Fundamental Computer Science

Interactive Proof Systems

Alastair Abbott
alastair.abbott@inria.fr

MoSIG1-M1Info, 2021

Outline

- Interactive proof systems
  - A different approach to computational complexity
  - Relate computational hardness and complexity of proofs
- Deterministic and probabilistic interactions
- Example: Graph non-isomorphism
- Public vs. private coins
- The complexity landscape through the eyes of interactive proof systems

NP Problem Certificates as Proofs

What do proofs have to do with computational complexity?

$\mathbf{NP}$ Problem Certificates as Proofs

What do proofs have to do with computational complexity?

Alternative definition of $\mathbf{NP}$

A language $L \in \mathbf{NP}$ if it has a polynomial time verifier V.

That is, V is a polynomial time TM such that:

- If $x \in L$ then there exists a certificate c such that $V(x, c) =$ accept;
- If $x \notin L$ then $V(x, c) =$ reject *for any string* c.

$\mathbf{NP}$ Problem Certificates as Proofs

What do proofs have to do with computational complexity?

Alternative definition of $\mathbf{NP}$

A language $L \in \mathbf{NP}$ if it has a polynomial time verifier V.

That is, V is a polynomial time TM such that:

- If $x \in L$ then there exists a certificate c such that $V(x, c) =$ accept;
- If $x \notin L$ then $V(x, c) =$ reject *for any string* c.

- The certificate c is a proof that $x \in L$.
- The verifier checks that this proof is correct.

Theorem Proving

"What is intuitively required from a theorem-proving procedure? First, that it is possible to "prove" a true theorem. Second, that it is impossible to "prove" a false theorem. Third, that communicating the proof should be efficient, in the following sense. It does not matter how long must the prover compute during the proving process, but it is essential that the computation required from the verifier is easy."

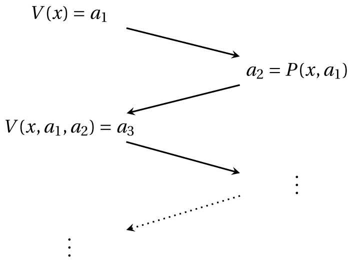
Goldwasser, Micali, Rackoff 1985

Complexity of Theorem Proving

What types of statements (or solutions to problems) can be proven in this way?

- How do we model an abstract theorem proving system in the most general way?
- What computational resources do we give the prover and verifier?
- What type of interaction can they have?

Interactive Proof Systems (Idea)



- P: has unbounded power, goal to convince V a theorem is true
  - e.g., that $x \in L$
- V: doesn't trust P, instead has to verify the proof provided by P with limited computational resources
  - e.g., polynomial time TM, ...
- P and V exchange messages with V eventually accepting P's proof, or rejecting it as incorrect.

Formalising the Interaction

Definition (k-round deterministic $V \leftrightarrow P$ interaction)

Let $k \geq 0$, $V : \{0,1\}^* \to \{0,1\}^* \cup \{\text{accept}, \text{reject}\}$ and $P : \{0,1\}^* \to \{0,1\}^*$.

A k-round deterministic $V \leftrightarrow P$ interaction on input $x \in \{0,1\}^*$, denoted $(V \leftrightarrow P)(x)$, is the sequence of strings $a_1, \ldots, a_k \in \{0,1\}^*$ such that:

$$
\begin{aligned}
a_1 &= V(x) \\
a_2 &= P(x, a_1) \\
&\vdots \\
a_{2i+1} &= V(x, a_1, \ldots, a_{2i}) && \text{for } 2i < k \\
a_{2i+2} &= P(x, a_{,1}, \ldots, a_{2i+1}) && \text{for } 2i+1 < k.
\end{aligned}
$$

The output of the interaction, denoted $\text{out}[(V \leftrightarrow P)(x)]$, is defined as $V(x, a_1, \ldots, a_k)$, which is required to be in $\{\text{accept}, \text{reject}\}$.

Deterministic Interactive Proof Systems

Definition ($\mathbf{dIP}$)

A language L has a k-round deterministic Interactive Proof system if there's a deterministic TM V that, on input $x, a_1, \dots, a_i$, runs in time $\text{poly}(n)$ and satisfies:

- Completeness: $x \in L \implies \exists P \ \text{out}[(V \leftrightarrow P)(x)] = \text{accept}$;
- Soundness: $x \notin L \implies \forall P \ \text{out}[(V \leftrightarrow P)(x)] = \text{reject}$.

The class $\mathbf{dIP}[k]$ contains all languages with a k-round deterministic Interactive Proof system. We define

$$\mathbf{dIP} = \mathbf{dIP}[\text{poly}(n)] := \bigcup_c \mathbf{dIP}[n^c],$$

allowing the number of rounds to depend on $n = |x|$.

Deterministic Interactive Proof Systems

Definition (dIP)

A language L has a k-round deterministic Interactive Proof system if there's a deterministic TM V that, on input $x, a_1, \ldots, a_i$, runs in time $\text{poly}(n)$ and satisfies:

- Completeness: $x \in L \implies \exists P \; \text{out}[(V \leftrightarrow P)(x)] = \text{accept};$
- Soundness: $x \notin L \implies \forall P \; \text{out}[(V \leftrightarrow P)(x)] = \text{reject}.$

The class $\mathbf{dIP}[k]$ contains all languages with a k-round deterministic Interactive Proof system. We define

$$\mathbf{dIP} = \mathbf{dIP}[\text{poly}(n)] := \bigcup_c \mathbf{dIP}[n^c],$$

allowing the number of rounds to depend on $n = |x|$.

- Recall: no assumption of computational power of P.
- Messages satisfy $|a_i| \in \text{poly}(n)$. (Why?)

Example: $\mathbf{dIP}$ protocol for 3SAT

Recall 3SAT: Boolean formulae $\mathcal{F} = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ where each clase C_i is disjunction of 3 literals, e.g. $C_1 = (x_1 \vee \bar{x}_4 \vee \bar{x}_3)$.

Example: $\mathbf{dIP}$ protocol for 3SAT

Recall 3SAT: Boolean formulae $\mathcal{F} = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ where each clase C_i is disjunction of 3 literals, e.g. $C_1 = (x_1 \vee \bar{x}_4 \vee \bar{x}_3)$.

A simple interactive proof system:

- The Verifier asks the Prover for the values of the literals in one clause at a time and records the answers.
- After $2m$ rounds, the verifier checks the provers answers are consistent and each clause evaluates to true.

Example: $\mathbf{dIP}$ protocol for 3SAT

Recall 3SAT: Boolean formulae $\mathcal{F} = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ where each clase C_i is disjunction of 3 literals, e.g. $C_1 = (x_1 \vee \bar{x}_4 \vee \bar{x}_3)$.

