Proof systems

\[ L = \{ (A, 1^k) : A \text{ is a true mathematical assertion with a proof of length } k \} \]

What is a “proof”?

complexity insight: meaningless unless can be efficiently verified

Interactive Proofs

- Two new ingredients:
  - randomness: verifier tosses coins, errs with some small probability
  - interaction: rather than “reading” proof, verifier interacts with computationally unbounded prover

- NP proof systems lie in this framework: prover sends proof, verifier does not use randomness
Interactive Proofs

• Interactive proof system for L is an interactive protocol (P, V)

\[ \begin{align*}
\text{Prover} & \quad \text{common input: } x \\
\text{Verifier} & \quad \text{# rounds } \leq \text{poly}(|x|) \\
\text{accept/reject} & 
\end{align*} \]

- Completeness: \( x \in L \Rightarrow \Pr[V \text{ accepts in } (P, V)(x)] \geq 2/3 \)
- Soundness: \( x \notin L \Rightarrow \forall P^* \Pr[V \text{ accepts in } (P^*, V)(x)] \leq 1/3 \)
- Efficiency: V is p.p.t. machine
- Repetition: can reduce error to any \( \varepsilon \)

Interactive Proofs

\[ \text{IP} = \{L : L \text{ has an interactive proof system}\} \]

• Observations/questions:
  - Philosophically interesting: captures more broadly what it means to be convinced a statement is true
  - Clearly \( \text{NP} \subseteq \text{IP} \). Potentially larger. How much larger?
  - If larger, randomness is essential (why?)

Graph Isomorphism

• Graphs \( G_0 = (V, E_0) \) and \( G_1 = (V, E_1) \) are isomorphic \( (G_0 \cong G_1) \) if exists a permutation \( \pi: V \rightarrow V \) for which

\[ (x, y) \in E_0 \iff (\pi(x), \pi(y)) \in E_1 \]

• \( \text{GI} = \{(G_0, G_1) : G_0 \cong G_1\} \)
  - In \( \text{NP} \)
  - Not known to be in \( \text{P} \), or \( \text{NP} \)-complete

• \( \text{GNI} = \text{complement of GI} \)
  - Not known to be in \( \text{NP} \)

**Theorem** (GMW): \( \text{GNI} \in \text{IP} \)

- Indication \( \text{IP} \) may be more powerful than \( \text{NP} \)

GNI in IP

• Interactive proof system for GNI:

\[ \begin{align*}
\text{Prover} & \quad \text{input: } (G_0, G_1) \\
\text{Verifier} & \quad \text{flip coin } c \in \{0, 1\}, \text{ pick random } \pi \\
\text{accept/} & \quad \text{if } r = c \\
\text{reject} & 
\end{align*} \]
GNI in IP

• completeness:
  – if $G_0$ not isomorphic to $G_1$ then $H$ is isomorphic to exactly one of $(G_0, G_1)$
  – prover will choose correct $r$

• soundness:
  – if $G_0 \cong G_1$ then prover sees same distribution on $H$ for $c = 0$, $c = 1$
  – no information on $c$ any prover $P^*$ can succeed with probability at most $1/2$

The power of IP

• We showed GNI $\in$ IP
• GNI $\in$ IP suggests IP more powerful than NP, since we don’t know how to show GNI in NP
• GNI in coNP

Theorem (LFKN): coNP $\subseteq$ IP

The power of IP

• Proof idea: input: $\varphi(x_1, x_2, \ldots, x_n)$
  – prover: "I claim $\varphi$ has $k$ satisfying assignments"
  – true iff
    • $\varphi(0, x_2, \ldots, x_n)$ has $k_0$ satisfying assignments
    • $\varphi(1, x_2, \ldots, x_n)$ has $k_1$ satisfying assignments
    • $k = k_0 + k_1$
  – prover sends $k_0, k_1$
  – verifier sends random $c \in \{0, 1\}$
  – prover recursively proves "$\varphi' = \varphi(c, x_2, \ldots, x_n)$ has $k_c$ satisfying assignments"
  – at end, verifier can check for itself.

