1 Interactive proofs with polynomial number of rounds

Last time we saw that Graph Non-Isomorphism problem is in AM. We also noted that if any coNP-complete problem were in AM, then the polytime hierarchy would collapse PH = Σ₂. Thus we’re unlikely to find an AM protocol for every problem in coNP.

We also know that any AM-style protocol with a constant number of rounds can be converted into an AM protocol with just 2 rounds. So to get some protocol for coNP, one seems to need a superconstant number of rounds.

We will see that a polynomial number of rounds yields an interactive proof for every problem in PSPACE. (Curiously, for the number k of rounds between a constant and polynomial, we don’t have any good characterization of the class AM[k]).

Before tackling PSPACE, we show an IP protocol for #SAT: the problem of counting the number of satisfying assignments of a given propositional formula.

2 Interactive Protocols for #SAT

Define #3SAT to be a function that maps a 3-CNF φ(x₁, ..., xₙ) into a number s that is equal to the number of satisfying assignments of φ. It’s not hard to see that #3SAT remains #P-complete!

We’ll show the following

Theorem 1 (Lund, Fortnow, Karloff, Nisan). #SAT is in IP = IP[poly].

The key ingredient in the proof of both results mentioned above is arithmetization of propositional formulas. Namely, the conversion of a given 3-CNF φ(x₁, ..., xₙ) into a an arithmetic formula computing a multivariate polynomial f(x₁, ..., xₙ) satisfying the following property. For any truth assignment a = (a₁, ..., aₙ) (which we view as a 0-1 vector), if φ(a) is True, then f(a) = 1; and if φ(a) is False, then f(a) = 0.

Such an arithmetization of a formula φ is carried out inductively. A variable x becomes the function x, and the literal ¯x becomes the function 1 − x. A formula φ₁ ∧ φ₂ becomes the function f₁ * f₂, where fᵢ is an arithmetization of φᵢ, i = 1, 2. Finally, a formula φ₁ ∨ φ₂
becomes the function \( 1 - (1 - f_1)(1 - f_2) \), where \( f_i \) is the arithmetization of \( \phi_i \), \( i = 1, 2 \). (To make sense of the last rule, recall that by de Morgan’s rule, \( \phi \lor \psi \equiv \neg(\neg\phi \land \neg\psi) \).

**Example:** Let \( \phi(x_1, x_2, x_3, x_4) = (x_1 \lor x_2 \lor x_3) \land (\bar{x}_1 \lor x_3 \lor x_4) \). Then the corresponding arithmetic formula will be \( f(x_1, x_2, x_3, x_4) = [1 - (1 - x_1)(1 - x_2)x_3][1 - x_1x_3(1 - x_4)] \), which is a polynomial of degree 6.

Note that the degree of the constructed polynomial is 3 times the number of clauses in \( \phi \). This is not a coincidence. It is easy to see that any 3-CNF \( \phi \) with \( m \) clauses is transformed by the arithmetization procedure described above into a polynomial of total degree at most \( 3m \).

**Lemma 2.** Let \( \phi(x_1, \ldots, x_n) \) be a 3-CNF with \( m \) clauses, and let \( f(x_1, \ldots, x_n) \) is the arithmetic formula obtained by arithmetizing \( \phi \). Then (1) the total degree of \( f \) is at most \( 3m \), (2) on any 0-1 vector \( a \), \( f(a) \in \{0, 1\} \), and (3) on any 0-1 vector \( a \), \( f(a) = 1 \) iff \( \phi(a) \) is True.

**Proof.** Exercise. (Hint: use induction.)

For a 3-CNF \( \phi \) and the corresponding arithmetization \( f \), let’s define \( \#\phi = (\# \text{ satisfying assignments of } \phi) \), and \( \#f = \sum_{x_1=0}^{1} \sum_{x_2=0}^{1} \cdots \sum_{x_n=0}^{1} f(x_1, x_2, \ldots, x_n) \). Then, by the lemma above, we get that \( \#\phi = \#f \). So, proving that \( \#\phi = s \) is equivalent to proving that \( \#f = s \).

The latter is what our IP protocol is going to do.

Note: \( 0 \leq \#\phi(x_1, \ldots, x_n) \leq 2^n \). So, if we take a prime \( p > 2^n \), then \( \#\phi( \text{ mod } p) = \#\phi \).

It will be convenient for the Verifier to get such a prime from the Prover, and do all the checking \( \text{ mod } p \), i.e., over the finite field \( \mathbb{Z}_p \).