A simple interactive proof system:

- The Verifier asks the Prover for the values of the literals in one clause at a time and records the answers.
- After $2m$ rounds, the verifier checks the provers answers are consistent and each clause evaluates to true.

Clearly correct:

- If $\mathcal{F}$ is satisfiable, P can prove it by correctly giving the values of the literals.
- If $\mathcal{F}$ not satisfiable, no way P can provide values making V accept.

Example: $\mathbf{dIP}$ protocol for 3SAT

Recall 3SAT: Boolean formulae $\mathcal{F} = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ where each clase C_i is disjunction of 3 literals, e.g. $C_1 = (x_1 \vee \bar{x}_4 \vee \bar{x}_3)$.

A simple interactive proof system:

- The Verifier asks the Prover for the values of the literals in one clause at a time and records the answers.
- After $2m$ rounds, the verifier checks the provers answers are consistent and each clause evaluates to true.

Clearly correct:

- If $\mathcal{F}$ is satisfiable, P can prove it by correctly giving the values of the literals.
- If $\mathcal{F}$ not satisfiable, no way P can provide values making V accept.

But do we really need multi-round interaction here?

dIP and NP

In the previous example we could just have asked for the full certificate. . .

dIP and NP

In the previous example we could just have asked for the full certificate...

Theorem

$\mathbf{NP} = \mathbf{dIP}$.

dIP and NP

In the previous example we could just have asked for the full certificate...

Theorem

$\mathbf{NP} = \mathbf{dIP}$.

Proof ($\mathbf{NP} \subseteq \mathbf{dIP}[2] \subseteq \mathbf{dIP}$):

- The certificate/verifier definition of $\mathbf{NP}$ is essentially a 2-round deterministic proof system.
- Given a polynomial time verifier V' for a language L:
  - $a_1 = V(x) = \epsilon$
  - $a_2 = P(x, \epsilon) = c$
  - $V(x, \epsilon, c) = V'(c) = \mathrm{accept}$ if c a valid certificate, reject otherwise.
- V clearly polynomial time; completeness and soundness satisfied by definition of polynomial time verifier.

$\mathbf{NP} = \mathbf{dIP}$

Proof ($\mathbf{dIP} \subseteq \mathbf{NP}$):

- Assume $L \in \mathbf{dIP}$ and let V be the dIP Verifier for L.
- For any $x \in L$ we use, as a polynomial-time verifiable certificate, the transcript of the k-round interaction causing V to accept: $c = (a_1, \ldots, a_k)$, where k is polynomial in n.
- This can be verified in polynomial time by checking that $V(x) = a_1$, $V(x, a_1, a_2) = a_3$, $\ldots$, $V(x, a_1, \ldots, a_k) = \text{accept}$.

$\mathbf{NP} = \mathbf{dIP}$

Proof ($\mathbf{dIP} \subseteq \mathbf{NP}$):

- Assume $L \in \mathbf{dIP}$ and let V be the dIP Verifier for L.
- For any $x \in L$ we use, as a polynomial-time verifiable certificate, the transcript of the k-round interaction causing V to accept: $c = (a_1, \ldots, a_k)$, where k is polynomial in n.
- This can be verified in polynomial time by checking that $V(x) = a_1$, $V(x, a_1, a_2) = a_3$, ..., $V(x, a_1, \ldots, a_k) = \text{accept}$.
- If $x \in L$ then such a certificate exists and can be verified in polynomial time.
- Conversely, if a certificate satisfies these conditions, we can define a prover P satisfying $P(x, a_1) = a_2$, $P(x, a_1, a_2, a_3) = a_4$, etc. This satisfies $\text{out}[(V \leftrightarrow P)(x)] = \text{accept}$ so $x \in L$.

Beyond Deterministic Proof Systems?

- The more complex interaction model we defined didn't give us any more power than the simple certificate-verifier proof system of $\mathbf{NP}$.
  - Is that surprising?

Beyond Deterministic Proof Systems?

- The more complex interaction model we defined didn't give us any more power than the simple certificate-verifier proof system of $\mathbf{NP}$.
  - Is that surprising?

- How could we go beyond $\mathbf{dIP}$ to obtain a benefit from interaction?

Beyond Deterministic Proof Systems?

- The more complex interaction model we defined didn't give us any more power than the simple certificate-verifier proof system of $\mathbf{NP}$.
  - Is that surprising?
- How could we go beyond $\mathbf{dIP}$ to obtain a benefit from interaction?
  - Give more power to the Verifier (e.g., non-determinism, probabilistic choices, ...).
  - Allow multiple independent provers.
  - ...

Beyond Deterministic Proof Systems?

- The more complex interaction model we defined didn't give us any more power than the simple certificate-verifier proof system of $\mathbf{NP}$.
  - Is that surprising?

- How could we go beyond $\mathbf{dIP}$ to obtain a benefit from interaction?
  - Give more power to the Verifier (e.g., non-determinism, probabilistic choices, . . .).
  - Allow multiple independent provers.
  - . . .

Probabilistic Verifiers

We will allow the Verifier to make probabilistic choices in verifying a proof.

- Intuitively: V may ask P questions at random, making it harder for P to cheat…
- but we'll need to modify the completeness and soundness requirements.

Probabilistic Verifiers

We will allow the Verifier to make probabilistic choices in verifying a proof.

- Intuitively: V may ask P questions at random, making it harder for P to cheat...
- but we'll need to modify the completeness and soundness requirements.

We allow V to be probabilistic Turing machine.

- Formally, $M = (K, \Sigma, \Gamma, \delta_0, \delta_1, s, H)$, where the transitions δ_0 and δ_1 are chosen with probability $1/2$.
- Equivalently: a probabilistic Turing machine is a deterministic Turing machine with an extra tape containing random bits.
  - Its output is a random variable over the random bits on that tape.

Probabilistic Verifiers

We will allow the Verifier to make probabilistic choices in verifying a proof.

- Intuitively: V may ask P questions at random, making it harder for P to cheat...
- but we'll need to modify the completeness and soundness requirements.

We allow V to be probabilistic Turing machine.

- Formally, $M = (K, \Sigma, \Gamma, \delta_0, \delta_1, s, H)$, where the transitions δ_0 and δ_1 are chosen with probability $1/2$.
- Equivalently: a probabilistic Turing machine is a deterministic Turing machine with an extra tape containing random bits.
  - Its output is a random variable over the random bits on that tape.

