Ended in the middle of proof of $\text{coNP} \subseteq \text{IP}$
(to illustrate ideas in result $\text{IP} = \text{PSPACE}$)

**Verifier**

Probabilistic poly time (in 1x1)
Private random string $r$

**Prover**

Computationally unbounded

\[
\begin{align*}
 a_1 &= f(x,r) \\
 a_2 &= g(x,a_1) \\
 a_3 &= f(x,r,a_1,a_2) \\
 a_4 &= g(x,a_1,a_2,a_3) \\
 &\vdots
\end{align*}
\]

Verifier announces decision
\[\text{out}_f \langle f,g \rangle (x) \in \{0,1\}^*\]

Language $L$ in $\text{IP}$ if $\exists$ protocol with polynomial (in 1x1) # rounds with

**Completeness**

\[x \in L \Rightarrow \exists \text{protocol } P \text{ \ \ \ } \Pr[\text{out}_v \langle V,P \rangle (x) = 1] \geq 2/3\]

**Soundness**

\[x \notin L \Rightarrow \forall P' \text{ \ \ \ } \Pr[\text{out}_v \langle V,P' \rangle (x) = 1] \leq 1/3\]
THEOREM 9 \[ \text{coNP} \subseteq \text{IP} \]

Construct protocol for more general problem

\[ \# \text{SAT}_D = \{ \langle \varphi, k \rangle \mid \varphi \text{ 3-CNF with exactly } k \text{ satisfying assignments} \} \]

\( k=0 \) gives 3-SAT as special case

Write clause \( C_i \) as polynomial \( p_j \)

\[ x_i \lor \overline{x}_j \lor x_k \mapsto 1 - (1-x_i) \overline{x}_j (1-x_k) \]

Write formula \( \varphi = \bigwedge_{j=1}^m C_j \) as polynomial

\[ P_\varphi = \prod_{j=1}^m P_j \quad (*) \]

Degree \( \leq 3m \)

Efficient representation in size \( O(m) \)

(arithmetic circuit)

Want to count \# satisfying assignments

\[ K = \sum_{b_1 \in \{0,1\}} \cdots \sum_{b_n \in \{0,1\}} P_\varphi (b_1, \ldots, b_n) \quad (**) \]

Do calculations mod prime \( p > 2^n \geq K \)

Observation: If \( g(x_1, \ldots, x_n) \) we plug in \( x_i = b_i \) for \( i=2, \ldots, n \), then get univariate polynomial. True also for

\[ h(x_i) = \sum_{b_2 \in \{0,1\}} \cdots \sum_{b_n \in \{0,1\}} g(x_1, b_2, \ldots, b_n) \quad (*) \]

(for \( g = P_\varphi \) or other polynomial)
We have that
\[ K = \sum_{b_1 \in \{0,1\}} \cdots \sum_{b_n \in \{0,1\}} g(b_1, \ldots, b_n) \] (1)
iff \[ h(0) + h(1) = K \] (obviously)

Idea of protocol
- Ask prover for prime \( p \in (2^n, 2^{2n}] \)
- Check that \( p \) prime
- Ask prover for \( h(x) \)
- Check \( h(0) + h(1) = K \)
- Check that prover was honest when giving \( h(x) \).

**SUMCHECK** \( (g, K, n) \)

\[ V: \]
\[ V: \] If \( n = 2 \), accept if \( g(0) + g(1) = K \), reject otherwise
\[ V: \] If \( n \geq 2 \), ask prover for \( h(x) \) in (1)

\[ P: \]
Sends \( s(x) \)

\[ V: \]
Check if \( s(0) + s(1) = K \), reject otherwise
Pick \( a \in \mathbb{R} \) \([0, p-1]\)
\[ K' := s(a) \]
\[ g'(x) := g(a, x_2, \ldots, x_n) \]
Run **SUMCHECK** \( (g', K', n-1) \)

Recursive call checks that \( s(x) = h(x) \) by verifying
\[ s(a) = \sum_{b_2 \in \{0,1\}} \cdots \sum_{b_n \in \{0,1\}} g(a, b_2, \ldots, b_n) \]
**Lemma 10**

