Proof Systems:

Given language \( L \): goal to prove \( x \in L \).

Proof system for \( L \) requires a verification algorithm \( V \):

Completeness: \( x \in L \Rightarrow \exists \ \text{proof}_x \ V \text{ accepts} (x, \text{proof}_x) \)

\( x \notin L \Rightarrow \forall \ \text{proof}' \ V \text{ rejects} (x, \text{proof}') \).

The prover asserts \( x \in L \):

True assertions have proofs
False assertions have no proofs.

Efficiency: \( \forall x, \text{proof} \ V((x, \text{proof})) \text{ runs in time } \text{poly}(1|x|) \).

\( L \in \text{NP} \) if
\( L = \{ x | \exists y \ 1 \leq |x|^k \ (x, y) \in R^3 \ \text{R \ & \ P,} \)
\( \text{Verifier} = R, \text{ proof} = y \).

New ingredients: randomness (Verifier can toss coins)
interaction — instead of reading the proof,
the verifier can ask questions.

Interactive proof systems:
both \( P \) (prover) and \( V \) (verifier) know \( x \).

\# rounds \( \leq \text{poly} (1|x|) \)
\( V \) can use a random string.
An Interactive Proof System for $L$ is an interactive protocol $(P, V)$

Completeness: $x \in L \Rightarrow \Pr[V\text{ accepts in } (P, V)(x)] \geq \frac{2}{3}$.

Soundness: $x \notin L \Rightarrow \forall P^* \Pr[V\text{ accepts in } (P^*, V)(x)] \leq \frac{1}{2}$

$V$ is a p.p.t. machine.

As usual, repetition can reduce the error to $\varepsilon$.

$|P| = \exists L | L$ has an interactive proof system $\exists$.

⇒ philosophically captures more broadly what it means to be convinced that a statement is true.

$NP \leq IP$ : Verifier receives only a single message and uses no random bits.

If $NP \not\leq IP$ then randomness is essential.

If the verifier is deterministic, the prover knows all of $V$'s questions in advance and can send all the answers in one shot.

**Graph Isomorphism**: $\delta_0 = (V_0, E_0)$ $\delta_1 = (V_1, E_1)$

graphs are isomorphic $\delta_0 \cong \delta_1$ iff $\exists \Pi: V \rightarrow V$

$(x, y) \in E_0 \iff (\Pi(x), \Pi(y)) \in E_1$

\[\delta_I = \exists (\delta_0, \delta_1) \mid \delta_0 \cong \delta_1\]
GI \in \text{NP but not known to be in P or NP-complete.}

GNI = \overline{GI} \text{ not known if GNI \in \text{NP.}}

**Theorem**: GNI \in \text{IP.} (Indication that IP may be more powerful than NP).

**Verification**

Flips coin c \in \{0,1\}

pick random \pi

Apply \pi to H_c

H = \pi(60)

\[ \begin{align*}
  & \text{if } \pi \equiv \theta_0 \quad r = 0 \\
  & \text{else} \quad r = 1
\end{align*} \]

Accept iff r = c.

**Completeness**: if \theta_0 \neq \theta_1, then \pi \equiv \theta_0 or \pi \equiv \theta_1 but not both. Prover can select the correct one.

**Soundness**: if \theta_0 \equiv \theta_1, then prover sees the same distribution regardless if c = 0, 1.

Prover gets no information about c.

Any prover can succeed w.p. \leq \frac{1}{2}.

GNI \subseteq \text{co-NP but not known if GNI is co-NP-complete.}

**Theorem**: co-NP \subseteq \text{IP.}

**Proof idea**: Will actually show the following language is in IP:

\[ \{ \phi(x_1\ldots x_n) \mid \phi(x_1\ldots x_n) \text{ has exactly k SAT assignments} \}\]
Prover claims that $\phi$ has exactly $k$ satisfying assignments.

This is true if:

\[ \phi(0, x_1, \ldots, x_n) \text{ has } k_0 \text{ sat assignments.} \]
\[ \phi(1, x_1, \ldots, x_n) \text{ has } k_1 \text{ sat assignments} \]
\[ k = k_0 + k_1. \]

Prover sends $k_0$ and $k_1$.

Then Verifier picks a random $c$ and asks Prover to recursively prove that $\phi = \phi(c, x_1, \ldots, x_n)$ has $k_c$ satisfying assignments.

At the end, Verifier can check.