The Prover will claim that

\[ \sum_{x_1=0}^{1} \sum_{x_2=0}^{1} \cdots \sum_{x_n=0}^{1} f(x_1, x_2, \ldots, x_n) = s, \tag{1} \]

for some number \( 0 \leq s \leq 2^n \). The verifier’s strategy will depend on the following way of “removing” the summation signs from the above arithmetic expression.

Define

\[ f_1(z) = \sum_{x_2=0}^{1} \sum_{x_3=0}^{1} \cdots \sum_{x_n=0}^{1} f(z, x_2, x_3, \ldots, x_n). \]

Note that \( f_1(z) \) is a univariate polynomial in variable \( z \) of degree at most that of \( f \), i.e., at most \( 3m \). Clearly, equality (1) is true iff

\[ f_1(0) + f_1(1) = s. \tag{2} \]

But how can we compute \( f_1(z) \)? It is very hard. The polytime verifier will not be able to do it (unless \( \text{NP} = \text{P} \) or something even more dramatic happens). However, the Prover can compute \( f_1(z) \). So, the Verifier can just ask the Prover to send the coefficients of \( f_1(z) \). Since \( f_1 \) has degree at most \( 3m \), the Prover needs to send at most \( 3m + 1 \) coefficients over \( \mathbb{Z}_p \), which is small enough amount of information that the polytime Verifier can handle.
If the Prover is honest, then he’ll send a correct \( s \) and \( f_1 \) so that equality (2) holds. What about a dishonest Prover? Suppose that the Prover sends a wrong \( s \), i.e., equality (1) is false. Then the prover sends \( g_1(z) \) (of degree at most \( 3m \)), and claims that \( g_1(z) \equiv f_1(z) \). Suppose that the Verifier checks first that \( g_1(0) + g_1(1) = s \). If the check does not pass, the Verifier rejects. Either the polynomial \( g_1 \) sent by the Prover does not pass this check and the Verifier rejects, or \( g_1(0) + g_1(1) = s \). In the latter case, since \( s \) is a wrong number, it must be the case that \( g_1(z) \neq f_1(z) \) (because \( f_1(0) + f_1(1) \neq s \)).

So, if the cheating Prover passes the Verifier’s check, then it must be the case that \( g_1(z) \) and \( f_1(z) \) are different polynomials. Since both polynomials are of degree at most \( 3m \), if the Verifier picks a random value \( r_1 \in \mathbb{Z}_p \), then with high probability (at least \( 1 - (3m)/p \)), \( g_1(r_1) \neq f_1(r_1) \).

What is \( f_1(r) \)? Note that \( f_1(r) = \sum_{x_2=0}^{1} \cdots \sum_{x_n=0}^{1} f(r, x_2, \ldots, x_n) \). The expression on the right hand side is of the same type as the initial equality in (1), except with one less summation sign. By setting \( s_1 = g_1(r_1) \), the Verifier reduces the original question about equality (1) to the new question about the equality with fewer summations:

\[
\sum_{x_2=0}^{1} \cdots \sum_{x_n=0}^{1} f(r, x_2, \ldots, x_n) = s_1. 
\] (3)

By what we argued above, the cheating Prover will have to give a wrong polynomial \( g_1(z) \) so that, with high probability, equality (3) is wrong.

Now the Prover and the Verifier can engage in the same protocol as before, but for a smaller instance - equality (3). After \( n \) rounds of communication, the Prover has sent to the Verifier a last polynomial \( g_n(z) \) which is supposed to be equal to \( f(r_1, r_2, \ldots, r_{n-1}, z) \), for \( r_1, \ldots, r_{n-1} \) chosen by the Verifier in the preceding rounds of the protocol. The Verifier again checks if \( g_n(0) + g_n(1) = s_{n-1} \), for the value \( s_{n-1} \) determined by the Verifier in the previous round. If the check does not pass, the Verifier rejects. Otherwise, the Verifier checks if \( g_n(z) \equiv f(r_1, \ldots, r_{n-1}, z) \) by testing if the two polynomials agree on \( 3m+1 \) distinct elements of \( \mathbb{Z}_p \). If they disagree, the Verifier rejects. Otherwise, the Verifier accepts.