If $g$ degree-$d$ polynomial and $p$ prime, then SUMCHECK $(g, K, n)$ has
- completeness 1
- soundness error $\leq \frac{d n}{p}$

- $m$ clauses $\Rightarrow \deg(P_g) \leq 3m$
- 3-CNF over $n$ variables $\Rightarrow m \leq 27n^3$
- Pick $p > 2^n$. Get soundness error $\leq \frac{d n}{p} < \frac{81n^4}{2^n} \Rightarrow 0$

So $\text{coNP} \subseteq \text{1P}$ follows from lemma 10

**Proof of Lemma 10**

Completeness: Obvious. Prover answers honestly, and all verifier checks pan out.

Soundness: By induction over $n$.

Base case ($n=1$): Want to detect if

$$\sum_{g(0), g(1)} g(0) \neq K$$

Compute $g(0) + g(1)$

0% probability of being fooled.
Inductive step: Want to detect if
\[ \sum_{g(1)} \ldots \sum_{g(n)} \neq K \]
\[ g(6) \in [0,15] \quad 6n \in [0,15] \]

Inductive hypothesis says that
\[ \text{SUMCHECK}(g', K, n-1) \] has soundness error \[ \leq \frac{dl}{p} \ (n-1) \]

Two cases:
(a) Prover honestly replies with \( h(x) \) as in (#)
   But then \( h(0) + h(1) \neq K \) and verifier has 0% probability of being fooled

(b) Prover replies with \( s(x) \neq h(x) \)
   \( \deg(s(x) - h(x)) \leq d \)
   \( \Rightarrow s(x) - h(x) \) has \( \leq d \) roots
   \( \Rightarrow \) at most \( d \) values for \( a \) such that \( s(a) = h(a) \)

(i) If prover is lucky and verifier picks \( s \) s.t. \( s(a) = h(a) \), then verifier fooled
(ii) Otherwise, get sumcheck instance for polynomial \( g' \) over \( n-1 \) variables with wrong value \( K \)

\[ \Pr[\text{verifier V fooled}] = \Pr[V \text{ fooled in case (i)}] + \Pr[V \text{ fooled in case (ii)}] \]
\[ \leq \frac{dl}{p} + \frac{dl}{p} \ (n-1) = \frac{dn}{p} \]

The lemma follows by the induction principle \( \square \)
We proved \( \text{coNP} \subseteq \text{IP} \).
Actually, most of what is needed for \( \text{PSPACE} \subseteq \text{IP} \) except for some extra twists.

What was the key idea? ARITHMETIZATION
CNF formula \( \Phi \) maps to polynomial \( P \).
Evaluate polynomial in much larger field \( \Rightarrow \)
makes it practically impossible for prover to cheat.

Can also define **MULTIPROVER INTERACTIVE PROTOCOLS (MIP)**. Provers agree beforehand on shared strategy but cannot communicate during protocol.

\[
\begin{align*}
\text{PROVER 1} & \quad \text{VERIFIER} & \quad \text{PROVER 2} \\
& \quad a_1 = f_1(x, r) & \quad b_1 = f_2(x, r) \\
& \quad a_2 = g_1(x, a_1) & \quad b_2 = g_2(x, b_2) \\
& \quad a_3 = f_1(x, r, a_1, a_2, b_2) & \quad b_3 = f_2(x, r, b_2, a_3, a_1) \\
\end{align*}
\]

Can allow up to polynomially many provers
(but verifier needs to have enough time to read all answers)

In fact, just going from 1 to 2 provers gives as much power as polynomially many provers.
Define MIP analogously to IP

Clearly, IP \subseteq MIP [can always ignore one prover]

**Thm 11** [Babai, Fortnow, Lund '90]

MIP = \text{NEXP}

Why are 2 provers more useful?
Can use 2nd prover to force non-adaptivity of 1st prover.