Problem: If $k$ is only $l$ bits long, Verifier will only catch this if V's random choices lead exactly to the leaf that is the source of the discrepancy.

Solution: Replace $\exists_0, \forall_1$ with $\exists F_3 \forall_1$.

Verifier substitutes a random field element at each step. Most majority of field elements at each step will catch a cheating prover (instead of just one).

\( \text{LFKN} \)

Theorem: \( L = \exists (\phi, k) \mid \text{CNF } \phi \text{ has exactly } k \text{ satisfying assignments} \).

1. Arithmetization: $\phi(x_1, \ldots, x_n) \Rightarrow P_\phi(x_1, \ldots, x_n)$

\[ \text{degree of polynomial on } F_q \text{ & prime } p \geq 2^n. \]
\[ x_i \rightarrow x_i \]
\[ \phi \land \phi' \rightarrow P_{\phi} \cdot P_{\phi'} \]
\[ \phi \lor \phi' \rightarrow 1 - (1 - P_{\phi})(1 - P_{\phi'}) \]

For all \( \alpha \in \mathbb{F}_2^n \), \( P_{\phi}(\alpha) = \phi(\alpha) \)

degree \( d \leq P_{\phi} \leq |\phi| \).

Can compute \( P_{\phi}(\alpha) \) in poly time, given \( \phi \circ \alpha \).

**Prover wishes to show:**
\[ k = \sum_{x_1 \in \mathbb{F}_2} \cdots \sum_{x_n \in \mathbb{F}_2} P_{\phi}(x_1 \cdots x_n) \]

**Define** \( k_2 = \sum_{x_2 \in \mathbb{F}_2} \cdots \sum_{x_n \in \mathbb{F}_2} P_{\phi}(x_2 \cdots x_n) \)

**Prover sends** \( k_2 \) for all \( x \in \mathbb{F}_2^n \)

**Verifier picks random** \( z \)

- Asks prover to prove that
  \[ k_2 = \sum_{x_2 \in \mathbb{F}_2} \cdots \sum_{x_n \in \mathbb{F}_2} P_{\phi}(z, x_2 \cdots x_n) \]
- Also checks that \( k_0 + k_1 = k \).

At end, verifier checks \( P(z_1, z_2, \ldots, z_n) = k_n \).

Actually... since \( q > 2^n \), prover can't send all of \( \phi \).

Sends the polynomial
\[ p(z) = \sum_{x_2} \cdots \sum_{x_n} P_{\phi}(z, x_2 \cdots x_n) \]

degree \( d' \leq d \leq |\phi| \).

Protocol shown on next page...
Input \((\phi, k)\)

Prover

\[
P_i(x) = \sum_{x_i \in \mathbb{F}_2} \phi(x, x_2, \ldots, x_n)
\]

\(i \in \{0, 1, \ldots, n\}\)

Verifier

\[
P_i(0) + P_i(1) = k ?
\]

pick random \(z_1 \in \mathbb{F}_2\)

\[
P_1(x) \rightarrow P_2(0) + P_2(z_1) = P_1(z_1)^2
\]

pick random \(z_2 \in \mathbb{F}_2\)

\[
P_2(x) \rightarrow P_3(0) + P_3(z_2) = P_2(z_2)^2
\]

\[
P_3(x) \rightarrow \ldots
\]

\[
P_n(0) + P_n(z_n) = P_{n-1}(z_{n-1})
\]

pick random \(z_n \in \mathbb{F}_2\)

\[
P_n(z_n) = \phi(z_1, \ldots, z_n)
\]

Completeness: \((\phi, k) \in L \Rightarrow \) honest prover will always cause the verifier to accept.

Soundness: \(P_i(x)\) is named poly., \(P_i^*(x)\) poly. sealed by prover.

Suppose \((\phi, k) \notin L \Rightarrow P_i(0) + P_i(1) \neq k\)

if \(P_i^*(0) + P_i^*(1) \neq k \Rightarrow V\) rejects.

\[
P_i^*(x) = \phi(x, x_2, \ldots, x_n)
\]

Bad case: \(V\) verifies if \(P_i^*(z_i) = P_i(z_i)\)

\[
P_{1/4} \leq P_i^*(z_i) = P_i(z_i) \leq 161/2^n
\]
Suppose we don’t have the bad event:

\[ P_i^*(z) \neq P_i(z) \]

Verifier sends \( P_2^*(z) \) if \( P_2^*(0) + P_2^*(1) \neq P_1^*(z) \) \( \Rightarrow \) reject.