The power of IP

• Analysis of proof idea:
  – Completeness: $\varphi(x_1, x_2, \ldots, x_n)$ has $k$ satisfying assignments $\Rightarrow$ accept with prob. 1
  – Soundness: $\varphi(x_1, x_2, \ldots, x_n)$ does not have $k$ satisfying assigns. $\Rightarrow$ accept prob. $\leq 1 - 2^n$

  – Why? It is possible that $k$ is only off by one; verifier only catches prover if coin flips $c$ are successive bits of this assignment

The power of IP

• Solution to problem (ideas):
  – replace $\{0, 1\}^n$ with $(F_q)^n$
  – verifier substitutes random field element at each step
  – vast majority of field elements catch cheating prover (rather than just 1)

Theorem: $L = \{ (\varphi, k) : \text{CNF } \varphi \text{ has exactly } k \text{ satisfying assignments} \}$ is in IP

The power of IP

• First step: arithmetization
  – transform $\varphi(x_1, \ldots, x_n)$ into polynomial $p_\varphi(x_1, x_2, \ldots x_n)$ of degree $d$ over a field $F_q$; $q$ prime $> 2^n$
  – recursively:
    • $x_i \rightarrow x_i$
    • $\varphi \rightarrow (1 - p_\varphi)$
    • $\varphi \land \varphi' \rightarrow (p_\varphi p_\varphi')$
    • $\varphi \lor \varphi' \rightarrow 1 - (1 - p_\varphi)(1 - p_\varphi')$
  – for all $x \in \{0, 1\}^n$ we have $p_\varphi(x) = \varphi(x)$
  – degree $d \leq |\varphi|$
  – can compute $p_\varphi(x)$ in poly time from $\varphi$ and $x$
The power of IP

- Prover wishes to prove:
  \[ k = \sum_{i=0}^{2} x_2 = 0.1 \quad \sum_{i=0}^{n} x_i \]
- Define: \( k = \sum_{i=0}^{2} x_2 = 0.1 \quad \sum_{i=0}^{n} x_i \)
- Prover sends: \( k \) for all \( z \in F_q \)
- Verifier: 
  - checks that \( k_0 + k_1 = k \)
  - sends random \( z \in F_q \)
- Continue with proof that \( k = \sum_{i=0}^{2} x_2 = 0.1 \quad \sum_{i=0}^{n} x_i \)
- At end: verifier checks for itself

The actual protocol

<table>
<thead>
<tr>
<th>Prover</th>
<th>Verifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ p_1(x) = \sum x_2 \cdot \sum_{i=0}^{n} x_i ]</td>
<td>[ p_0(0) + p_1(1) ]</td>
</tr>
<tr>
<td>[ p_2(x) = \sum x_2 \cdot \sum_{i=0}^{n} x_i ]</td>
<td>[ z_1 ]</td>
</tr>
<tr>
<td>[ p_3(x) = \sum x_2 \cdot \sum_{i=0}^{n} x_i ]</td>
<td>[ z_2 ]</td>
</tr>
<tr>
<td>[ p_4(x) = \sum x_2 \cdot \sum_{i=0}^{n} x_i ]</td>
<td>[ z_n ]</td>
</tr>
<tr>
<td>[ p_5(x) = \sum x_2 \cdot \sum_{i=0}^{n} x_i ]</td>
<td>[ ]</td>
</tr>
</tbody>
</table>

Analysis of protocol

- Soundness:
  - Let \( p(x) \) be the correct polynomials
  - Let \( p^*(x) \) be the polynomials sent by (cheating) prover
  - if \( (\varphi, k) \notin L \Rightarrow p_1(0) + p_1(1) \neq k \)
  - Either \( p_1(0)^* + p_1(1)^* \neq k \) (and V rejects)
  - Or \( p_i(0)^* \neq i \Rightarrow P_{rel} p_i(z) = p_i(z) \leq d/q \leq |\varphi|/2^n \)
  - Assume \( p_i(0)^* + p_i(1)^* = \) \( p_i(z) \)
  - Either \( p_i(0)^* \neq i \Rightarrow P_{rel} p_i(z) = p_i(z) \) (and V rejects)
  - Or \( p_i(0)^* \neq i \Rightarrow P_{rel} p_i(z) = p_i(z) \) \leq |\varphi|/2^n \)

