

Indian Association for the Cultivation of Science

(Deemed to be University under *de novo* Category)

Master's/Integrated Master's-PhD Program/ Integrated Bachelor's-Master's Program/PhD Course

Theory of Computation II: COM 5108

Lecture Adv-1-IP

Instructor: Goutam Biswas *Autumn Semester 2026*

1 Introduction

The class **NP** is characterized by the efficient (*polynomial-time*) and *deterministic* verifier of the proof of the assertion of the membership of an object in a set. The supplier of the proof, the *prover*, has no real role to play other than the polynomial bound on the length of the proof.
It was found that if a *probabilistic* verifier is adopted, the proof system may be used to characterize much bigger class (most likely) with a low probability of error. The error can be made negligible by repeated application of the verification procedure.

The collection of sets that have *Interactive Proof System* (**IP**), is found to be same as the deterministic polynomial space bounded class **PSPACE** which is believed to be larger than **NP**.

An **NP** proof gives information that possibly cannot be computed efficiently. As an example, a satisfying truth value assignment is a proof of *satisfiability* of a Boolean formula ϕ. But there is no known efficient algorithm to compute it. There are interactive proofs that can convince the the verifier the correctness of an assertion without revealing any other information. It is called a *zero knowledge proof*.

A standard proof of an assertion can be transformed to another polynomial size proof (with redundancy). The assertion can be validated by a probabilistic procedure that access (oracle) a few locations of the transformed proof. A valid proof is accepted with certainty. But an invalid proof of a false assertion is rejected with a probability $\geq \frac{1}{2}$.

A *proof* system is (in)formally defined in logic as a finite set of *axioms*, and a set of *transformation* or *derivation* rules. Axioms may be considered as the *basis* of a proof system. A *proposition* is a formal statement[1] which may be *true* or *false*. An *assertion* is a *proposition* that is claimed to be true. A *proof* of an assertion is a sequence of statements, derived from the axioms using the

[1] In a system of mathematical logic, there is a formal definition of a statement over some alphabet and binding rules.

derivation rules, and ends with an assertion.
The *verification* of a proof is determined by the formal or informal definitions of the proof system. In computing, the notion of an efficient verification procedure is connected with the notion of efficient (polynomial time) computation. The verifier for the class **NP** has the pair (a, p_a) as input, where a is an assertion, and p_a is the expected certificate or proof of a. The proof can be verified in time polynomial in the length of the assertion a.

But it has been proved that if *probabilistic polynomial-time* verification is allowed, then a substantial benefit can be extracted as a few mentioned earlier.

2 Interactive Proof System

The *proof* in this system is not a *fixed object* like a classical proof in mathematics, or a proof supplied to the **NP**-verifier. It is more like an argument, a sequence of queries from the *verifier*, and answers by the *prover*. The role of the *prover* is to convince the *verifier* that the assertion is correct (even if it is false!). The role of the *verifier* is to check the validity of the claim of the *prover*. The validation is not certain, but established with a high probability. We start with an example.
Example 1. We consider the decision problems of *graph isomorphism* and its complement.

$$GI = \{< G_0, G_1 >:\ G_0 \cong G_1\},\ \overline{GI} = \{< G_0, G_1 >:\ G_0 \ncong G_1\},$$

where $< G_0, G_1 >$ is the encoding of two graphs.
If the edge sets of the graphs are represented by $n \times n$ *adjacency matrices* M_0 and M_1, and by a permutation of the vertices $\pi : V_0 \to V_1$, M_0 turns out to be same as M_1, then the graphs are isomorphic and $< G_0, G_1 >\in GI$. Otherwise, $< G_0, G_1 >\in \overline{GI}$.

The language $GI \in \mathbf{NP}$ as the permutation π is a polynomial length proof of isomorphism, and it can be verified in polynomial-time. But there is no known polynomial-time algorithm to generate this proof[2].
On the other hand, even if G_0 in not isomorphic to G_1, there is no known polynomial length proof of that fact. The decision problem $\overline{GI} \in \mathbf{coNP}$.

But there is an interactive proof of the membership of $< G_0, G_1 >$ in $\overline{GI}$ with high certainty. The exchange between the *prover* and the *verifier* is as follows. The input (to both) is $< G_0, G_1 >$.

