CMPUT 675: Computational Complexity Theory **Winter 2019**

Lecture 13 (Feb. 26): IP = PSPACE

Lecturer: Zachary Friggstad *Scribe: Tim Put*

13.1 Easy direction: IP $\subseteq$ PSPACE

IP $\subseteq$ PSPACE because PSPACE gives us sufficient power to simulate the polynomial time verifier against all possible proof strategies, and determine if the best strategy is sufficient. Note that we do not need to actually compute what the best strategy is, merely that the best strategy is good enough to convince the verifier more than two-thirds of the time.

Suppose $L \in$ IP. Then there exists an interactive proof protocol with a prover P and a verifier V for L. To decide L in PSPACE, we will simulate the (polynomial time) verifier on all the (at most polynomial length) prover schemes. Prover schemes can be encoded as a polynomial length list of polynomial length messages, which itself is just a polynomial length bit string. Represent the collection of possible IP computations to decide the given string x, as a tree where each node of even depth represents a message from the prover, and each node of odd depth represents message from the verifier. This tree has polynomial depth, since the proofs are of polynomially many rounds. Traverse the tree in a depth first order, reusing space, and assign a 1 to each node where the verifier accepts and a 0 to each node where the verifier rejects. For any internal prover node, once all of its children have been assigned weights (on the way 'back up'), assign it the maximum of the weights of its children, since this corresponds to choosing the optimal prover message. For any internal verifier node, once all of its children have been assigned weights (on the way 'back up'), assign it the average of the weights of its children. Since the verifier randomly chooses its message, the prover can expect its average performance from this node, to be the average of its performances on the child nodes. Once the tree has been traversed, the root node will have a single weight assigned to it. If this value is greater than 2/3, accept x, otherwise reject.

13.2 Warm-up on #SAT

Definition 1 $\#\text{SAT} = \{(\phi, k) : \phi \textit{ is a CNF with exactly } k \textit{ satisfying assignments}\}$

We begin with a warm up proof that #SAT $\in$ IP, by specifying an interactive proof procedure for #SAT instances. The procedure comes in two conceptual steps: an encoding of the problem instance as a polynomial equivalence test, and a polytime polynomial equivalence proof procedure.

13.2.1 Preprocessing

Given a #SAT problem instance ϕ, we "Arithmatize" the CNF ϕ, mapping the Boolean forumla to a polynomial over $\mathbb{Z}$, giving us access to algebraic tools. We treat conjunctions as multiplication, and use de Morgan's law to transform disjunctions into conjunctions. Negation, naturally, is expressed as the only (up to extensionality) fixed-point free map on $\mathbb{Z}_2$.

- $\mathsf{Arith}(\perp) \mapsto 0$

- $\mathsf{Arith}(\top) \mapsto 1$
- $\mathsf{Arith}(\bar{x}_i) \mapsto (1 - \mathsf{Arith}(x_i))$
- $\mathsf{Arith}(c_i \wedge c_j) \mapsto \mathsf{Arith}(c_i) \cdot \mathsf{Arith}(c_i)$

e.g. $x_1 \vee \bar{x}_3 \vee \bar{x}_4 \mapsto 1 - (1 - x_1) \cdot x_3 \cdot x_4$, up to an abuse of notation on the variable names.

It is clear by inspection that a CNF is satisfied by a given assignment iff the corresponding assignment makes the arithmatized CNF equal to 1. Without loss of generality we assume that each variable appears at most once in each clause of a given #SAT instance (otherwise, if the variables appear with opposite valence, the clause reduces to $\top$, and if with the same valence, then the duplicate may be deleted). We write $\mathsf{Arith}_\phi(x_1, \ldots, x_n)$ for the polynomial resulting from the arithmatization of a CNF ϕ. Note that the degree of the polynomial is a most $d = n \cdot m$ where n is the number of variables in the CNF, and m is the number of clauses.

Then:

$$(\phi, k) \in \#\mathsf{SAT} \iff \sum_{x_1 \in \{0,1\}} \sum_{x_2 \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} \mathsf{Arith}_\phi(x_1, x_2, \ldots, x_n) = k \tag{13.1}$$

13.2.2 Protocol

Here we describe an interactive proof protocol for verifying the arithmetic expression above. Here, we will use m to denote the number of clauses, n the number of variables, and $d = n \cdot m$ to be an upper bound on the polynomial $\mathsf{Arith}_\phi(x_1, x_2, \ldots, x_n)$.

13.2.2.1 Protocol initialization

- Prover: choose a prime $p \in (d \cdot 2^n, d \cdot 2^{n+1}]$ (this is always possible, by Chebyshev's theorem).
- Verifier: check that p is prime, reject if it is not, otherwise continue (this is possible in polynomial time by [AKS04], though even a probabilistic check would suffice).