Claim

For every polynomial-time probabilistic Turing machine M there exists a deterministic Turing machine N and a computable polynomial p such that

$$\forall x, y \in \{0,1\}^*,\ \Pr[M(x) = y] = \Pr_{r \in \{0,1\}^{p(n)}}[N(x,r) = y].$$

(Probabilistic) Interactive Proof Systems

k-round probabilistic $V \leftrightarrow P$ interaction

Let $k \geq 0$ and p be a computable polynomial. On input $x \in \{0,1\}^*$, the interaction now proceeds as follows:

1. V is given (or chooses) a random string $r \in \{0,1\}^{p(n)}$ with probability $\Pr(r) = 1/2^{p(n)}$.
2. V and P exchange messages to obtain the following sequences of strings $a_1, \ldots, a_k \in \{0,1\}^*$:

$$
\begin{aligned}
a_1 &= V(x, r) \\
a_2 &= P(x, a_1) \\
a_3 &= V(x, r, a_1, a_2) \\
a_4 &= P(x, a_1, a_2, a_3) \\
&\vdots
\end{aligned}
$$

The output of the interaction, $\mathrm{out}[(V \leftrightarrow P)(x)] = V(x, r, a_1, \ldots, a_k)$ is now a random variable over r.

Probabilistic Verifiers and IP

We can no longer ask for perfect completeness and soundness: we would recover $\mathbf{dIP}$.

Probabilistic Verifiers and IP

We can no longer ask for perfect completeness and soundness: we would recover **dIP**.

- Instead: V should correctly accept/reject with high probability.

Probabilistic Verifiers and IP

We can no longer ask for perfect completeness and soundness: we would recover $\mathbf{dIP}$.

- Instead: V should correctly accept/reject with high probability.

Definition ($\mathbf{IP}$)

A language L has a k-round probabilistic Interactive Proof system if there's a deterministic TM V and polynomial p that, on inputs $x, r, a_1, \ldots, a_k$ runs in time $\mathrm{poly}(n)$ and satisfies:

- Completeness: $x \in L \implies \exists P \;\; \Pr[\mathrm{out}(V \leftrightarrow P)(x) = \mathrm{accept}] \geq 2/3$;
- Soundness: $x \notin L \implies \forall P \;\; \Pr[\mathrm{out}(V \leftrightarrow P)(x)] = \mathrm{reject}] \geq 2/3$,

where the probabilities are over $r \in \{0,1\}^{p(n)}$.

The class $\mathbf{IP}[k]$ contains all languages with a k-round probabilistic Interactive Proof system. We define

$$\mathbf{IP} = \mathbf{IP}[\mathrm{poly}(n)] := \bigcup_c \mathbf{IP}[n^c].$$

Boosting the Correctness

The bounds of $2/3$ are arbitrary!

Lemma

The definition of $\mathbf{IP}$ remains unchanged if we replace $2/3$ by $1 - 1/2^{n^c}$ for any fixed $c > 0$.

Boosting the Correctness

The bounds of $2/3$ are arbitrary!

Lemma

The definition of $\mathbf{IP}$ remains unchanged if we replace $2/3$ by $1 - 1/2^{n^c}$ for any fixed $c > 0$.

Proof (idea):

- V and P repeat their interaction protocol some number m times.
- V finally takes the majority output from the m repetitions.
- By the Chernoff bound, the protocol succeeds with probability $1 - 1/2^{\Omega(m)}$.
- m can be taken to be polynomial in n while maintaining an overall polynomial time protocol.

Boosting the Correctness

The bounds of $2/3$ are arbitrary!

Lemma

The definition of $\mathbf{IP}$ remains unchanged if we replace $2/3$ by $1 - 1/2^{n^c}$ for any fixed $c > 0$.

Proof (idea):

- V and P repeat their interaction protocol some number m times.
- V finally takes the majority output from the m repetitions.
- By the Chernoff bound, the protocol succeeds with probability $1 - 1/2^{\Omega(m)}$.
- m can be taken to be polynomial in n while maintaining an overall polynomial time protocol.

- This is a standard argument in the analysis of probabilistic algorithms.
- Actually, here the repetitions can even be done in parallel to keep the same number of rounds k.

Does Randomness Help?

Does going from $\mathbf{dIP}$ to $\mathbf{IP}$ actually add anything?

Does Randomness Help?

Does going from $\mathbf{dIP}$ to $\mathbf{IP}$ actually add anything?

- Probabilistic Turing machines are not thought to be substantially more powerful than deterministic Turing machines.
  - $\mathbf{P}$ vs. $\mathbf{BPP}$
  - Does $\mathbf{IP}$ contain anything not in $\mathbf{NP}$?

Example: Graph (Non)isomorphism

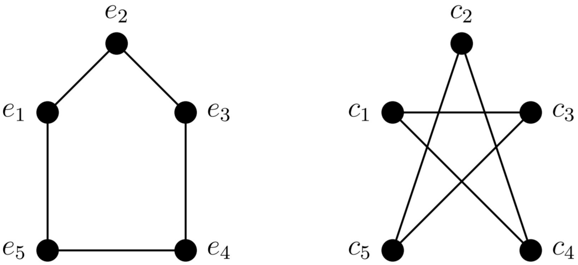
Definition

Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic if they are the same up to a relabelling of the vertices; i.e., if $G_1 = \pi(G_2)$ for some permutation π of the labels of the vertices. We write $G_1 \cong G_2$.

Example: Graph (Non)isomorphism

Definition

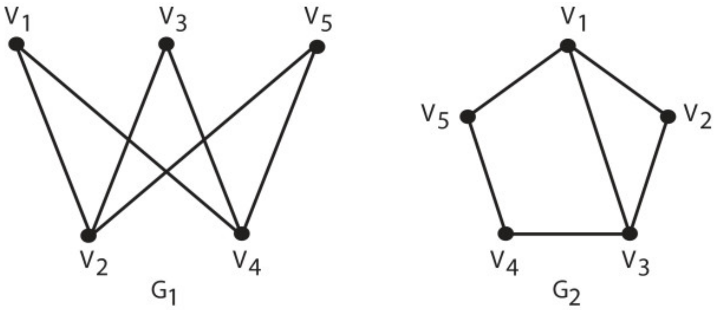
Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic if they are the same up to a relabelling of the vertices; i.e., if $G_1 = \pi(G_2)$ for some permutation π of the labels of the vertices. We write $G_1 \cong G_2$.



Example: Graph (Non)isomorphism

Definition

Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic if they are the same up to a relabelling of the vertices; i.e., if $G_1 = \pi(G_2)$ for some permutation π of the labels of the vertices. We write $G_1 \cong G_2$.



Example: Graph (Non)isomorphism

Consider the languages:

$$\mathrm{ISO} = \{\langle G_1, G_2\rangle \mid G_1 \cong G_2\}, \qquad \mathrm{NONISO} = \{\langle G_1, G_2\rangle \mid G_1 \ncong G_2\}.$$

- $\mathrm{ISO} \in \mathbf{NP}$: the permutation π is a certificate.
- ISO not though to be in $\mathbf{P}$, but also not thought to be $\mathbf{NP}$-complete!
- $\mathrm{NONISO} \in \mathbf{coNP}$, but not thought to be in $\mathbf{NP}$.

