אוניברסיטת
אריאל
בשומרון

Computational Complexity

Lecturer: Edward A. Hirsch

https://edwardahirsch.github.io/edwardahirsch

edward.a.hirsch@gmail.com

January 14, 2026

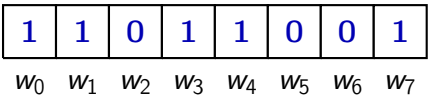
Lecture plan:

- The PCP Theorem and hardness of approximation.
  - Statement of the theorem.
  - Approximation algorithms.
  - Inapproximability as a corollary to the PCP Theorem.
  - Proof of the theorem.
- More examples of interactive protocols.

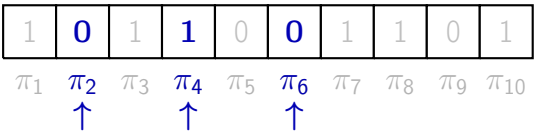
Probabilistically Checkable Proofs (PCP)

Key idea:

For SAT, a proof is a satisfying assignment $w \in \{0,1\}^n$.
One needs to read all its bits to verify it.



We are looking for a proof π (in a different format) that we do not need to read.
It is enough to inspect $O(1)$ bits of such a proof in order to verify it with high prob.



Probabilistically Checkable Proofs (PCP)

Language L, input x of length n.

Prover (actually, a $\exists$ quantifier): claims that $x \in L$,
publishes a proof π for x (does not send it as we do not want to read it).

Verifier (notation: $V^{\pi}(x)$): accepts or rejects (x, π), can use $\rho(n)$ random bits, polynomial time $p(n)$, can access $q(n)$ bits of π by index: asks about bit i, gets $\pi[i]$ instantly .

There is no other communication, Prover cannot change the published proof.

Probabilistically Checkable Proofs (PCP)

Language L, input x of length n.

Prover (actually, a $\exists$ quantifier): claims that $x \in L$,
publishes a proof π for x (does not send it as we do not want to read it).

Verifier (notation: $V^{\pi}(x)$): accepts or rejects (x, π), can use $\rho(n)$ random bits, polynomial time $p(n)$, can access $q(n)$ bits of π by index: asks about bit i, gets $\pi[i]$ instantly .

There is no other communication, Prover cannot change the published proof.

Definition

$L \in \mathbf{PCP}(\rho(n), q(n))$, if there is verifier V (as above) such that $\forall x$

(completeness) $x \in L \Rightarrow \exists \pi \;\; \Pr\{V^{\pi}(x) = 1\} = 1$,

(soundness) $x \notin L \Rightarrow \forall \pi' \;\; \Pr\{V^{\pi'}(x) = 1\} < \frac{1}{2}$.

By definition, $\mathbf{NP} = \mathbf{PCP}(0, \mathrm{poly}(n))$, but randomness can help:

Theorem (PCP Theorem; Arora, Lund, Motwani, Sudan, and Szegedy 1992)

$\mathbf{NP} = \mathbf{PCP}(O(\log n), O(1))$.

Stated with perfect completeness. In fact, *three* bits of the proof are enough for $1 - \delta$ vs $\frac{1}{2} + \delta$.

Reminder: Optimization problems, Approximation algorithms

Optimization problem = search problem R + goal function f.

Example

`MAX–SAT`: Given CNF F, find assignment A, goal function $f(F, A)$ = the number of satisfied clauses.

Exact algorithm for maximization problem given x finds an optimal solution s_E, i.e.,

- $(x, s_E) \in R$,
- $f(x, s_E) = \max_{s:(x,s)\in R} f(x, s)$.

ρ-approximation algorithm finds a solution $s_\approx$ s.t.

- $(x, s_\approx) \in R$,
- $f(x, s_\approx) \geqslant \rho \cdot f(x, s_E)$, where s_E is the optimal solution.

Example

Deterministic $\frac{1}{2}$-approximation algorithm for `MAX–SAT`:

```
-- choose the best of:  the all-0 assignment and the all-1 assignment.
```

Random sampling gives $\frac{7}{8}$-approximation if each clause contains at least 3 literals.