0.D.

\[ P_2^*(0) + P_2^*(1) = P_1^*(z) \neq P_1(z) = P_1(0) + P_1(1) \]

\( \Rightarrow \) \( P_2^* \neq P_2. \)

If \( P_i(z) = P^*_i(z) \) \( \Rightarrow \) BAD!

In general: Bad event: \( P_i(z) = P^*_i(z) \)

Probability any bad event occurs: \( \leq n \frac{1}{2^n} < \text{small}. \)

At the end, we have \( P_n^*(z) \) which the prover claims is equal to \( P_n^*(z) = P_n(z_1, \ldots, z_{n-1}, z) \) if no “bad” event occurs, these polynomials are actually different.

Verifier selects a random \( z_n \) and checks.

\[ P_n^*(z) = P_n(z_1, \ldots, z_n) \]

Fails to detect difference w/ pr \( \leq \frac{1}{2^n}. \)

If no bad even occurs, verifier will detect the difference:

\[ L = 3 (\phi, k) \]

CNF \( \phi \) has exactly \( k \) satisfying assignments \( 3 \) is in \( \text{IP} \)

\( L \) is \( \text{coNP-hard} \) so \( \text{coNP} \leq \text{IP}. \)
NP, co-NP \subseteq \text{IP}. How much more is in \text{IP}.

**Theorem (Shamir)\text{ IP = PSPACE.}**

\text{IP} \subseteq \text{PSPACE} is easy: enumerate all possible interactions, explicitly calculate acceptance probability.

**Interactin very powerful!**

Can interact with another player \text{ if generalized geography and determine if she can win, even if you can not compute the optimal moves).**

Need to prove \text{PSPACE} \subseteq \text{IP} \implies (\text{i.e. QSAT} \in \text{IP}).

Same basic idea as \text{co-NP} proof - plus a few additional ingredients.

First assume no occurrence of \( x \) separated by more than one \( A \) from point of quantification:

\[ \exists x \in A \text{ (occurrence of } x) \quad \text{no occurrence of } x. \]

This helps ensure degree of any single variable \text{O}(1/\phi) \uparrow

Arithmeticize \( \phi \rightarrow \text{P} \phi \)

\[ \exists x: \phi \rightarrow \sum_{X_i=0,1} \text{P} \phi(x_i \ldots) \]

\[ \forall x: \phi \rightarrow \prod_{X_i=0,1} \text{P} \phi(x_i, \ldots). \text{ this can double the degree and square the size of } \phi. \]
Quantified Boolean expression $\phi$ is true if $P\phi > 0$.

**Problem:** the $\forall \left( \exists \right)$ terms may cause $P\phi \geq 2^{2^{|\phi|}}$ to be too large to compute.

**Solution:** evaluate mod $q$ such that $2^n < q < 2^{2n}$.

Prover sends a "good" $q$ in the first round.

**Claim:** a good $q$ exists because the number of primes in the range is at least $2^n$.

Since the product of all these primes is $P\phi$, it can't be that $P\phi \mod q = 0$ for all the $q$ in the range.

CSA protocol:

**Verifier**

```
Input $\phi$

Proof

$K, q, P_1(x)$

$P_1(x)$: remove order $\exists \left( \forall \right)$ from $\phi$

$P_2(x)$: remove order $\forall \left( \exists \right)$ from $\phi$[x_i = z_i]

Output

$P_1(x)$
```

**Verifier**

```
$P_1(b) + P_1(z) = k$ or
$P_1(b) \cdot P_1(z) = k$

Pick random $z_i \in \mathbb{F}_q$
```

**Prover**

```
$P_2(x)$

Check $P_2(b) + P_2(z) = P_1(z_i)$

or $P_2(b) \cdot P_2(z) = P_1(z_i)$

Pick random $z_i \in \mathbb{F}_q$
```

```
$P_n(x)$

Check $P_n(b) + P_n(z) = P_{n-1}(z_{n-1})$ or
$P_n(b) \cdot P_n(z) = P_{n-1}(z_{n-1})$

Pick random $z_i \in \mathbb{F}_q$
```

```
$P_2(x)$
```

```
$P_n(x)$
```

```
$P_1(x)$
```
Note: because of condition (\#) $P_n(x)$ is bounded in size, so it can be sent to $V$.

Also, $P_q\left[ x_1 \leq t_1, \ldots, x_n \leq t_n \right]$ has no quantifiers since all the $x_i$ are fixed so this can be computed in poly time.