Example: Graph (Non)isomorphism

Consider the languages:

$$\mathrm{ISO} = \{\langle G_1, G_2\rangle \mid G_1 \cong G_2\}, \qquad \mathrm{NONISO} = \{\langle G_1, G_2\rangle \mid G_1 \ncong G_2\}.$$

- $\mathrm{ISO} \in \mathbf{NP}$: the permutation π is a certificate.
- ISO not though to be in $\mathbf{P}$, but also not thought to be $\mathbf{NP}$-complete!
- $\mathrm{NONISO} \in \mathbf{coNP}$, but not thought to be in $\mathbf{NP}$.

Theorem

$\mathrm{NONISO} \in \mathbf{IP}$.

Example: Graph (Non)isomorphism

The protocol is remarkably simple:

V: Pick $i \in \{1, 2\}$ uniformly at random. Choose a random permutation π and permute the vertices of G_i to obtain $H = \pi(G_i)$. Send H to P.

P: Identify which of G_1 or G_2 was used to produce H. Let G_j be that graph. Send j to V.

V: Accept if $i = j$; otherwise reject.

Example: Graph (Non)isomorphism

The protocol is remarkably simple:

V: Pick $i \in \{1, 2\}$ uniformly at random. Choose a random permutation π and permute the vertices of G_i to obtain $H = \pi(G_i)$. Send H to P.

P: Identify which of G_1 or G_2 was used to produce H. Let G_j be that graph. Send j to V.

V: Accept if $i = j$; otherwise reject.

Proof of Completeness

Completeness: $x \in L \implies \exists P \ \Pr[\mathrm{out}(V \leftrightarrow P)(x) = \mathrm{accept}] \geq 2/3.$

Example: Graph (Non)isomorphism

The protocol is remarkably simple:

V: Pick $i \in \{1, 2\}$ uniformly at random. Choose a random permutation π and permute the vertices of G_i to obtain $H = \pi(G_i)$. Send H to P.

P: Identify which of G_1 or G_2 was used to produce H. Let G_j be that graph. Send j to V.

V: Accept if $i = j$; otherwise reject.

Proof of Completeness

Completeness: $x \in L \implies \exists P \ \Pr[\text{out}(V \leftrightarrow P)(x) = \text{accept}] \geq 2/3$.

Proof: If $G_1 \ncong G_2$ then the above protocol gives $\Pr[V \text{ accepts}] = 1 \geq 2/3$. Indeed, since P can have unbounded power and H must be isomorphic to exactly one of G_1 and G_2, this P can always exist (e.g., P can try all $\mathcal{O}(n!)$ permutations).

Example: Graph (Non)isomorphism

- V: Pick $i \in \{1, 2\}$ uniformly at random. Choose a random permutation π and permute the vertices of G_i to obtain $H = \pi(G_i)$. Send H to P.
- P: Identify which of G_1 or G_2 was used to produce H. Let G_j be that graph. Send j to V.
- V: Accept if $i = j$; otherwise reject.

Proof of Soundness

Soundness: $x \notin L \implies \forall P \ \Pr[\mathrm{out}(V \leftrightarrow P)(x) = \mathrm{reject}] \geq 2/3$.

Example: Graph (Non)isomorphism

V: Pick $i \in \{1, 2\}$ uniformly at random. Choose a random permutation π and permute the vertices of G_i to obtain $H = \pi(G_i)$. Send H to P.

P: Identify which of G_1 or G_2 was used to produce H. Let G_j be that graph. Send j to V.

V: Accept if $i = j$; otherwise reject.

Proof of Soundness

Soundness: $x \notin L \implies \forall P \; \Pr[\text{out}(V \leftrightarrow P)(x) = \text{reject}] \geq 2/3$.

Proof:

- If $G_1 \cong G_2$ then H is isomorphic to both G_1 and G_2, and could have been obtained from either graph.
- Hence, P can do no better than guessing i at random: $\Pr[V \text{ reject}] \geq 1/2$.
- This can be increased above $2/3$ by repeating the protocol (in parallel or sequentially), and accepting only if P is correct every time. For m repetitions, one has $\Pr[V \text{ accepts}] \leq 1/2^m$.

Discussion on NONISO

Since we don't think $\text{NONISO} \in \mathbf{NP}$, it seems randomness really does help!

- We only needed 2 rounds to solve NONISO.
  - Is using $\text{poly}(n)$ rounds ever useful?
- How much more powerful are interactive proof systems?

Discussion on NONISO

Since we don't think $\mathrm{NONISO} \in \mathbf{NP}$, it seems randomness really does help!

- We only needed 2 rounds to solve NONISO.
  - Is using $\mathrm{poly}(n)$ rounds ever useful?
- How much more powerful are interactive proof systems?

If $G_1 \not\cong G_2$, note that V only learns this fact, but not how G_1 and G_2 are related (what π is)

- This is the basis for zero-knowledge proofs, which have important applications for cryptography
- E.g., proving you know a password without revealing *what* the password is.

Public vs. Private Coins

Our protocol for NONISO relied crucially on the fact that P didn't know which $i \in \{1, 2\}$ V chose.

- We defined private coin interactive proof systems: only V was given the random string r.

Public vs. Private Coins

Our protocol for NONISO relied crucially on the fact that P didn't know which $i \in \{1,2\}$ V chose.

- We defined private coin interactive proof systems: only V was given the random string r.

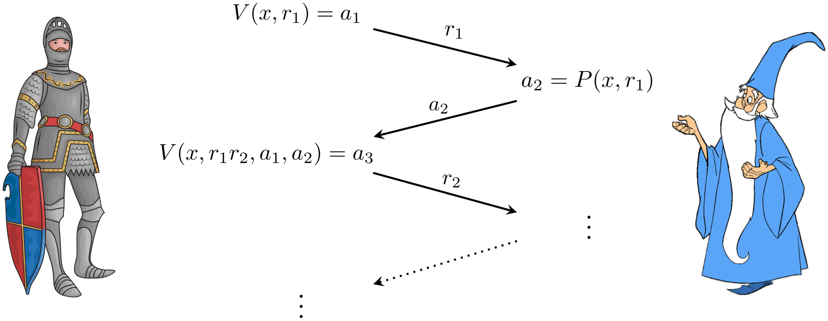
We can also consider public coin interactive proof systems, where P can see also the random bits V uses between each round.