Corollary (The PCP Theorem restated)

$\exists \delta > 0$ s.t. $\forall L \in \mathbf{NP}$ there is a poly-time computable reduction $S\colon \{0,1\}^* \to \{\text{CNF}\}$ s.t.

$$x \in L \quad \Rightarrow \quad S(x) \in \texttt{SAT},$$

$$x \notin L \quad \Rightarrow \quad \text{no assignment satisfies a share} \geqslant (1-\delta) \text{ of clauses of } S(x).$$

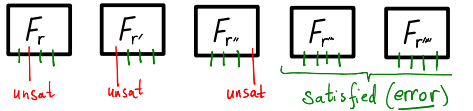
Corollary (The PCP Theorem restated)

$\exists \delta > 0$ s.t. $\forall L \in \mathbf{NP}$ there is a poly-time computable reduction $S\colon \{0,1\}^* \to \{\text{CNF}\}$ s.t.

$$x \in L \quad \Rightarrow \quad S(x) \in \texttt{SAT},$$
$$x \notin L \quad \Rightarrow \quad \text{no assignment satisfies a share} \geqslant (1-\delta) \text{ of clauses of } S(x).$$

Reduction S (input x):

- Convert V to non-adaptive algorithm: if the old version makes c queries, the new version makes all queries it could make irrespectively of the answers ($c' := 2^c$ is still a constant),
- Consider $V(x, r, w)$ as a function of input x, random string $r \in \{0,1\}^{\rho(n)}$, answers $w \in \{0,1\}^{c'}$.
- The input x is fixed, for each r, we get a function $V_{x,r}\colon \{0,1\}^{c'} \to \{0,1\}$.
  Write a constant-size CNF F_r in variables $w_1, \ldots, w_{c'}$ and auxiliary variables s.t.
  $F_r|_{w=w_*}$ is satisfiable iff $V_{x,r}(w_*) = 1$.
  Using truth tables the number of clauses is at most $2^{c'}$.
- Output $\bigwedge_{r \in \{0,1\}^{\rho(n)}} F_r$.



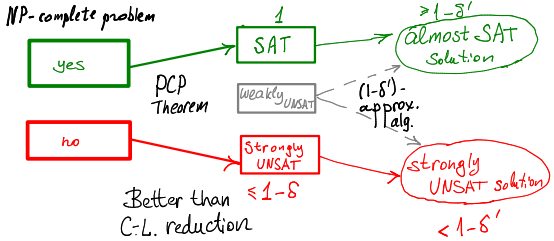
If $x \in L$ and π is a good proof, all F_r are satisfied by π (split into w's).

If $x \notin L$, at least **a half of** r**'s** force $V(x, r) = 0$.

Thus a half of F_r's are UNSAT, every assignment falsifies at least one clause in each F_r. Thus $\delta \geqslant 1/2^{c'+1}$.

Corollary

$\mathbf{P} \neq \mathbf{NP} \Rightarrow \nexists$ poly-time α-approximation algorithm for MAX–SAT with $\alpha > 1 - \delta$.



Assume X is a δ-approximation algorithm.

Poly-time algorithm for SAT (input x):

```
-- Produce CNF F := S(x).
-- If X(F) satisfies a share of > (1 − δ) clauses, accept.
-- Otherwise reject.
```

If $x \in$ SAT, then $X(F)$ satisfies a share of $\geqslant \alpha \cdot 1 > 1 - \delta$.

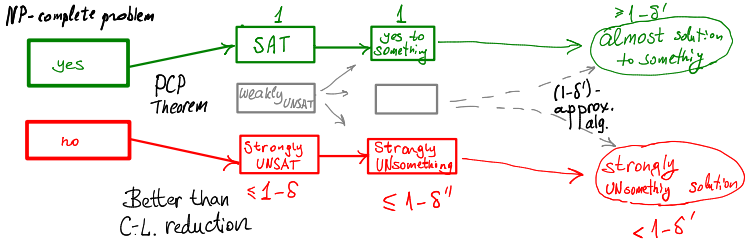
If $x \notin$ SAT, then $X(F)$ satisfies a share of at most $1 - \delta$ (strongly unsatisfiable).

(In)approximability of other problems

MAX-3-SAT

Inapproximability of MAX-3-SAT:

Recall formulas F_r on c' variables.
Convert SAT to 3-SAT (using auxiliary variables), extending it at most c' times.



