Notes on Complexity Theory: Fall 2005 Last updated: November, 2005

Lecture 11

Jonathan Katz

1 The Power of $\mathcal{IP}$

In the last lecture we showed a (surprising!) interactive proof for graph non-isomorphism. This begs the question: how powerful is $\mathcal{IP}$? We have mentioned already that $\mathcal{IP} \subseteq \mathsf{PSPACE}$; in this lecture we show equality by proving $\mathsf{PSPACE} \subseteq \mathcal{IP}$.

1.1 co$\mathcal{NP} \subseteq \mathcal{IP}$

We begin with a "warm-up" and show that co$\mathcal{NP} \subseteq \mathcal{IP}$. Recall from last time that co$\mathcal{NP}$ is unlikely to have a constant-round interactive proof system (since this would imply[1] that the polynomial hierarchy collapses); indeed, the proof system we show here will have a *linear* number of rounds.

The basic idea is the following: we *arithmetize* a $3CNF$ formula ϕ to obtain a formula which evaluates to 0 iff ϕ has no satisfying assignments. We then show how to give an interactive proof demonstrating that the formula indeed evaluates to 0. To arithmetize ϕ, we proceed as follows: identify 0 with "false" and positive integers with "true." The literal x_i becomes variable x_i, and the literal $\bar{x}_i$ becomes $(1 - x_i)$. We replace "$\wedge$" by multiplication, and "$\vee$" by addition. Let Φ denote the polynomial that results from this arithmetization (this is an n-variate polynomial in the variables $x_1, \ldots, x_n$, of degree at most the number of clauses in ϕ).

Now consider what happens when the $\{x_i\}$ are assigned boolean values: all literals take the value 1 if they evaluate to "true," and 0 if they evaluate to "false." Any clause (which is a disjunction of literals) takes a positive value iff at least one of its literals is true; thus, a clause takes a positive value iff it evaluates to "true." Finally, note that Φ itself (which is a conjunction of clauses) takes on a positive value iff all of its constituent clauses are positive. We can summarize this as: $\Phi(x_1, \ldots, x_n) > 0$ if $\phi(x_1, \ldots, x_n) = \text{TRUE}$, and $\Phi(x_1, \ldots, x_n) = 0$ if $\phi(x_1, \ldots, x_n) = \text{FALSE}$. Summing over all possible (boolean) settings to the variables, we see that

$$\phi \notin SAT \Leftrightarrow \sum_{x_1 \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} \Phi(x_1, \ldots, x_n) = 0.$$

We have said already that if ϕ has m clauses, then Φ has degree (at most) m (where the [total] degree of a polynomial is the maximum degree on any of its monomials, and the degree of a monomial is the sum of the degrees of its constituent variables). Furthermore, the sum above is at most $2^n \cdot 3^m$. So, if we work modulo a prime $q > 2^n \cdot 3^m$ the above is equivalent to:

$$\phi \notin SAT \Leftrightarrow \sum_{x_1 \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} \Phi(x_1, \ldots, x_n) = 0 \bmod q.$$

[1] In more detail: a constant-round proof system for co$\mathcal{NP}$ would imply a constant-round *public-coin* proof system for co$\mathcal{NP}$, which would in turn imply co$\mathcal{NP} \subseteq$ **AM**. We showed last time that the latter implies the collapse of PH.

Working modulo a prime (rather than over the integers) confers two advantages: it keeps the numbers from getting too large (since all numbers will be reduced modulo q; note that $|q| = \log q$ is polynomial) and it means that we are working over the finite field $\mathbb{F}_q$ (which simplifies the analysis).

We have now reduced the question of whether ϕ is unsatisfiable to the question of proving that a particular expression sums to 0(!) Hopefully, this already hints at the power of arithmetization: it transforms questions of logic (e.g., satisfiability) into questions of abstract mathematics (polynomials, group theory, algebraic geometry, ...) and we can then use all the powerful tools of mathematics to attack our problem. Luckily, for the present proof the only "deep" mathematical result we need to rely on is that a non-zero polynomial of degree m over a field has at most m roots. An easy corollary is that two different polynomials each of degree (at most) m can agree on at most m points.

