Interactive Proofs

Lecture 16

What the all-powerful can convince mere mortals of

Recap

Recap

- Non-deterministic Computation

Recap

- Non-deterministic Computation
- Polynomial Hierarchy

Recap

- Non-deterministic Computation
- Polynomial Hierarchy
  - Non-determinism on steroids!

Recap

- Non-deterministic Computation
- Polynomial Hierarchy
  - Non-determinism on steroids!
- Non-uniform computation

Recap

- Non-deterministic Computation
- Polynomial Hierarchy
  - Non-determinism on steroids!
- Non-uniform computation
- Probabilistic Computation

Recap

- Non-deterministic Computation
- Polynomial Hierarchy
  - Non-determinism on steroids!
- Non-uniform computation
- Probabilistic Computation
- Today: Interactive Proofs

Recap

- Non-deterministic Computation
- Polynomial Hierarchy
    - Non-determinism on steroids!
- Non-uniform computation
- Probabilistic Computation
- Today: Interactive Proofs
    - Non-determinism and Probabilistic computation on steroids!

Interactive Proofs

Interactive Proofs

- **Prover** wants to convince **verifier** that x has some property

Interactive Proofs

- **Prover** wants to convince **verifier** that x has some property
  - i.e. x is in language L

Interactive Proofs

- **Prover** wants to convince **verifier** that x has some property
  - i.e. x is in language L

Interactive Proofs

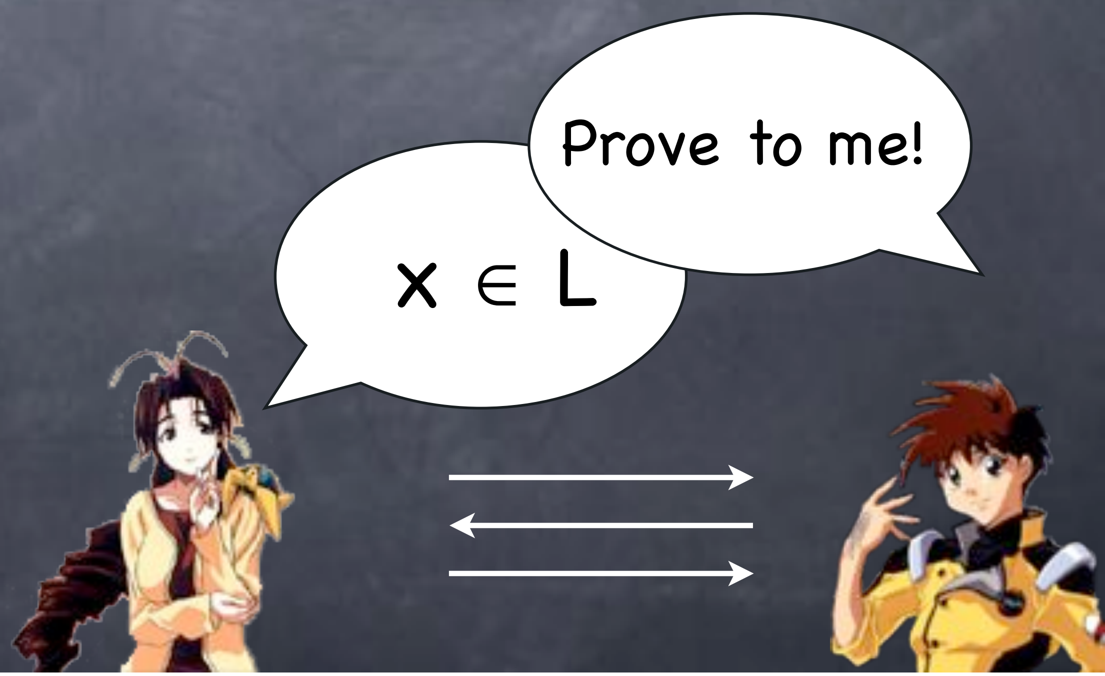
- **Prover** wants to convince **verifier** that x has some property
  - i.e. x is in language L

Interactive Proofs

- **Prover** wants to convince **verifier** that x has some property
    - i.e. x is in language L

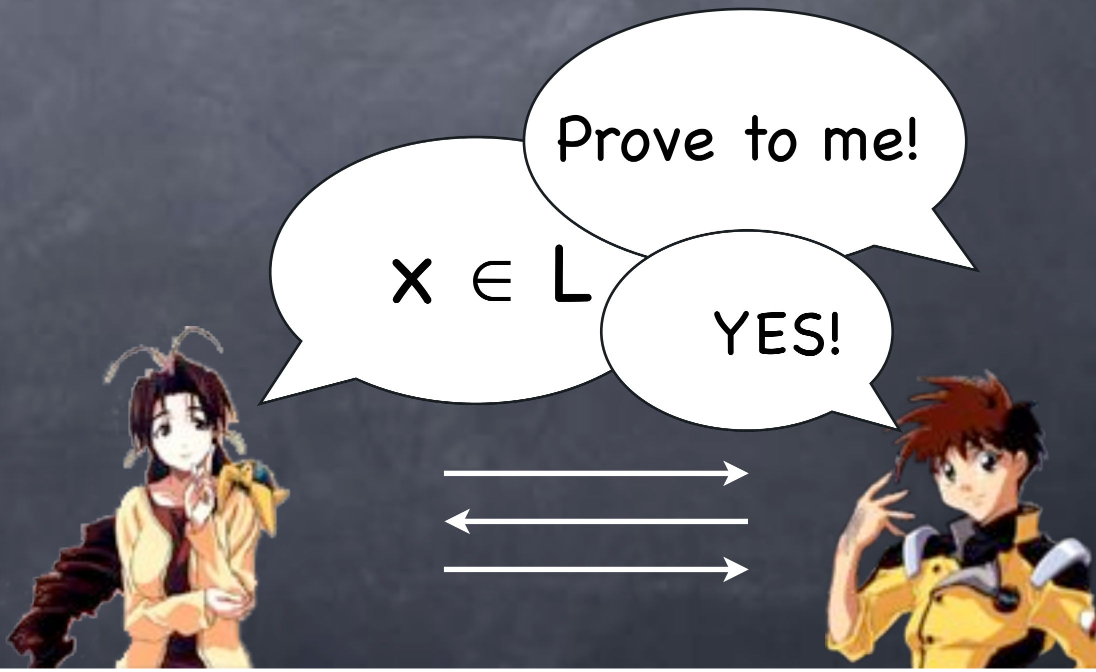
Interactive Proofs

- **Prover** wants to convince **verifier** that x has some property
  - i.e. x is in language L