Fact: For MAX–3–ESAT (all clauses of length exactly 3)
there is a 7/8-approximation, but no $(7/8+\varepsilon)$-approximation algorithm for any constant $\varepsilon > 0$. [Håstad]

MAX-3-SAT can be reduced to MAX-IS so that $\#$ max. satisfied clauses $=|$ max IS size $|$.

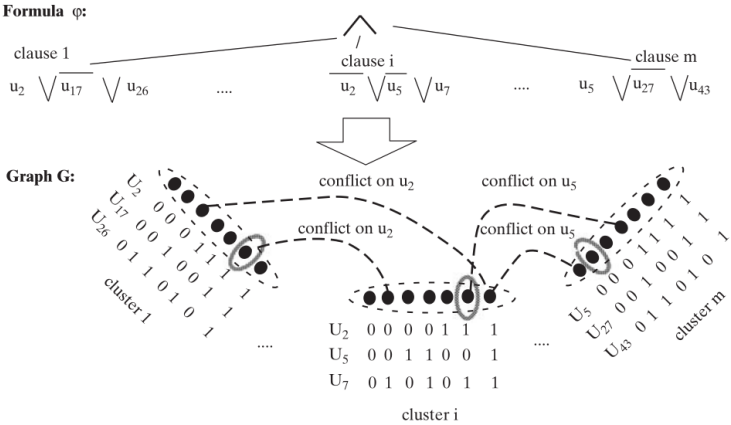


Figure 2.5. We transform a 3*CNF* formula φ with m clauses into a graph G with $7m$ vertices as follows: each clause C is associated with a cluster of 7 vertices corresponding to the 7 possible satisfying assignments to the variables C depends on. We put edges between any two vertices in the same cluster and any two vertices corresponding to *inconsistent* partial assignments. The graph G will have an independent set of size m if and only if φ was satisfiable. The figure above contains only a sample of the edges. The three circled vertices form an independent set.

(In)approximability of other problems

MAX-INDEPENDENT SET: nonconstant inapproximability

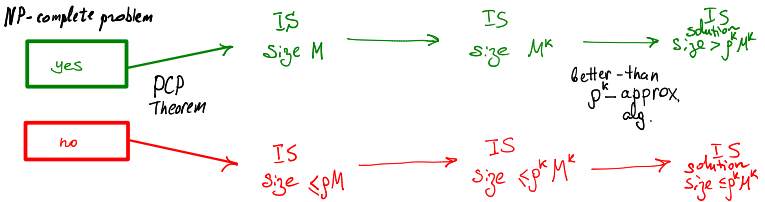
Given $G = (V, E)$, consider new graph G_k:

- $V_k = \{S \subseteq V : |S| = k\}$,
- $(S_1, S_2) \in E_k \iff S_1 \cup S_2$ indep. in G.

$$|\text{ Max IS of } G_k\ | = \binom{|\text{ Max IS of } G\ |}{k}$$

For ρ-smaller solution it would be $\binom{\rho|\text{ Max IS of } G\ |}{k} \sim \rho^k \binom{|\text{ Max IS of } G\ |}{k}$

This is a much worse approximation ρ^k.



(In)approximability of other problems

MIN-VERTEX COVER

Let $G = (V, E)$.

$$| \text{ min vertex cover } | = |V| - | \text{ max independent set } |$$

Same kind of bounds for minimization problems.

NB: terminology may change from α-approximation to $\frac{1}{\alpha}$-approximation.

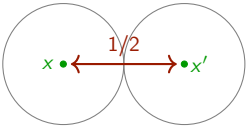
Let us prove the PCP Theorem!

1. The 1st part: $\mathbf{PCP}(\mathrm{poly}(n), O(1))$. ⟵ will prove
2. The 2nd part: $\mathbf{PCP}(\log(n), O(1))$.

Walsh–Hadamard code

- Everything is done modulo 2, scalar product $< x, y >= \sum_{i=1}^{n} x_i y_i \mod 2$.
- The code of $x \in \{0,1\}^n$ is $\mathtt{WH}(x) = (< x, y >)_{y \in \{0,1\}^n} : \{0,1\}^n \to \{0,1\}$ (very long! 2^n bits).
- Error correcting code:

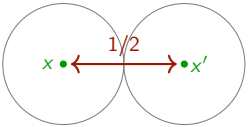
$$\forall x, x' \ \Pr_r\{\mathtt{WH}(x)(r) \neq \mathtt{WH}(x')(r)\} \geqslant \frac{1}{2}.$$



Walsh–Hadamard code

- Everything is done modulo 2, scalar product $<x, y>= \sum_{i=1}^{n} x_i y_i \mod 2$.
- The code of $x \in \{0,1\}^n$ is $\mathtt{WH}(x) = (<x, y>)_{y \in \{0,1\}^n} : \{0,1\}^n \to \{0,1\}$ (very long! 2^n bits).
- Error correcting code:

$$\forall x, x' \ \Pr_r\{\mathtt{WH}(x)(r) \neq \mathtt{WH}(x')(r)\} \geqslant \frac{1}{2}.$$



- Given $f : \{0,1\}^n \to \{0,1\}$, how close is some codeword? What is it?
- Testing for δ-closeness to a codeword (linear function):
  - f, g are called δ-close (for $\delta < 1/2$) if $\Pr_y\{f(y) \neq g(y)\} < \delta$.
  - For $\delta < 1/3$, if f is not δ-close to a linear function $\Rightarrow$

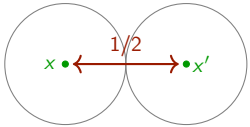
$$\Pr_{y,z}\{f(y+z) \neq f(y) + f(z)\} > \delta/2. \quad \text{[PROOF DELAYED]}$$

  - Repeat $O(1)$ times, get 0.001-closeness with any small error.

Walsh–Hadamard code

- Everything is done modulo 2, scalar product $<x, y>= \sum_{i=1}^{n} x_i y_i \mod 2$.
- The code of $x \in \{0,1\}^n$ is $\mathtt{WH}(x) = (<x, y>)_{y \in \{0,1\}^n} : \{0,1\}^n \to \{0,1\}$ (very long! 2^n bits).
- Error correcting code:

$$\forall x, x' \ \Pr_r\{\mathtt{WH}(x)(r) \neq \mathtt{WH}(x')(r)\} \geqslant \frac{1}{2}.$$



- Given $f : \{0,1\}^n \to \{0,1\}$, how close is some codeword? What is it?
- Testing for δ-closeness to a codeword (linear function):
  - f, g are called δ-close (for $\delta < 1/2$) if $\Pr_y\{f(y) \neq g(y)\} < \delta$.
  - For $\delta < 1/3$, if f is not δ-close to a linear function $\Rightarrow$

$$\Pr_{y,z}\{f(y+z) \neq f(y) + f(z)\} > \delta/2. \quad \text{[PROOF DELAYED]}$$

  - Repeat $O(1)$ times, get 0.001-closeness with any small error.
- Self-correction: for $\delta < \frac{1}{4}$, if f is δ-close to linear $\tilde{f}$, then

$$\forall y \quad \Pr_{y'}\{\tilde{f}(y) = f(y') + f(y+y')\} \geqslant 1 - 2\delta$$

$\mathbf{NP} \subseteq \mathbf{PCP}(\mathrm{poly}(n), O(1))$

Exercise #1: it suffices to show it for one **NP**-complete language.

Exercise #2: solvable systems of quadratic equations mod 2 is **NP**-complete (use equations like $z_{x=y} = xy + (1-x)(1-y)$ to define new variables).

- **NP**-complete language: satisfiable systems of quadratic equations mod 2.
- Input: a system presumably satisfied by $u = (u_1, \ldots, u_n) \in \{0,1\}^n$.

$\mathbf{NP} \subseteq \mathbf{PCP}(\mathrm{poly}(n), O(1))$

Exercise #1: it suffices to show it for one **NP**-complete language.

Exercise #2: solvable systems of quadratic equations mod 2 is **NP**-complete (use equations like $z_{x=y} = xy + (1-x)(1-y)$ to define new variables).

- **NP**-complete language: satisfiable systems of quadratic equations mod 2.
- Input: a system presumably satisfied by $u = (u_1, \ldots, u_n) \in \{0,1\}^n$.
- Proof: $\mathtt{WH}(u)$ and $\mathtt{WH}(u \otimes u)$, where $(y \otimes z)_{ij} = y_i z_j$.