- Not unreasonable if we assume an all-powerful Prover and want to know what proofs he can convince the Verifier to believe.
- *A priori* this could make the prover more powerful or, conversely, restrict how the Verifier can check the proof.

Arthur-Merlin Interactions

Public coin interactive protocols are usually called Arthur-Merlin protocols.

- Arthur has a random string $r = (r_1, r_2, \dots)$, with $r_i \in \{0,1\}^*$.
- He sends the string r_i he uses in each round.



- We can consider that Arthur just successively sends random bits to Merlin.
- At the end, he performs a final computation to decide to accept or reject.

Arthur-Merlin

Definition ($\mathbf{AM}$)

The class $\mathbf{AM}[k]$ is defined as the subset of $\mathbf{IP}[k]$ obtained when we restrict ourselves to Verifiers V that are given random strings $r = (r_1, \ldots, r_{\lceil k/2 \rceil})$ and satisfy, for all $i \geq 0$, $V(x, r, r_1, a_2, r_2, \ldots, a_{2i}) = r_i$ for all strings $a_2, a_4, \ldots, a_{2i} \in \{0,1\}^*$.

We call $\mathbf{AM} = \mathbf{AM}[2]$.

Arthur-Merlin

Definition ($\mathbf{AM}$)

The class $\mathbf{AM}[k]$ is defined as the subset of $\mathbf{IP}[k]$ obtained when we restrict ourselves to Verifiers V that are given random strings $r = (r_1, \ldots, r_{\lceil k/2 \rceil})$ and satisfy, for all $i \geq 0$, $V(x, r, r_1, a_2, r_2, \ldots, a_{2i}) = r_i$ for all strings $a_2, a_4, \ldots, a_{2i} \in \{0,1\}^*$.

We call $\mathbf{AM} = \mathbf{AM}[2]$.

That is:

- $V(x, r) = r_1$, $V(x, r, r_1, a_2) = r_2$, etc.
- $\Pr[V(x, r, r_1, a_2, \ldots, a_k) = \text{accept}] \geq 2/3$, etc.

Arthur-Merlin

Definition ($\mathbf{AM}$)

The class $\mathbf{AM}[k]$ is defined as the subset of $\mathbf{IP}[k]$ obtained when we restrict ourselves to Verifiers V that are given random strings $r = (r_1, \ldots, r_{\lceil k/2 \rceil})$ and satisfy, for all $i \geq 0$, $V(x, r, r_1, a_2, r_2, \ldots, a_{2i}) = r_i$ for all strings $a_2, a_4, \ldots, a_{2i} \in \{0,1\}^*$.

We call $\mathbf{AM} = \mathbf{AM}[2]$.

That is:

- $V(x, r) = r_1$, $V(x, r, r_1, a_2) = r_2$, etc.
- $\Pr[V(x, r, r_1, a_2, \ldots, a_k) = \text{accept}] \geq 2/3$, etc.

- Clearly, for all k $\mathbf{AM}[k] \subseteq \mathbf{IP}[k]$.
- How much power do we lose by going to public keys?

Simulating Private Coins with Public Coins

Theorem (Goldwasser-Sipser, 1987)

For every $f : \mathbb{N} \to \mathbb{N}$ with $f(n)$ computable in time $poly(n)$

$$\mathbf{IP}[f(n)] \subseteq \mathbf{AM}[f(n) + 2].$$

Simulating Private Coins with Public Coins

Theorem (Goldwasser-Sipser, 1987)

For every $f : \mathbb{N} \to \mathbb{N}$ with $f(n)$ computable in time $poly(n)$

$$\mathbf{IP}[f(n)] \subseteq \mathbf{AM}[f(n) + 2].$$

- Public coins can simulate private ones with only 2 extra rounds of communication!
- This should be surprising.

Simulating Private Coins with Public Coins

Theorem (Goldwasser-Sipser, 1987)

For every $f : \mathbb{N} \to \mathbb{N}$ with $f(n)$ computable in time $poly(n)$

$$\mathbf{IP}[f(n)] \subseteq \mathbf{AM}[f(n) + 2].$$

- Public coins can simulate private ones with only 2 extra rounds of communication!
- This should be surprising.

Corollary

$\mathbf{AM}[\text{poly}(n)] = \mathbf{IP}$.

- Using private coins is convenient, but not change much.

NONISO with Public Keys

Rather than proving the Goldwasser-Sipser theorem, let us instead sketch a proof of the following result:

Theorem

$\mathrm{NONISO} \in \mathbf{AM}$.

NONISO with Public Keys

Rather than proving the Goldwasser-Sipser theorem, let us instead sketch a proof of the following result:

Theorem

$\mathrm{NONISO} \in \mathbf{AM}$.

Proof (idea):

- Rephrase the problem quantitatively:

$$S = \{H \mid H \cong G_1 \text{ or } H \cong G_2\}.$$

- The size of S tells us is $G_1 \cong G_2$:

$$\text{if } G_1 \not\cong G_2 \text{ then } |S| = 2n!, \qquad \text{if } G_1 \cong G_2 \text{ then } |S| = n!$$

- We assume for simplicity that G_1, G_2 each have exactly $n!$ isomorphic graphs.

NONISO with Public Keys

Rather than proving the Goldwasser-Sipser theorem, let us instead sketch a proof of the following result:

Theorem

NONISO $\in$ $\mathbf{AM}$.

Proof (idea):

- Rephrase the problem quantitatively:

$$S = \{H \mid H \cong G_1 \text{ or } H \cong G_2\}.$$

- The size of S tells us is $G_1 \cong G_2$:

$$\text{if } G_1 \ncong G_2 \text{ then } |S| = 2n!, \qquad \text{if } G_1 \cong G_2 \text{ then } |S| = n!$$

- We assume for simplicity that G_1, G_2 each have exactly $n!$ isomorphic graphs.
- P must convince V that $|S|$ is much larger than $n!$
- Note that we can efficiently certify that a graph H is in S.

Tool: Pairwise Independent Hash Functions

To do this, we make use of Hash functions.

Definition (Pairwise independent hash functions)

Let $\mathcal{H}_{n,k} \subseteq \{h \mid h : \{0,1\}^n \to \{0,1\}^k\}$ be a set of functions. We say that $\mathcal{H}_{n,k}$ is pairwise independent if for all $x, x' \in \{0,1\}^n$ with $x \neq x'$ and all $y, y' \in \{0,1\}^k$:

$$\Pr_{h \in \mathcal{H}_{n,k}} [h(x) = y \wedge h(x') = y'] = 1/2^{2k}.$$

- This implies that $\Pr_h[h(x) = y] = 1/2^k$ for all x, y.

Tool: Pairwise Independent Hash Functions

To do this, we make use of Hash functions.