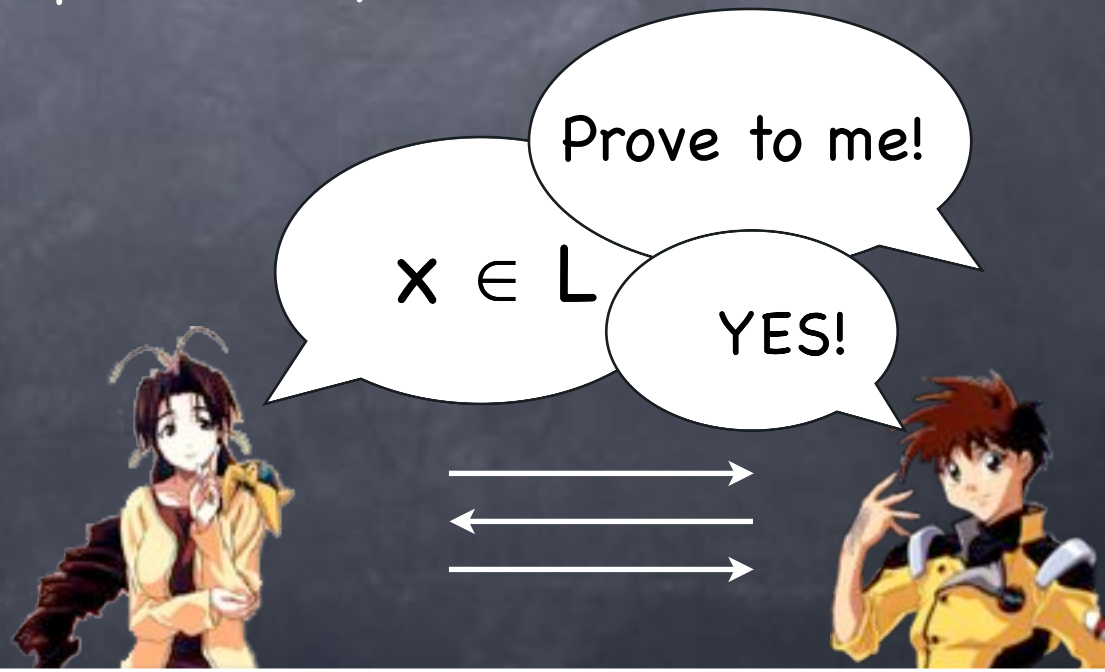
Interactive Proofs

- **Prover** wants to convince **verifier** that x has some property
  - i.e. x is in language L



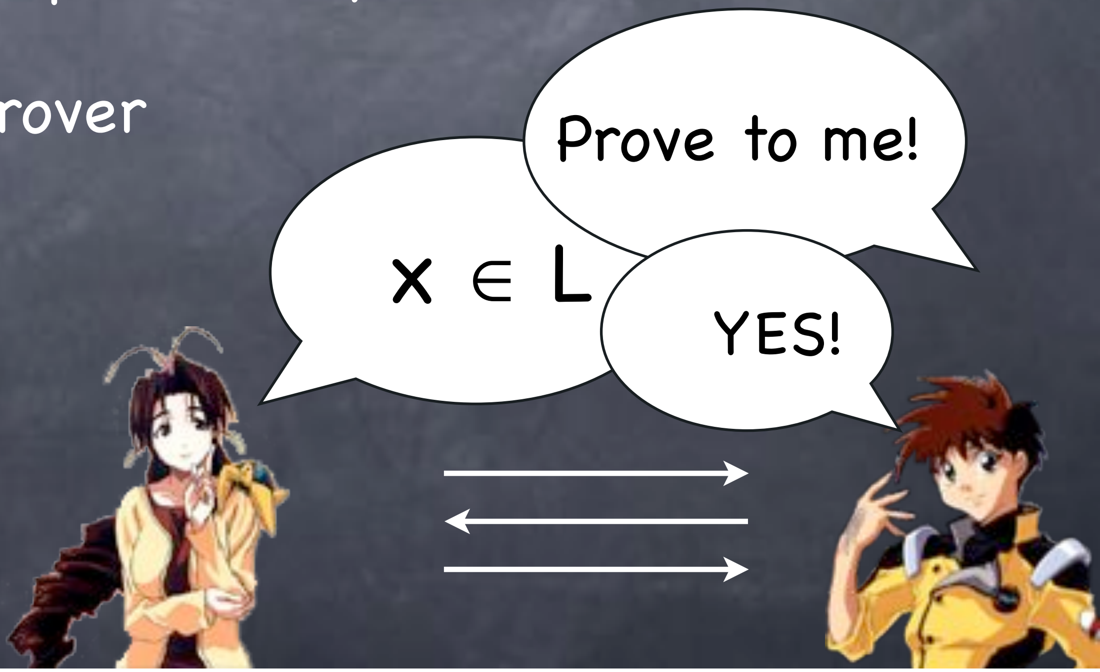
Interactive Proofs

- **Prover** wants to convince **verifier** that x has some property
  - i.e. x is in language L
- All powerful prover, computationally bounded verifier



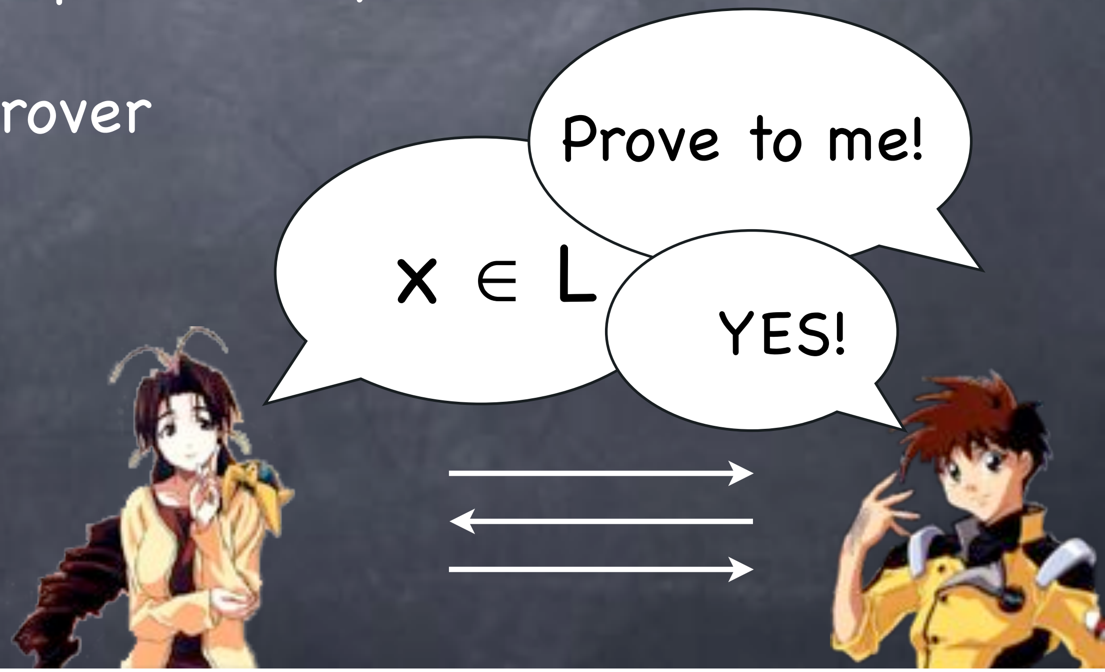
Interactive Proofs

- **Prover** wants to convince **verifier** that x has some property
  - i.e. x is in language L
- All powerful prover, computationally bounded verifier
- Verifier doesn't trust prover



Interactive Proofs

- **Prover** wants to convince **verifier** that x has some property
  - i.e. x is in language L
- All powerful prover, computationally bounded verifier
- Verifier doesn't trust prover
  - Limits the power



Interactive Proofs

Interactive Proofs

- **Completeness**

Interactive Proofs

- **Completeness**
  - If x in L, honest Prover should convince honest Verifier

Interactive Proofs

- **Completeness**
  - If x in L, honest Prover should convince honest Verifier
- **Soundness**

Interactive Proofs

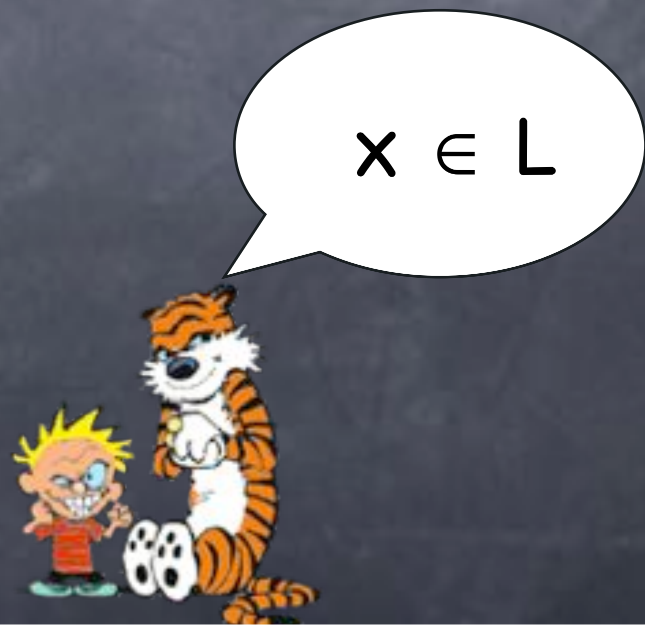
- **Completeness**
  - If x in L, honest Prover should convince honest Verifier
- **Soundness**
  - If x not in L, honest Verifier won't accept any purported proof


