Lecture 7 — Interactive Proofs & the PCP Theorem

> Course **2-INF-135/15 Pravdepodobnostné algoritmy**, LS 2025/26. Source slides: `RA_slidy_IPaPSPACEPCPstart_tlac.pdf` (38 p., IP = PSPACE + start of PCP), `RA_PCP_komplet.pdf` (42 p., the full PCP proof), book chapter `PCP_kniha.pdf` [Crescenzi–Kann style text] (46 p., ch. 7 "The PCP theorem"). Štátnicové syllabus topics covered: *interactive proofs and IP = PSPACE; the sum-check protocol and arithmetization; the PCP theorem NP = PCP[O(log n), O(1)]; the two readings of it (locally checkable proofs and hardness of approximation); the proof via linear / low-degree encodings and the composition lemma; inapproximability of Max-3-SAT and of the maximum independent set / clique.*

The one-paragraph map of the whole lecture

There are two big theorems here, and they are siblings: both replace a *combinatorial* check ("is this formula satisfiable?") by an *algebraic* one ("do these polynomials agree?").

The first is **IP = PSPACE**: a polynomial-time, coin-flipping **verifier** chatting with an all-powerful but untrusted **prover** can be convinced of exactly the PSPACE statements. The engine is **arithmetization** — turn a Boolean formula into a polynomial — plus the **sum-check protocol**, where the verifier never computes a giant sum itself but forces the prover to "open up" the sum one variable at a time, spot-checking each step at a random point.

The second, the centrepiece, is the **PCP theorem**: $\mathrm{NP} = \mathrm{PCP}[\,O(\log n),\, O(1)\,]$**.** Every NP statement has a (polynomially long) **written proof** that a verifier can check by tossing $O(\log n)$ coins and **reading only a constant number of its bits** — yet still catch any false "proof" with probability $\geq \frac{1}{2}$. Read one way, this is astonishing: correctness becomes a *local* property. Read the other way — the way the exam cares about — it says **approximating** NP-hard optimization problems (Max-3-SAT, maximum clique, maximum independent set) is itself NP-hard. The proof builds proofs you can check locally out of three ingredients: **linear-function encodings** (giving long, exponential proofs), **low-degree-polynomial encodings** (giving short, polynomial proofs with polylog queries), and a **composition lemma** that glues a long-but-few-queries verifier inside a short-but-many-queries one to get the best of both: logarithmic randomness *and* constant queries.

Part A — Interactive proofs and IP = PSPACE

A.1 What an interactive proof is

Picture a **student** (the prover P) trying to convince a **teacher** (the verifier V) that "$x \in L$." It is an exam by conversation: they exchange messages, and **the teacher has the last word.**

- **Verifier V** — a *probabilistic, polynomial-time* algorithm. It tosses **private** coins (the prover does **not** see V's random bits) and sends/receives polynomially many messages of polynomial length.
- **Prover P** — *unbounded* computational power, but **untrusted**. It will say anything to be believed.

The pair (P, V) is an **interactive protocol** for L if:

$$x \in L \;\Rightarrow\; \exists P:\ \Pr[\,V(P,x) = 1\,] = 1 \qquad \text{(completeness)}$$

$$x \notin L \;\Rightarrow\; \forall P:\ \Pr[\,V(P,x) = 1\,] \le \tfrac{1}{2} \qquad \text{(soundness)}$$

> **Intuition.** If the statement is true, *some* honest prover convinces V every time. If it is false, then *no* prover — however clever or malicious — fools V more than half the time; any false "proof" gets caught by V's coins with probability $\ge \frac{1}{2}$. The private coins are the teacher's secret: the student must commit to answers before learning what the teacher will check.

$\mathbf{IP}$ is the class of languages with such a protocol. The $\frac{1}{2}$ is cosmetic — repeat and take majority to push the error to 2^{-k}.

A.2 The easy half: IP ⊆ PSPACE

A PSPACE machine can simulate the *whole game tree of conversations* without writing it down all at once. The verifier uses $r(n)$ random bits and the prover sends messages of length $m(n)$. We want to know: $\forall r\ \exists m:\ V^{(r)}(P(m), x) = 1$? More precisely, the optimal prover maximizes V's acceptance probability, and that maximum is $\mathbf{max}_{\text{prover strategy}}\ \Pr_r[\,V \text{ accepts}\,]$. Polynomial space suffices to recurse over all message sequences $m \in \{0,1\}^{m(n)}$ and average over all coin sequences $r \in \{0,1\}^{r(n)}$ — exponentially many, but explored one branch at a time, reusing space. So $\mathbf{IP} \subseteq \mathbf{PSPACE}$.

A.3 The hard half: PSPACE ⊆ IP, via the PSPACE-complete problem TQBF

It suffices to give a protocol for **one** PSPACE-complete problem; everything else reduces to it. That problem is **TQBF** (true quantified Boolean formulas): $B = \forall x_1 \, \exists x_2 \, \forall x_3 \cdots \exists x_n \, \Phi(x_1, \ldots, x_n)$. The whole trick of this part is to **arithmetize** and then run **sum-check**. We warm up on a counting version where the idea is cleanest.

Warm-up: arithmetization and #3SAT$_D$ ∈ IP

Arithmetize a Boolean formula φ — translate logic into a polynomial P_φ over the integers (working mod a prime later):
$\neg x \rightsquigarrow (1 - x), \qquad x \wedge y \rightsquigarrow x \cdot y, \qquad x \vee y \rightsquigarrow 1 - (1 - x)(1 - y)$. For a 3-CNF φ with m clauses, $P_\varphi(X_1, \ldots, X_n) = \prod_{j \le m} p_j(X_1, \ldots, X_n)$, and crucially
$\varphi(x_1, \ldots, x_n) = 1 \iff P_\varphi(X_1, \ldots, X_n) = 1 \quad \text{on } 0/1 \text{ inputs.}$

> **Example.** $\varphi = (x \vee y \vee z) \wedge (\neg x \vee y \vee z) \wedge (x \vee \neg y \vee z)$ gives
> $P_\varphi = (1 - (1 - X)(1 - Y)(1 - Z))(1 - X(1 - Y)(1 - Z))(1 - (1 - X)Y(1 - Z))$.