We now show the interactive proof that ϕ is not satisfiable.

- Both prover and verifier have ϕ. They both compute the polynomial Φ. The prover wants to show that $0 = \sum_{x_1 \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} \Phi(x_1, \ldots, x_n)$ Note that although the verifier can evaluate Φ at any point, the verifier *cannot* evaluate the summation since it involves a sum over *exponentially-many* terms.
- The prover sends a prime q such that $q > 2^n \cdot 3^m$. The verifier checks the primality of q.
- The verifier initializes $v_0 = 0$.
- The following is repeated as i runs from 1 to n:
  - The prover sends a polynomial $\hat{P}_i$ (in one variable) of degree at most m.
  - The verifier checks that $\hat{P}_i$ has degree at most m and that $\hat{P}_i(0) + \hat{P}_i(1) = v_{i-1}$ (addition is done in $\mathbb{F}_q$). If not, the verifier rejects. Otherwise, the verifier chooses a random $r_i \in \mathbb{F}_q$, computes $v_i = \hat{P}_i(r_i)$, and sends r_i to the prover.
- The verifier accepts if $\Phi(r_1, \ldots, r_n) = v_n$ and rejects otherwise. (Note that even though we originally only "cared" about the values Φ takes when its inputs are *boolean*, there is nothing stopping us from evaluating Φ at any points in our field.)

We now show completeness and soundness.

Completeness: For every $1 \leq i \leq n$ (and given the verifier's choices of $r_1, \ldots, r_{i-1}$) define the polynomial:

$$P_i(x_i) \stackrel{\text{def}}{=} \sum_{x_{i+1} \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} \Phi(r_1, \ldots, r_{i-1}, x_i, x_{i+1}, \ldots, x_n).$$

We claim that if ϕ is unsatisfiable and the prover always sends $\hat{P}_i = P_i$, then the verifier will always accept. To see this, first note that P_i always has degree at most m so the verifier never rejects for that reason. In the first iteration ($i = 1$), we have

$$P_1(0) + P_1(1) = \sum_{x_1 \in \{0,1\}} \left(\sum_{x_2 \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} \Phi(x_1, \ldots, x_n) \right) = 0 = v_0,$$

since ϕ is unsatisfiable. For $i > 1$ we have:

$$\begin{aligned} P_i(0) + P_i(1) &= \sum_{x_i \in \{0,1\}} \sum_{x_{i+1} \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} \Phi(r_1, \ldots, r_{i-1}, x_i, \ldots, x_n) \\ &= P_{i-1}(r_{i-1}) = v_{i-1}. \end{aligned}$$

Finally, $v_n \stackrel{\text{def}}{=} P_n(r_n) = \Phi(r_1, \ldots, r_n)$. So, the verifier will accept.

Soundness: To prove soundness we will rely on the following lemma:

Lemma 1 *For an execution of the protocol above, define the values Q_i (for $0 \leq i \leq n$) via:*

$$\begin{aligned} Q_i &\stackrel{\text{def}}{=} \sum_{x_{i+1} \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} \Phi(r_1, \ldots, r_i, x_{i+1}, \ldots, x_n) \\ &= P_i(r_i) \end{aligned}$$

(where P_i is the same polynomial defined earlier). For $0 \leq i < n$, if $v_i \neq Q_i$ the probability that the verifier does not reject but $v_{i+1} = Q_{i+1}$ is at most $\frac{m}{q}$ (even for a cheating, all-powerful prover).

