ and reject if either does not hold. Now if Equation (1) is true, then the prover can just send us the correct polynomial $g'(x) = g(x)$ and everything will be fine. If Equation (1) is not true, then if the prover wants to have a prayer of getting us to accept, then it's forced to send an incorrect polynomial $g'(x) \neq g(x)$. Since the two polynomials agree in at most $m$ places, we can pick a random number $\xi_1$ from a reasonably small set $I$ and with high probability, $g'(\xi_1) \neq g(\xi_1)$. If we had $g(x)$ then we could just evaluate $g(\xi_1)$ and $g'(\xi_1)$ and with high probability catch the cheating prover. Of course, the whole point is that we don't have $g(x)$, but we have now reduced our original problem, verifying Equation (1), to a simpler instance of the same problem, namely verifying

$$\sum_{x_2=0}^{1} \cdots \sum_{x_n=0}^{1} \tilde{\phi}(\xi_1, x_2, \ldots, x_n) = k_1 \tag{2}$$

where $k_1 = g'(\xi_1)$. By sending us $g'(x)$, the prover has implicitly claimed that Equation (2) holds. It had no way of knowing what number $\xi_1$ we would pick, though, which gives us an edge. If Equation (1) holds, then Equation (2) holds when the prover just sends us the correct polynomial. If Equation (1) does not hold, then with high probability Equation (2) does not hold, regardless of the prover's behavior. In either case we are back to where we started, except that now there are only $n-1$ variables being summed over. We iterate this process until our task is reduced to the problem of verifying

$$\tilde{\phi}(\xi_1, \xi_2, \ldots, \xi_n) = k_n \tag{3}$$

for some numbers $\xi_1, \ldots, \xi_n \in I$ and $k_n$. Now we use the other property required of $\tilde{\phi}$, namely that is can be evaluated efficiently, to explicitly check that Equation (3) holds. We accept if it holds and reject otherwise. This is the only time in the protocol when we are willing to accept.

**Theorem 6.** *The sumcheck protocol is a public-coin interactive proof system that verifies Equation (1) with perfect completeness and polynomially small soundness, provided $\tilde{\phi}$ has degree at most $m$ in every variable and can be evaluated in polynomial time.*

*Proof.* That the protocol is public-coin is clear — in each step the verifier uses its randomness only to pick the number $\xi_i$, and then it sends $\xi_i$ to the prover. That the protocol has perfect completeness is also straightforward to see — if Equation (1) holds then the prover can just send us the correct polynomial $g'(x) = g(x)$ in every phase and the consistency check $g'(0) + g'(1) = k_i$ will always pass and all the implicit claims of the form of Equation (2) will be true, so the final check of Equation (3) will also pass, leading to acceptance with probability 1.

Now we argue the soundness. Suppose Equation (1) does not hold, but nevertheless we accept. In the last phase, the claim we're checking, Equation (3), is true (otherwise we would reject), but in the first phase, the claim we're checking, Equation (1), is false. Thus there must be some phase where the current claim transitions from being false to being true. In this phase, the prover sends us some $g'(x) \neq g(x)$ (since otherwise we would reject after seeing that the consistency check $g'(0) + g'(1) = k_i$ fails). For the claim generated by this phase to be true, it must be the case that $g'(\xi_{i+1}) = g(\xi_{i+1})$, which happens with probability at most $m/|I|$. The probability that we accept is at most the probability that this happens in some phase, which by a union bound is at most $mn/|I|$. We can choose $|I|$ to be polynomially large to make the soundness polynomially small.

The only thing left to verify is that the protocol runs in polynomial time. The final check can be done in polynomial time by hypothesis, so the only issue is that the coefficients of the $g(x)$ polynomials might become too large. One can argue that the coefficients do not become too large. We can avoid that analysis by letting $I$ be the integers mod $p$ for a small prime $p$, and letting all computations take place in the field of integers mod $p$. Now we need to worry about the soundness, though; it could be the case that the originally claimed value $k_0$ is congruent to the true value mod $p$, and in that case the prover can make the verifier always accept. However, the difference between $k_0$ and the true value cannot have too many prime factors (certainly fewer than $n$). The density of primes is great enough that picking $p$ at random from the set of primes of polynomial magnitude (and hence logarithmic bit length) affects the soundness by only a polynomially small amount. Thus the overall soundness is still polynomially small, and computations mod $p$ can certainly be done efficiently. Selecting $p$ at random is not a problem since the verifier can enumerate all primes in the desired range by brute force. □

## 4 Next Lecture

For next time we will continue our discussion of interactive proof systems. We will use the so-called *sumcheck protocol* We will use the so-called sumcheck protocol to prove that $\mathrm{P}^{\#\mathrm{P}} \subseteq \mathrm{IP}$. Building on this technique, we strengthen the result and show that IP = PSPACE, i.e., polynomial space exactly captures the power of interactive proof systems. Then we introduce the notion of multiple prover interactive proof systems and relate them to probabilistically checkable proofs, in which the interaction between prover and verifier is eliminated but the proof is allowed to be very long. There are some extremely profound and involved results in these areas, which we briefly discuss.