- *Prover:* Claims that the graphs are non-isomorphic (even if it is not the case).
- *Verifier:* The verifier selects at random G_i, $i \in \{0, 1\}$. It generates a random permutation π of the vertex set of V_i. The new adjacency matrix is M_i^π, such that $M_i^\pi[\pi[u], \pi[v]] = M_i[u, v]$. It sends $G_i^\pi = (V_i, M_i^\pi)$ to the prover. Note that the computation of the verifier is randomized, but bounded by polynomial time.
- *Prover:* It has unlimited computation power. If the input pair is non-isomorphic, it can detect with certainty whether G_i^π is isomorphic to G_j, $j \in \{0, 1\}$, and returns j.

[2] GI is not known to be **NP**-*hard*.

But if the input pair is isomorphic, then G_i^π received can be a permutation of G_0 as well as G_1. The *prover* selects $j \in \{0, 1\}$ at random (to fool the verifier), and returns j. The probability that $i = j$ is $\frac{1}{2}$

- *Verifier:* On receiving j, will *accept* prover's claim of non-isomorphism, if $i = j$, otherwise, it *rejects*.

If $G_0 \ncong G_1$, prover will always be able to send $j = i$, and verifier will *accept* with certainty. The proof system is *complete*.
But if $G_0 \cong G_1$, the prover will be able to able to send the correct value j with the probability $\frac{1}{2}$.

The following are a few important points about a proof system.

- *Soundness:* A verification system should not accept any *false proof* of a *false assertion*.
- *Completeness:* A prover should be able to establish the validity of a true statement[3].
- If the proof can be generated in polynomial time, it is great. The class **P** has this property.
- But if there is no known polynomial time prover, we ignore the time complexity of the proof generation. But the verification of the proof should be efficient (polynomial-time). The class **NP** has this property.
- In an interactive proof system, the verifier does not *trust* the prover. The prover tries to convince the verifier.

2.1 Definition

An interactive proof may be viewed as a game between the *verifier* and a *prover*. The *prover* is equipped with unlimited computational power, and tries to convince the probabilistic polynomial-time-bounded *verifier* about the validity of an assertion (*true* or *false*). If the assertion is true, the *prover* will always be able to convince the *verifier*. The *verifier* rejects a false assertion with a probability $\geq \frac{1}{2}$.

The interaction between the prover and the verifier, known as their strategies, is a pair of functions. The move of the verifier depends on the input (assertion), the sequence of its coin tosses, and the sequence of messages received from the prover. The similar is true for the prover.

Let the strategies of two parties be A and B on the common input x and the randomness r_A and r_B respectively. The message sequence generated by A is $\alpha_1, \alpha_2, \cdots$, and that of B is $\beta_1, \beta_2, \cdots$. If we assume that A makes the first move, and B makes the last move, then $\alpha_i = A(x, r_A, \beta_1, \cdots, \beta_{i-1})$ and $\beta_i = B(x, r_B, \alpha_1, \cdots, \alpha_i)$. If A takes a decision after t rounds, then it is defined as $A(x, r_a, \beta_1, \cdots, \beta_t)$.

The strategy is probabilistic and polynomial-time-bounded if the next message can be computed in *poly(|x|)* steps. Naturally, the number of messages, their lengths, and the number of coin tosses are all bounded by *poly(|x|)*.

[3]We already know that there may not be a proof system for every set. Think of the *halting problem*.

Definition 1. An *interactive proof system (IP)* for a set S is a *prover-verifier* game, where the *verifier* is using a probabilistic polynomial-time strategy V. But the *prover* can use a computationally unbounded strategy, P. Both have the common input x, satisfying the following conditions.

- *Completeness:* if $x \in S$, the *verifier* V, after interacting with the *prover* *accepts* it.
- *Soundness:* if $x \notin S$, then on every strategy P of the *prover*, the *verifier* *rejects* x with a probability $\geq \frac{1}{2}$.

The class **IP** is the collection of sets having an *interactive proof system*. It was proved that **IP** = **PSPACE**.

2.2 Arithmetization of Boolean (CNF) Formulae

We have already shown an example where the membership problem of a set in **coNP**[4] can be decided by an *interactive proof system*.
The class **NP** $\subseteq$ **IP**, as the *prover* can present the complete polynomial-length proof of an assertion to the verifier. It is verified by the *verifier* in polynomial time.
We show that **coNP** $\subseteq$ **IP** by constructing an *interactive proof system* for $\overline{SAT} = \{\phi : \phi \text{ is an unsatisfiable Boolean formula in CNF}\}$[5]. So **NP** $\cup$ **coNP** $\subseteq$ **IP**.

A Boolean formula is transformed into a formula of integer variables. The process is known as *arithmetization*. A *Boolean assertion* is translated to an *arithmetic assertion*. The *interactive proof system* works on the the arithmetic assertions to establish the *unsatisfiability* of a Boolean formula.
The transformation rules are as follows.