Interactive Proofs

- **Completeness**
  - If x in L, honest Prover should convince honest Verifier
- **Soundness**
  - If x not in L, honest Verifier won't accept any purported proof

Interactive Proofs

- **Completeness**
  - If x in L, honest Prover should convince honest Verifier
- **Soundness**
  - If x not in L, honest Verifier won't accept any purported proof

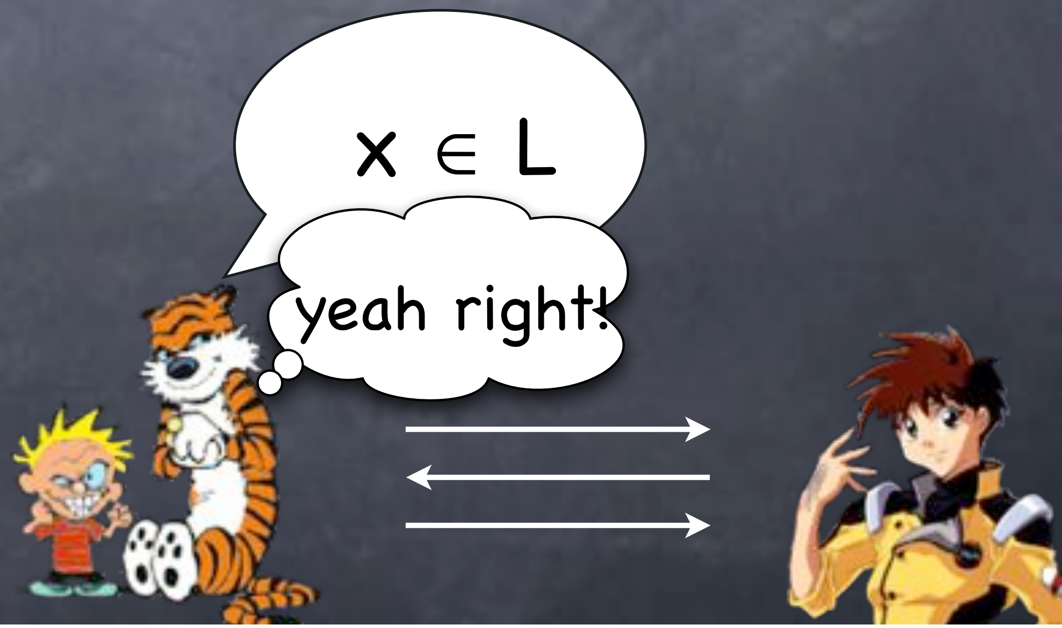


Interactive Proofs

- **Completeness**
  - If x in L, honest Prover should convince honest Verifier
- **Soundness**
  - If x not in L, honest Verifier won't accept any purported proof

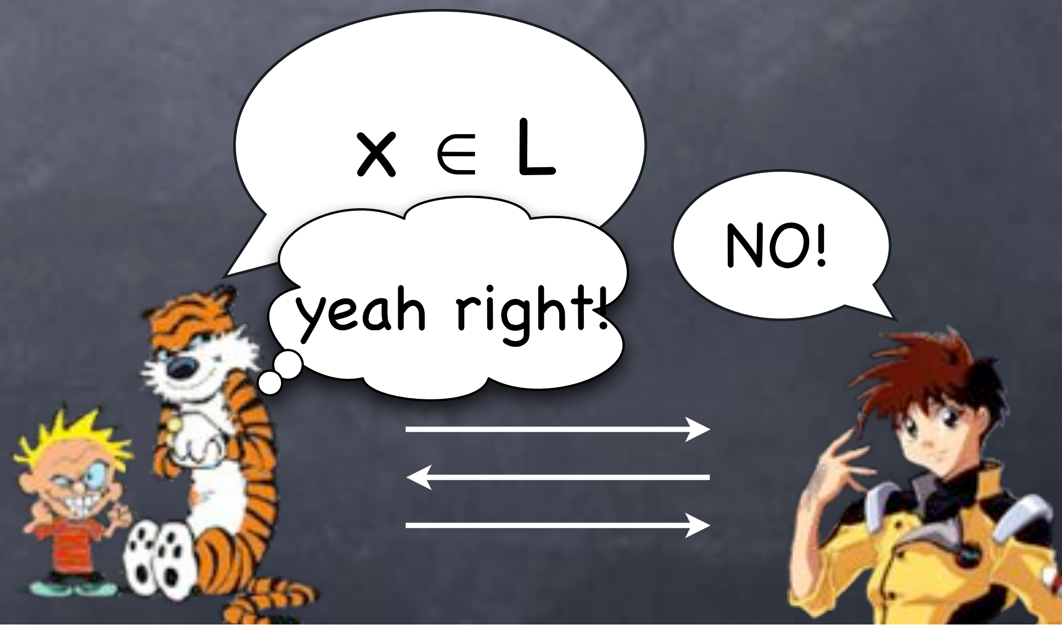
Interactive Proofs

- **Completeness**
  - If x in L, honest Prover should convince honest Verifier
- **Soundness**
  - If x not in L, honest Verifier won't accept any purported proof



Interactive Proofs

- **Completeness**
  - If x in L, honest Prover should convince honest Verifier
- **Soundness**
  - If x not in L, honest Verifier won't accept any purported proof



An Example

An Example

- Coke in bottle or can

An Example

- **Coke in bottle or can**
  - Prover claims: coke in bottle and coke in can are different

An Example

- **Coke in bottle or can**
    - Prover claims: coke in bottle and coke in can are different
- IP protocol:

An Example

- **Coke in bottle or can**
  - Prover claims: coke in bottle and coke in can are different
- IP protocol:

An Example

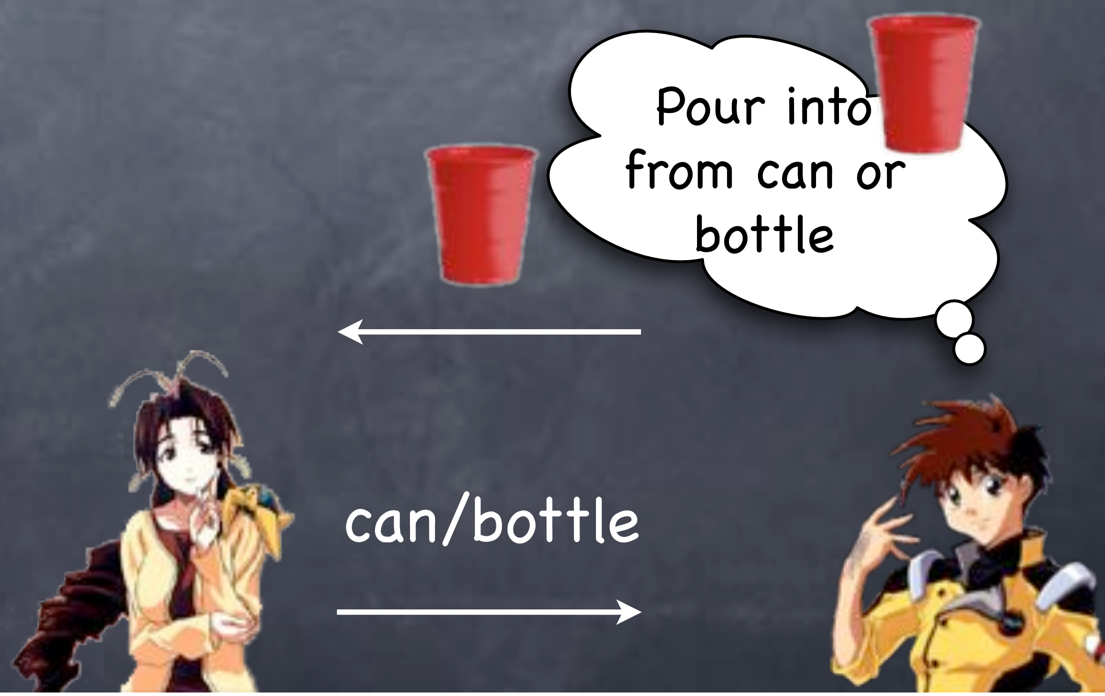
- **Coke in bottle or can**
  - Prover claims: coke in bottle and coke in can are different
- IP protocol:

An Example

- **Coke in bottle or can**
  - Prover claims: coke in bottle and coke in can are different
- IP protocol:

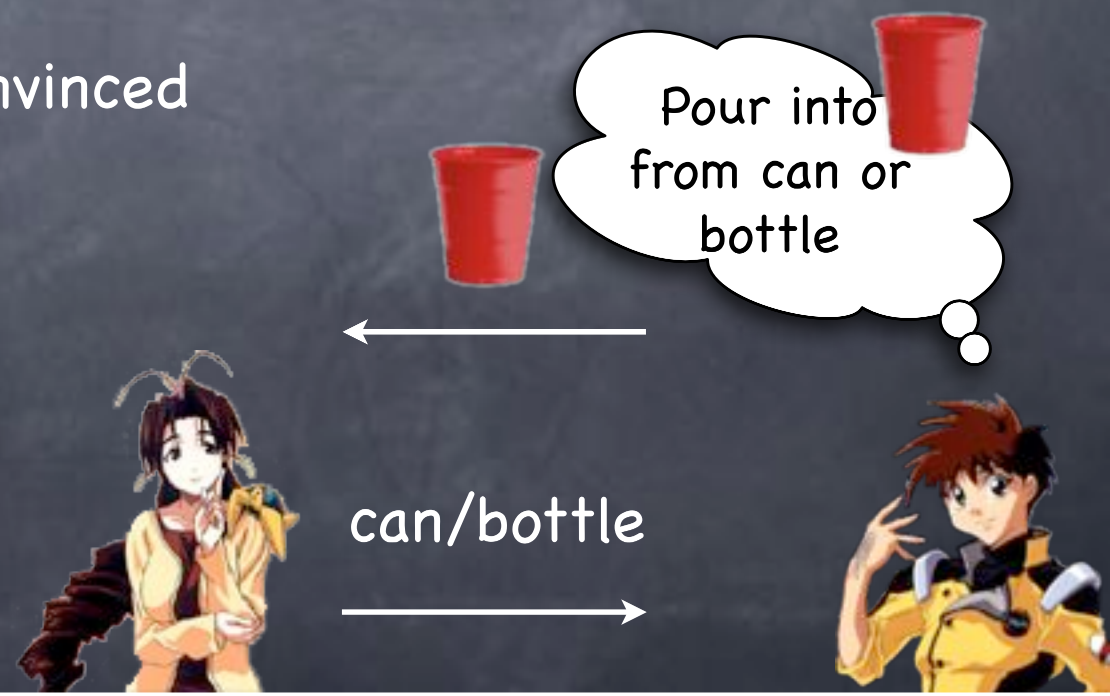
An Example

- **Coke in bottle or can**
  - Prover claims: coke in bottle and coke in can are different
- IP protocol:
  - prover tells whether cup was filled from can or bottle



An Example

- **Coke in bottle or can**
  - Prover claims: coke in bottle and coke in can are different
- IP protocol:
  - prover tells whether cup was filled from can or bottle
  - repeat till verifier is convinced



An Example

An Example

- **Graph non-isomorphism (GNI)**

An Example

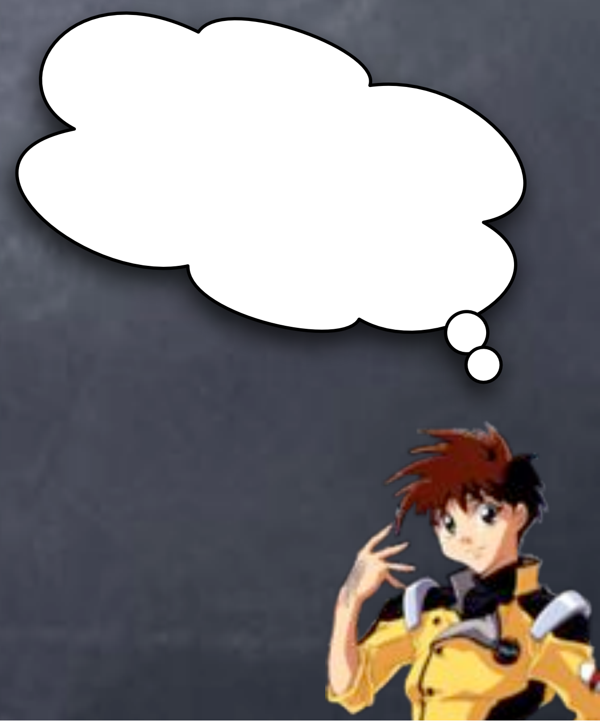
- **Graph non-isomorphism (GNI)**
  - Prover claims: G_0 not isomorphic to G_1

An Example

- **Graph non-isomorphism (GNI)**
  - Prover claims: G_0 not isomorphic to G_1
- IP protocol:

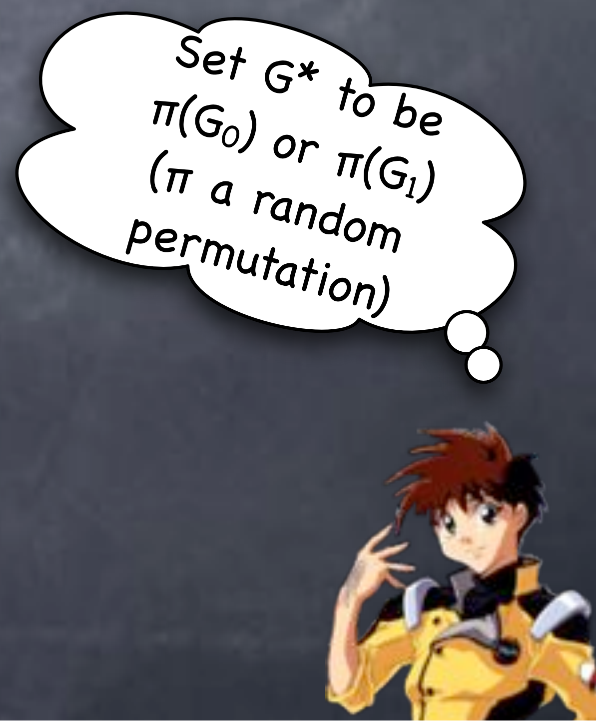
An Example

- **Graph non-isomorphism (GNI)**
  - Prover claims: G_0 not isomorphic to G_1
- IP protocol:



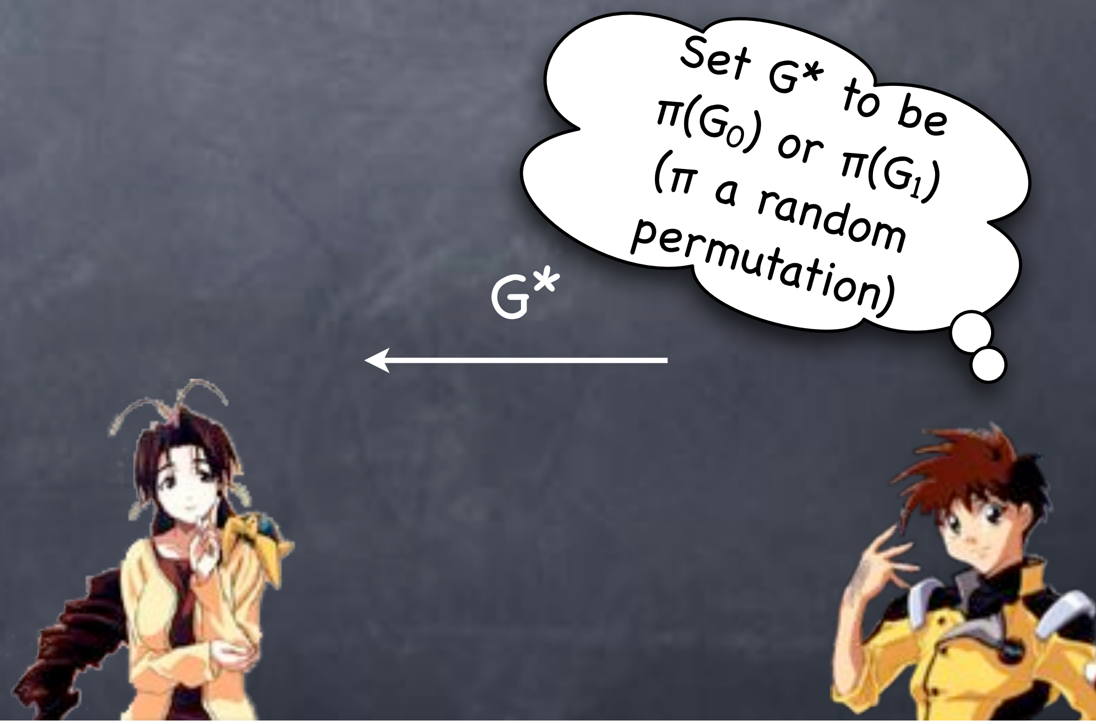
An Example

- **Graph non-isomorphism (GNI)**
  - Prover claims: G_0 not isomorphic to G_1
- IP protocol:



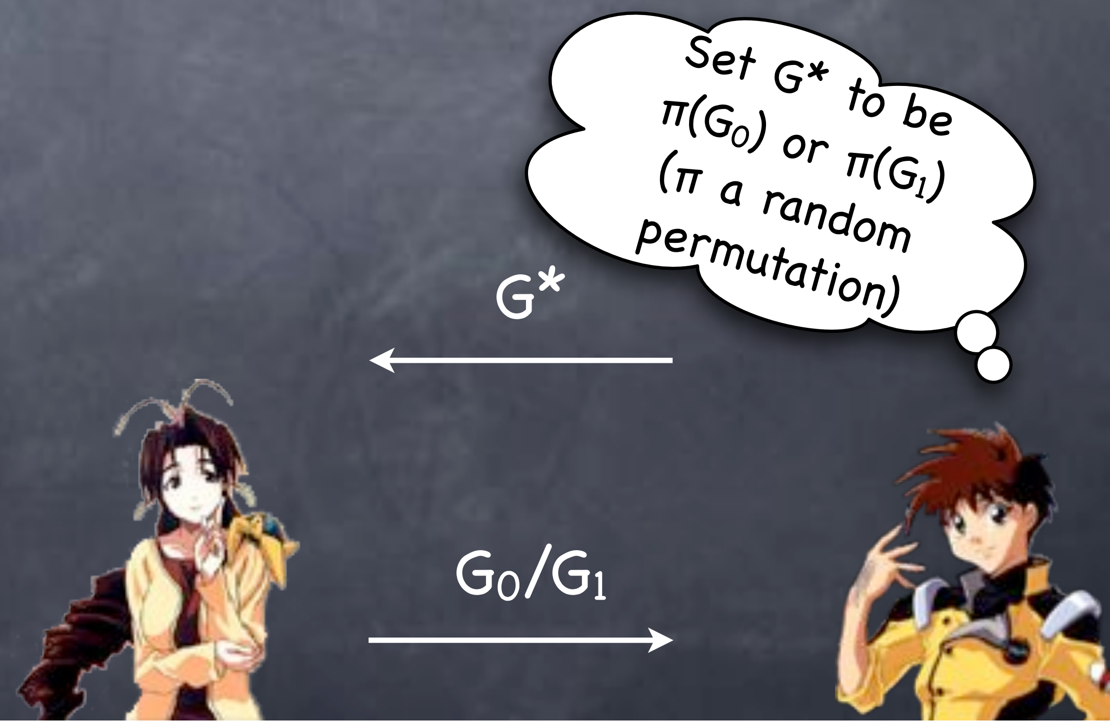
An Example

- **Graph non-isomorphism (GNI)**
  - Prover claims: G_0 not isomorphic to G_1
- IP protocol:



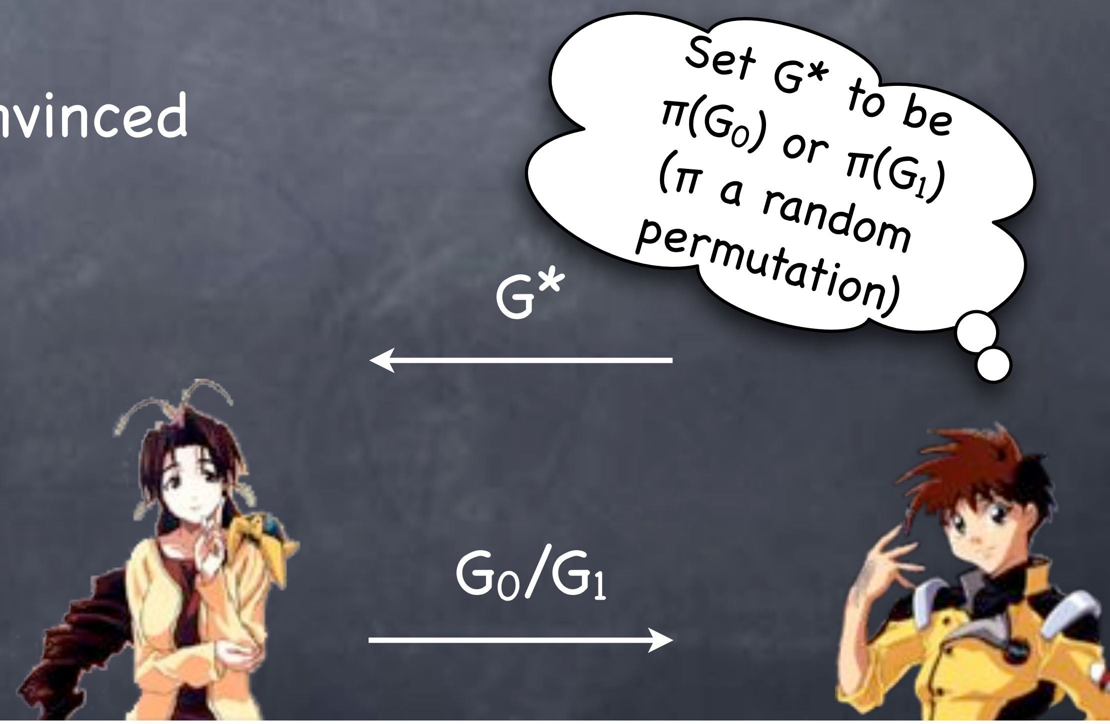
An Example

- **Graph non-isomorphism (GNI)**
  - Prover claims: G_0 not isomorphic to G_1
- IP protocol:
  - prover tells whether G^* came from G_0 or G_1



An Example

- **Graph non-isomorphism (GNI)**
  - Prover claims: G_0 not isomorphic to G_1
- IP protocol:
  - prover tells whether G^* came from G_0 or G_1
  - repeat till verifier is convinced



Interactive Proofs

Interactive Proofs

- **Completeness**

Interactive Proofs

- **Completeness**
  - If x in L, honest Prover will convince honest Verifier

Interactive Proofs

- **Completeness**
  - If x in L, honest Prover will convince honest Verifier
  - With probability at least 2/3

Interactive Proofs

- **Completeness**
  - If x in L, honest Prover will convince honest Verifier
  - With probability at least 2/3
- **Soundness**

Interactive Proofs

- **Completeness**
  - If x in L, honest Prover will convince honest Verifier
  - With probability at least 2/3
- **Soundness**
  - If x not in L, honest Verifier won't accept any purported proof

Interactive Proofs

- **Completeness**
  - If x in L, honest Prover will convince honest Verifier
  - With probability at least 2/3
- **Soundness**
  - If x not in L, honest Verifier won't accept any purported proof
  - Except with probability at most 1/3

Deterministic IP?