Now **#3SAT**$_D = \{(\Phi, K) \mid \Phi$ is a 3-CNF with exactly K satisfying assignments$\}$. The number of satisfying assignments is just a **sum of the polynomial over the cube**:
$\#\Phi = \sum_{b_1 \in \{0,1\}} \cdots \sum_{b_n \in \{0,1\}} P_\Phi(b_1, \ldots, b_n)$. So $(\Phi, K) \in$ #3SAT$_D$ iff this sum equals K. The verifier cannot compute a sum over 2^n points — but the **sum-check protocol** lets it verify the claim while reading almost nothing itself.

The sum-check protocol

Setup: a polynomial $g(X_1, \ldots, X_n)$ of total degree d, an integer K, a prime p. We want to verify $K \equiv_p \sum_{b_1 \in \{0,1\}} \cdots \sum_{b_n \in \{0,1\}} g(b_1, \ldots, b_n)$.

The verifier "peels one variable at a time":

- **If $n = 1$:** V just checks $g(0) + g(1) = K$ directly and accepts/rejects.
- **If $n \ge 2$:** V wants the univariate polynomial obtained by summing out *all but the first* variable, $h(X_1) = \sum_{b_2} \cdots \sum_{b_n} g(X_1, b_2, \ldots, b_n)$. V asks P for it. P sends some univariate $s(X_1)$ (claimed to equal h).
- V checks the **consistency at the boundary**: $s(0) + s(1) = K$? If not, **reject**.
- If yes, V picks a **random** point $a \in_R \mathrm{GF}(p)$ and **recurses**: verify that $s(a) = \sum_{b_2} \cdots \sum_{b_n} g(a, b_2, \ldots, b_n)$, i.e. a sum-check on one fewer variable with new target $K' := s(a)$.

> **Why this is sound — the punchline.** Suppose the prover lied, $s \neq h$. Two non-equal univariate polynomials of degree $\leq d$ over $\mathrm{GF}(p)$ agree on at most d points, so a *random* a has $s(a) \neq h(a)$ with probability $\geq 1 - d/p$. But if $s(a) \neq h(a)$, the prover is now stuck having to prove a **new false statement** about $g(a, \cdot)$ — the lie is *pushed down* one level, never erased. Over n rounds, $\Pr[V \text{ rejects a false claim}] \geq \left(1 - \frac{d}{p}\right)^n \geq 1 - \frac{nd}{p}$, which is close to 1 once p is a polynomially long prime. The prover's only escape would be to guess V's random a in advance — impossible with private coins.

This is the whole soul of interactive proofs: **the verifier never does the big computation; it forces the prover to reveal it step by step and audits each step at a random point.**

TQBF ∈ IP — the two complications and their fixes

For $B = \forall x_1 \exists x_2 \cdots \Phi$, arithmetize quantifiers too: negation only on variables ($\neg x \rightsquigarrow 1 - x$), then $\wedge \rightsquigarrow \cdot, \quad \vee \rightsquigarrow +, \quad \exists \rightsquigarrow \sum, \quad \forall \rightsquigarrow \prod$. Then B is true iff its arithmetization $A = \prod_{b_1} \sum_{b_2} \cdots \sum_{b_n} P_\Phi(b_1, \ldots, b_n) \neq 0$, and we run a sum-check-style protocol (now mixing $\sum$ for ∃ and $\prod$ for ∀), peeling the **leftmost** quantifier each round, turning the bound variable into a free one and sending a univariate polynomial $g(z)$ with $g(0)\, J\, g(1) = K$, where $J \in \{+, \cdot\}$ is the operator being peeled.

Two things can go wrong, and the fixes are the examinable insight:

1. **The numbers explode.** A product of n factors can reach value $O(2^{2^n})$ — doubly exponential, too big to send. **Fix:** work **modulo a prime** p of polynomial bit-length. Such a p exists with $A \not\equiv_p 0 \iff B$ true: if primes $p_1, \ldots, p_m$ each divided A then their product would divide A, but $p_1 \cdots p_m = \Omega(2^{2^{nd}})$ while $A = O(2^{2^n})$ — a contradiction once we take enough primes. The prover sends p together with a **certificate of primality**.
2. **The degree explodes.** Each $\prod$ over a variable can double the degree of remaining polynomials, so after many ∀'s the univariate messages would have exponential degree. **Fix:** put B into **"simple" form** — between any variable and *its own* quantifier there is at most one ∀. Any QBF can be transformed to simple form with only polynomial growth (re-introduce a fresh copy $x_i \rightsquigarrow x_{j_i}$ after each ∀, enforcing equality $x_i = x_j \rightsquigarrow z_i z_j + (1 - z_i)(1 - z_j)$). For a simple QBF, the degree of the functional-form polynomial grows only **linearly** in $|B|$, so all messages stay polynomial.

> **Correctness bound.** If B is *false*, the verifier accepts with probability at most t/p, where t is the degree of the functional form (linear in $|B|$). Polynomial t, polynomially-long prime $p \Rightarrow$ negligible error. Hence $\mathbf{PSPACE} \subseteq \mathbf{IP}$, and combined with A.2, $\boxed{\mathbf{IP} = \mathbf{PSPACE}.}$

This matters for PCP because it immediately gives one boundary point of the PCP world: $\mathbf{PSPACE} \subseteq \mathbf{PCP}[\mathbf{poly}, \mathbf{poly}]$ (below).

Part B — PCP: definition and meaning

B.1 The definition

A **probabilistically checkable proof** flips the picture from Part A: instead of a *live* conversation, the prover writes down a **static proof string** Π once, and the verifier gets **random access** to it (an oracle) but only **peeks at a few bits**.

> $L \in \mathrm{PCP}[r(n), q(n)]$ if there is a probabilistic polynomial-time verifier V that, on input x, uses $r(|x|)$ **random bits** and reads $q(|x|)$ **bits** of a proof Π, with
> $x \in L \Rightarrow \exists \pi : \ \Pr_r[V(x,r) = 1] = 1 \quad (\mathbf{completeness}),$
> $x \notin L \Rightarrow \forall \pi : \ \Pr_r[V(x,r) = 1] < \tfrac{1}{2} \quad (\mathbf{soundness}).$

The two parameters are **randomness** r and **query complexity** q — and we measure efficiency by how *stingy* the verifier is with both.