- Constants: *true* $\mapsto 1$, *false* $\mapsto 0$.
- Operators: $x \vee y \mapsto x + y$, $x \wedge y \mapsto x \cdot y$, and $\neg x \mapsto 1 - x$.
- Expression value: 0 (*false*), $n > 0$ (*true*).

Example 2. Let

$$\phi_1 = (x \vee y \vee z) \wedge (x \vee y \vee \neg z) \wedge (\neg x \vee y \vee z) \wedge (\neg x \vee y \vee \neg z)$$

There are 4 satisfying assignments, and one of them is $(0, 1, 0)$.

$$\phi_2 = (x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee z) \wedge (\neg x \vee \neg z)$$

There is no satisfying assignment.

After the arithmetization:

$$\phi_1^A = (x + y + z) \cdot (x + y + (1 - z)) \cdot ((1 - x) + y + z) \cdot ((1 - x) + y + (1 - z)) \cdot$$

The value of ϕ_1^A for the assignment $(0, 1, 0)$ is $1 \cdot 2 \cdot 2 \cdot 3 = 12$. So the original formula is *satisfiable*.
But the value of

$$\phi_2^A = (x + y) \cdot (x + (1 - y)) \cdot ((1 - x) + z) \cdot ((1 - x) + (1 - z)).$$

[4] $GI \in$ **NP**, so $\overline{GI} \in$ **coNP**.
[5] $\overline{SAT}$ is a *complete problem* for the class **coNP**.

is zero for any assignment, as one of the factors will always be zero.

Let in a formula $\phi(x_1, \cdots, x_n)$, the maximum number of literals per clause be l, and m be the number of clauses.
The formula $\phi^A(x_1, \cdots, x_n)$ is a polynimial over $x_1, \cdots, x_n$. The total degree of the polynomial is m, the number of clauses.
If the the value per clause is v, then $0 \leq v \leq l$. The value of the formula is v_ϕ, $0 \leq v_\phi \leq l^m$.
The sum of the total value added over all possible assignments of n variables is $\sum_{x_1,\cdots,x_n \in \{0,1\}} \phi^A$. It is in the range 0 to $2^n \times l^m$.
A formula is unsatisfiable if $\sum_{x_1,\cdots,x_n \in \{0,1\}} \phi^A = 0$.

The set $\mathbb{Z}_q = \{0, 1, \cdots, q-1\}$ forms a finite field $\mathbb{F}_q =< \mathbb{Z}_q, +_q, \times_q, 0, 1 >$, when q is a prime. If two integers $m, n \in \{0, 1, \cdots, M\}$, and $q > M$, then $m = n$ if and only if $m \equiv n \pmod{q}$.

A Boolean formula ϕ in CNF is unsatisfiable if the summation of the corresponding arithmetic formula evaluates to zero in the *finite field* $\mathbb{F}_q$.

$$\sum_{x_1=0,1} \cdots \sum_{x_n=0,1} \phi_A(x_1, \cdots, x_n) \equiv 0 \pmod{q},$$

where the prime $q \geq 2^n \times l^m$.

2.3 Outline of the Protocol

The following is an outline of the *Interactive Proof (IP)* protocol to establish the *unsatisfiability* of a Boolean formula in CNF. If the formula ϕ is *unsatisfiable*, it will be established with certainty. If it is *satisfiable*, then that can be detected with high probability. In the protocol, the *prover* with its unbounded computational power tries to prove that

$$\sum_{x_1=0,1} \cdots \sum_{x_n=0,1} \phi_A(x_1, \cdots, x_n) \equiv 0 \pmod{q},$$

(even if it is not). The *verifier* using randomization and polynomial-time bounded computing resources tries to verify the claim. We start with an example.
Example 3. Let

$$\phi(x, y, z) = (x) \wedge (\neg x \vee y) \wedge (\neg y) \wedge (z).$$

Here the number of variables, $n = 3$, maximum number of literals per clause, $l = 2$, and the number of clauses is $m = 4$.