Deterministic IP?

- Deterministic Verifier IP

Deterministic IP?

- Deterministic Verifier IP
  - Prover can construct the entire transcript, which verifier can verify deterministically

Deterministic IP?

- Deterministic Verifier IP
  - Prover can construct the entire transcript, which verifier can verify deterministically
  - NP certificate

Deterministic IP?

- Deterministic Verifier IP
  - Prover can construct the entire transcript, which verifier can verify deterministically
  - NP certificate
  - Deterministic Verifier IP = NP

Deterministic IP?

- Deterministic Verifier IP
  - Prover can construct the entire transcript, which verifier can verify deterministically
  - NP certificate
  - Deterministic Verifier IP = NP
- Deterministic Prover IP = IP

Deterministic IP?

- Deterministic Verifier IP
  - Prover can construct the entire transcript, which verifier can verify deterministically
  - NP certificate
  - Deterministic Verifier IP = NP
- Deterministic Prover IP = IP
  - For each input prover can choose the random tape which maximizes Pr[yes] (probability over honest verifier's randomness)

Public and Private Coins

Public and Private Coins

- Public coins: Prover sees verifier's coin tosses

Public and Private Coins

- Public coins: Prover sees verifier's coin tosses
  - Verifier might as well send nothing but the coins to the prover

Public and Private Coins

- Public coins: Prover sees verifier's coin tosses
  - Verifier might as well send nothing but the coins to the prover
- Private coins: Verifier does not send everything about the coins

Public and Private Coins

- Public coins: Prover sees verifier's coin tosses
  - Verifier might as well send nothing but the coins to the prover
- Private coins: Verifier does not send everything about the coins
  - e.g. GNI protocol: verifier keeps coin tosses hidden; uses it to create challenge

Arthur Merlin Proofs

Arthur Merlin Proofs

- Arthur-Merlin proof-systems

Arthur Merlin Proofs

- Arthur-Merlin proof-systems
  - Arthur: polynomial time verifier

Arthur Merlin Proofs

- Arthur-Merlin proof-systems
  - Arthur: polynomial time verifier

Arthur Merlin Proofs

- Arthur-Merlin proof-systems
  - Arthur: polynomial time verifier
  - Merlin: unbounded prover

Arthur Merlin Proofs

- Arthur-Merlin proof-systems
  - Arthur: polynomial time verifier
  - Merlin: unbounded prover

Arthur Merlin Proofs

- Arthur-Merlin proof-systems
  - Arthur: polynomial time verifier
  - Merlin: unbounded prover
  - Random coins come from a beacon

Arthur Merlin Proofs

- Arthur-Merlin proof-systems
  - Arthur: polynomial time verifier
  - Merlin: unbounded prover
  - Random coins come from a beacon

Arthur Merlin Proofs

- Arthur-Merlin proof-systems
  - Arthur: polynomial time verifier
  - Merlin: unbounded prover
  - Random coins come from a beacon
    - Public coin proof-system

Arthur Merlin Proofs

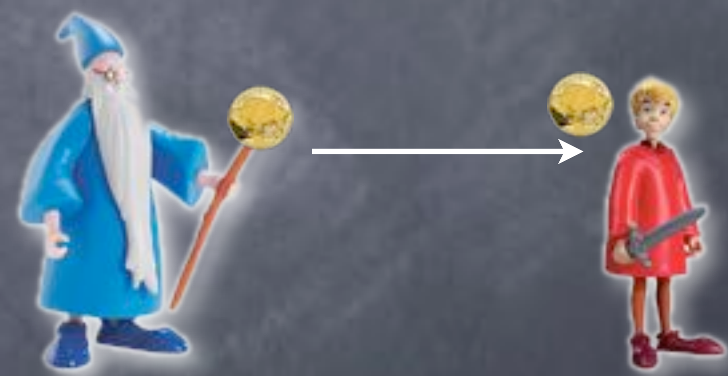
- Arthur-Merlin proof-systems
  - Arthur: polynomial time verifier
  - Merlin: unbounded prover
  - Random coins come from a beacon
    - Public coin proof-system
    - Arthur sends no messages nor flips any coins

Arthur Merlin Proofs

- Arthur-Merlin proof-systems
  - Arthur: polynomial time verifier
  - Merlin: unbounded prover
  - Random coins come from a beacon
    - Public coin proof-system
    - Arthur sends no messages nor flips any coins

Arthur Merlin Proofs

- Arthur-Merlin proof-systems
  - Arthur: polynomial time verifier
  - Merlin: unbounded prover
  - Random coins come from a beacon
    - Public coin proof-system
    - Arthur sends no messages nor flips any coins

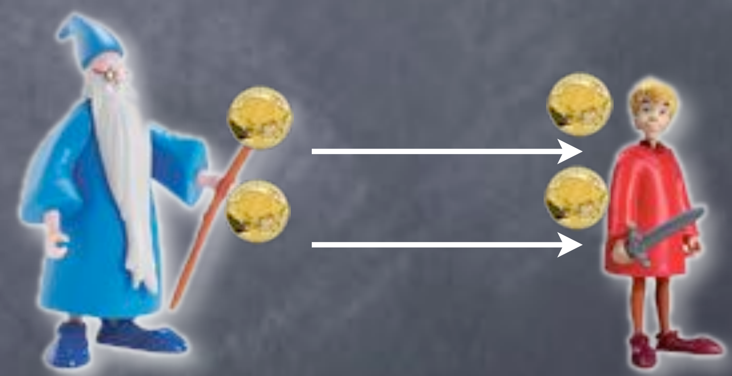


Arthur Merlin Proofs

- Arthur-Merlin proof-systems
  - Arthur: polynomial time verifier
  - Merlin: unbounded prover
  - Random coins come from a beacon
    - Public coin proof-system
    - Arthur sends no messages nor flips any coins

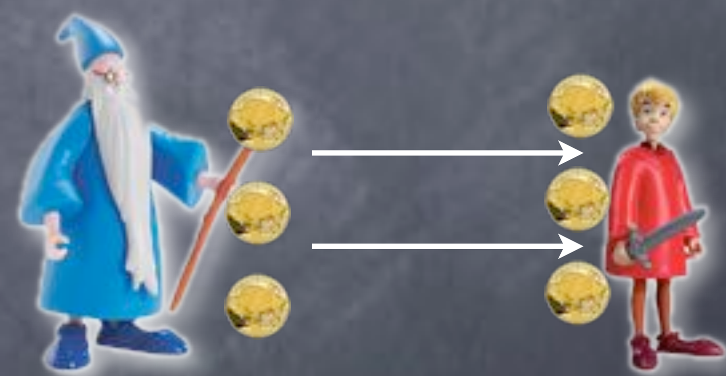
Arthur Merlin Proofs

- Arthur-Merlin proof-systems
  - Arthur: polynomial time verifier
  - Merlin: unbounded prover
  - Random coins come from a beacon
    - Public coin proof-system
    - Arthur sends no messages nor flips any coins