Proof Assume $v_i \neq Q_i$ and the prover sends a degree-m polynomial $\hat{P}_{i+1}(x_{i+1})$. There are two cases: either $\hat{P}_{i+1} = P_{i+1}$ (as polynomials) or not. If they are equal, the verifier will immediately reject since

$$\begin{aligned} \hat{P}_{i+1}(0) + \hat{P}_{i+1}(1) &= P_{i+1}(0) + P_{i+1}(1) \\ &= \sum_{x_{i+1} \in \{0,1\}} \sum_{x_{i+2} \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} \Phi(r_1, \ldots, r_i, x_{i+1}, \ldots, x_n) \\ &\stackrel{\text{def}}{=} Q_i \neq v_i. \end{aligned}$$

If $\hat{P}_{i+1} \neq P_{i+1}$, we know that these polynomials can agree on at most m points. Therefore, except with probability $\frac{m}{q}$ the verifier chooses a point r_{i+1} for which

$$v_{i+1} \stackrel{\text{def}}{=} \hat{P}_{i+1}(r_{i+1}) \neq P_{i+1}(r_{i+1}) \stackrel{\text{def}}{=} Q_{i+1}.$$

This completes the proof. ∎

Soundness now follows easily: if ϕ is satisfiable (and so the prover is trying to cheat) then $0 = v_0 \neq Q_0$. Using the lemma repeatedly and applying a union bound, we see that $v_n \neq Q_n$ except with probability at most $\frac{nm}{q}$. But $Q_n \stackrel{\text{def}}{=} \Phi(r_1, \ldots, r_n)$, and so $v_n \neq Q_n$ implies that the verifier rejects at the end of the protocol.

1.2 $\#\mathcal{P} \subseteq \mathcal{IP}$

It is straightforward to extend the protocol of the previous section to show that $\#\mathcal{P} \subseteq \mathcal{IP}$. The only substantive difference is in the way we arithmetize ϕ: now we want our arithmetization Φ to evaluate to *exactly* 1 on any satisfying assignment to ϕ, and to 0 otherwise. For literals we proceed as before, transforming x_i to x_i and $\bar{x}_i$ to $1 - x_i$. For clauses, we do something different: given clause $a \vee b \vee c$ (where a, b, c are literals), we construct the polynomial:

$$1 - (1 - \hat{a})(1 - \hat{b})(1 - \hat{c}),$$

where $\hat{a}$ represents the arithmetization of a, etc. Note that if all of a, b, c are set to "false" (i.e., $\hat{a} = \hat{b} = \hat{c} = 0$) the above evaluates to 0 (i.e., false), while if any of a, b, c are "true" the above evaluates to 1 (i.e., true). Finally, the entire formula ϕ (which is the "and" of a bunch of clauses) is simply the product of the arithmetization of its clauses. This gives the arithmetization Φ with the desired properties. Note that the degree of Φ is now (at most) $3m$.

Using the above arithmetization, a formula ϕ has exactly K satisfying assignments iff:

$$K = \sum_{x_1 \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} \Phi(x_1, \ldots, x_n).$$

Using the exact same protocol as before, except with $q > 2^n$ (since the above summation can now be at most 2^n) and setting $v_0 = K$ (the claimed number of satisfying assignments), gives an interactive proof for $\#SAT$ with soundness error $3mn/q$.

2 $\mathcal{IP} = \mathsf{PSPACE}$

Before we show that $\mathsf{PSPACE} \subseteq \mathcal{IP}$, we need to find a PSPACE-complete problem to work with. It should not be surprising that the problem we pick will be a variant of satisfiability. However, the proof that the problem is PSPACE-complete is not entirely trivial.

2.1 A PSPACE-Complete Problem

The problem we will consider is *totally quantified boolean formulae* (denoted $QSAT$) which consists of quantified formulae of the form:

$$\forall x_1 \exists x_2 \cdots Q_n x_n \quad \phi(x_1, \ldots, x_n),$$

where $Q_n = \forall$ if n is odd, and $Q_n = \exists$ if n is even, and an expression of the above form is in $QSAT$ if it is true: that is, if it is the case that "for all $x_1 \in \{0,1\}$, there exists an $x_2 \in \{0,1\}, \ldots$ such that $\phi(x_1, \ldots, x_n)$ evaluates to true." (Note that it does not make a difference whether the "$\forall$" quantifier or the "$\exists$" quantifier comes first, since we can always add an extra "dummy" variable.) While this looks very similar to the polynomial hierarchy we have seen earlier, the key difference here is that the number of alternations is *unbounded* (but polynomial in the input size); in contrast, the k^{th} level of the polynomial hierarchy allows only a *constant* (i.e., k) number of alternations.