The formula after arithmetization is

$$\phi^A(x, y, z) = x((1 - x) + y)(1 - y)z = x(1 - x + y)(1 - y)z.$$

The maximum value of sum over all possible valuations of variables of $\phi^A(x, y, z)$ cannot exceed $2^n \times l^m = 2^3 \times 2^4 = 128$.
But in our example there are three unit clauses and one clause with two litererals. So the value cannot exceed 2, and the value of the sum cannot exceed $2^3 \times 2 = 16$.
We can take the prime $q = 17$, and compute in $\mathbb{F}_{17}$. If $\phi(x, y, z)$ is unsatisfiable, then

$$\sum_{x=0,1} \sum_{y=0,1} \sum_{z=0,1} x(1 - x + y)(1 - y)z \equiv 0 \pmod{17}.$$

At the beginning of the first iteration, the prover strips the summation over x, evaluates

$$\sum_{y=0,1}\sum_{z=0,1} x(1-x+y)(1-y)z$$

as a univariate polynomial of x on $\mathbb{F}_q$.
In the example, the polynomial is $p_1(x) = x - x^2$. The polynomial is sent to the verifier. The verifier checks that $p_1(0)+p_1(1) = 0$, as expected for a unsatisfiable formula ($v_0 = 0$). Then it randomly chooses an $r_1 \in \mathbb{F}_q$, evaluates $v_1 = p_1(r_1) \mod q$. Let the chosen $r_1 = 3$, and $v_1 = p_1(3) = 3 - 3^2 \equiv -6 = 11 \pmod{17}$. The verifier sends $r_1(= 11)$ to the prover.

In the next round, the prover has the formula

$$\sum_{y=0,1}\sum_{z=0,1} \phi^A(3, y, z) = \sum_{y=0,1}\sum_{z=0,1} 3(1-3+y)(1-y)z.$$

The summation on the variable y is removed, the univariate polynomial of y over $\mathbb{F}_q$ is evaluated, and sent to the verifier. The polynomial is

$$p_2(y) = -3y^2 + 9y - 6 \equiv 14y^2 + 9y + 11 \equiv \pmod{17}.$$

The verifier verifies that $p_2(0) + p_2(1) \equiv 11 + 14 + 9 + 11 \equiv 11 \pmod q$. It is the same as v_1 it calculated and sent.
The verifier again, randomly chooses $r_2 = 4 \in \mathbb{F}_q$, computes $v_2 = p_2(4) = 14 \cdot 4^2 + 9 \cdot 4 + 11 \equiv -1 \pmod{17}$ and sends $r_2(= 4)$ to the prover.

In the third round, the prover has the formula

$$\sum_{z=0,1} \phi^A(3, 4, z) \equiv \sum_{z=0,1} 3(1-3+4)(1-4)z \equiv \sum_{z=0,1} -z.$$

It sends the verifier $p_3(z) = -z$.
The verifier evaluates evaluates $p_3(0) + p_3(1) \equiv -1 \pmod q$, satisfied, selects at random $r_3 = 2$, evaluates $v_3 = -2$, and sends $r_3(= 2)$ to the prover.

At the end prover evaluates and sends $p_3(2) = -1$ and the verifier accepts $\phi(z, y, z)$ as unsatisfiable.

Example 4. We consider a satisfiable formula that the prover is trying to establish as unsatisfiable,

$$\phi(x, y, z) = (x) \wedge (\neg x \vee y) \wedge (\neg y \vee z) \wedge (z).$$

Here the number of variables, $n = 3$, maximum number of literals per clause, $l = 2$, and the number of clauses is $m = 4$. The formula after arithmetization is

$$\phi^A(x, y, z) = x((1-x)+y)(1-y+z)z = x(1-x+y)(1-y+z)z.$$

The maximum value of sum over all possible valuations of variables of $\phi^A(x, y, z)$ cannot exceed $2^n \times l^m = 2^3 \times 2^4 = 128$. We can take $q = 131$, and work in $\mathbb{F}_{131}$. But again it is safe to work with a prime larger than $2^3 \times (1 \cdot 2 \cdot 2 \cdot 1) = 32$. We take $q = 37$. If $\phi(x, y, z)$ is unsatisfiable, then

$$\sum_{x=0,1}\sum_{y=0,1}\sum_{z=0,1} x(1-x+y)(1-y+z)z \equiv 0 \pmod{37},\ \textit{i.e.}\ v_0 = 0.$$

The prover should send the polynomial

$$p_1(x) = 2x(1-x) + x(2-x) = -3x^2 + 4x.$$

But then $p_1(0) + p_1(1) \equiv 0 + 1 \equiv 1 \pmod 1$1. The verifier *rejects* the unsatisfiability claim.

But the unscrupulous prover sends

$$p_1'(x) = -3x^2 + 3x?$$

In this round, the verifier will accept the polynomial as $p'(0) + p'(1) = 0$. Then it randomly choose $r_1 = 2 \in \mathbb{F}_{37}$. The verifier evaluates $v_1 = p_1'(2) = -3 \times 2^2 + 3 \times 2 \equiv -6 \equiv 31 \pmod{37}$, and sends $r_1(= 2)$ to the prover.