A first example — PCP for graph non-isomorphism (GNI)

You are given two n-vertex graphs G_0, G_1 and want to be convinced they are **not** isomorphic. The proof Π is a giant table: for **every** n-vertex graph H, $\Pi(H)$ records which of G_0, G_1 it is isomorphic to (if any): $\Pi(H) = i \quad \text{if } H \cong G_i$**.** The verifier: pick $b \in_R \{0,1\}$, a random permutation τ, form $H = \tau(G_b)$ (a random relabelling of G_b), and **check** $\Pi(H) = b$**.**

> **Why it works.** If $G_0 \ncong G_1$, an honest table answers correctly and V always accepts. If $G_0 \cong G_1$, then $H = \tau(G_b)$ is isomorphic to *both*, so its scrambled form leaks **no information** about which b was used — any table is right with probability exactly $\tfrac{1}{2}$. The randomness hides b ; the verifier reads a single table entry.

B.2 The theorem and its two readings

$$\boxed{\mathrm{NP} = \mathrm{PCP}[\,O(\log n),\, O(1)\,].}$$

Reading 1 — locally checkable proofs. Every NP statement has a polynomial-length proof that can be verified by tossing $O(\log n)$ coins and reading a **constant** number of its symbols. Correctness, normally a *global* property of a proof, can be made **local and spot-checkable**: a wrong proof is wrong "almost everywhere," so a constant-size random sample exposes it.

Reading 2 — hardness of approximation (the exam's favourite). PCP is equivalent to a **gap-producing reduction**:

> **Theorem.** There is a constant $\rho < 1$ such that for every $L \in \mathbf{NP}$ there is a polynomial-time f mapping instances to 3-CNF formulas with
> $x \in L \Rightarrow \mathbf{val}(f(x)) = 1, \qquad x \notin L \Rightarrow \mathbf{val}(f(x)) < \rho,$ where $\mathbf{val}$ is the maximum fraction of simultaneously satisfiable clauses.

The reduction creates a **gap**: satisfiable formulas stay fully satisfiable, unsatisfiable ones become *robustly* unsatisfiable (you cannot even get a ρ-fraction). Consequences:

- **Corollary.** If there is a ρ-approximation algorithm for **Max-3-SAT**, then $\mathbf{P} = \mathbf{NP}$. (Run it on $f(x)$: a value $\geq \rho$ means $x \in L$, $< \rho$ means $x \notin L$ — the gap is exactly what an approximation could not cross.)
- Hence (Kráľovič's inapproximability results): if $\mathbf{P} \neq \mathbf{NP}$, then **Max-3-SAT** $\notin$ **PTAS** and **MaxClique** $\notin$ **PTAS** — no polynomial-time approximation scheme exists. We work the independent-set / clique version out fully in Part D, because two of the practice exams ask for exactly it.

B.3 The easy containments (and the exam-2 problem)

These you can prove by hand; they bracket the theorem and one of them is a standalone exam question.

- $\mathbf{PSPACE} \subseteq \mathbf{PCP}[\mathrm{poly}, \mathrm{poly}]$ — because $\mathbf{IP} = \mathbf{PSPACE}$ (Part A) and an interactive proof with public randomness can be written down as a checkable proof.
- **The proof is never longer than it needs to be:** the only proof positions that can ever be read are those queried for *some* random string, so effectively $|\Pi| \leq q(n) \cdot 2^{r(n)}$.
- $\mathbf{PCP}[r, q] \subseteq \mathbf{NTIME}\left(2^{O(r)} \cdot q\right)$ — a nondeterministic machine **guesses** the (length-$\leq q\,2^r$) proof, then **enumerates all 2^r random strings** and accepts iff V accepts on every one.
- $\mathbf{PCP}[\log n, 1] \subseteq \mathbf{NP}$ — special case: $2^{O(\log n)} = \mathrm{poly}$, so the guessed proof is polynomial and all poly random strings are checkable in polynomial time.
- $\mathbf{PCP}[\log n, \mathrm{poly}] = \mathbf{NP}$ — the matching lower bound; "$\supseteq$" is the harder construction (Part C, short proofs).

We may always assume soundness $\frac{1}{2}$ (amplify to 2^{-c} by repeating c times) and a **non-adaptive** verifier — one that fixes *all* its queries up front as a function of its coins, rather than letting later queries depend on earlier answers. (With a constant number of queries, adaptive vs. non-adaptive makes no difference.)

> **Exam connection (exam-2, Problem 2).** "*Show, without invoking the PCP theorem, that if 3-SAT* $\in \mathbf{PCP}[\frac{1}{2}\log n,\ c]$ *then* $\mathbf{P} = \mathbf{NP}$." The mechanism is the bound above. With $r = \frac{1}{2}\log n$ there are only $2^r = \sqrt{n}$ random strings, and the verifier reads c bits for each, so **only** $\leq c\sqrt{n}$ **proof positions matter**. Each random string ρ imposes one constraint on c of those proof bits; perfect completeness means $x \in \mathbf{3\text{-}SAT}$ iff there is an assignment to those proof bits satisfying **all** $\sqrt{n}$ constraints, while $x \notin \mathbf{3\text{-}SAT}$ leaves every assignment failing more than half of them. So 3-SAT reduces to deciding a tiny constraint system over $O(\sqrt{n})$ Boolean variables — and the point the examiner wants is that the proof has collapsed to polynomial size with a verifier whose coins are *almost gone*, turning the probabilistic check into a deterministic search you can carry out directly. State the key inequality $|\Pi| \leq q \cdot 2^r$, build the constraint system, and argue the gap makes the decision unambiguous.

B.4 The bridge: NP ⊆ PCP[log n, 1] ⟺ 3-SAT has an amplifying reduction

This lemma is how PCP and inapproximability are literally the same statement.