Rather than checking 13.1, the protocol will check the following equation. This equation holds exactly when 13.1 holds, since $p > d \cdot 2^n > k$ and the polynomial evaluates to 0 or 1 when all inputs are 0 or 1.

$$\sum_{x_1 \in \{0,1\}} \sum_{x_2 \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} \mathsf{Arith}_\phi(x_1, x_2, \ldots, x_n) =_p k \tag{13.2}$$

The notation $=_p$ means the quantities are equal modulo p.

However, rather than directly checking (13.2), the verifier will interact with the prover to and iteratively try to reduce the number of variables in the expression in a "probably sound".

Eventually the procedure bottom out on a polynomial equation small enough for the verifier to check directly with its own computational power.

13.2.2.2 Protocol interaction

Verifier: computes the following "expression" g in this compact form (i.e. without opening up the parenthesis in the polynomial $\mathsf{Arith}_\phi(x_1, x_2, \ldots, x_n)$ or evaluating the sums). The goal is to understand if the claim $g =_p k$

is true or false:

$$g := \sum_{x_1 \in \{0,1\}} \sum_{x_2 \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} \mathsf{Arith}_\phi(x_1, x_2, \ldots, x_n) \tag{13.3}$$

Then the following steps are repeated, each step eliminates one of the sums by directly replacing some x_i with a fixed value.

Prover: provides the following *univariate* polynomial (as a list of coefficients modulo p), as a witness for the previous claim.

$$s_1(x_1) = \sum_{x_2 \in \{0,1\}} \sum_{x_3 \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} \mathsf{Arith}_\phi(z_1, x_2, \ldots, x_n) \tag{13.4}$$

Verifier: checks that $s_1(0) + s_1(1) =_p k$, and that s_1 is of degree at most d; rejects if the check fails. The verifier can safely reject in this case: if $s_1(0) + s_1(1) \neq_p k$ then either the prover did not send the true univariate polynomial obtained by evaluating the last $n-1$ sums in g, or it did but then $(\phi, k) \notin \#\text{SAT}$.

Otherwise, the verifier samples z_1 uniformly from $\mathbb{Z}_p$ (integers modulo p), sends z_1 to the prover, and repeats the protocol except with asking if

$$s_1(z_1) = \sum_{x_2 \in \{0,1\}} \sum_{x_3 \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} \mathsf{Arith}_\phi(z_1, x_2, x_3, \ldots, x_n).$$

The verifier then asks the prover to send a polynomial $s_2(x_2)$ such that:

$$s_2(x_2) = \sum_{x_3 \in \{0,1\}} \sum_{x_4 \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} \mathsf{Arith}_\phi(z_1, x_2, x_3, \ldots, x_n) \tag{13.5}$$

Verifier: checks that

$$s_1(z_1) = s_2(0) + s_2(1) \tag{13.6}$$

The protocol iterates in this fashion having the prover send a polynomial $s_i(x_i)$ with degree at most d with the intent that

$$s_i(x_i) = \sum_{x_{i+1} \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} \mathsf{Arith}_\phi(z_1, z_2, \ldots, z_{i-1}, z_i, x_{i+1}, \ldots, x_n) \tag{13.7}$$

and having the verifier check $s_i(0) + s_i(1) = s_{i-1}(z_{i-1})$ (reject if it fails) and then randomly sampling $z_i \in \mathbb{Z}_p$ if it succeeds and sending this to the prover.

This continues until the verifier is left with checking

$$s_n(z_n) = \mathsf{Arith}_\phi(z_1, \ldots, z_n).$$

The verifier can directly check this in polynomial time without further help.

13.2.2.3 Completeness

$(\phi, k) \in \#\text{SAT}$:
In this case the protocol proceeds without trouble and the verifier accepts with probability 1, by construction. The prover simply needs to provide the polynomials (13.7) it has been requested to send.

13.2.2.4 Soundness

$(\phi, k) \notin \#\text{SAT}$:
In this case, in the first step, either the first polynomial $s_1(z_1)$ has $s_1(0) + s_1(1) \neq k$, in which case the verifier immediately rejects, or $s_1(0) + s_1(1) = k$ in which case $s_1(0) + s_1(1) \neq g$. In the latter case, we have the polynomial $s_1(x_1)$ is distinct from the univariate polynomial (over x_1) obtained by partially evaluating g at the last $n-1$ sums.

But two distinct degree d polynomials can agree on at most d points (since the difference of two distinct degree d polynomials is nonzero and has degree d). So when sampling z_1 the probability that

$$s(z_1) \neq \sum_{x_2 \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} \mathsf{Arith}_\phi(z_1, x_2, \ldots, x_n)$$

is at least $1 - d/p$. If so, then the next iteration begins with a false claim to be verified.