Definition (Pairwise independent hash functions)

Let $\mathcal{H}_{n,k} \subseteq \{h \mid h : \{0,1\}^n \to \{0,1\}^k\}$ be a set of functions. We say that $\mathcal{H}_{n,k}$ is pairwise independent if for all $x, x' \in \{0,1\}^n$ with $x \neq x'$ and all $y, y' \in \{0,1\}^k$:

$$\Pr_{h \in \mathcal{H}_{n,k}}[h(x) = y \wedge h(x') = y'] = 1/2^{2k}.$$

- This implies that $\Pr_h[h(x) = y] = 1/2^k$ for all x, y.

Lemma

For all $n, k > 0$ there exist efficiently computable pairwise independent hash functions.

Set Lower Bound Protocol

We want a public coin protocol for the following problem:

Set Lower Bound Problem

Let $S \subseteq \{0,1\}^n$ be a set such that the membership $s \in S$ can be efficiently certified, and $K \in \mathbb{N}$.

Goal: P wants to convince V that $|S| \geq K$. V should reject with good probability if $|S| \leq K/2$.

Clearly, solving this problem allows us to solve NONISO.

Set Lower Bound Protocol

We want a public coin protocol for the following problem:

Set Lower Bound Problem

Let $S \subseteq \{0,1\}^n$ be a set such that the membership $s \in S$ can be efficiently certified, and $K \in \mathbb{N}$.

Goal: P wants to convince V that $|S| \geq K$. V should reject with good probability if $|S| \leq K/2$.

Clearly, solving this problem allows us to solve NONISO.

Idea:

- V asks P to find a $x \in S$ such that $h(x) = y$, for randomly chosen h, y.
- Such an x may not exist, but the probability it does exist is larger if $|S|$ is larger.
- If x exists, P can find it (unbounded power), and V can easily check $h(x) = y$.
- This occurs with higher probability if $|S| \geq K$.

Set Lower Bound Protocol

Let k be an integer satisfying $2^{k-2} < K \leq 2^{k-1}$ so that $1/4 < K/2^k \leq 1/2$.

V: Randomly pick a $h \in \mathcal{H}_{n,k}$ and a $y \in \{0,1\}^k$. Send h, y to P (equivalently, the coins used to pick them).

P: Try to find an $x \in S$ such that $h(x) = y$. Send x to V, and also a certificate that $x \in S$.

V: Verify efficiently that $x \in S$ and $h(x) = y$ then accept; otherwise reject.

Set Lower Bound Protocol

Let k be an integer satisfying $2^{k-2} < K \leq 2^{k-1}$ so that $1/4 < K/2^k \leq 1/2$.

V: Randomly pick a $h \in \mathcal{H}_{n,k}$ and a $y \in \{0,1\}^k$. Send h, y to P (equivalently, the coins used to pick them).

P: Try to find an $x \in S$ such that $h(x) = y$. Send x to V, and also a certificate that $x \in S$.

V: Verify efficiently that $x \in S$ and $h(x) = y$ then accept; otherwise reject.

Proof (soundness): Assume $|S| \leq K/2$ (so V should reject).

- V can only be made to accept if $\exists x : h(x) = y$, i.e. if $y \in h(S)$.

Set Lower Bound Protocol

Let k be an integer satisfying $2^{k-2} < K \leq 2^{k-1}$ so that $1/4 < K/2^k \leq 1/2$.

V: Randomly pick a $h \in \mathcal{H}_{n,k}$ and a $y \in \{0,1\}^k$. Send h, y to P (equivalently, the coins used to pick them).

P: Try to find an $x \in S$ such that $h(x) = y$. Send x to V, and also a certificate that $x \in S$.

V: Verify efficiently that $x \in S$ and $h(x) = y$ then accept; otherwise reject.

Proof (soundness): Assume $|S| \leq K/2$ (so V should reject).

- V can only be made to accept if $\exists x : h(x) = y$, i.e. if $y \in h(S)$.

$$\begin{aligned}
\Pr[V \text{ accepts}] &= \Pr[\text{randomly chosen } y \in h(S)] \\
&= |h(S)|/2^k \\
&\leq |S|/2^k \\
&\leq \frac{K}{2 \cdot 2^k} = \left(\frac{1}{2}\right)\left(\frac{K}{2^k}\right).
\end{aligned}$$

Set Lower Bound Protocol

Proof (completeness): Assume $|S| \geq K$ (so V should accept).

- For simplicity, further assume $|S| \leq 2^{k-1}$ (we had $K \leq 2^{k-1}$).
- Assume P follows the protocol. What is the probability V accepts?

Set Lower Bound Protocol

Proof (completeness): Assume $|S| \geq K$ (so V should accept).

- For simplicity, further assume $|S| \leq 2^{k-1}$ (we had $K \leq 2^{k-1}$).
- Assume P follows the protocol. What is the probability V accepts?

$$\begin{aligned}
\Pr[V \text{ accepts}] &= \Pr[\exists x \in S : h(x) = y] \\
&= \Pr[\vee_{x \in S}\ h(x) = y] \\
&\geq \sum_{x \in S} \Pr[h(x) = y] - \sum_{x,x' \in S : x \neq x'} \Pr[h(x) = y \text{ and } h(x') = y] \\
&= \sum_{x \in S} 2^{-k} - \sum_{x,x' \in S : x \neq x'} 2^{-2k} \\
&= |S| 2^{-k} - \frac{|S|(|S|-1)}{2} 2^{-2k} \\
&\geq |S|/2^k \cdot \left(1 - |S|/2^{k+1}\right) \\
&\geq K/2^k \cdot (1 - 2^{k-1}/2^{k+1}) \\
&= \left(\frac{3}{4}\right)\left(\frac{K}{2^k}\right).
\end{aligned}$$

Set Lower Bound Protocol

We have:

- if $|S| \leq K/2$, $\Pr[V \text{ accepts}] \leq \frac{1}{2} \cdot \frac{K}{2^k}$;
- if $|S| \geq K$, $\Pr[V \text{ accepts}] \geq \frac{3}{4} \cdot \frac{K}{2^k}$.

- Since $1/4 < K/2^k \leq 1/2$, the difference $\frac{1}{4} \cdot \frac{K}{2^k} > 1/16$.

Set Lower Bound Protocol

We have:

- if $|S| \leq K/2$, $\Pr[V \text{ accepts}] \leq \frac{1}{2} \cdot \frac{K}{2^k}$;
- if $|S| \geq K$, $\Pr[V \text{ accepts}] \geq \frac{3}{4} \cdot \frac{K}{2^k}$.

- Since $1/4 < K/2^k \leq 1/2$, the difference $\frac{1}{4} \cdot \frac{K}{2^k} > 1/16$.