$\mathbf{NP} \subseteq \mathbf{PCP}(\mathrm{poly}(n), O(1))$

Exercise #1: it suffices to show it for one **NP**-complete language.

Exercise #2: solvable systems of quadratic equations mod 2 is **NP**-complete (use equations like $z_{x=y} = xy + (1-x)(1-y)$ to define new variables).

- **NP**-complete language: satisfiable systems of quadratic equations mod 2.
- Input: a system presumably satisfied by $u = (u_1, \ldots, u_n) \in \{0,1\}^n$.
- Proof: $\mathtt{WH}(u)$ and $\mathtt{WH}(u \otimes u)$, where $(y \otimes z)_{ij} = y_i z_j$.
- Verifier:
  1. Test the proof for 0.001-closeness to linear, then use linear functions f and g by self-correction.
  2. Check that f and g are related as presumed. No $\Rightarrow \Pr\{f(r)f(r') \neq g(r \otimes r')\} \geqslant 0.250$: Indeed, $\Pr_r\{rW \neq rW'\} \geqslant \frac{1}{2}$, take random subsum again.
  3. Verify that f satisfies the system: take a random subsum of equations: coefficients a, rhs c; f does not satisfy $\Rightarrow \Pr\{g(a) \neq c\} \geqslant 0.5$.

The delayed proof

For $\delta < 1/3$, f is not δ-close to a linear function $\Rightarrow$

$$\Pr_{y,z}\{f(y+z) \neq f(y) + f(z)\} > \delta/2.$$

The delayed proof

For $\delta < 1/3$, if

$$\Pr_{y,z}\{f(y+z) = f(y) + f(z)\} \geqslant 1 - \delta/2,$$

then f is δ-close to a linear function.
Here is this linear function:

$$\tilde{f}(x) = \operatorname*{maj}_r \left(f(x+r) + f(r)\right).$$

Let $p_x = \Pr_r\{\tilde{f}(x) = f(x+r) + f(r)\}$.
By construction $p_x \geqslant 1/2$. Let us prove $p_x \geqslant 1 - \delta$.

$$\begin{aligned} \Pr_{r,s}\{f(x+r) + f(s) \neq f(x+r+s)\} &\leqslant \delta/2, \\ \Pr_{r,s}\{f(r) + f(x+s) \neq f(x+r+s)\} &\leqslant \delta/2, \text{ i.e.} \end{aligned}$$

$$1 - \delta \leqslant \Pr_{r,s}\{f(x+r) + f(s) = f(x+s) + f(r)\} = \Pr_{r,s}\{f(x+s) + f(s) = f(x+r) + f(r)\} =$$

$$\sum_{b=0}^{1} (\Pr_r\{f(x+r) + f(r) = b\})^2 = p_x^2 + (1-p_x)^2 \leqslant p_x^2 + p_x(1-p_x) = p_x.$$

The delayed proof

For $\delta < 1/3$, if

$$\Pr_{y,z}\{f(y+z) = f(y) + f(z)\} \geqslant 1 - \delta/2,$$

then f is δ-close to a linear function.
Here is this linear function:

$$\tilde{f}(x) = \operatorname*{maj}_r \left(f(x+r) + f(r)\right).$$

Overall, $\Pr_r\{\tilde{f}(x) \neq f(x+r) + f(r)\} < \delta$.

- Prove linearity of $\tilde{f}$:

$$\begin{aligned}
\Pr_r\{\underline{\tilde{f}(x) + \tilde{f}(y)} + f(r) &\neq f(x+r) + \tilde{f}(y)\} < \delta, \\
\Pr_r\{f(x+r) + \tilde{f}(y) &\neq f(x+y+r)\} < \delta, \\
\Pr_r\{f(x+y+r) &\neq \underline{\tilde{f}(x+y)} + f(r)\} < \delta,
\end{aligned}$$

  i.e. $\Pr_r\{\tilde{f}(x+y) = \tilde{f}(x) + \tilde{f}(y)\} > 0$, i.e. $= 1$.

The delayed proof

For $\delta < 1/3$, if

$$\Pr_{y,z}\{f(y+z) = f(y) + f(z)\} \geqslant 1 - \delta/2,$$

then f is δ-close to a linear function.
Here is this linear function:

$$\tilde{f}(x) = \operatorname*{maj}_r \left(f(x+r) + f(r)\right).$$

Overall, $\Pr_r\{\tilde{f}(x) \neq f(x+r) + f(r)\} < \delta$.