In general if $k_i \neq \sum_{x_{i+1} \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} \mathsf{Arith}_\phi(z'_1, \ldots, z'_i, x_{i+1}, \ldots, x_{i-1}\}$ where $k_i := s_i(z'_i)$ then either the verifier will reject s_{i+1} or, with probability at least $1 - d/p$, the subsequent round will begin with the false claim $k_{i+1} \neq \sum_{x_{i+2} \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} \mathsf{Arith}_\phi(z'_1, \ldots, z'_i, x_{i+1}, \ldots, x_{i-1}\}$.

So if the base case is reached, then with probability $(1 - d/p)^n \geq (1 - 1/2^n)^n \geq 2/3$ (for $n \geq 3$) the statement $s_n(z_n) =_p [\mathsf{Arith}_\phi(z_1, \ldots, z_n)$ is false and the verifier will determine this through direct calculation.

13.3 IP = PSPACE

The warm up on #SAT gave us a protocol for deciding polynomial equality in IP for polynomials with polynomial bit complexity. To show that PSPACE $\subseteq$ IP we will extend the "arithmatization" trick to formulas with universal, as well as existential quantifiers. This will allow us to use the protocol for TQBF instances. We will, however, need one more trick to ensure that the polynomials exchanged retain only polynomial bit complexity: the $\forall$ will correspond to multiplication and this could cause a true TQBF instance to evaluate to an integer much greater than 2^n if we leave $\exists$ corresponding simply to a sum. The trick is to use a different substitution for $\exists$ so the final expression is guaranteed to be 0 or 1.

We ammend the "arithmatization" from before with two new mapping rules:

- $\mathsf{Arith}(\exists x_i : \phi(x_1, \ldots, x_n)) \mapsto (1 - (1 - \mathsf{Arith}_\phi(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)) \cdot (1 - \mathsf{Arith}_\phi(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n))$
- $\mathsf{Arith}(\forall x_i : \phi(x_1, \ldots, x_n)) \mapsto \mathsf{Arith}_\phi(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n) \cdot \mathsf{Arith}_\phi(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$

Note that the mapping for the existential quantifier corresponds to the de Morgan transformed OR we saw earlier, while the mapping for the universal quantifier corresponds to our encoding of AND. We can now encode TQBF instances as polynomials, which evaluate to one iff the corresponding quantified Boolean formula is true. However, the universal quantifier mapping rapdily inflates the degree of the polynomial produced, with each universal quantifier potentially doubling the degree of the resulting polynomial. In general, a formula which maps to a degree d polynomial, when wrapped in n universal quantifiers maps to a polynomial of degree $2^n d$ which cannot be sent as a polynomial length message in an IP protocol.

Observe that $x \in \{0,1\} \Rightarrow x^2 = x \Rightarrow x^n = x$, for $n > 0$. So, so long as we are only concerned with $\{0,1\}$ arguments, we can reduce the complexity of polynomial by linearizing the variables.
Let R_i be the operator on polynomials which linearizes the ith argument of the given polynomial, then

$\forall \hat{x} \in \{0,1\}^n R_i p(\hat{x}) = p(\hat{x})$. Note that $R_i p$ can be computed in time polynomial in p if p is given as an explicit sum of monomials: simply traverse the representation of the polynomial, and replace each instance of x_i^k with x_i.

Then, if we interleave appropriate calls of the reduction operators R_i into the protocol, we can restrict the exchanged intermediate polynomials to polynomials of polynomial bit complexity, since each polynomial is of degree at most 2 over at most n variables, since at most one degree doubling will occur between each reduction, and after reduction the polynomial is a multilinear polynomial.

From here, the protocol to decide TQBF is similar to that for #SAT:

13.3.1 Protocol

- Prover: choose a prime $p \in (2^n, 2^{n+1}]$ (this is always possible, by Chebyshev's theorem).
- Verifier: check that p is prime, reject if it is not, otherwise continue (this is possible in polynomial time by [AKS04], though even a probabilistic check would suffice).

The protocol will check the following equation:

$$O_1 R_1 O_2 R_1 R_2 \dots O_n R_1 \dots R_n \ \mathsf{Arith}_\phi(x_1, x_2, \dots, x_n) =_p 1 \tag{13.8}$$

Where $O_i \in \{\exists x_i, \forall x_i\}$

- Verifier: if no quantifiers or reduction operators appear in the instance being decided, simply check if $\mathsf{Arith}_\phi(z_1, z_2, \dots, z_n) =_p 1$, accept if that holds, and reject otherwise. Else proceed by case analysis on the outermost operator or quantifier.