This isn't quite what we want, but:

- Recall V knows $K/2^k$.
- Since there's a nonzero gap between the probabilities, it can be amplified:
  - V and P run the protocol m times in parallel.
  - V accepts if the protocol accepts in at least $\frac{5}{8} \times \frac{K}{2^k}$ of the repetitions.
  - One then has $\Pr[V \text{ accepts}] > 1/2$ if $|S| \geq K$, and $\Pr[V \text{ accepts}] < 1/2$ if $|S| \leq K/2$.
  - This can be amplified to above $2/3$ with further repetition.

Remarks on the Goldwasser-Sipser Theorem

- By doing all repetition in parallel, we have a 2-round protocol for the Set Lower Bound problem.
- We then indeed have $\mathrm{NONISO} \in \mathbf{AM}$.

Remarks on the Goldwasser-Sipser Theorem

- By doing all repetition in parallel, we have a 2-round protocol for the Set Lower Bound problem.
- We then indeed have $\mathrm{NONISO} \in \mathbf{AM}$.

A similar approach can be used to prove $\mathbf{IP}[k] \subseteq \mathbf{AM}[k+2]$.

- S corresponds (roughly) to the set of possible messages a private coin verifier could send.
- P has to prove that certain messages are likely to be sent by a private coin verifier, and that a private coin verifier is likely to accept.
- One proceeds round by round... There are some technicalities, but this captures the main idea.

Remarks on the Goldwasser-Sipser Theorem

- By doing all repetition in parallel, we have a 2-round protocol for the Set Lower Bound problem.
- We then indeed have $\mathrm{NONISO} \in \mathbf{AM}$.

A similar approach can be used to prove $\mathbf{IP}[k] \subseteq \mathbf{AM}[k+2]$.

- S corresponds (roughly) to the set of possible messages a private coin verifier could send.
- P has to prove that certain messages are likely to be sent by a private coin verifier, and that a private coin verifier is likely to accept.
- One proceeds round by round... There are some technicalities, but this captures the main idea.

Even though it seems like public coins make it harder for the Verifier, this isn't really the case !

Collapsing the Hierarchy

- We could solve NONISO with a constant number of rounds (2).
- What kind of problem might we need more rounds for?

Collapsing the Hierarchy

- We could solve NONISO with a constant number of rounds (2).
- What kind of problem might we need more rounds for?

Theorem

For $k \geq 2$ a constant, $\mathbf{AM}[k] = \mathbf{AM}[2] = \mathbf{AM}$.

Collapsing the Hierarchy

- We could solve NONISO with a constant number of rounds (2).
- What kind of problem might we need more rounds for?

Theorem

For $k \geq 2$ a constant, $\mathbf{AM}[k] = \mathbf{AM}[2] = \mathbf{AM}$.

- This collapse is again somewhat surprising!
- We leave proving it as an exercise for later.

Collapsing the Hierarchy

- We could solve NONISO with a constant number of rounds (2).
- What kind of problem might we need more rounds for?

Theorem

For $k \geq 2$ a constant, $\mathbf{AM}[k] = \mathbf{AM}[2] = \mathbf{AM}$.

- This collapse is again somewhat surprising!
- We leave proving it as an exercise for later.

Corollary

For all $k \geq 2$ a constant, $\mathbf{IP}[k] \subseteq \mathbf{AM}[k+2] = \mathbf{AM}$.

- Careful: Only for constant k; doesn't imply $\mathbf{IP} = \mathbf{AM}[\mathrm{poly}(n)] = \mathbf{AM}$!

Reflection on Arthur-Merlin Proofs

- Despite the apparent differences, we saw public and private coins were roughly as useful as each other for theorem proving.
- We have the following inclusions:

$$\mathbf{NP} \subseteq \mathbf{AM}[2] = \mathbf{AM}[k] \subseteq \mathbf{IP}$$

Reflection on Arthur-Merlin Proofs

- Despite the apparent differences, we saw public and private coins were roughly as useful as each other for theorem proving.
- We have the following inclusions:

$$\mathbf{NP} \subseteq \mathbf{AM}[2] = \mathbf{AM}[k] \subseteq \mathbf{IP}$$

- Another natural intermediate class is $\mathbf{MA}$ (between $\mathbf{NP}$ and $\mathbf{AM}$).
    - "Probabilistic analogue of $\mathbf{NP}$".
    - We'll look at that as an exercise if we have time.

Reflection on Arthur-Merlin Proofs

- Despite the apparent differences, we saw public and private coins were roughly as useful as each other for theorem proving.
- We have the following inclusions:

$$\mathbf{NP} \subseteq \mathbf{AM}[2] = \mathbf{AM}[k] \subseteq \mathbf{IP} \subseteq ?$$

- Another natural intermediate class is $\mathbf{MA}$ (between $\mathbf{NP}$ and $\mathbf{AM}$).
  - "Probabilistic analogue of $\mathbf{NP}$".
  - We'll look at that as an exercise if we have time.

- How big is $\mathbf{IP}$?

$\mathbf{IP} = \mathbf{PSPACE}$

It seems difficult to find problems in $\mathbf{IP}$ that require a number of rounds growing with n.

- For some time, it was suspected that $\mathbf{AM} = \mathbf{IP}\ldots$

IP = PSPACE

It seems difficult to find problems in $\mathbf{IP}$ that require a number of rounds growing with n.

- For some time, it was suspected that $\mathbf{AM} = \mathbf{IP}\ldots$

Theorem (Shamir, 1990)

$\mathbf{IP} = \mathbf{PSPACE}$.

$\mathbf{IP} = \mathbf{PSPACE}$

It seems difficult to find problems in $\mathbf{IP}$ that require a number of rounds growing with n.

- For some time, it was suspected that $\mathbf{AM} = \mathbf{IP}\ldots$

Theorem (Shamir, 1990)

$\mathbf{IP} = \mathbf{PSPACE}$.

- We will sketch only $\mathbf{IP} \subseteq \mathbf{PSPACE}$.
- Proving the other direction is somewhat more involved.
  - See, e.g., Sipser's textbook, or "Computational Complexity: A Modern Approach" by Arora and Barak.
  - Involves interesting techniques, e.g., arithmetization of Boolean formulas.

IP $\subseteq$ PSPACE

Proof (idea):

- The difficulty comes from fact that the Prover may have unbounded computational power.
- Easier if we return to the private coin setting.

IP $\subseteq$ PSPACE

Proof (idea):

- The difficulty comes from fact that the Prover may have unbounded computational power.
- Easier if we return to the private coin setting.
- Assume $L \in \mathbf{IP}$ and L's verifier V uses exactly $k \in \mathrm{poly}(n)$ rounds.
- Note we can assume also that each message a_i has a polynomially bounded length (say n^c).
- Then, the entire transcript $A_k = (a_1, \ldots, a_k)$ is of polynomial size.