It is not too difficult to see that $QSAT \in \mathsf{PSPACE}$ (basically, in polynomial space we can try all settings of all the variables and keep track of whether the quantified expression is true). It is a bit trickier to show that $QSAT$ is PSPACE-complete. Given a PSPACE machine M deciding some language L, we will reduce the computation of M on some input x to an instance of $QSAT$. Since M uses space n^k and runs in time 2^{n^k} for some constant k, we may encode configurations of M on some initial input x (with $|x| = n$) using $O(n^k)$ bits. We will construct a sequence of formulae $\psi_i(a, b)$ which evaluate to true iff there is a path (i.e., sequence of steps of M) of length at most 2^i. Letting a_0 denote the initial state of M (on input x), and letting z denote the (unique) accepting state of M, deciding whether M accepts x is equivalent to deciding whether $\psi_{n^k}(a_0, z)$ is true.

We need now to construct the ψ. For ψ_0 this is easy: to evaluate $\psi_0(a, b)$ we simply test whether $a = b$ or whether configuration b follows from configuration a in one step. Now, given ψ_i we construct ψ_{i+1}. The "obvious" way of doing this would be as in the proof of Savitch's theorem: namely, to define $\psi_{i+1}(a, b)$ as:

$$\exists c : \quad \psi_i(a, c) \wedge \psi_i(c, b).$$

Doing so, however, would result in a formula ψ_{n^k} of *exponential* size! (To see this, note that expanding everything [and collapsing the existential quantifiers] we would get an expression like:

$$\exists c_1, \ldots, c_{2^{n^k}-1} : \psi_0(a_0, c_1) \wedge \cdots \wedge \psi_0(c_{2^{n^k}-1}, z),$$

which is exponential in $|x|$. We may also note that we have not made any use of universal quantifiers in the above.) Instead, we proceed a bit more cleverly and "encode" $\psi_i(a,c) \wedge \psi_i(c,b)$ in a single expression. In particular, define $\psi_{i+1}(a,b)$ as:

$$\exists c \forall x,y: \quad \Big(\big((x,y)=(a,c)\big) \vee \big((x,y)=(c,b)\big) \Big) \Rightarrow \psi_i(x,y).$$

(Note that the logical implication $a \Rightarrow b$ is equivalent to the expression $b \vee \bar{a}$.) The key point is that whereas previously the size of ψ_{i+1} was double the size of ψ_i, here the size of ψ_{i+1} is only a constant (additive) factor larger than ψ_i.

A few technical points remain to be addressed: first, the quantifiers of ψ_i are "buried" inside the expression for ψ_{i+1} (whereas $QSAT$ requires that these quantifiers come up front); however, it is easy to see that the quantifiers of ψ_i can be migrated to the front without changing the truth of the overall expression. A second problem is that ψ_{n^k} is not in CNF form. However, since the inner expression of ψ_{n^k} can be evaluated by a polynomial-size circuit, we can easily convert this inner expression to the required form (as in the proof that circuit-SAT is $\mathcal{NP}$-complete). See [2] for further details.