- Prove linearity of $\tilde{f}$:

$$\begin{aligned}
\Pr_r\{\underline{\tilde{f}(x) + \tilde{f}(y)} + f(r) &\neq f(x+r) + \tilde{f}(y)\} < \delta,\\
\Pr_r\{f(x+r) + \tilde{f}(y) &\neq f(x+y+r)\} < \delta,\\
\Pr_r\{f(x+y+r) &\neq \underline{\tilde{f}(x+y)} + f(r)\} < \delta,
\end{aligned}$$

  i.e. $\Pr_r\{\tilde{f}(x+y) = \tilde{f}(x) + \tilde{f}(y)\} > 0$, i.e. $= 1$.

- Prove that $f, \tilde{f}$ are δ-close: assume $\Pr_x\{f(x) \neq \tilde{f}(x)\} > \delta$,
  but by construction $\Pr_r\{\tilde{f}(x) = f(x+r) + f(r)\} \geqslant 1/2$, thus
  $\Pr_{x,r}\{f(x) \neq f(x+r) + f(r)\} > \delta/2$, contradiction.

More examples of interactive protocols (**IP**)

Sumcheck protocol

Polynomial g in n variables. Prover claims: $K = \sum_{b_1=0}^{1} \sum_{b_2=0}^{1} \dots \sum_{b_n=0}^{1} g(b_1, \dots, b_n)$

Round for n variables:

`Prover:`

`-- send coefficients of the polynomial` $h(x_1) := \sum_{b_2=0}^{1} \dots \sum_{b_n=0}^{1} g(x_1, b_2, \dots, b_n)$

`Verifier:`

`-- check` $h(0) + h(1) = K$ `(otherwise reject),`

`-- pick and send random` a`,`

Then they recursively check $K' = \sum_{b_2=0}^{1} \dots \sum_{b_n=0}^{1} g'(b_2, \dots, b_n)$

for $(n-1)$-variable polynomial $g' = g|_{x_1=a}$ and $K' = h(a)$.

Last round ($n = 1$):

`-- check` $g(0) + g(1) = K$`, thus accept or reject.`

Error prob. in one round is $\leqslant \frac{d}{\text{field size}} \leqslant \frac{1}{n^2}$ for a field of size $\geqslant dn^2$. In total (n rounds) $\Pr \leqslant n \cdot \frac{1}{n^2} \leqslant \frac{1}{n}$.

Interactive protocol for computing the permanent of an integer matrix

Like a determinant, but without signs (very hard!): $\operatorname{perm} A = \sum_{\sigma \in S_n} \prod_{i=1}^{n} a_{i,\sigma(i)}$

- Downward self-reduction: $\operatorname{perm} A = a \iff \sum_{j=1}^{n} a_{1j} \cdot \operatorname{perm} A_{1j} = a$.
  - Prover sends $d_j = \operatorname{perm} A_{1j}$.
  - Verifier checks $\sum_j a_{1j} d_j = a$ and recursively checks $\operatorname{perm} A_{1j} = d_j$.
- We need to merge checking two problems: $\operatorname{perm} B = b$ and $\operatorname{perm} C = c$:
  - Prover sends the coefficients of $p(x) = \operatorname{perm}(Bx + C(1-x))$.
  - Verifier checks $p(0) = c$ and $p(1) = d$ and ...
- Computations modulo a prime $\geqslant n^4$.

Interactive protocol for computing the permanent of an integer matrix

Like a determinant, but without signs (very hard!): $\operatorname{perm} A = \sum_{\sigma \in S_n} \prod_{i=1}^{n} a_{i,\sigma(i)}$