Completion: if $\phi \in \text{SAT}$
then an honest prover will cause $V$ to accept.

Soundness: let $P_i(x)$ be correct polynomial
let $P_i^*(x)$ be polynomial sent by prover.

If $\phi \in \text{QSAT} \Rightarrow$ then $P_i(0) + \ldots + P_i(t) = 0 \neq k$.
If $P_i^*(0) + \ldots + P_i^*(t) \neq k$ then $V$ will reject.
If this doesn't happen then $P_i^* \neq P_i$.

$$\Pr_{z_i}[P_i^*(z_i) = P_i(z_i)] \leq \frac{2^{1/\phi}}{2^n} \quad \text{this because } \phi \in \text{SAT}.$$ 

In general: assume $P_i(0) \neq P_i^*(0) = P_i(1) + \ldots + P_i(t)$.
If $P_i^*(0) + \ldots + P_i^*(t) \neq P_i^*(z_i)$ then $V$ rejects.
Otherwise $P_i^* = P_i$.

$$\Pr_{z_i}[P_i(z_i) = P_i^*(z_i)] \leq \frac{2^{1/\phi}}{2^n}.$$ 

At the end, we have $P_n(z_n) \neq P_n^*(z_n)$

$\phi$ can calculate this from the original $\phi$. Given $q_{z_{n-1}} \ldots z_n$
\[ V \text{ will reject as long as } p_i(\bar{z}_i) \neq p_i^*(\bar{z}_i) \text{ for all } i. \]

For each \( i \), the probability \( p_i(\bar{z}_i) = p_i^*(\bar{z}_i) \leq \frac{2|\Phi|}{2^n} \approx \frac{1}{3}. \)

\[ \text{Prob that for any } i, \quad p_i(\bar{z}_i) = p_i^*(\bar{z}_i) \leq \frac{2n|\Phi|}{2^n} \ll \frac{1}{3}. \]

\[ \implies \text{QSAT is in IP.} \]

**Example:** \[ \Phi = \forall x \exists y ((x \lor y) \land \forall z ((x \land z) \lor (y \land \neg z))) \]

\[ p_{\Phi} = \prod_{x=0,1} \sum_{y=0,1} \left[ (x+y) \ast \prod_{z=0,1} \left[ (x+z + y(1-z)) + \sum_{w=0,1} (z + y(1-w)) \right] \right] \]

\[ \prod_{x=0,1} \sum_{y=0,1} \left[ (x+y) \ast \prod_{z=0,1} \left[ (x+z + y(1-z)) + 2z + y \right] \right] \]

\[ \prod_{x=0,1} \sum_{y=0,1} \left[ (x+y) \ast (2y)(x+2+y) \right] \]

\[ \prod_{x=0,1} (x+1)(2)(x+3) = (1\cdot2\cdot3)(2\cdot2\cdot4) = 96 \Rightarrow p_\Phi \]

Prover claims \( p_\Phi > 0 \).

Prover sends \( q=13 \); claim \( p_\Phi = 96 \mod 13 = 5 \); sends \( k=5 \).

Prover removes even \( \prod_{x=0,1} \) and sends \( (x+1)2(x+3) = 2x^2 + 8x + 6 \equiv q(x) \)

Verifier checks \( p_{\Phi}(0)p_{\Phi}(1) = (6)(16) = 96 \equiv 5 \mod 13. \)

Verifier picks randomly \( z_1 = 9 \).
\[
\text{Prover sends } P_2(y) = 2y^3 + y^2 + 3y
\]

**Verifier checks**

\[P_2(0) + P_2(1) = P_1(9) \equiv 6 \equiv 6\]

\[P_3(3) = \left(9 + 3(1-3)\right) + 2(2+3) = 82 + 6.\]

\[\Rightarrow 12 \cdot P_3(0) \cdot P_3(1) \text{ should be } P_2(3)\]

**Prover and verifier both know** \(12 \ldots \).

**Prover sends** \(P_3(2)\).

**Verifier checks**

\[12 \cdot P_3(0) \cdot P_3(1) = 12 \cdot 14 \cdot 6 \equiv 13 = 7\]
Verifying given random 2402.
Final check: \[ 12 \left[ (9 \cdot 7) + 3(1+7) + (7 + 3(1-2)) \right] = 12 \left[ 6 + 74(2) \right] \]

Arthur-Merlin Games: IP permits verifier to keep coin flips private... is this necessary?