Suppose prover 1 gets questions

$q_1, q_2, \ldots, q_m$

Prover 1 sees context and can choose answer to $q_i$ depending on $q_1, \ldots, q_{i-1}$

But if verifier randomly picks $i \in \{1, \ldots, m\}$

and asks $q_i$ from prover 2, and requires both provers 1 & 2 should give same answer to $q_i$, then prover 1 can no longer answer adaptively (because prover 2 cannot answer adaptively).

So provers might as well write down and publish big table with answers to all possible questions. [This needs a formal argument, of course.]

Verifier questions = random look-ups in table
PCP[r, q] = set of languages that can be decided by q random checks in table of size 2^r
[Informal definition]

Can restate Thm 11 as

\[ \text{NEXP} = \bigcup_{\text{poly}} \text{PCP[poly, poly]} = \bigcup_{n \in \mathbb{N}} \text{PCP[n, n^c]} \]

Can be "scaled down" to

\[ \text{NP} = \bigcup_{\text{polylog}} \text{PCP[polylog, polylog]} \]

And further improved (with lots of work)

\[ \text{THM 12 PCP THEOREM [Arora-Safra '92] [Arora-Lund-Motwani-Sudan-Szegedy '92]} \]

\[ \text{NP} = \text{PCP[O(log n), O(1)]} \]

Means that for any language \( L \subseteq \text{NP} \) can write down proofs \( \Pi \) of \( x \in L \) s.t.

- It has size \( \text{poly}(1|x|) \)
- It can be checked by reading constant \#bits (independent of size of \( x \))
- If \( x \in L \), accept w.h.p.
- If \( x \notin L \), reject w.h.p.

Now if this isn't magic...

Proof is highly nontrivial and would take several lectures even just for an overview.
Example 13 Graph Non-Isomorphism & PCP\[\text{poly}(n), O(1)\]

\[\text{GNI} = \{ \langle G_0, G_1 \rangle \mid G_0 \neq G_1 \}\]

Graphs on \(n\) vertices

Represent by adjacency matrix

Binary string of length \(n^2\) \(\leftrightarrow\) number in \([0, 2^{n^2} - 1]\)

Proof 11: Binary string of length \(2^{n^2}\)

Let position \(p \in [0, 2^{n^2} - 1]\) correspond to graph \(H_p\).

Expected format of proof

Bit in position \(p\) is:

a) 0 if \(H_p \cong G_0\)

b) 1 if \(H_p \cong G_1\)

c) don't care otherwise

Verifier test

1. Flip \(b \in \{0, 1\}\)

2. Choose random permutation \(\pi : [n] \rightarrow [n]\)

3. Let \(H_p = \pi(G_0)\)

4. Look up bit \(b'\) in position \(p\)

5. Accept if \(b = b'\); reject otherwise

Analysis

Completeness: If \(G_0 \neq G_1\), proof confirms

Table 11 according to specification.

Verifier's test will always accept
Soundness (sketch)

If \( G_0 \neq G_1 \), then probability of checking position \( p \) is independent of \( G \).
So, mentally, we can
(i) Choose random \( \sigma \)
(ii) Look up \( b \) in position \( p \) for \( H_{p} = \pi(G_i) \)
(iii) Only now flip \( b \in \{0,1\} \)
(iv) Accept if \( b = b' \) [with probability = \( \frac{1}{2} \)]

More formal proof of soundness

Suppose \( G_0 \neq G_1 \)

Consider for \( i \in \{0,1\} \) distributions
\[
D_i = \{ \pi(G_i) \mid \pi : [n] \rightarrow [n] \text{ uniformly sampled random permutations} \}
\]

Then \( D_0 \) and \( D_1 \) are identical.