**Analysis** First, note that an honest Prover, by answering truthfully to all challenges, will convince the Verifier to accept with probability 1. Now, suppose that a Prover is dishonest, and gives a wrong value of \( s \) to the Verifier. Then, either the Prover does not pass one of the Verifier’s checks of the form \( g_i(0) + g_i(1) = s_{i-1} \) in round \( i, 1 \leq i \leq n \), or all such checks are passed by the Prover. In the former case, the Verifier will reject, correctly. In the latter case, if the Prover cheated in round \( i - 1 \), he’ll be forced to cheat in round \( i \), with probability at least \( 1 - (3m)/p \). So the probability that equality of type (1) holds in any of the \( n \) rounds is at most \( n(3m)/p \), exponentially small. Thus, with high probability, in the last round of the protocol, \( g_n(z) \neq f(r_1, \ldots, r_{n-1}, z) \), and this will be discovered by the Verifier. So, if the Prover cheats, then with high probability, the Verifier will reject.

**Remark 3.** Note that the Verifier in the described protocol just sends random strings to the Prover. So, the described protocol is actually of the Arthur-Merlin type.
3 Generalization to the $\mathsf{IP = PSPACE}$

Since TQBF is a complete problem for $\mathsf{PSPACE}$, it suffices to give an IP protocol for TQBF. A QBF $\phi$ can be of the form $\forall x_1 \forall x_2 \exists x_3 \ldots \phi(x_1, x_2, \ldots, x_n)$. As before, we can arithmetize the formula $\phi$, obtaining a multivariate polynomial $f(x_1, \ldots, x_n)$ that agrees with $\phi$ over any Boolean $n$-bit input. By induction, it’s easy to show that the QBF $\phi$ is True iff $\prod_{x_1=0}^{1} \prod_{x_2=0}^{1} \prod_{x_3=0}^{1} \cdots f(x_1, x_2, \ldots, x_n) > 0$. That is, we replace each $\forall x_i$ quantifier with $\prod_{x_i=0}^{1}$, and each $\exists x_j$ with $\sum_{x_j=0}^{1}$. Let’s call the resulting arithmetic expression $A$.

As in the #3SAT protocol in the previous section, we could try to remove the $\sum$’s and $\prod$’s, one by one, and doing the checks of the form $g_i(0) + g_i(1) = s_{i-1}$ or $g_i(0) * g_i(1) = s_{i-1}$, respectively.

There are several problems with this approach. First, the value of $A$ can be as big as $2^{2^n}$, since for the formula with $n$ universal quantifiers, we need to multiply together $f(a_1, \ldots, a_n)$, for all $2^n$ possible binary vectors $(a_1, \ldots, a_n)$. To deal with this problem, we can use modular arithmetic (as in the case of Polynomial Identity Testing for Arithmetic Circuits). So, we pick a random prime $p$ of about $\text{poly}(n)$ bit-size, and do all our verification mod $p$.

Another problem is the following. Suppose we “remove” the left-most $\sum x_1$ (or $\prod x_1$), and consider the univariate polynomial $f_1(z)$ equal to the original expression $A$ where $x_1$ is replaced by $z$ and the quantification over $x_1$ is dropped. This univariate polynomial in $z$ can have degree as large as $2^{n-1}$, since there may be $n-1 \prod$’s between the quantified $x_1$ and the occurrence of $x_1$ in the QBF formula $\phi$, and each such $\prod$ doubles the degree of $z$ in the polynomial $f_1(z)$. So, a polytime Verifier cannot ask the Prover for the coefficients of $f_1(z)$ — there are just too many of them!

This is a much more serious problem than the first. Upon closer study, one observes that this problem is due to the fact that a QBF may have an unbounded number of $\forall$-quantifiers between a quantifier for a variable $x$ and the occurrence of $x$ in the formula. If we could somehow transform any given QBF so that between any quantifier for $x$ and the occurrence of $x$ in the formula there is at most one universal quantifier, we would be able to argue that the degree of polynomial $f_1(z)$ is at most $2^k$(the degree of $f(x_1, \ldots, x_n)$), which is small enough for the Verifier to be able to receive the coefficients of $f_1(z)$.

It turns out that such a transformation is indeed possible! Here’s how. For every occurrence of the situation $Qx \ldots \forall y \ldots x$, we introduce a new “place-holder” variable $x'$ for $x$, and write the equivalent QBF $Qx \ldots \exists x'((x' \leftrightarrow x) \land \forall y \ldots x')$. Note that after this transformation, there is one universal quantifier between $\exists x'$ and the occurrence of $x'$ in the formula. At the same time, the number of universal quantifiers between $Qx$ and the occurrence of $x$ in the new formula is decreased by 1. So, continuing in the same way, we can convert any given QBF to a QBF of the desired form, where there is at most one universal quantifier between any $Qx$ and the occurrence of $x$ in the formula. Observe that this transformation results in a QBF which may not be in prenex form with all the quantifiers in the front of the formula, but it’s OK for our purposes — see the remark below.