Analysis of protocol

- Soundness (continued):
  - If verifier does not reject, there must be some \( i \) for which:
    \[ p_i^* \neq p_i \text{ and yet } P_{rel} p_i(z) = p_i(z) \]
  - For each \( i \), probability is \( \leq |\varphi|/2^n \)
  - Union bound: probability that there exists an \( i \) for which the bad event occurs is \( \leq n|\varphi|/2^n \leq \text{poly}(n)/2^n \ll 1/3 \)
Analysis of protocol

- Conclude: \( L = \{ (\varphi, k): \text{CNF } \varphi \text{ has exactly } k \text{ satisfying assignments} \} \) is in IP
  
  - \( L \) is coNP-hard, so coNP \subseteq IP
  
  - Question remains:
    - NP, coNP \subseteq IP. Potentially larger. How much larger?

IP = PSPACE

**Theorem:** (Shamir) IP = PSPACE

- Note: IP \subset PSPACE
  - enumerate all possible interactions, explicitly calculate acceptance probability
  - interaction extremely powerful!
  - An implication: you can interact with master player of Generalized Geography and determine if she can win from the current configuration even if you do not have the power to compute optimal moves!

IP = PSPACE

- need to prove PSPACE \subseteq IP
  - use same type of protocol as for coNP
    - some modifications needed

IP = PSPACE

- protocol for QSAT
  - arithmetization step produces arithmetic expression \( p_\varphi \):
    - \( (\exists x) \varphi \rightarrow \Sigma_{x_i = 0,1} p_\varphi \)
    - \( (\forall x) \varphi \rightarrow \prod_{x_i = 0,1} p_\varphi \)
  - start with QSAT formula in special form ("simple")
    - no occurrence of \( x_i \) separated by more than one "\( \land \)" from point of quantification

IP = PSPACE

- quantified Boolean expression \( \varphi \) is true if and only if \( p_\varphi > 0 \)
- Problem: \( \prod \)'s may cause \( p_\varphi > 2^{2^{2^{2n}}} \)
- Solution: evaluate mod \( 2^n \leq q \leq 2^{3n} \)
- prover sends "good" \( q \) in first round
  - "good" \( q \) is one for which \( p_\varphi \mod q > 0 \)
- Claim: good \( q \) exists
  - # primes in range is at least \( 2^n \)

The QSAT protocol

```
Prover \( k, q, p(x) \)
\( p_c(x) \): remove outer \( \Sigma \) or \( \Pi \) from \( p_\varphi \)
\( p_c(x) \): remove outer \( \Sigma \) or \( \Pi \) from \( p_\varphi [x_i = z_i] \)
\( p_c(x) \): remove outer \( \Sigma \) or \( \Pi \) from \( p_\varphi [x_i = z_i, x_j = z_j] \)
... 
```

Verifier

```
\( p_c(0) + p_c(1) = k? \) or \( p_c(0)p_c(1) = k? \)
pick random \( z_i \) in \( F_q \)
\( p_c(0) + p_c(1) = p_c(z_i)? \) or \( p_c(0)p_c(1) = p_c(z_i)? \)
pick random \( z_i \) in \( F_q \)
... 
```

```
\( p_c(z_i) = p_\varphi [x_1 = z_1, \ldots, x_n = z_n] \)
```
Analysis of the QSAT protocol

• Completeness:
  – if $\phi \in \text{QSAT}$ then honest prover on previous slide will always cause verifier to accept