Because following two experiments give same distribution

1. Pick random permutation \( \sigma : [n] \rightarrow [n] \) and return \( \sigma \)
2. Fix arbitrary permutation \( \sigma^* : [n] \rightarrow [n] \)
   Pick random permutation \( \sigma : [n] \rightarrow [n] \)
   Return \( \sigma \circ \sigma^* \)

So we can let \( \sigma^* \) be permutation such that \( \sigma^* (G_0) = G_1 \) [exact equality]
\[
\Pr[\text{accept}] = \sum_{\text{pos in tape}} \Pr[\text{read pos } p] \cdot \Pr[\text{read bit} = 0 \mid \text{read pos } p]
\]

\[
\Pr[\text{read pos } p] \text{ independent of } b \text{ by argument above}
\]

Hence, what bit \( b' = b[p] \) verifier reads is independent of coin flip \( b \). So

\[
(*) = \sum_{\text{pos } p} \Pr[\text{read pos } p] \cdot \Pr[b = \text{some fixed bit}] 
\]

\[
= \Pr[b = \text{some fixed bit}] \cdot \sum_{\text{pos } p} \Pr[\text{read } p] 
\]

\[
= \Pr[b = \text{some fixed bit}] 
\]

\[
= \frac{1}{2} \text{ as claimed}
\]
More on to
CRYPTOGRAPHY
Just scratch the surface
DD2448 Foundations of Cryptography
given in spring
[Necessary for a well-rounded T)CS
education, if you ask me]

"HUMAN INGENUITY CANNOT CONCOCT
A CIPHER WHICH HUMAN INGENUITY
CANNOT RESOLVE"
Edgar Allan Poe 1841

Cat-and-mouse game throughout
the ages.

Shannon (late '40s): rigorous definition
of security

1970s: Birth of modern cryptography
Connection to computational complexity theory
Make code breaking a computationally
hard problem (so hardness = good news!)

Cross-fertilization: Many ideas from
crypto have turned out to be extremely
useful in complexity theory
**Basic Task**

key $k \in \mathbb{E}_R \{0,1\}^n$

Alice

$x \in \mathbb{E}_R \{0,1\}^n$

\[ y = E_k(x) \]

\[ x' = D_k(y) \]

Bob

\[ x' = D_k(E_k(x)) = x \]

**Eve**

**Definition:** Encryption scheme is perfectly secret if $\forall x, x' \in \{0,1\}^n$ the distributions $E_{k_1}(x)$ and $E_{k_2}(x')$ are identical.

Recall: $U_n$ = uniformly random $n$-bit string

**Example 15** Suppose $n = m$

Let $y = \text{bitwise XOR of } x \text{ and } k$

$(y_i = x_i \oplus k_i \pmod{2})$ ONE-TIME PAD

Not hard to prove $E_k(x)$ looks perfectly random to outside observer.

**Claim 16** If $(E, D)$ is an encryption scheme with $n < m$, then it is not perfectly secret.

Proof: Nice exercise
Solution? Drop perfect secrecy
require secrecy only w. r. t. computationally bounded adversaries

[ Even NSA is computationally bounded ]

If this is to be possible, need P ≠ NP
(see Lem 9.2 in Amra-Barak)
But this is not sufficient

Let us very briefly sketch a basic assumption of modern crypto and
some consequences of it that allow us to recover a "computationally
secure one-time pad"

**DEF 17** Function \( \varepsilon : \mathbb{N} \to [0, 1] \) is
**NEGligible** if \( \varepsilon(n) = n^{-\omega(1)} \)
That is, \( \forall c \in \mathbb{N} \) s. t. \( \forall n > N \)
\( \varepsilon(n) < n^{-c} \)

**DEF 18** A poly-time computable function
\( f : \mathbb{E} \leftarrow \mathbb{E} \) is a **ONE-WAY
FUNCTION** if for every probabilistic
poly-time algorithm \( A \) there is a
negligible function \( \varepsilon \) s. t.
\[ \Pr_{x \in \mathbb{E}} \left[ A(y) = x \mid s. t. f(x') = y \right] < \varepsilon(n) \]
CONJECTURE/ASSUMPTION 19
One-way functions exist

CLAIM 20
If one-way functions exist, then P≠NP

Proof: Good exercise; not hard.

And if one-way functions exist, then computationally secure encryption schemes exist.

Will talk a little bit more about this (and some other aspects of cryptography) next lecture.