CS6160 Complexity

Topic: $\mathsf{AM}, \mathsf{BPL} \subseteq \mathsf{P}, \#\mathsf{SAT}_D \in \mathsf{IP}$ | Date: Oct 26, 2026

Lecturer: Wei-Kai Lin | Scriber: Wish Shao

1 Introduction to the Class AM

Zizheng introduced the class AM (Arthur–Merlin).

Definition 1 (AM)**.** [AB09, Definition 8.10] $\mathsf{AM}[k]$ is defined as the subset of $\mathsf{IP}[k]$ obtained when we restrict the verifier's message to be random bits, and not allowing it to use any other random bits that re not contained in these message. We also call such proof a public coin proof or Arthur-Merlin proof.

Theorem 2. *[AB09, Theorem 8.12]* $\mathsf{GNI} \in \mathsf{AM}[2]$.

Proof sketch. Given two graphs G_1, G_2 on n vertices, there are $n!$ possible vertex permutations, and each permutation produces a graph isomorphic to the original graph.

However, different permutations may produce the same labeled graph. An *automorphism* is a permutation that leaves the graph unchanged. Define

$$\mathrm{aut}(G) = \{\pi : \pi(G) = G\}.$$

A nontrivial automorphism is one that is not the identity permutation. A graph has fewer than $n!$ distinct isomorphic copies if and only if it has a nontrivial automorphism.

First, consider the set

$$S_0 = \{H : H \cong G_1 \vee H \cong G_2\}.$$

If neither graph has a nontrivial automorphism, then each has exactly $n!$ distinct isomorphic copies. Thus,

$$|S_0| = \begin{cases} n!, & G_1 \cong G_2, \\ 2n!, & G_1 \ncong G_2. \end{cases}$$

For the general case, a graph G has $n!/|\mathrm{aut}(G)|$ distinct isomorphic copies. To account for automorphisms, we redefine the set to include pairs of graphs and their automorphisms:

$$S = \{(H, \pi) : (H \cong G_1 \vee H \cong G_2) \wedge \pi \in \mathrm{Aut}(H)\}.$$

Since isomorphic graphs have the same number of automorphisms, each isomorphism class contributes exactly

$$\frac{n!}{|\mathrm{Aut}(G)|} \cdot |\mathrm{Aut}(G)| = n!$$

pairs to S. Therefore,

$$|S| = \begin{cases} n!, & G_1 \cong G_2, \\ 2n!, & G_1 \ncong G_2. \end{cases}$$

This reduces GNI to distinguishing these two possible sizes of S. We will later introduce an interactive protocol for proving a lower bound on the size of a set. □

2 Quiz 1 Review: $\mathsf{BPL} \subseteq \mathsf{P}$

In Quiz 1, we were asked to prove $\mathsf{BPL} \subseteq \mathsf{P}$. A common mistake is to assume that logarithmic space immediately implies polynomial running time.

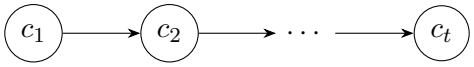
Recall that a Turing machine using $O(\log n)$ space has at most

$$2^{O(\log n)} = \text{poly}(n)$$

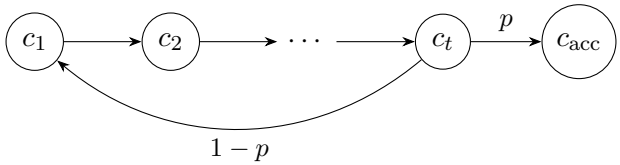
configurations. However, having polynomially many configurations does **not** necessarily imply polynomial run time for a randomized machine.

The key distinction is whether a computation can revisit the same configuration.

For a **deterministic machine** that always halts, revisiting a configuration would cause an infinite loop. Thus, its computation cannot repeat any configuration and must terminate in polynomial time:



However, for a **randomized machine**, the same configuration may be revisited with some probability, while another random choice allows the computation to proceed and eventually terminate:



Consequently, a randomized machine with logarithmic space is not necessarily polynomial-time bounded unless a time bound is imposed separately.

3 $\#\mathsf{SAT}_D \in \mathsf{IP}$

3.1 Preliminaries

Recall the decision version of #SAT:

$$\#\mathsf{SAT}_D = \{\langle \phi, k \rangle : \phi \text{ has exactly } k \text{ satisfying assignments}\}.$$

In the previous lecture, we introduced arithmetization [AB09, Section 8.3.1], which converts a Boolean formula ϕ into a polynomial g such that, for every Boolean assignment $b \in \{0,1\}^n$,

$$\phi(b) = g(b).$$

Therefore, $\langle \phi, k \rangle \in \#\mathsf{SAT}_D$ if and only if

$$\sum_{b_1,\ldots,b_n \in \{0,1\}} g(b_1, \ldots, b_n) = k.$$

For a 3CNF formula with m clauses, the total degree of g is at most $3m = \text{poly}(n)$. We denote this degree bound by d.