2.2 An Interactive Proof for $QSAT$

We now show an interactive proof system for $QSAT$; coupled with the fact that $QSAT$ is PSPACE-complete, this shows that $\mathsf{PSPACE} \subseteq \mathcal{IP}$. We use the same arithmetization we used for $\#\mathcal{P}$; recall, for a boolean formula ϕ this results in a degree-$3m$ polynomial Φ such that $\Phi(x_1,\ldots,x_n)=1$ if $\phi(x_1,\ldots,x_n)$ is true, and $\Phi(x_1,\ldots,x_n)=0$ if $\phi(x_1,\ldots,x_n)$ is false. But now we need to worry about quantifiers. The arithmetization of an expression of the form $\forall x_n \phi(x_1,\ldots,x_n)$ will be:

$$\prod_{x_n} \Phi(x_1,\ldots,x_n) \stackrel{\text{def}}{=} \Phi(x_1,\ldots,x_{n-1},0) \cdot \Phi(x_1,\ldots,x_{n-1},1)$$

(where Φ is the arithmetization of ϕ). Note that if we fix values for $x_1,\ldots,x_{n-1}$ then the above evaluates to 1 if the expression $\forall x_n \phi(x_1,\ldots,x_n)$ is true, and evaluates to 0 if this expression is false. The arithmetization of an expression of the form $\exists x_n \phi(x_1,\ldots,x_n)$ will be

$$\coprod_{x_n} \Phi(x_1,\ldots,x_n) \stackrel{\text{def}}{=} 1 - \big(1-\Phi(x_1,\ldots,x_{n-1},0)\big) \cdot \big(1-\Phi(x_1,\ldots,x_{n-1},1)\big)$$

(where, again, Φ is the arithmetization of ϕ). Note again that if we fix values for $x_1,\ldots,x_{n-1}$ then the above evaluates to 1 if the expression $\exists x_n \phi(x_1,\ldots,x_n)$ is true, and evaluates to 0 if this expression is false. Proceeding in this way, a quantified boolean formula $\exists x_1 \forall x_2 \cdots \forall x_n \phi(x_1,\ldots,x_n)$ is true iff

$$1 = \coprod_{x_1} \prod_{x_2} \cdots \prod_{x_n} \Phi(x_1,\ldots,x_n) \tag{1}$$

(where Φ is our original arithmetization of ϕ).

A natural idea is to "plug in" Eq. (1) to the existing protocols we have seen for co$\mathcal{NP}$ and $\#\mathcal{P}$, and to have the prover help the verifier evaluate the above expression by "stripping off" operators one-by-one. However, things are not quite this easy: the problem is that the degrees of the intermediate polynomials are too large. For example, following the ideas used in the previous protocols we would want the honest prover to send the polynomial:

$$P(x_1) \stackrel{\text{def}}{=} \prod_{x_2} \cdots \prod_{x_n} \Phi(x_1,\ldots,x_n)$$

(and then the verifier would check that $\coprod_{x_1} P(x_1) = 1$, etc.). But the degree of x_1 doubles each time a $\prod$ or $\coprod$ operator is applied, meaning that the degree of $P(x_1)$ would be exponential in n. Besides whatever effect this will have on soundness, this is even a problem for completeness since a polynomially-bounded verifier simply cannot process an exponentially-large polynomial (i.e., with exponentially-many non-zero coefficients).

To address the above issue, we use a simple[2] trick. Note that we only care about evaluating Eq. (1) for $\{x_i\}$ taking on *boolean* values. But for any $k > 0$ we have $x_i^k = x_i$ when $x_i \in \{0, 1\}$. So we can in fact reduce the degree of every variable in any intermediate polynomial to (at most) 1. (For example, the polynomial $x_1^5 x_2^4 + x_1^6 + x_1^7 x_2$ would become $2x_1x_2 + x_1$.) Let R_{x_i} denote a *reduce operator* denoting this "degree reduction" operation applied to variable x_i. Thus, the prover needs to convince the verifier that:

$$1 = \coprod_{x_1} R_{x_1} \prod_{x_2} R_{x_1} R_{x_2} \coprod_{x_3} \cdots R_{x_1} \cdots R_{x_{n-1}} \prod_{x_n} R_{x_1} \cdots R_{x_n} \Phi(x_1, \ldots, x_n).$$

As in the previous protocols, we will actually evaluate the above modulo some prime q. Note that the above expression evaluates to either 0 or 1 (as long as Φ is derived by arithmetizing a boolean expression), and so soundness is only affected by choice of q in that the soundness will end up being proportional to $1/q$ (see below).