> A polynomial-time f on 3-CNF formulas is a **c-amplifying reduction** (for $c < 1$) if
> $$\mathbf{maxSAT}(\varphi) = 1 \Rightarrow \mathbf{maxSAT}(f(\varphi)) = 1, \qquad \mathbf{maxSAT}(\varphi) < 1 \Rightarrow \mathbf{maxSAT}(f(\varphi)) < c.$$
> Satisfiable stays satisfiable; unsatisfiable becomes "at most a c-fraction satisfiable."

> **Lemma.** $\mathbf{NP} \subseteq \mathbf{PCP}[\log n, 1] \iff$ 3-SAT has an amplifying reduction.

(⇐) reduction ⇒ verifier. Given f, the **proof** is a satisfying assignment of $f(\varphi)$. The verifier picks a **random clause** of $f(\varphi)$ (that costs $O(\log n)$ coins), reads the **3 bits** $\Pi(i), \Pi(j), \Pi(k)$ for its variables, and checks the clause is satisfied. If $\varphi \in$ SAT, some assignment satisfies all clauses ⇒ accept always. If $\varphi \notin$ SAT, at most a c-fraction of clauses are satisfied ⇒ reject probability $\geq 1 - c$; repeat to push acceptance below $\frac{1}{2}$. Constant queries, log randomness. ✓

(⇒) verifier ⇒ reduction. Given a verifier V for SAT using $c\log n$ coins and t queries, construct f: for **each** random string $r \in \{0,1\}^{c\log n}$, the queried positions $i_1, \ldots, i_t$ and the accept predicate define a Boolean function $\varphi'_r(x_{i_1}, \ldots, x_{i_t})$ with $\varphi'_r = 1 \iff V$ accepts on r. Convert each φ'_r into

3-CNF φ_r (auxiliary variables), and set $f(\varphi) := \bigwedge_{r \in \{0,1\}^{c \log n}} \varphi_r$. If $\varphi \in$ SAT, the good proof satisfies all ⇒ $f(\varphi) \in$ SAT. If $\varphi \notin$ SAT, soundness says $\Pr_r[\varphi_r(\alpha) = 0] \geq \frac{1}{2}$ for every assignment α, so with $t' = \max_r(\#\text{clauses in } \varphi_r)$ at least a $\frac{1}{2t'}$-fraction of clauses of $f(\varphi)$ is **unsatisfied** — i.e. $\text{maxSAT}(f(\varphi)) < c$ for $c = 1 - \frac{1}{2t'}$. ✓

Part C — Proving the PCP theorem

C.1 The three-step plan

$$(1)\ \text{long proof} \xrightarrow{\text{linear functions}} (2)\ \text{short proof} \xrightarrow{\text{low-degree polys}} (3)\ \text{composition}$$

1. **Every NP problem has an exponentially long proof checkable with $O(1)$ queries.** Tool: arithmetization of Boolean formulas + **linear functions** and self-correction. Result: $\text{NP} \subseteq \text{PCP}[O(n^3), O(1)]$.
2. **Every NP problem has a polynomially long proof checkable with polylog queries.** Tool: **low-degree polynomials** (two distinct low-degree polynomials agree on few points)
   - low-degree test + sum-check. Result: $\text{NP} \subseteq \text{PCP}[O(\log n), O(\log^4 n)]$.
3. **The composition lemma** glues a verifier *inside* another to combine "log randomness" with "constant queries," landing on $\text{NP} = \text{PCP}[O(\log n), O(1)]$.

The recurring slogans:

- *Arithmetization* reduces satisfiability to an **algebraic** property.
- *Linear / low-degree functions* are **testable** (you can check from a few samples that a table is close to one) and **self-correctable** (you can recover a true value despite errors).
- It is enough to do this for the NP-complete **3-CNF-SAT** — validity there gives validity for all of NP by reduction.

We need one shared definition first.

> **δ-close functions.** For finite sets D, R and $0 < \delta < 1$, functions $f, g : D \to R$ are **δ-close** if they disagree on at most a δ-fraction of inputs: $\Pr_{x \in D}[\, f(x) \neq g(x) \,] \leq \delta$. A **linear function** $f : \mathbb{Z}_2^m \to \mathbb{Z}_2$ satisfies $f(x + y) = f(x) + f(y)$ for all x, y. (Over $\mathbb{Z}_2$, this is the same as a degree-1 polynomial, i.e. $f(x) = a \cdot x$.)

C.2 The long proof: NP ⊆ PCP[O(n³), O(1)]

C.2.1 The linearity test (Blum–Luby–Rubinfeld) and self-correction

> **Lemma (closeness to linear).** Let $\delta < \frac{1}{3}$ and $g : \mathbb{Z}_2^m \to \mathbb{Z}_2$ with $\Pr_{x,y}[\, g(x+y) \neq g(x) + g(y)\,] \leq \delta/2$. Then there is a **linear** f that is δ-close to g.

The witness is the **majority vote**: $f(x) := \text{the } b \in \mathbb{Z}_2 \text{ for which } \Pr_y[\, g(x+y) - g(y) = b] \geq \frac{1}{2}$. The proof has three moves: (1) f, g are δ-close (else the rarely-violated additivity would be violated too often — contradiction); (2) the vote is overwhelming, $p_a := \Pr_x[f(a) = g(a+x) - g(x)] \geq 1 - \delta$, shown by a two-term expansion $1 - \delta \leq \sum_{z \in \mathbb{Z}_2} \left(\Pr_x[g(x+a) - g(x) = z]\right)^2 \leq p_a$; (3) **linearity of f**: for fixed a, b, applying $p. \geq 1 - \delta$ three times gives $\Pr_x[\, f(a) + f(b) + g(x) = f(a+b) + g(x)\,] \geq 1 - 3\delta > 0 \quad (\delta < \frac{1}{3})$, and since the event $f(a) + f(b) = f(a+b)$ does **not depend on** x, its probability is 0 or 1 — and being > 0, it is 1. So f is linear. □

Program LT (linearity test). Repeat $k = \lceil 2/\delta \rceil$ times: pick random $x, y \in \mathbb{Z}_2^m$, and if $g(x) + g(y) \neq g(x+y)$ return NO; else YES.

- If g is linear ⇒ always YES.
- If g is **not** δ-close to any linear function ⇒ $\Pr[\text{NO}] \geq \frac{1}{2}$.