We choose a prime $p \in (2^n, 2^{2n}]$. This prevents different satisfying assignment counts from becoming equal modulo p. For example, if $n = 3$ and $p = 5$, a formula with 7 satisfying assignments would be indistinguishable from one with 2 satisfying assignments, since $7 \equiv 2 \pmod 5$. Choosing $p > 2^n$ avoids this issue, as the number of satisfying assignments is at most 2^n. A sufficiently large field also ensures a small soundness error.

3.2 Sumcheck Protocol

Base case: $n = 1$**.** The verifier V directly checks whether

$$g(0) + g(1) = k.$$

It accepts if the equality holds and rejects otherwise.

Inductive case: $n > 1$**.** The prover sends a univariate polynomial $s_1(x_1)$, claimed to be

$$s_1(x_1) = \sum_{b_2,\ldots,b_n \in \{0,1\}} g(x_1, b_2, \ldots, b_n),$$

where $\deg(s_1) \leq d$.

The verifier checks that

$$s_1(0) + s_1(1) = k$$

and $\deg(s_1) \leq d$. If either check fails, V rejects. Otherwise, V samples $a_1 \leftarrow \mathbb{F}_p$, computes $s_1(a_1)$, and sends a_1 to P.

The protocol then recurses on the claim

$$\sum_{b_2,\ldots,b_n \in \{0,1\}} g(a_1, b_2, \ldots, b_n) = s_1(a_1).$$

Just to demonstrate the next round: P sends a polynomial

$$s_2(X_2) = \sum_{b_3,\ldots,b_n \in \{0,1\}} g(a_1, X_2, b_3, \ldots, b_n).$$

The verifier checks

$$s_2(0) + s_2(1) = s_1(a_1)$$

and $\deg(s_2) \leq d$, then samples $a_2 \leftarrow \mathbb{F}_p$.

The protocol continues recursively until only one variable remains, at which point V directly evaluates the polynomial as in the base case.

Claim 3. *For any false statement and any prover P^*, the Verifier rejects with probability at least $(1 - \frac{d}{p})^{n-1}$.*

Proof sketch. Let

$$h_1(X_1) = \sum_{b_2,\ldots,b_n \in \{0,1\}} g(X_1, b_2, \ldots, b_n)$$

be the correct polynomial.

Both s_1, h_1 have degree at most d. Thus, the nonzero polynomial $s_1 - h_1$ has at most d roots over $\mathbb{F}_p$. Consequently,

$$\Pr[s_1(a_1) = h_1(a_1) \mid s_1 \neq h_1] \leq \frac{d}{p}.$$

Therefore, with probability at least $1 - d/p$, the next recursive claim remains false. The same argument applies to every next round, so with probability at least $(1 - d/p)^{n-1}$, the claim remains false throughout all rounds. Since the final check is deterministic, the verifier must reject.

□

Theorem 4 (Shamir [LFKN92] [Sha92])**.** PSPACE $\subseteq$ IP.

We will complete the proof in the next lecture by constructing an interactive proof for the PSPACE-complete problem True Quantified Boolean Formula (TQBF).

Again, we first arithmetize the Boolean formula ϕ into a polynomial g. We replace existential quantifiers ($\exists$) with summation and universal quantifiers ($\forall$) with multiplication.

For example, a quantified Boolean formula $\forall x_1 \exists x_2 \forall x_3 \cdots \exists x_n \, \phi(x_1, \ldots, x_n)$ is true if and only if

$$\prod_{b_1 \in \{0,1\}} \sum_{b_2 \in \{0,1\}} \prod_{b_3 \in \{0,1\}} \cdots \sum_{b_n \in \{0,1\}} g(b_1, \ldots, b_n) > 0.$$

Here, the expression is evaluated over the integers: summation is positive if at least one assignment satisfies the remaining formula, while multiplication is positive only if both assignments do.

However, repeated multiplication can cause the intermediate values to grow doubly exponentially, up to $2^{2^{O(n)}}$. Representing such large integers would require exponentially many bits, making direct evaluation infeasible for a polynomial-time verifier. We will discuss more in the next lecture...

Acknowledgement

References

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[LFKN92] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM (JACM)*, 39(4):859–868, 1992.

[Sha92] Adi Shamir. Ip= pspace. *Journal of the ACM (JACM)*, 39(4):869–877, 1992.