Arthur Merlin Proofs

- Arthur-Merlin proof-systems
  - Arthur: polynomial time verifier
  - Merlin: unbounded prover
  - Random coins come from a beacon
    - Public coin proof-system
    - Arthur sends no messages nor flips any coins



MA and AM

MA and AM

- Class of languages with two message Arthur-Merlin protocols

MA and AM

- Class of languages with two message Arthur-Merlin protocols
  - AM (or AM[2]): One message from beacon, followed by one message from Merlin

MA and AM

- Class of languages with two message Arthur-Merlin protocols
  - AM (or AM[2]): One message from beacon, followed by one message from Merlin
  - MA (or MA[2]): One message from Merlin followed by one message from beacon

MA and AM

- Class of languages with two message Arthur-Merlin protocols
  - AM (or AM[2]): One message from beacon, followed by one message from Merlin
  - MA (or MA[2]): One message from Merlin followed by one message from beacon
- Contain NP and BPP

Multiple-message proofs

Multiple-message proofs

- AM[k], MA[k], IP[k]: k(n) messages

Multiple-message proofs

- AM[k], MA[k], IP[k]: k(n) messages
  - Turns out IP[k] $\subseteq$ AM[k+2]!

Multiple-message proofs

- AM[k], MA[k], IP[k]: k(n) messages
  - Turns out IP[k] $\subseteq$ AM[k+2]!
  - Turns out IP[const] = AM[const] = AM[2]!

Multiple-message proofs

- AM[k], MA[k], IP[k]: k(n) messages
  - Turns out IP[k] $\subseteq$ AM[k+2]!
  - Turns out IP[const] = AM[const] = AM[2]!
    - Called AM

Multiple-message proofs

- AM[k], MA[k], IP[k]: k(n) messages
  - Turns out IP[k] ⊆ AM[k+2]!
  - Turns out IP[const] = AM[const] = AM[2]!
    - Called AM
  - Turns out IP[poly] = AM[poly] = PSPACE!

Multiple-message proofs

- AM[k], MA[k], IP[k]: k(n) messages
  - Turns out IP[k] ⊆ AM[k+2]!
  - Turns out IP[const] = AM[const] = AM[2]!
    - Called AM
  - Turns out IP[poly] = AM[poly] = PSPACE!
    - Called IP (= PSPACE)

Multiple-message proofs

- AM[k], MA[k], IP[k]: k(n) messages
  - Turns out IP[k] $\subseteq$ AM[k+2]!
  - Turns out IP[const] = AM[const] = AM[2]!
    - Called AM
  - Turns out IP[poly] = AM[poly] = PSPACE!
    - Called IP (= PSPACE)
  - Later.

How can private coins be avoided?

How can private coins be avoided?

- Example: GNI

How can private coins be avoided?

- Example: GNI
  - Recall GNI protocol used private coins

How can private coins be avoided?

- Example: GNI
  - Recall GNI protocol used private coins
- An alternate view of GNI

How can private coins be avoided?

- Example: GNI
  - Recall GNI protocol used private coins
- An alternate view of GNI
  - Each of G_0 and G_1 has n! isomorphic graphs

How can private coins be avoided?

- Example: GNI
  - Recall GNI protocol used private coins
- An alternate view of GNI
  - Each of G_0 and G_1 has n! isomorphic graphs
    - (Assuming no automorphisms. Else, count with multiplicity.)

How can private coins be avoided?

- Example: GNI
  - Recall GNI protocol used private coins
- An alternate view of GNI
  - Each of G_0 and G_1 has n! isomorphic graphs
    - (Assuming no automorphisms. Else, count with multiplicity.)
  - If G_0 and G_1 isomorphic, same set of n! isomorphic graphs

How can private coins be avoided?

- Example: GNI
  - Recall GNI protocol used private coins
- An alternate view of GNI
  - Each of G_0 and G_1 has n! isomorphic graphs
    - (Assuming no automorphisms. Else, count with multiplicity.)
  - If G_0 and G_1 isomorphic, same set of n! isomorphic graphs
  - Else 2(n!) isomorphic graphs

How can private coins be avoided?

- Example: GNI
  - Recall GNI protocol used private coins
- An alternate view of GNI
  - Each of G_0 and G_1 has n! isomorphic graphs
    - (Assuming no automorphisms. Else, count with multiplicity.)
  - If G_0 and G_1 isomorphic, same set of n! isomorphic graphs
  - Else 2(n!) isomorphic graphs
  - Prover to prove that $|\{H: H \equiv G_0 \text{ or } H \equiv G_1\}| > n!$

Set Lower-bound

Set Lower-bound

- Prover wants to prove that $|S| > K$, for a set S such that $|S| \geq 2K$

Set Lower-bound

- Prover wants to prove that $|S| > K$, for a set S such that $|S| \geq 2K$
- $S \subseteq U$, a sampleable universe, membership in S certifiable

Set Lower-bound

- Prover wants to prove that $|S| > K$, for a set S such that $|S| \geq 2K$
- $S \subseteq U$, a sampleable universe, membership in S certifiable
- Suppose K large (say $K=|U|/3$). Then simple protocol:

Set Lower-bound

- Prover wants to prove that $|S| > K$, for a set S such that $|S| \geq 2K$
- $S \subseteq U$, a sampleable universe, membership in S certifiable
- Suppose K large (say $K=|U|/3$). Then simple protocol:
  - Verifier picks a random element $x \in U$

Set Lower-bound

- Prover wants to prove that $|S| > K$, for a set S such that $|S| \geq 2K$
- $S \subseteq U$, a sampleable universe, membership in S certifiable
- Suppose K large (say $K=|U|/3$). Then simple protocol:
    - Verifier picks a random element $x \in U$
    - If $x \in S$, prover returns certificate

Set Lower-bound

- Prover wants to prove that $|S| > K$, for a set S such that $|S| \geq 2K$
- $S \subseteq U$, a sampleable universe, membership in S certifiable
- Suppose K large (say $K=|U|/3$). Then simple protocol:
  - Verifier picks a random element $x \in U$
  - If $x \in S$, prover returns certificate
  - If certificate valid, verifier accepts

Set Lower-bound

- Prover wants to prove that $|S| > K$, for a set S such that $|S| \geq 2K$
- $S \subseteq U$, a sampleable universe, membership in S certifiable
- Suppose K large (say $K=|U|/3$). Then simple protocol:
  - Verifier picks a random element $x \in U$
  - If $x \in S$, prover returns certificate
  - If certificate valid, verifier accepts
- If $|S| > 2K$, Pr[yes] > 2/3. If $|S| \leq K$, Pr[yes] $\leq$ 1/3

Set Lower-bound

- Prover wants to prove that $|S| > K$, for a set S such that $|S| \geq 2K$
- $S \subseteq U$, a sampleable universe, membership in S certifiable
- Suppose K large (say $K=|U|/3$). Then simple protocol:
  - Verifier picks a random element $x \in U$
  - If $x \in S$, prover returns certificate
  - If certificate valid, verifier accepts
- If $|S| > 2K$, Pr[yes] > 2/3. If $|S| \leq K$, Pr[yes] $\leq$ 1/3
- But what if $K/|U|$ is exponentially small?