- Downward self-reduction: $\operatorname{perm} A = a \iff \sum_{j=1}^{n} a_{1j} \cdot \operatorname{perm} A_{1j} = a$.
  - Prover sends $d_j = \operatorname{perm} A_{1j}$.
  - Verifier checks $\sum_j a_{1j} d_j = a$ and recursively checks $\operatorname{perm} A_{1j} = d_j$.
- We need to merge checking two problems: $\operatorname{perm} B = b$ and $\operatorname{perm} C = c$:
  - Prover sends the coefficients of $p(x) = \operatorname{perm}(Bx + C(1-x))$.
  - Verifier checks $p(0) = c$ and $p(1) = d$ and ...
  - checks recursively $\operatorname{perm}(Br + C(1-r)) = p(r)$

    for a random r, error prob. $\leqslant \dfrac{\deg\left(p(x) - \operatorname{perm}(Bx + C(1-x))\right)}{\text{field size}}$.
- Computations modulo a prime $\geqslant n^4$.

$$\begin{array}{lcl} \text{formula} & \mapsto & \text{polynomial} \\ \texttt{False} & \mapsto & 0 \\ \texttt{True} & \mapsto & 1 \\ \alpha \wedge \beta & \mapsto & \alpha \cdot \beta \\ \neg\alpha & \mapsto & 1-\alpha \\ \alpha \vee \beta & \mapsto & 1-(1-\alpha)(1-\beta) =: \alpha \odot \beta \end{array}$$

Introduce operators (working on polynomials) instead of quantifiers:

$$\begin{aligned} (A_x P)(\ldots) &= P(0,\ldots)\cdot P(1,\ldots), \\ (E_x P)(\ldots) &= P(0,\ldots)\odot P(1,\ldots); \end{aligned}$$

One more operator (not for a quantifier): multilinearization (just remove the degrees!)

$$(L_x P)(x,\ldots) = P \mod (x^2 - x).$$

We need to prove

$$q^{(1)}_{x_1} \quad L_{x_1} \quad q^{(2)}_{x_2} \quad L_{x_1}L_{x_2} \quad q^{(x_3)} \quad \ldots \quad q^{(n)}_{x_n} \quad L_{x_1}\ldots L_{x_n} \quad P(x_1\ldots x_n) = 1.$$

Let d be the input formula size.
Recursive protocol for proving that the formula is true:

$$q_{x_i} q'_{\dots} \dots q''_{\dots} P(x_1, \dots, x_{i-1}, x_i \dots x_n)|_{x_1 = r_1, \dots, x_{i-1} = r_{i-1}} = c :$$

Denote

$$R(x_1, \dots, x_i) = q'_{\dots} \dots q''_{\dots} P(x_1, \dots, x_i).$$

Prover gives coefficients of this polynomial $s(x_i)$ of degree at most d.

$q = A$. If $s(0)s(1) \neq c$, Verifier rejects.
Otherwise checks recursively $R(r_1 \dots r_{i-1}, r_i) = s(r_i)$ for a random r_i.

$q = E$. Analogously, checking $s(0) \odot s(1) = c$.

Error prob. at each step is $\leqslant \dfrac{d}{\text{field size}}$. In total ($d^2$ steps) error prob. $\leqslant \dfrac{1}{d}$ for a field of size $\geqslant d^4$.

Let d be the input formula size.
Recursive protocol for proving that the formula is true:

$$q_{x_i} q'_{\dots} \dots q''_{\dots} P(x_1, \dots, x_{i-1}, x_i \dots x_n)|_{x_1 = r_1, \dots, x_{i-1} = r_{i-1}} = c :$$

Denote

$$R(x_1, \dots, x_i) = q'_{\dots} \dots q''_{\dots} P(x_1, \dots, x_i).$$

Prover gives coefficients of this polynomial $s(x_i)$ of degree at most d.

$q = A$. If $s(0)s(1) \neq c$, Verifier rejects.
Otherwise checks recursively $R(r_1 \dots r_{i-1}, r_i) = s(r_i)$ for a random r_i.

$q = E$. Analogously, checking $s(0) \odot s(1) = c$.

$q = L$. Analogously, but we already have some value r_i!
Verifier checks $s(0) + (s(1) - s(0))r_i = c$ and checks recursively that $R(r_1, \dots, r_{i-1}, r'_i) = s(r'_i)$ for a new random r'_i.

Error prob. at each step is $\leqslant \dfrac{d}{\text{field size}}$. In total ($d^2$ steps) error prob. $\leqslant \dfrac{1}{d}$ for a field of size $\geqslant d^4$.

The End

(of the course)

Do not hesitate to ask `edward.a.hirsch@gmail.com` for admission hours