

# CMPT 710 - Complexity Theory: Lecture 22

Valentine Kabanets

November 22, 2007

## 1 Interactive Protocols for #SAT

We'll continue with the proof of the following

**Theorem 1** (Lund, Fortnow, Karloff, Nisan). *#3SAT is in IP.*

Recall that we take a 3-CNF  $\phi$  and arithmetize it, obtaining a polynomial  $f$ . We get  $\#\phi =$  ( $\#$  satisfying assignments of  $\phi$ ) is the same as  $\#f = \sum_{x_1=0}^1 \sum_{x_2=0}^1 \cdots \sum_{x_n=0}^1 f(x_1, x_2, \dots, x_n)$ . So we need to design an IP protocol for proving that  $\#f = s$ .

Note:  $0 \leq \#\phi(x_1, \dots, x_n) \leq 2^n$ . So, if we take a prime  $p > 2^n$ , then  $\#\phi \pmod p = \#\phi$ . It will be convenient for the Verifier to get such a prime from the Prover, and do all the checking  $\pmod p$ , i.e., over the finite field  $\mathbb{Z}_p$ .

The Prover will claim that

$$\sum_{x_1=0}^1 \sum_{x_2=0}^1 \cdots \sum_{x_n=0}^1 f(x_1, x_2, \dots, x_n) = s, \quad (1)$$

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

$$f_1(z) = \sum_{x_2=0}^1 \sum_{x_3=0}^1 \cdots \sum_{x_n=0}^1 f(z, x_2, x_3, \dots, x_n).$$

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

$$f_1(0) + f_1(1) = s. \quad (2)$$

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  $g_1(z) \not\equiv f_1(z)$  (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  $f_1(r) = \sum_{x_2=0}^1 \cdots \sum_{x_n=0}^1 f(r, x_2, \dots, x_n)$ . 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:

$$\sum_{x_2=0}^1 \cdots \sum_{x_n=0}^1 f(r, x_2, \dots, x_n) = s_1. \quad (3)$$

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  $f(r_1, r_2, \dots, r_{n-1}, z)$ , for  $r_1, \dots, r_{n-1}$  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  $g_n(z) \equiv f(r_1, \dots, r_{n-1}, z)$  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,  $g_n(z) \not\equiv f(r_1, \dots, r_{n-1}, z)$ , and this will be discovered by the Verifier. So, if the Prover cheats, then with high probability, the Verifier will reject.

**Remark** 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.

## 2 Generalization to the $IP = PSPACE$

Since TQBF is a complete problem for PSPACE, it suffices to give an IP protocol for TQBF. A QBF  $\phi$  can be of the form  $\forall x_1 \forall x_2 \exists x_3 \dots \phi(x_1, x_2, \dots, x_n)$ . As before, we can arithmetize the formula  $\phi$ , obtaining a multivariate polynomial  $f(x_1, \dots, x_n)$  that agrees with  $\phi$  over any Boolean  $n$ -bit input. By induction, it's easy to show that the QBF  $\phi$  is True iff  $\prod_{x_1=0}^1 \prod_{x_2=0}^1 \sum_{x_3=0}^1 \dots f(x_1, x_2, \dots, x_n) > 0$ . 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  $f(a_1, \dots, a_n)$ , for all  $2^n$  possible binary vectors  $(a_1, \dots, a_n)$ . 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  $\pmod p$ .

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  $2 * (\text{the degree of } f(x_1, \dots, x_n))$ , 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  $Qx \dots \forall y \dots x$ , we introduce a new “place-holder” variable  $x'$  for  $x$ , and write the equivalent QBF  $Qx \dots \exists x' ((x' \leftrightarrow x) \wedge \forall y \dots x')$ . 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.

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.

### 3 PCP theorem

To talk about PCPs, we need the definition of a probabilistic polytime verifier  $V$  that is  $(r(n), q(n))$ -restricted, i.e., on input of size  $n$ ,

1.  $V$  uses at most  $O(r(n))$  random bits, and
2.  $V$  queries at most  $O(q(n))$  bits of the proof string.