Set Lower-bound

Set Lower-bound

- Prover wants to prove that $|S| > K$, for a set S such that $|S| \geq 2K$

Set Lower-bound

- Prover wants to prove that $|S| > K$, for a set S such that $|S| \geq 2K$
- But K can be very small (say $|U|=2^n$, $K=2^{n/2}$)

Set Lower-bound

- Prover wants to prove that $|S| > K$, for a set S such that $|S| \geq 2K$

- But K can be very small (say $|U|=2^n$, $K=2^{n/2}$)

- Idea: First "hash down" U to almost size 2K, so that small sets (like S) do not shrink much (and of course, do not grow). Then, prove that H(S) is a large subset of H(U):

Set Lower-bound

- Prover wants to prove that $|S| > K$, for a set S such that $|S| \geq 2K$
- But K can be very small (say $|U|=2^n$, $K=2^{n/2}$)
- Idea: First "hash down" U to almost size $2K$, so that small sets (like S) do not shrink much (and of course, do not grow). Then, prove that $H(S)$ is a large subset of $H(U)$:
  - Verifier picks a random element $y \in H(U)$

Set Lower-bound

- Prover wants to prove that $|S| > K$, for a set S such that $|S| \geq 2K$
- But K can be very small (say $|U|=2^n$, $K=2^{n/2}$)
- Idea: First "hash down" U to almost size 2K, so that small sets (like S) do not shrink much (and of course, do not grow). Then, prove that H(S) is a large subset of H(U):
  - Verifier picks a random element $y \in H(U)$
  - If $y \in H(S)$, prover returns certificate: $x \in S$ (+cert.), $y=H(x)$

Set Lower-bound

- Prover wants to prove that $|S| > K$, for a set S such that $|S| \geq 2K$
- But K can be very small (say $|U|=2^n$, $K=2^{n/2}$)
- Idea: First "hash down" U to almost size 2K, so that small sets (like S) do not shrink much (and of course, do not grow). Then, prove that H(S) is a large subset of H(U):
  - Verifier picks a random element $y \in H(U)$
  - If $y \in H(S)$, prover returns certificate: $x \in S$ (+cert.), $y=H(x)$
  - If certificate valid, verifier accepts

Set Lower-bound

- Prover wants to prove that $|S| > K$, for a set S such that $|S| \geq 2K$
- But K can be very small (say $|U|=2^n$, $K=2^{n/2}$)
- Idea: First "hash down" U to almost size 2K, so that small sets (like S) do not shrink much (and of course, do not grow). Then, prove that H(S) is a large subset of H(U):
  - Verifier picks a random element $y \in H(U)$
  - If $y \in H(S)$, prover returns certificate: $x \in S$ (+cert.), $y=H(x)$
  - If certificate valid, verifier accepts
- Is there such a hash function for all small sets S?

Set Lower-bound

- Prover wants to prove that $|S| > K$, for a set S such that $|S| \geq 2K$
- But K can be very small (say $|U|=2^n$, $K=2^{n/2}$)
- Idea: First "hash down" U to almost size 2K, so that small sets (like S) do not shrink much (and of course, do not grow). Then, prove that H(S) is a large subset of H(U):
  - Verifier picks a random element $y \in H(U)$
  - If $y \in H(S)$, prover returns certificate: $x \in S$ (+cert.), $y=H(x)$
  - If certificate valid, verifier accepts
- Is there such a hash function for all small sets S?
  - Clearly no single function for all S!

Hash Function Family

Hash Function Family

- A family of hash functions

Hash Function Family

- A family of hash functions
  - Given any small subset S, a random function h from the family will not shrink it much (say by 3/4) with high probability

Hash Function Family

- A family of hash functions
  - Given any small subset S, a random function h from the family will not shrink it much (say by 3/4) with high probability
  - (Though every h shrinks <u>some</u> small sets)

Hash Function Family

- A family of hash functions
  - Given any small subset S, a random function h from the family will not shrink it much (say by 3/4) with high probability
  - (Though every h shrinks <u>some</u> small sets)
  - Relate shrinking to "hash collision probability"

Hash Function Family

- A family of hash functions
  - Given any small subset S, a random function h from the family will not shrink it much (say by 3/4) with high probability
  - (Though every h shrinks <u>some</u> small sets)
  - Relate shrinking to "hash collision probability"
    - $\Pr_h[h(x)=h(x')]$ (max over $x \neq x'$)

Hash Function Family

- A family of hash functions
  - Given any small subset S, a random function h from the family will not shrink it much (say by 3/4) with high probability
  - (Though every h shrinks <u>some</u> small sets)
  - Relate shrinking to "hash collision probability"
    - $\Pr_h[h(x)=h(x')]$ (max over $x \neq x'$)
    - Exercise!

2-Universal Hash Family

2-Universal Hash Family

- (a.k.a pairwise-independent hashing)

2-Universal Hash Family

- (a.k.a pairwise-independent hashing)
- Family of functions h: U → R

2-Universal Hash Family

- (a.k.a pairwise-independent hashing)
- Family of functions $h: U \to R$
- $Pr_h[h(x)=y] = 1/|R|$ for all $x \in U$ and $y \in R$

2-Universal Hash Family

- (a.k.a pairwise-independent hashing)
- Family of functions $h: U \rightarrow R$
- $Pr_h[h(x)=y] = 1/|R|$ for all $x \in U$ and $y \in R$
- $Pr_h[h(x)=y \ \& \ h(x')=y'] = 1/|R|^2$ for all $x \neq x' \in U$ and $y, y' \in R$

2-Universal Hash Family

- (a.k.a pairwise-independent hashing)
- Family of functions $h: U \to R$
- $Pr_h[h(x)=y] = 1/|R|$ for all $x \in U$ and $y \in R$
- $Pr_h[h(x)=y \ \& \ h(x')=y'] = 1/|R|^2$ for all $x \neq x' \in U$ and $y, y' \in R$
  - E.g. in exercise

2-Universal Hash Family

- (a.k.a pairwise-independent hashing)
- Family of functions $h: U \to R$
- $Pr_h[h(x)=y] = 1/|R|$ for all $x \in U$ and $y \in R$
- $Pr_h[h(x)=y \ \&\ h(x')=y'] = 1/|R|^2$ for all $x \neq x' \in U$ and $y, y' \in R$
    - E.g. in exercise
- Hash collision probability = $1/|R|$

Public-coin protocol for Set lower-bound

Public-coin protocol for Set lower-bound

- Given a description of S and size K, to prove $|S|>K$ (if $|S|>2K$)

Public-coin protocol for Set lower-bound

- Given a description of S and size K, to prove |S|>K (if |S|>2K)
  - Verifier picks a random hash function h from a 2UHF family from U to R, with |R| = 8K (say), and a random element y in R, and sends (h,y) to Prover.

Public-coin protocol for Set lower-bound

- Given a description of S and size K, to prove |S|>K (if |S|>2K)
  - Verifier picks a random hash function h from a 2UHF family from U to R, with |R| = 8K (say), and a random element y in R, and sends (h,y) to Prover.
  - Prover sends back (if possible) $x \in S$ s.t. h(x)=y, with a certificate for $x \in S$

Public-coin protocol for Set lower-bound

- Given a description of S and size K, to prove $|S|>K$ (if $|S|>2K$)
  - Verifier picks a random hash function h from a 2UHF family from U to R, with $|R| = 8K$ (say), and a random element y in R, and sends (h,y) to Prover.
  - Prover sends back (if possible) $x \in S$ s.t. $h(x)=y$, with a certificate for $x \in S$
  - Verifier verifies $x \in S$ and $h(x)=y$, and if so outputs Yes

Public-coin protocol for Set lower-bound

- Given a description of S and size K, to prove |S|>K (if |S|>2K)
  - Verifier picks a random hash function h from a 2UHF family from U to R, with |R| = 8K (say), and a random element y in R, and sends (h,y) to Prover.
  - Prover sends back (if possible) $x \in S$ s.t. h(x)=y, with a certificate for $x \in S$
  - Verifier verifies $x \in S$ and h(x)=y, and if so outputs Yes
- Pr[Yes] has a constant gap between |S| > 2K and |S| < K [Exercise]