IP $\subseteq$ PSPACE

Proof (idea):

- The difficulty comes from fact that the Prover may have unbounded computational power.
- Easier if we return to the private coin setting.
- Assume $L \in \mathbf{IP}$ and L's verifier V uses exactly $k \in \text{poly}(n)$ rounds.
- Note we can assume also that each message a_i has a polynomially bounded length (say n^c).
- Then, the entire transcript $A_k = (a_1, \ldots, a_k)$ is of polynomial size.

- Approach: we recursively enumerate all possible transcripts checking their consistency with V and calculating the maximum probability that V would accept x.

$\mathbf{IP} \subseteq \mathbf{PSPACE}$: More Details

Define: $\Pr[V \text{ accepts } x] = \max_P \Pr_r[\text{out}(V \leftrightarrow P)(x) = \text{accept}]$.

- If $x \in L$, then $\Pr[V \text{ accepts } x] \geq 2/3$;
- If $x \notin L$, then $\Pr[V \text{ accepts } x] \leq 1/3$.

$\mathbf{IP} \subseteq \mathbf{PSPACE}$: More Details

Define: $\Pr[V \text{ accepts } x] = \max_P \Pr_r[\mathrm{out}(V \leftrightarrow P)(x) = \mathrm{accept}]$.

- If $x \in L$, then $\Pr[V \text{ accepts } x] \geq 2/3$;
- If $x \notin L$, then $\Pr[V \text{ accepts } x] \leq 1/3$.

We compute this, by recursively computing, for $A_i = (a_1, \ldots, a_i)$,

$$\Pr[V \text{ accepts } x \mid A_i] := \max_P \Pr_r[\mathrm{out}(V \leftrightarrow P)(x, A_i) = \mathrm{accept}],$$

where $\mathrm{out}(V \leftrightarrow P)(x, A_i)$ is the output of the probabilistic interaction beginning with the transcript A_i.

- If A_i is inconsistent with V, this is given probability 0.
- Note that $\Pr[V \text{ accepts } x] = \Pr[V \text{ accepts } x \mid A_0]$.

$\mathbf{IP} \subseteq \mathbf{PSPACE}$: More Details

We then compute this recursively:

- Base case: $\Pr[V \text{ accepts } x \mid A_k] = \Pr_r[V(x, r, a_1, \ldots, a_k) = \text{accept}]$.

$\mathbf{IP} \subseteq \mathbf{PSPACE}$: More Details

We then compute this recursively:

- Base case: $\Pr[V \text{ accepts } x \mid A_k] = \Pr_r[V(x, r, a_1, \ldots, a_k) = \text{accept}]$.
- For $i < k$ odd:

$$\Pr[V \text{ accepts } x \mid A_i] = \max_{a_{i+1}} \Pr[V \text{ accepts } x \mid (A_i, a_{i+1})];$$

$\mathbf{IP} \subseteq \mathbf{PSPACE}$: More Details

We then compute this recursively:

- Base case: $\Pr[V \text{ accepts } x \mid A_k] = \Pr_r[V(x, r, a_1, \ldots, a_k) = \text{accept}]$.
- For $i < k$ odd:

$$\Pr[V \text{ accepts } x \mid A_i] = \max_{a_{i+1}} \Pr[V \text{ accepts } x \mid (A_i, a_{i+1})];$$

- For $i < k$ even:

$$\Pr[V \text{ accepts } x \mid A_i] = \sum_{a_{i+1}} \Pr_r[V(x, r, A_i) = a_{i+1}] \Pr[V \text{ accepts } x \mid (A_i, a_{i+1})].$$

IP ⊆ PSPACE: More Details

We then compute this recursively:

- Base case: $\Pr[V \text{ accepts } x \mid A_k] = \Pr_r[V(x, r, a_1, \ldots, a_k) = \text{accept}]$.
- For $i < k$ odd:

$$\Pr[V \text{ accepts } x \mid A_i] = \max_{a_{i+1}} \Pr[V \text{ accepts } x \mid (A_i, a_{i+1})];$$

- For $i < k$ even:

$$\Pr[V \text{ accepts } x \mid A_i] = \sum_{a_{i+1}} \Pr_r[V(x, r, A_i) = a_{i+1}] \Pr[V \text{ accepts } x \mid (A_i, a_{i+1})].$$

We thus calculate recursively the performance of the optimal prover.

- Since the recursion has polynomial depth we compute $\Pr[V \text{ accepts } x]$ in polynomial space.

Discussion on Interactive Proofs

- Interactive proof systems are actually very powerful, in particular when we're allowed a polynomial number of interactions.
  - Can certify proofs for problems (seemingly) much more difficult than $\mathbf{NP}$.

Discussion on Interactive Proofs

- Interactive proof systems are actually very powerful, in particular when we're allowed a polynomial number of interactions.
  - Can certify proofs for problems (seemingly) much more difficult than $\mathbf{NP}$.

- Since $\mathbf{IP} = \mathbf{PSPACE}$, the Prover doesn't need unbounded power: polynomial space is enough!
- But in other cases (e.g., the public coin protocol for NONISO), it seems the prover needs to solve a harder problem than NONISO.
  - It needed to find a graph H isomorphic to either G_1 or G_2 such that $h(\langle H \rangle) = y$.

More Directions with Interactive Proofs

There are many further directions one can go with interactive proofs:

- Zero-knowledge proofs (e.g., for cryptography).

More Directions with Interactive Proofs

There are many further directions one can go with interactive proofs:

- Zero-knowledge proofs (e.g., for cryptography).
- Multiple provers.
  - By providing random questions to several independent Provers, one can try and play the Provers off against each other.
  - This turns out to significantly increase the power of Verifier.

More Directions with Interactive Proofs

There are many further directions one can go with interactive proofs:

- Zero-knowledge proofs (e.g., for cryptography).
- Multiple provers.
  - By providing random questions to several independent Provers, one can try and play the Provers off against each other.
  - This turns out to significantly increase the power of Verifier.
- Probabilistically checkable proofs (PCP).
  - Can one certify a proof just by checking parts of it at random, but without seeing all of it?

More Directions with Interactive Proofs

There are many further directions one can go with interactive proofs:

- Zero-knowledge proofs (e.g., for cryptography).
- Multiple provers.
  - By providing random questions to several independent Provers, one can try and play the Provers off against each other.
  - This turns out to significantly increase the power of Verifier.
- Probabilistically checkable proofs (PCP).
  - Can one certify a proof just by checking parts of it at random, but without seeing all of it?
- Quantum interactive proof systems.
  - Used to provide the natural quantum analogue of $\mathbf{NP}$.