So, after doing the aforementioned transformation of a given QBF, and doing all verification modulo a large enough prime, we can design an IP protocol for TQBF which is very
similar to the one for \#3SAT described earlier. This proves that $PSPACE \subseteq IP$. The other inclusion $IP \subseteq PSPACE$ is fairly straightforward: in $PSPACE$ we can compute the probability that a Verifier accepts a given input; the details are left as an exercise.

Remark 4. We can define the process of arithmetizing a given qbf inductively as follows:

- a variable $x$ becomes the polynomial $x$, and a negated variable $\bar{x}$ becomes $1 - x$ (we assume that negations are applied to variables only);
- $\phi \land \psi$ becomes $f_\phi \ast f_\psi$, where $f_\phi$ and $f_\psi$ are the arithmetic expressions corresponding to the subformulas $\phi$ and $\psi$, respectively;
- $\phi \lor \psi$ becomes $f_\phi + f_\psi$, where $f_\phi$ and $f_\psi$ are the arithmetic expressions corresponding to the subformulas $\phi$ and $\psi$, respectively (note this is different, simpler, from the way we arithmetized a cnf in the \#SAT case\(^1\));
- $\exists x \phi(x, \ldots)$ becomes $\sum_{x=0}^{1} f_\phi(x, \ldots)$, where $f_\phi$ is the arithmetic expression corresponding to the subformula $\phi()$;
- $\forall x \phi(x, \ldots)$ becomes $\prod_{x=0}^{1} f_\phi(x, \ldots)$, where $f_\phi$ is the arithmetic expression corresponding to the subformula $\phi()$.

It is easy to prove that the described arithmetization yields an arithmetic expression that is 0 if the initial qbf is false, and is greater than 0 if the initial qbf is true.

When we run an IP protocol, we are peeling off $\sum$ and $\prod$ quantifiers one at a time, similarly to the case of the IP protocol for \#SAT. After we replace a current variable with a random number, we simplify the arithmetic expression. For example, when we remove the $\exists$ in the subformula $\exists x'(x \leftrightarrow x') \land \forall y \ldots x'$, the resulting arithmetic expression simplifies to $c \ast f'$, where $c$ is the integer value of the expression $x \leftrightarrow x'$ after we plug in a random value for $x'$ (note that $x$ must have already been replaced by a random number by this point), and $f'$ is the arithmetic expression corresponding to the subformula $\forall y \ldots x'$. Suppose Merlin claims that this expression equals $s$. If $c = 0$, we accept if $s = 0$ and reject otherwise. If $c \neq 0$, we set $s' = s/c$, and inductively proceed to check that the expression $f'$ is equal to $s'$.

4 Multiple Provers

Can we get some kind of interactive proofs for classes above $PSPACE$, say $EXP$ or $NEXP$? Clearly, $IP$ is not powerful enough as $IP \subseteq PSPACE$. What if we allow multiple, say a constant (or even polynomial) number of provers?

We imagine a verifier that can speak with several provers while the provers can’t talk to each other. It turns out that just two provers are enough!

\(^1\)Observe that replacing an “OR” by “+” is good enough in our case, as this will ensure that the false formula yields the value 0, whereas a true formula yields a value greater than 0.
Let’s denote by $\text{MIP}$ the class of languages that can be decided by 2-prover protocol where a randomized polytime verifier exchanges a polynomial number of messages with two provers that don’t talk to each other.

We have the following:

**Theorem 5** (Babai, Fortnow, Lund). $\text{NEXP} = \text{MIP}$.

That is, if we allow two provers, we jump in complexity from $\text{PSPACE}$ all the way to $\text{NEXP}$.

**Remark 6.** It can be shown that rather than interacting with two provers, the verifier may be just given a full transcript listing the answers to all possible questions by the verifier. That is, the strategy followed by the two provers can be written down and given to the verifier, and the provers are no longer needed! In this equivalent setup, the verifier randomly decides what questions to ask, then looks up the answers in the book of answers he was given (using his random access to the book, meaning that he’s free to open a book on any page of his choosing), asks more questions, and so on for a polynomial number of steps, and then finally the verifier decides to either accept or reject.

The theorem about $\text{MIP} = \text{NEXP}$ above has been “scaled down” to $\text{NP}$, resulting in the famous PCP Theorem, which we’ll discuss next time.