Now we can apply the basic idea from the previous protocols to construct a new protocol in which, in each round, the prover helps the verifier "strip" one operator from the above expression. Denote the above expression abstractly by:

$$F_\phi = \mathcal{O}_1 \mathcal{O}_2 \cdots \mathcal{O}_\ell \; \Phi(x_1, \ldots, x_n) \bmod q\,,$$

where $\ell = \sum_{i=1}^n (i+1)$ and each O_j is one of $\prod_{x_i}$, $\coprod_{x_i}$, or R_{x_i} (for some i). Describing the protocol in a bit more detail, at every round k the verifier holds some value v_k and the prover wants to convince the verifier that

$$v_k = \mathcal{O}_{k+1} \cdots \mathcal{O}_\ell \; \Phi_k \bmod q,$$

where Φ_k is some polynomial. At the end of the round the verifier will compute some v_{k+1} and will then want to be convinced that

$$v_{k+1} = \mathcal{O}_{k+2} \cdots \mathcal{O}_\ell \; \Phi_{k+1} \bmod q,$$

for some Φ_{k+1}. We explain how this is done below. At the beginning of the protocol we start with $v_0 = 1$ and $\Phi_0 = \Phi$ (so that the prover wants to convince the verifier that the given quantified formula is true); at the end of the protocol the verifier will be able to compute Φ_ℓ itself and check whether this is equal to v_ℓ.

It only remains to describe each of the individual rounds. There are three cases corresponding to the three types of operators (we omit the " mod q" from our expressions from now on, for simplicity):

Case 1: $\mathcal{O}_{k+1} = \prod_{x_i}$ (for some i). Here, the prover wants to convince the verifier that

$$v_k = \prod_{x_i} R_{x_1} \cdots \coprod_{x_{i+1}} \cdots \prod_{x_n} R_{x_1} \cdots R_{x_n} \Phi(r_1, \ldots, r_{i-1}, x_i, \ldots, x_n). \tag{2}$$

(*Important technical note*: when we write an expression like the above, we really mean:

$$\left(\prod_{x_i} R_{x_1} \cdots \coprod_{x_{i+1}} \cdots \prod_{x_n} R_{x_1} \cdots R_{x_n} \Phi(x_1, \ldots, x_{i-1}, x_i, \ldots, x_n) \right) [r_1, \ldots, r_{i-1}].$$

[2]Of course, it seems simple in retrospect…

That is: first the expression is computed symbolically, and then the resulting expression is evaluated by setting $x_1 = r_1, \ldots, x_{i-1} = r_{i-1}$.) This is done in the following way:

- The prover sends a degree-1 polynomial $\hat{P}(x_i)$.
- The verifier checks that $v_k = \prod_{x_i} \hat{P}(x_i)$. If not, reject. Otherwise, choose random r_i, set $v_{k+1} = \hat{P}(r_i)$, and enter the next round with the prover trying to convince the verifier that:

$$v_{k+1} = R_{x_1} \cdots \coprod_{x_{i+1}} \cdots \prod_{x_n} R_{x_1} \cdots R_{x_n} \Phi(r_1, \ldots, r_{i-1}, r_i, x_{i+1}, \ldots, x_n). \tag{3}$$

We show completeness and an analogue of Lemma 1 (which will be used to prove soundness) for the above step. For completeness, assuming Eq. (2) is true the prover can send

$$\hat{P}(x_i) = P(x_i) \stackrel{\text{def}}{=} R_{x_1} \cdots \coprod_{x_{i+1}} \cdots \prod_{x_n} R_{x_1} \cdots R_{x_n} \Phi(r_1, \ldots, r_{i-1}, x_i, \ldots, x_n)$$

and then the verifier will not reject and also Eq. (3) will hold for any choice of r_i. As for soundness, if Eq. (2) does *not* hold then the prover must send $\hat{P}(x_i) \neq P(x_i)$ (or else the verifier rejects right away); but then Eq. (3) will not hold except with probability $1/q$.