We say that  $L \in \text{PCP}[r(n), q(n)]$  if there is a probabilistic polytime  $(r(n), q(n))$ -restricted verifier  $V$  such that

1. (**completeness**)  $x \in L \Rightarrow \exists \pi_x \Pr[V^{\pi_x}(x) = 1] = 1$ ,
2. (**soundness**)  $x \notin L \Rightarrow \forall \pi \Pr[V^\pi(x) = 1] \leq 1/2$ .

Here, the probabilities are taken over the random choices of the verifier, and the notation  $V^\pi$  means that the verifier  $V$  has random access to the string  $\pi$ , i.e.,  $V$  may request to see the  $i$ th bit of the string  $\pi$ , for any  $i$ .

A verifier  $V$  can be either *non-adaptive* or *adaptive*. A non-adaptive verifier, after flipping its  $O(r(n))$  random coins, decides upon  $O(q(n))$  locations in the proof string that  $V$  wants to see. In other words, a non-adaptive verifier decides upon what locations to see in advance.

On the other hand, an adaptive verifier, after flipping  $O(r(n))$  random coins, decides upon a first location of the proof string that  $V$  wants to see. Then, after seeing the contents of this location,  $V$  decides which location to see next. After seeing the contents of the second location,  $V$  decides which location to see third, and so on for  $O(q(n))$  locations.

In general, when talking about PCP verifiers, we mean adaptive verifiers. However, most verifiers we'll see will be non-adaptive.

A couple of remarks about the definition of PCP. First, note that a non-adaptive  $(r(n), q(n))$ -restricted verifier can access (over all of its random string of length  $O(r(n))$ ) at most  $2^{O(r(n))}O(q(n))$  bits of the proof string. So, if  $r(n) = \log n$ , and  $q(n) \in \text{poly}(n)$ , then such a verifier can see at most a polynomial number of bits of the proof, and so we may as well assume that the proof string is of polynomial length (since if it's longer, the verifier won't be able to see the extra part anyway).

The case of an adaptive verifier is trickier. Once  $V$  has flipped its random coins, we can think of the *strategy* of  $V$  for requesting the bits of the proof string as a binary tree of height at most  $O(q(n))$ . The root  $u$  of the tree is labeled by the first location of the proof string that  $V$  wants to see. The left child  $v$  of  $u$  is labeled by the next location in the proof that  $V$  wants to see if the contents of the first location is 0; the right child  $w$  of  $u$  is labeled by the next location in the proof that  $V$  wants to see if the contents of the first location is 1. The children of  $v$  and  $w$  are defined similarly. Since  $V$  asks at most  $O(q(n))$  queries, the height of this strategy tree is  $O(q(n))$ . Finally, the children of the last query in a given path of the strategy tree are two leaves: left (corresponding to the contents 0) and right (corresponding

to the contents 1). The leaves are labeled by Accept or Reject, according to the decision of the verifier  $V$ .

For example, if  $q(n) = 2$ , then for some random string of  $V$ , the strategy tree of  $V$  may look as follows: root  $u$  is labeled 13, the left child  $v$  is labeled 5, and the right child  $w$  is labeled 7. Both children of  $v$  are labeled Accept; and the left child of  $w$  is labeled Accept, whereas the right child of  $w$  is labeled Reject. This tree corresponds to the strategy: if proof bit 13 is 0, then see proof bit 5; otherwise, see proof bit 7. If bit 13 is 1, and bit 7 is 1, then Reject; in all other cases, Accept.

We now state the famous PCP theorem, which is one of the deepest results in theoretical computer science in the last decade.

**Theorem 2** (PCP Theorem).  $\text{NP} = \text{PCP}[\log n, 1]$ .

The PCP Theorem was proved by Arora, Safra, and Arora, Lund, Motwani, Sudan, Szegedy in two papers that in turn built upon rich body of work on interactive proof systems done earlier by many researchers. Recently, the authors of these two papers have been honored with the Gödel prize for their contribution to computer science.