Case:

1. $O_i = \forall x_i$:
   The prover sends a linear polynomial $s(x_i)$. The verifier checks that $s(0) \cdot s(1) =_p k$ and rejects if this fails. Otherwise the verifier samples z_i uniformly from $\mathbb{Z}_p$, sends it to the prover. The protocol then recurses with O_i eliminated, and the fixed z_i substitued in for x_i.

   The intent is that s should be the univariate polynomial obtained if all operators except $\forall x_i$ one were applied. The next round continues with this operator removed and with the new value $s(0) \cdot s(1)$ for k.

2. $O_i = \exists x_i$:
   The prover sends a linear polynomial $s(x_i)$. The verifier checks that $1 - (1 - s(0)) \cdot (1 - s(1)) =_p k$ and rejects if this fails. Otherwise the verifier samples z_i uniformly from $\mathbb{Z}_p$, sends it to the prover. The protocol then recurses with O_i eliminated, and the fixed z_i substitued in for x_i.

   Again, the intent is that s should be the univariate polynomial obtained if all operators except $\exists x_i$ one were applied. The next round continues with this operator removed and with the new value $s(0) + s(1)$ for k.

3. R_i:
   This takes just a bit more explaining. In general, some values $z_1, \dots, z_j$ have already been fixed when an operator of the form $R_i (i \leq j)$ is the leftmost operator that should be removed. At this point, we are trying to verify

   $$k = [R_i R_{i-1} \cdots R_1][O_{j+1} R_{j+1} \cdots R_1][O_{j+2} R_{j+2} \cdots R_1] \cdots [O_n R_n \cdots R_1] \mathsf{Arith}_\Phi(z_1, \dots, z_{j-1}, x_j, \dots, x_n).$$

The square brackets are added just for emphasis. Here, each $O_{j'}$ is of the form $\forall x_{j'}$ or $\exists x_{j'}$.

The polynomial s that is requested by the verifier is supposed to be the univariate polynomial (over x_i) obtained by first applying the remaining operators $[R_{i-1} \cdots R_1][O_{j+1} R_{j+1} \cdots R_1] \cdots [O_n R_n \cdots R_1]$ to $\mathsf{Arith}_\Phi(x_1, \ldots, x_n)$ and then plugging in all $z_1, \ldots, z_j$ **except** z_i, leaving x_i as a variable.

The prover sends an at most quadratic polynomial $s(x_i)$ (since we have at most doubled the degree of the polynomial since the last multilinearization). The verifier checks that $[R_i s](z_i) =_p k$ and rejects if this fails. The verifier the resamples z_i uniformly from $\mathbb{Z}_p$, sends it to the prover. The protocol then recurses with this call to R_i eliminated, and the new z_i substituted in for for the old z_i and with the new value for k being $s(z_i)$.

Note that each case reduces the number of quantifiers plus reductions by one, and so the recursion terminates.

13.3.1.1 Completeness

Similar to the #SAT case, by construction, if the instance is a valid TQBF instance, then at each step, the prover can provide an honest answer which ensures the verifier does not reject, and that the next claim is also true. So in the true case, the verifier will accept with probability one.

13.3.1.2 Soundness

Again similar to the #SAT case, in each step of the recursion, if the prover's claim going into the recursion was false, then even if it avoids being rejected in that round, it proceeds to the next recursion with a false claim with high probability. In particular, if the prover's claim is false, it either provides a polynomial which fails the verifiers check, or it provides a distinct at most degree 2 polynomial. But, again, distinct degree 2 polynomial agree at at most 2 points. So the verifier catches the prover with probablilty $\geq 1 - 2/p$. Hence the probability that the verifier rejects, given that the intial claim by the prover was false is $\geq (1 - 2/p)^{n^2} \geq 2/3$ for large enough n. Here, n^2 is an upper bound on the number of operators that are to be removed.

Therefore TQBF $\in \mathsf{IP}$, hence $\mathsf{PSPACE} \subseteq \mathsf{IP}$, thus $\mathsf{PSPACE} = \mathsf{IP}$.

Comment: Observe that in the "yes" instance case, the verifier can always convince the prover to accept. So in the definition of IP we could require the verifier to always accept some prover and this would not change the set of languages that can be decided with an interactive proof.

References

AKS04 M. Agrawal and N. Kayal and N. Saxena, PRIMES Is in P, *Annals of Mathematics* **2** Vol. 160, 2004, pp.781–793.

SHA92 A. Shamir, $\mathsf{IP} = \mathsf{PSPACE}$, *J. ACM* **4** Vol. 39, 1992, pp.869–877.

SHE92 A. Shen, $\mathsf{IP} = \mathsf{PSPACE}$: Simplified Proof, *J. ACM* **4** Vol. 39, 1992, pp.878–880.