Note that the BNTI protocol breaks without it.

Arthur-Merlin game: interactive protocol in which coin flips are public (although not known in advance).

Arthur may just as well send the results of his coin flips and ask Merlin to do whatever computation he would have done.

Clearly, Arthur-Merlin \( \subseteq \) IP.

"Private coins are at least as powerful as public coins."

The proof of \( IP = \text{PSPACE} \): actually shows.

\( \text{PSPACE} \subseteq \text{Arthur Merlin} \subseteq \text{NP} \subseteq \text{PSPACE} \)

"Public coins are at least as powerful as private coins!"

Limiting # of rounds: AM \[ k \] # rounds = \( k \), Arthur (V) goes first.

MA \[ k \] # rounds = \( k \), Merlin (P) goes first.

Theorem: \( \text{AM} [k] = \text{AM} [k] \) with perfect completeness
MA \[ k \] = MA \[ k \] with perfect completeness

\( \Rightarrow \text{accept w.p. 1} \).
Theorem: \( \forall k \geq 2 \quad AN[k] = AN[2] \).


Then can move all of Arthur messages to beginning.
\[
ANAMAN \subseteq ANANMAM \ldots \subseteq ANANMAM
\]

Proof: given \( L \in MA[2] \).
\[
\forall x \in L \Rightarrow \exists m \quad Pr[ (x, m, r) \in R ] = 1.
\]
\[
Pr[ \exists m \quad (x, m, r) \in R ] = 1.
\]
\[
\forall x \notin L \Rightarrow \forall m \quad Pr[ (x, m, r) \in R ] \leq \epsilon.
\]
\[
\Rightarrow Pr[ \exists m \quad (x, m, r) \in R ] \leq 2^{|m|} \epsilon.
\]

by \( t \) repetitions, can get \( \epsilon < 2^{-t} \)
repeat \( s^t \) independent random shuffles: \( t = m + 1 \)
\[
2^{|m|} \epsilon < \frac{1}{2}.
\]

Two important classes: \( MA = MA[2] \)
\( AN = AN[2] \).
Definitions w/ reference to interaction:
\( L \leq MA \) iff \( \exists \) poly-time language \( R \):
\[ x \in L \implies \exists m \Pr_r [(x, m, r) \in R] = 1 \]
\[ \forall m \Pr_r [(x, m, r) \in R] \leq \frac{1}{2} \]

\( L \leq AM \) iff \( \exists \) poly-time language \( R \):
\[ x \in L \implies \Pr_r [\exists m (x, m, r) \in R] = 1 \]
\[ \Pr_r [\exists m (x, m, r) \in R] \leq \frac{1}{2} \]

\( AM, MA \leq NP \) (can elect to ignore \( r \)).

AM, MA \leq \Pi_2^P
\[ x \in L \iff \forall r \exists m (x, m, r) \in R. \]

MA \leq AM \leq \Pi_2^P
\[ \text{from above.} \]

\[ \Pi_2^P \]
\[ \Sigma_2^P \]
\[ \text{Theorem: co-NP \leq \text{AM} \implies PH = \text{AM}.} \]

Suffice to show: \( \Sigma_2^P \leq \text{AM} \)

We showed earlier that this

Causo PH = \( \Sigma_2^P \)

Proof: \( \text{co-NP \leq \text{AM} \implies } \Sigma_2^P \leq \text{AM} \).

\[ L \leq \Sigma_2^P \implies \exists \text{ poly-time } R \]
\[ x \in L \implies \exists y \forall z (x, y, z) \in R. \]
\[ x \not\in L \implies \forall y \exists z (x, y, z) \notin R. \]
Merlin sends y
1 AM round decides co-NP query \( \forall z (x, y, z) \in R \).

3 rounds: M AM since MA \subseteq AM
\[ \Rightarrow AM \subseteq AM. \]

We know Arthur-Merlin = IP

**Theorem** \( IP[k] \subseteq AM[O(k)] \).

implies \( \forall k \geq 2 \) \( IP[k] = AM[O(k)] = AM[2] \)

\( GNI \in IP[2] = AM \)

\[ \Rightarrow \text{ Not known if } GNI \text{ is NP-complete.} \]
If it is \( GNI \) is co-NP-complete.
\[ \Rightarrow \text{ co-NP} \subseteq AM \Rightarrow AM = PH \ (\text{unlikely!}). \]