Program SCF (self-correction). To read the *true* value $f(x)$ from a corrupted table g that is δ-close to linear f: pick random y and return $g(x+y) - g(y)$.

- Returns $f(x)$ with probability $\geq 1 - 2\delta$ (because both y and $x + y$ are uniform, each is a good point except with probability δ, and f linear ⇒ $f(x) = f(x+y) - f(y)$).

> **Why self-correction, not just "read $g(x)$"?** Reading $g(x)$ directly errs with probability $\leq \delta$ — *better* than 2δ. But SCF **randomizes the access**: the point actually queried, $x + y$, is uniformly random, so different corrected reads are (nearly) independent. That independence is what lets us union-bound over several reads in the consistency and satisfiability tests.

C.2.2 Arithmetization for the long proof

Turn 3-SAT φ into a degree-3 polynomial P_φ over $\mathbb{Z}_2$: $\text{literal } u \mapsto p_u = 1 - x_u, \quad \neg u \mapsto p_u = x_u, \quad \text{clause } C = l_1 \vee l_2 \vee l_3 \mapsto P_C = p_{l_1} p_{l_2} p_{l_3}$, $\varphi = C_1 \wedge \cdots \wedge C_m \mapsto P_\varphi = \sum_{i=1}^m P_{C_i}$. Now $P_C(a) = 0$ exactly when clause C is satisfied, so $\varphi(a) = 1 \Rightarrow P_\varphi(a) = 0$, but the converse fails — $P_\varphi(a)$ counts (mod 2) the *parity* of unsatisfied

clauses, so an even number of failures hides itself. **Fix with a random combination:** for $r \in \mathbb{Z}_2^m$, $P_\varphi^r = \sum_{i=1}^m r_i P_{C_i}$. Using the fact that for $v \neq 0$, $\Pr_r[\sum_i r_i v_i = 1] = \frac{1}{2}$:
$\varphi(a) = 1 \Rightarrow \forall r:\ P_\varphi^r(a) = 0; \qquad \varphi(a) = 0 \Rightarrow \Pr_r[P_\varphi^r(a) = 1] = \frac{1}{2}$. So a random r catches an unsatisfying assignment with probability $\frac{1}{2}$. (Picking a random *input* a instead wouldn't help — we'd have to test all a.)

The key structural theorem lets us **evaluate any degree-3 polynomial using three linear functions of the assignment**:

> **Theorem.** For $a = (a_1, \ldots, a_n) \in \mathbb{Z}_2^n$ there exist three **linear** functions $A_a : \mathbb{Z}_2^n \to \mathbb{Z}_2$, $B_a : \mathbb{Z}_2^{n^2} \to \mathbb{Z}_2$, $C_a : \mathbb{Z}_2^{n^3} \to \mathbb{Z}_2$, such that **every** degree-3 polynomial p over n variables satisfies $p(a_1, \ldots, a_n) = \alpha_p + A_a(q_{p,1}) + B_a(q_{p,2}) + C_a(q_{p,3})$, where α_p and the index-vectors $q_{p,i}$ depend only on p and are polynomial-time computable.

Here $A_a(x) = \sum_i a_i x_i$ evaluates linear terms, $B_a(y) = \sum_{i,j} a_i a_j y_{ij}$ the quadratic terms, $C_a(z) = \sum_{i,j,k} a_i a_j a_k z_{ijk}$ the cubic terms; and $q_{p,i}$ is the characteristic vector of which monomials appear in p. The clever part: A_a, B_a, C_a depend only on the **assignment** a (so they can be *pre-tabulated as the proof*), while the query vectors $q_{p,i}$ depend only on the **polynomial** p (so the verifier computes them itself).

C.2.3 The verifier (Π = three linear-function tables)

The proof is a concatenation $\Pi = A'\,B'\,C'$ where $|A'| = 2^n$, $|B'| = 2^{n^2}$, $|C'| = 2^{n^3}$ — the full value-tables of A_a, B_a, C_a for the (claimed) satisfying a. The verifier runs three checks:

1. **Linearity** (Program LT on each of A', B', C'): each is δ-close to *some* linear function — else caught with probability $\geq \frac{1}{2}$.
2. **Consistency** (Program CT): even if all three are linear, they must come from the **same** a, i.e. $\tilde{b}_{(i-1)n+j} = \tilde{a}_i \tilde{a}_j$ and $\tilde{c}_{(i-1)n^2+(j-1)n+k} = \tilde{a}_i \tilde{a}_j \tilde{a}_k$. CT picks random x, x', uses SCF to read $a = A'(x)$, $a' = A'(x')$, $b = B'(x \circ x')$ (where $(x \circ x')_{(i-1)n+j} = x_i x'_j$), and checks $a \cdot a' = b$; similarly checks $A' \cdot B'$ vs C'. (SCF's randomization is what makes the three reads independent enough to bound the error.)

   > **Lemma.** For $\delta < \frac{1}{24}$ there is a constant k so that if **no** single a makes A', B', C' δ-close to the linear functions with coefficients $a,\ a \circ a,\ a \circ a \circ a$, then one of k runs of LT/CT returns NO with probability $\geq 1 - \delta$.

3. **Satisfiability** (Program CSAT): pick random $r \in \mathbb{Z}_2^m$, compute $\alpha_{P_\varphi^r}$ and the query vectors, use SCF to read $a = A'(q_1)$, $b = B'(q_2)$, $c = C'(q_3)$, and check whether $\alpha + a + b + c = 1$ (i.e. $P_\varphi^r(a) = 1$, a *violation*). If so return NO. An unsatisfying a is caught with probability $\geq \frac{1}{2}$ by

the random r.

Each test uses $O(n^3)$ random bits (a 3-CNF has $\le n^3$ clauses) and $O(1)$ queries; repeat a constant number of times. Therefore $\boxed{\mathbf{NP} \subseteq \mathbf{PCP}[O(n^3), O(1)].}$ The proof is **exponentially long** (tables of size up to 2^{n^3}) — which is the drawback we fix next.

C.3 The short proof: NP ⊆ PCP[O(log n), O(log^4 n)]

The same three-part skeleton (testable encoding → arithmetize → assemble), but the linear functions are replaced by **low-degree polynomials over a larger field**, which encode the same information in **polynomial** length.