Case 2: $\mathcal{O}_{k+1} = \coprod_{x_i}$ (for some i). This case and its analysis are similar to the above and are therefore omitted.

Case 3: $\mathcal{O}_{k+1} = R_{x_i}$ (for some i). Here, the prover wants to convince the verifier that

$$v_k = R_{x_i} \cdots R_{x_j} \mathcal{O}_{x_{j+1}} \cdots \prod_{x_n} R_{x_1} \cdots R_{x_n} \Phi(r_1, \ldots, r_j, x_{j+1}, \ldots, x_n), \tag{4}$$

where $j \geq i$ and $\mathcal{O} \in \{\prod, \coprod\}$. This case is a little different from anything we have seen before. This round proceeds as follows:

- The prover sends a polynomial $\hat{P}(x_i)$ of appropriate degree (see below).
- The verifier checks that $\left(R_{x_i} \hat{P}(x_i)\right)[r_i] = v_k$. If not, reject. Otherwise, choose a **new** random r_i, set $v_{k+1} = \hat{P}(r_i)$, and enter the next round with the prover trying to convince the verifier that:

$$v_{k+1} = \mathcal{O}_{k+2} \cdots \prod_{x_n} R_{x_1} \cdots R_{x_n} \Phi(r_1, \ldots, r_{i-1}, r_i, r_{i+1}, \ldots, r_j, x_{j+1}, \ldots, x_n) \tag{5}$$

  (and this falls into one of the three possible cases).

Completeness is again easy to see: assuming Eq. (4) is true, the prover can simply send:

$$\hat{P}(x_i) = P(x_i) \stackrel{\text{def}}{=} \mathcal{O}_{k+2} \cdots \prod_{x_n} R_{x_1} \cdots R_{x_n} \Phi(r_1, \ldots, r_{i-1}, x_i, r_{i+1}, \ldots, r_j, x_{j+1}, \ldots, x_n)$$

and then the verifier will not reject and also Eq. (5) will hold for any choice of (new) r_i. As for soundness, if Eq. (4) does *not* hold then the prover must send $\hat{P}(x_i) \neq P(x_i)$; but then Eq. (5) will not hold except with probability d/q where d is the degree of $\hat{P}$.

This brings us to the last point, which is what the degree of $\hat{P}$ should be. We can see that except for the final n reduce operators, the degree of the intermediate polynomial can be at most 2; for the final n operators, the degree can be up to $3m$. We may now compute the soundness error of the entire protocol: we obtain a $1/q$ error for each of the n operators of type $\prod$ or $\coprod$, a $3m/q$ error for each of the final n reduce operators, and a $2/q$ error for all other reduce operators. Applying a union bound, we see that the soundness error is:

$$\frac{n}{q} + \frac{3mn}{q} + \frac{2}{q} \cdot \sum_{i=1}^{n-1} i = \frac{3mn + n^2}{q}.$$

Bibliographic Notes

The result that $\mathsf{PSPACE} \subseteq \mathcal{IP}$ is due to Shamir [3], building on [1]. The "simplified" proof given here is from [4].

References

[1] C. Lund, L. Fortnow, H.J. Karloff, and N. Nisan. Algebraic Methods for Interactive Proof Systems. *J. ACM* 39(4): 859–868 (1992). The result originally appeared in FOCS '90.

[2] C.H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1995.

[3] A. Shamir. $\mathcal{IP} = \mathsf{PSPACE}$. *J. ACM* 39(4): 869–877 (1992). The result originally appeared in FOCS '90.

[4] A. Shen. $\mathcal{IP} = \mathsf{PSPACE}$: Simplified Proof. *J. ACM* 39(4): 878–880 (1992).