Analysis of the QSAT protocol

• Soundness:
  – let $p_i(x)$ be the correct polynomials
  – let $p_i^*(x)$ be the polynomials sent by (cheating) prover
  – $\phi \in \text{QSAT} \Rightarrow 0 = p_i(0) + p_i(1) \neq k$ -- $\phi$ is "simple"
  – either $p_i^*(0) + p_i^*(1) \neq k$ (and V rejects)
  – or $p_i^* \neq p_i$ for some $i$ such that:
    – $p_i^*(z_i) = p_i(z_i)$
    – assume $(p_i+1(0) + p_i+1(1) = p_i(z_i)) \neq p_i^*(z_i)$
  – or $p_i^* \neq p_i$ for some $i$ such that:
    – $p_i^*(0) + p_i^*(1) \neq p_i^*(z_i)$ (and V rejects)
    – or $p_i^* \neq p_i$ for some $i$ such that:
      – $p_i^*(z_i) = p_i(z_i)$
      – assume $(p_i+1(0) + p_i+1(1) = p_i(z_i)) \neq p_i^*(z_i)$

• Conclude: QSAT is in IP

Example

$\phi = \forall x \exists y (x \lor \neg y) \land \forall z (x \lor z) \land \exists w (\neg z \lor y \lor \neg w))$

$p_\phi = \prod_{x=0,1} \Sigma_{y=0,1} \left[(x \lor y) \land \prod_{z=0,1} \left[(x \lor z) \lor (z \lor y \lor \neg w)\right]\right]$
Example

\( p_1(0) = \sum_{y=0,1} [(9 + y) \cdot \prod_{z=0,1} [(9z + y(1-z)) + \sum_{w=0,1} (z + y(1-w))] ] \)

Round 2: (prover claims this = 6)
- prover removes outermost “\( \Sigma \)”; sends
  \( p_2(y) = 2y^3 + y^2 + 3y \)
- verifier checks:
  \( p_2(0) + p_2(1) = 0 + 6 = 6 \equiv 6 \pmod{13} \)
- verifier picks randomly: \( z_2 = 3 \)

Example

\( \phi = \forall x \exists y (x \cdot y) \land \exists z ((x \cdot z) \lor (y \cdot z)) \lor \exists w (z \cdot (y \lor z)) \)

\( p_\phi = \prod_{x=0,1} [\sum_{y=0,1} [(x + y) \cdot \prod_{z=0,1} [(xz + y(1-z)) + \sum_{w=0,1} (z + y(1-w))] ] ] \)

Round 3: (prover claims this = 7)
- everyone agrees expression = 12*(…)
- prover removes outermost “\( \prod \)”; sends
  \( p_3(z) = 8z + 6 \)
- verifier checks:
  \( p_3(0) \cdot p_3(1) = 6(14) = 84; 12*84 = 7 \pmod{13} \)
- verifier picks randomly: \( z_3 = 7 \)

Example

\( 12 \cdot p_3(7) = 12 \cdot [(9^7 + 3(1-7)) + \sum_{w=0,1} (7 + 3(1-w))] \)

Round 4: (prover claims = 12*10)
- everyone agrees expression = 12*[6+…]
- prover removes outermost “\( \Sigma \)”; sends
  \( p_4(w) = 10w + 10 \)
- verifier checks:
  \( p_4(0) + p_4(1) = 10 + 20 = 30; 12*[6+30] = 12*10 \pmod{13} \)
- verifier picks randomly: \( z_4 = 2 \)
- Final check:
  \( 12*[9^7 + 3(1-7)] + 7 + 3(1-2)] \equiv 12*[6 + p_4(2)] = 12*[6 + 30] \)

Arthur-Merlin Games

- **IP** permits verifier to keep coin-flips private
  - necessary feature?
  - GNI protocol breaks without it

- **Arthur-Merlin game**: interactive protocol in which coin-flips are public
  - Arthur (verifier) may as well just send results of coin-flips and ask Merlin (prover) to perform any computation Arthur would have done
Arthur-Merlin Games

- Clearly Arthur-Merlin ⊆ IP
  - “private coins are at least as powerful as public coins”

- Proof that IP = PSPACE actually shows
  \[ \text{PSPACE} \subseteq \text{Arthur-Merlin} \subseteq \text{IP} \subseteq \text{PSPACE} \]
  - “public coins are at least as powerful as private coins”!