Parameters (worth memorizing the scale)

symbol	meaning	value
n	# Boolean variables	≥ 3
q	a prime	$\approx 100\lceil \log^4 n \rceil$
$F = \mathbb{Z}_q$	finite field	$
$H \subseteq F$	subset ${0,\dots,	H
k	# variables of the polynomials	$\approx \log n / \log\log n$
d	total degree	$O(k
$\mathcal{F}_{d,k}$	k-variate polynomials of degree d	—

C.3.1 Why polynomials encode well — the agreement lemma

> **Lemma (Schwartz–Zippel-style).** Two distinct polynomials in $\mathcal{F}_{d,k}$ agree on at most $d\,q^{k-1}$ of the q^k points of F^k; equivalently a nonzero degree-d polynomial has $\le dq^{k-1}$ roots.

So the **agreement fraction** is $\le d/q < \frac{1}{2}$. Consequently a function that is δ-close (for $\delta < \frac{1}{4}$) to *some* low-degree polynomial is close to a **unique** one: polynomials make good error-correcting **codes**.

Low-degree extension (the encoding). A satisfying assignment is a function $a : H^k \to \{0,1\}$ (we use $|H^k| \ge n$, so k-tuples of H index the n bits). Its encoding is the unique low-degree polynomial agreeing with it on H^k:

> **Theorem.** For $f : H^t \to \{0,1\}$ there is a **unique** $p_f \in \mathcal{F}_{t|H|,t}$ with $p_f(y) = f(y)$ for all $y \in H^t$, namely $p_f(x) = \sum_{h \in H^t} S_h(x) f(h)$, where the **selector** S_h is 1 at h and 0 on the rest of H^t.

This encodes n bits as q^k field elements — a **polynomial** blow-up (vs. 2^n before). A correct proof is exactly *a low-degree polynomial encoding a satisfying assignment*.

C.3.2 The low-degree test and its correction

A polynomial restricted to any **line** $\ell_{b,s} = \{b + st \mid t \in F\}$ is a univariate degree-d polynomial — and conversely: $g \in \mathcal{F}_{d,k} \iff \forall b, s :\ g_{b,s}(t) = g(b + st) \in \mathcal{F}_{d,1}$.

Program LDT. With an auxiliary **line-table** $T : F^{2k} \to F^{d+1}$ supplying, for each line, the coefficients of the best-fitting univariate degree-d polynomial $P_{b,s}$: repeat $\lceil 3/\delta \rceil$ times — pick a random line (b, s) and random point t, and reject if $P_{b,s}(t) \neq g(b + st)$.

- $g \in \mathcal{F}_{d,k}$ ⇒ a line-table makes LDT always accept.
- g not δ-close to any degree-d polynomial ⇒ for every T, LDT rejects with probability $\geq \frac{3}{4}$. (Relies on a deep but here-omitted theorem: high *success rate* on random lines ⇒ globally close to a low-degree polynomial.)
- Cost: $O(k \log q)$ random bits, $O(1)$ queries to g and T.

Program CLDP (correction, the analogue of SCF): to read $f(x)$, pick a random line through x, check the table agrees at a random point, and return $P_{x,s}(0)$. It returns the true $f(x)$ with probability $\geq 1 - 2\sqrt{\delta} - d/q$.

C.3.3 Arithmetization revisited — characteristic functions and zero-testers

As before, each clause becomes one of four degree-3 monomials $p_0, \ldots, p_3$ (indexed by how many variables are negated; assume negated variables first and indices increasing). Define **clause-characteristic functions** $\chi_\varphi^j : H^{3k} \to \{0,1\}$:
$\chi_\varphi^j(i_1, i_2, i_3) = 1 \iff \varphi \text{ has a type-}j \text{ clause on } u_{i_1}, u_{i_2}, u_{i_3}$. Then a satisfies φ iff for all j and all (i_1, i_2, i_3), $f_\varphi^j(i_1, i_2, i_3) = \chi_\varphi^j(i_1, i_2, i_3) \cdot p_j(a_{i_1}, a_{i_2}, a_{i_3}) = 0$. Replace χ_φ^j and p_j by their low-degree extensions g_φ^j. Satisfaction becomes: $g_\varphi^j \equiv 0$ **on all of** H^{3k}. Note χ_φ^j depends only on φ, so the verifier can compute it — each query to g_φ^j becomes **3 queries to** f_a.

Zero-testers — turn "identically zero" into "a sum is zero."

> **Lemma.** There is a family R of q^{3k} polynomials (zero-testers) in $\mathcal{F}_{3k|H|,3k}$ such that for any $f : H^{3k} \to F$ **not** identically zero, $\Pr_{R \in \mathcal{R}}\left[\sum_{h \in H^{3k}} R(h) f(h) = 0\right] \leq \frac{3}{100}$. Constructible in time $q^{O(k)} = \text{poly}(n)$.

Idea: form $g(t_1,\dots,t_{3k}) = \sum_h f(h)\prod_i t_i^{h_i}$; then $g \equiv 0 \iff f \equiv 0$ on H^{3k}, and by the agreement lemma a nonzero g vanishes on $\le 3|H|k/q < \frac{3}{100}$ of F^{3k}. Each $R_b(x) = s(b,x)$ has $\sum_h R_b(h)f(h) = g(b)$, so a random b (i.e. random zero-tester) catches a nonzero f.

C.3.4 The sum-check (again!), and assembling the short verifier

Checking $g_\varphi^j \equiv 0$ now reduces to checking a **single sum** $\sum_{h\in H^{3k}} R(h)g_\varphi^j(h) = 0$, done by **sum-check** — the same peel-one-variable idea as Part A, but now over the field F and the cube H^{3k}. Define partial-sum polynomials $g_i(x_1,\dots,x_i) = \sum_{y_{i+1},\dots,y_{3k}\in H} f(x_1,\dots,x_i,y_{i+1},\dots,y_{3k})$, which satisfy $g_i(x_1,\dots,x_i) = \sum_{x\in H} g_{i+1}(x_1,\dots,x_i,x)$ and $\sum_h f(h) = \sum_{x_1\in H} g_1(x_1)$.