In the second round the prover computes

$$\sum_{y=0,1} \sum_{z=0,1} 2(1-2+y)(1-y+z)z \equiv 0 \pmod{37},$$

So $p_2(y) = -2y^2 + 6y - 6$. But $p_2(0) + p_2(1) = -6 + (-2) + 6 - 6 = -8 \not\equiv v_1 = -6$. So the prover will have to get a polynomial $p_2'(y)$ so that $p_2'(0) + p_2'(1) = -6$. The polynomials $p_2()$ and $p_2'()$ shouls have the same value at the randonly chosen $r_1 \in \mathbb{F}_{37}$. The probability of getting such low degree polynomials is low.

Completeness: If the claim is true, the prover will send the correct polynomial, and eventually, the verifier will accept it.

Soundness: The probability of getting two distinct univariate polynomials $p_1(x)$ and $p_2(x)$ over the field $\mathbb{F}_q$, such that for some randomly chosen $r \in \mathbb{F}_q$, $p_1(r) = p_2(r) \leq \frac{d}{q}$, where $d \leq \max\{deg(p_1(x)), deg(p_2(x))\}$. It is low if $d \ll q$[6].

In general, at the beginning of the i^{th} iteration, the assertion to prove is

$$\sum_{x_i=0,1} \sum_{x_{i+1}=0,1} \cdots \sum_{x_n=0,1} \phi_A(r_1, \cdots, r_{i-1}, x_i, x_{i+1}, \cdots, x_n) \equiv v_{i-1} \pmod q.$$

The *prover* removes the summation over the variable x_i, partially evaluates the expression, and creates a single variable polynomial of x_i over the field $\mathbb{F}_q$. The coefficients of different terms of $p_i(x_i)$ are computed by the the *prover* with

$$x_1 \leftarrow r_1, \cdots, x_{i-1} \leftarrow r_{i-1} \text{ and } \sum_{x_{i+1}=0,1} \cdots \sum_{x_n=0,1} \text{ over } x_{i+1} \cdots x_n.$$

$$p_i(x_i) = \sum_{x_{i+1}=0,1} \cdots \sum_{x_n=0,1} \phi_A(r_1, \cdots, r_{i-1}, x_i, x_{i+1}, \cdots, x_n).$$

The values $r_1, \cdots, r_{i-1} \in \mathbb{Z}_q$ were supplied by the *verifier* during the earlier rounds. The value of v_{i-1} was also computed by the *verifier* in the $(i-1)^{th}$ round. The prover sends the polynomial $p_i(x_i)$ to the verifier. Its length cannot be more than a polynomial of the length of ϕ^A, and the degree cannot be more than the number of clauses of ϕ.

The *verifier* does the following computation on receiving the polynomial $p_i(x_i)$.

[6]Note: Let $p_1(x)$ and $p_2(x)$ be two distinct polynomials over $\mathbb{F}_q$, and let $p(x) = p_1(x) - p_2(x)$. The $deg(p) \leq d = \max\{deg(p_1), deg(p_2)\}$. For all $r \in \mathbb{F}_q$, $p_1(r) = p_2(r) \Rightarrow p(r) = 0$. The number of roots of $p(x) = 0$ in $\mathbb{F}_q$ is $\leq d$.
The probability, $\Pr_{r \in \mathbb{F}_q}[p(r) = 0] \leq \frac{d}{q}$. If the degree of the polynomial is small compared to the size of the finite field, the probability for the false positive is low.

(i) Checks the degree of the polynomial. It should not exceed the number of clauses.

(ii) Verifies that, $p_i(0)+p_i(1) = v_{i-1}$ (computed by the verifier in the previous iteration). Note that $v_0 = 0$.

(iii) Selects uniformly at random $r_i \in \mathbb{F}_q$.

(iv) Computes $p_i(r_i) = v_i$.

(v) Sends r_i to the prover.

The verifier rejects the prover's claim, if $p_i(0) + p_i(1) \neq v_{i-1}$.

References

[OD] Computational Complexity *A Conceptual Perspective* by *Oded Goldreich*, Pub. Cambridge University Press, 2008, ISBN 978-0-521-88473-0.

[MS] *Theory of Computation* by Michael Sipser, (3rd. ed.) Pub. Cengage Learning, 2013, ISBN 978-81-315-2529-6a .

[SABB] *Computational Complexity, A Modern Approach* by Sanjeev Arora & Boaz Barak, Pub. Cambridge University Press, 2009, ISBN 978-0-521-42426-4.