Program Sum-Check (with table T of the partial-sum polynomials): check $\sum_{x\in H} g_1'(x) = 0$; then for $i = 2..3k$, pick random $r_i \in F$ and check $\sum_{x\in H} g_i'(x) = g_{i-1}'(r_{i-1})$; finally check $f(r_1,\dots,r_{3k}) = g_{3k}'(r_{3k})$.

- correct f summing to 0 ⇒ accepts; nonzero sum ⇒ rejects with prob $\ge \frac{3}{4}$ (each round a wrong partial polynomial is exposed at the random r_i with prob $\ge 1 - d/q$, and the lie propagates).
- Cost: $O(k\log q)$ random bits, one value of f and $3k(d+1)$ rows of T.

The short verifier. Proof = the low-degree extension f_a, its line-table T_a, and the partial-sum tables T_0, T_1, T_2, T_3 (one per clause type, for the product with a zero-tester). The verifier (1) runs LDT to check f_a is close to a degree-$\le d$ polynomial; (2) for each j, picks a random zero-tester R and runs Sum-Check on $R\cdot g_\varphi^j$ (substituting 3 queries to f_a for each query to g_φ^j, using CLDP to read corrected values). Counting:

- randomness: $O(k\log q) = O(\log n)$;
- queries: $O(1)$ values of f_a (length $O(\log q) = O(\log\log n)$ each) + $O(1)$ entries of T_a (length $O(k|H|\log q) = O(\log^2 n)$) + $O(\log^2 n)$ rows of T_j (length $O(\log^2 n)$) ⇒ $O(\log^4 n)$ bits total.

$$\boxed{\mathrm{NP} \subseteq \mathrm{PCP}[O(\log n), O(\log^4 n)].}$$

C.4 The composition lemma — getting both at once

We now have two verifiers: one with **constant queries but polynomial randomness** (long proof), one with **logarithmic randomness but polylog queries** (short proof). Composition plugs the first *inside* the second to inherit the good parameter from each.

> **Composition lemma (informal).** If $\mathrm{NP} \subseteq \mathrm{PCP}[O(r_1), O(q_1)]$ **and** there is an (r_2, q_2)-**normal-form** verifier for 3-SAT, then

> $\mathbf{NP} \subseteq \mathbf{PCP}\big[O(r_1(n) + r_2(kq_1(n))),\ O(q_2(kq_1(n)))\big], \quad k \text{ constant.}$

Why a "normal form" is needed. The composed verifier wants to check "A_1 would accept after reading these d_1 words of Π_1" — but it must **not read those words itself** (that would re-introduce the large query count). The escape: have a second verifier A_2 check this, where A_2 has access to a **split encoding** of the proof (an encoding of *each row* read by A_1, not of the whole proof). A **normal-form verifier** is exactly one wired to operate on such an encoding:

> **(r, q)-normal form (sketch).** It has an $(\ell, k \cdot q)$-**encoding scheme** E (codewords are tables, with minimum distance $\delta_{\min}$, $\ell(n) \le 2^{hr(n)}$), uses exactly $hr(n)$ random bits, reads exactly d rows (a constant) whose indices depend only on the coins, and behaves as: if the c encoding-tables are codewords decoding to a satisfying assignment ⇒ accepts w.p. 1; if some table is **far** ($\ge \delta_{\min}/3$) from any codeword ⇒ accepts w.p. $< \frac{1}{2}$; if all tables are close but decode to a non-satisfying assignment ⇒ accepts w.p. $< \frac{1}{2}$.

Proof idea of the lemma. Let A_1 witness $\mathbf{NP} \subseteq \mathbf{PCP}[O(r_1), O(q_1)]$ (on input of length n, uses $h_1 r_1(n)$ coins, reads d_1 words of length $k_1 q_1(n)$). For each random string r, let $y_1^r, \dots, y_{d_1}^r$ be the words A_1 reads and define $L' = \{(y_1, \dots, y_{d_1}) \mid A_1 \text{ accepts when it reads them}\}$. A_1 runs in polynomial time, so $L' \in \mathbf{P}$; let $\Phi_{A_{L'}}$ be the 3-CNF from Cook–Levin for it. Because there is a normal-form verifier A_2 for 3-SAT, we can attach to each random string r a sub-proof Π_2^r that A_2 accepts (with the encoded rows $E_2(y_1^r), \dots, E_2(y_{d_1}^r)$ as oracle) iff $(y_1^r, \dots, y_{d_1}^r) \in L'$. The **composed proof table** has two parts: the Π_2^r's (first part) and the E_2-encodings of the rows of Π_1 (second part). Its dimensions multiply out to $\ell(n) \le 2 \cdot 2^{h_1 r_1(n)} \cdot 2^{h_2 r_2(k_1 q_1(n))}$ rows.

The composed verifier A: simulate A_1 with coins r to get the row indices; then simulate A_2 with coins r', oracle the encoded rows + Π_2^r; accept iff A_2 accepts. Total randomness $h_1 r_1(n) + h_2 r_2(k_1 q_1(n))$.

- **Completeness:** $\varphi \in$ SAT ⇒ honest Π_1 makes A_1 accept always ⇒ for every r the encoded rows + Π_2^r make A_2 accept w.p. 1 ⇒ A accepts w.p. 1.
- **Soundness:** $\varphi \notin$ SAT ⇒ decoding the second part gives a table Π' with $\Pr[A_1 \text{ accepts } \Pi'] < \frac{1}{2}$, so A_1 rejects w.p. $> \frac{1}{2}$. For a rejecting r, the read rows $\notin L'$, so A_2 rejects w.p. $> \frac{1}{2}$. Hence $\Pr[A \text{ rejects}] \ge \Pr[A_1 \text{ rej}] \cdot \Pr[A_2 \text{ rej}] > \frac{1}{4}$ — amplify below $\frac{1}{2}$. □

The five-step assembly (this is the actual proof of the theorem):

1. Composition lemma (above).
2. $\mathbf{NP} \subseteq \mathbf{PCP}[O(\mathrm{poly}(n)), O(1)]$ — the long proof (C.2), put in normal form (a $(\mathrm{poly}(n), 1)$-normal-form verifier exists).
3. $\mathbf{NP} \subseteq \mathbf{PCP}[O(\log n), O(\mathrm{polylog}(n))]$ — the short proof (C.3), put in normal form (a $(\log n, \mathrm{polylog}(n))$-normal-form verifier exists).

4. Compose 1+3+3: $\mathbf{NP} \subseteq \mathbf{PCP}[O(\log n), O(\mathbf{polyloglog}(n))]$, since $\log(k\,\mathbf{polylog}\,n) = O(\log n)$ and $\mathbf{polylog}(k\,\mathbf{polylog}\,n) = \mathbf{polyloglog}(n)$.
5. Compose 1+2+4: $\mathbf{NP} \subseteq \mathbf{PCP}[O(\log n + \mathbf{poly}(k\,\mathbf{polyloglog}\,n)), O(1)]$, and since the randomness is still $O(\log n)$, $\boxed{\mathbf{NP} = \mathbf{PCP}[O(\log n), O(1)].}$

> **The punchline of composition.** Step 4 shrinks queries from polylog to polyloglog while keeping log randomness; step 5 shrinks them to **constant** at the cost of only $O(\log n)$ extra randomness. Each composition "spends a little randomness to buy back a lot of queries," and the parameters were chosen so the randomness never escapes $O(\log n)$.

Part D — The payoff: inapproximability (exam-1 P4, exam-3 P4)

This is what the PCP theorem is *for*, and two practice exams ask it directly: **approximating the maximum independent set is NP-hard.** The bridge is the **FGLSS graph**.

> **Theorem.** There is a constant $0 < \rho < 1$ such that, **if** $\mathbf{P} \neq \mathbf{NP}$, no polynomial-time algorithm can, for every graph G, output an independent set of size $\geq \rho \cdot \alpha(G)$ (where $\alpha(G)$ is the maximum independent-set size).

Construction. Take an NP-complete L and its PCP verifier V ($r = O(\log n)$ coins, $q = O(1)$ queries, completeness 1, soundness $< \frac{1}{2}$). Build a graph H_x:

- **Vertices** = **accepting local views**: pairs (ρ, a) where ρ is a random string and a is an assignment to the q proof bits V reads on ρ such that V **accepts**. There are $\leq 2^r \cdot 2^q = \mathbf{poly}(n)$ vertices (since $r = O(\log n)$, $q = O(1)$).
- **Edges** connect **conflicting** views: (ρ, a) and (ρ', a') are adjacent if they are inconsistent — they assign **different values to some proof position they both query** (also, two views with the same ρ conflict, so an independent set picks ≤ 1 view per random string).

Key facts.

- A proof Π yields an independent set: for each ρ on which V accepts, take the view (ρ, a) matching Π. These are mutually consistent (all agree with Π) ⇒ independent. Its size = number of accepted random strings.
- $x \in L$**:** the honest Π is accepted on **all** 2^r strings ⇒ $\alpha(H_x) \geq 2^r$.

- $x \notin L$**:** any independent set is a set of mutually consistent accepting views, i.e. it defines a partial proof accepted on each chosen ρ; soundness says no proof is accepted on $\geq \frac{1}{2}$ of strings $\Rightarrow \alpha(H_x) < 2^r/2$.

So $\alpha(H_x)$ jumps by a factor 2 between yes- and no-instances. A polynomial-time algorithm approximating α within a factor better than 2 would **cross the gap** and decide $L \Rightarrow \mathbf{P} = \mathbf{NP}$. Repeating the verifier (or graph products) pushes the gap to any constant, giving inapproximability within every constant $\rho < 1$.

> **Exam framing.** State the vertex set (ρ, q), define the conflict edges precisely, then prove the two bounds: $x \in L \Rightarrow \alpha(H_x) \geq 2^r$ (one consistent view per string) and $x \notin L \Rightarrow \alpha(H_x) < 2^r/2$ (soundness). Conclude that a ρ-approximation separates the cases. Clique is the complement: $\alpha(G) = \omega(\bar{G})$, so the same gap proves MaxClique inapproximable.

Closing themes table

Theme	Where it appears	The one-line idea
Arithmetization	IP=PSPACE; both PCP proofs	Logic → polynomials; satisfiability → an algebraic identity.
Sum-check / peel-a-variable	#3SAT$_D$, TQBF, short PCP	Don't compute the giant sum; make the prover open it one variable at a time, audit at a random point.
Random spot-check of a lie	sum-check, LT, LDT	Two distinct low-degree polynomials agree rarely ⇒ a random point exposes a wrong one.
Testability	linearity test, low-degree test	From $O(1)$ samples, decide if a table is *close* to a (linear / low-degree) function.
Self-correction	SCF, CLDP	Recover a true value from a corrupted-but-close table — *and* randomize the access so reads are independent.
Encoding = error-correcting code	linear (long), low-degree (short)	A "proof" is a codeword; being wrong is being far from every codeword, hence locally visible.
Composition	final assembly	Spend a little randomness to buy back many queries; iterate to reach $O(\log n)$ coins, $O(1)$ queries.
Gap = inapproximability	Part D, exams	PCP soundness $< \frac{1}{2}$ becomes a factor-2 gap in $\alpha(G)$/Max-3-SAT ⇒ approximation is NP-hard.
**$	\Pi	\le q\cdot 2^r$**

Exam connections recap

- **exam-2 Problem 2** (PCP$[\frac{1}{2}\log n, c] \Rightarrow$ P=NP): use $|\Pi| \le q \cdot 2^r$; with $r = \frac{1}{2}\log n$ only $O(\sqrt{n})$ proof bits matter and the verifier nearly derandomizes — see §B.3.
- **exam-1 Problem 4 / exam-3 Problem 4** (independent-set hardness via PCP): the FGLSS graph H_x with vertices = accepting views, conflict edges, and the 2^r vs $2^r/2$ gap — see Part D.
- **Definitions to have ready:** PCP$[r, q]$ (completeness 1, soundness $< \frac{1}{2}$); IP (private coins, completeness/soundness); the gap statement of the PCP theorem; the amplifying-reduction equivalence (§B.4).