Interactive Proofs

Lecture 19

And Beyond

So far

So far

- IP = PSPACE = AM[poly]

So far

- IP = PSPACE = AM[poly]
  - PSPACE enough to calculate max Pr[yes]

So far

- IP = PSPACE = AM[poly]
  - PSPACE enough to calculate max Pr[yes]
  - AM[poly] protocol for TQBF using arithmetization

So far

- IP = PSPACE = AM[poly]
  - PSPACE enough to calculate max Pr[yes]
  - AM[poly] protocol for TQBF using arithmetization
- In fact IP[k] ⊆ AM[k+2] for all k(n)

So far

- IP = PSPACE = AM[poly]
  - PSPACE enough to calculate max Pr[yes]
  - AM[poly] protocol for TQBF using arithmetization
- In fact IP[k] ⊆ AM[k+2] for all k(n)
  - Using a public-coin set lower-bound proof

So far

- IP = PSPACE = AM[poly]
  - PSPACE enough to calculate max Pr[yes]
  - AM[poly] protocol for TQBF using arithmetization
- In fact IP[k] ⊆ AM[k+2] for all k(n)
  - Using a public-coin set lower-bound proof
- AM[k] = AM for constant k ≥ 2

So far

- IP = PSPACE = AM[poly]
  - PSPACE enough to calculate max Pr[yes]
  - AM[poly] protocol for TQBF using arithmetization
- In fact IP[k] ⊆ AM[k+2] for all k(n)
  - Using a public-coin set lower-bound proof
- AM[k] = AM for constant k ≥ 2
  - Using MA ⊆ AM and alternate characterization in terms of pairs of complementary ATTMs

So far

- IP = PSPACE = AM[poly]
  - PSPACE enough to calculate max Pr[yes]
  - AM[poly] protocol for TQBF using arithmetization
- In fact IP[k] ⊆ AM[k+2] for all k(n)
  - Using a public-coin set lower-bound proof
- AM[k] = AM for constant k ≥ 2
  - Using MA ⊆ AM and alternate characterization in terms of pairs of complementary ATTMs
- Perfect completeness: One-sided-error-AM = AM

So far

- IP = PSPACE = AM[poly]
  - PSPACE enough to calculate max Pr[yes]
  - AM[poly] protocol for TQBF using arithmetization
- In fact IP[k] ⊆ AM[k+2] for all k(n)
  - Using a public-coin set lower-bound proof
- AM[k] = AM for constant k ≥ 2
  - Using MA ⊆ AM and alternate characterization in terms of pairs of complementary ATTMs
- Perfect completeness: One-sided-error-AM = AM
  - Similar to BPP ⊆ Σ_2^P (yields MAM protocol; MAM=AM)

AM $\subseteq \Pi_2^P$

$AM \subseteq \Pi_2^P$

- Consider any L with an AM protocol

AM $\subseteq \Pi_2^P$

- Consider any L with an AM protocol
- By perfect completeness:

$AM \subseteq \Pi_2^P$

- Consider any L with an AM protocol
- By perfect completeness:
  - $x \in L \Rightarrow \forall\, y_{Arthur}\ \exists\, z_{Merlin}\ R(x, y_{Arthur}, z_{Merlin}) = 1$

$AM \subseteq \Pi_2^P$

- Consider any L with an AM protocol
- By perfect completeness:
  - $x \in L \Rightarrow \forall\, y_{Arthur}\ \exists\, z_{Merlin}\ R(x, y_{Arthur}, z_{Merlin}) = 1$
- And by (any positive) soundness:

$AM \subseteq \Pi_2^P$

- Consider any L with an AM protocol
- By perfect completeness:
  - $x \in L \Rightarrow \forall\, y_{Arthur}\ \exists\, z_{Merlin}\ R(x, y_{Arthur}, z_{Merlin}) = 1$
- And by (any positive) soundness:
  - $x \notin L \Rightarrow \exists\, y_{Arthur}\ \forall\, z_{Merlin}\ R(x, y_{Arthur}, z_{Merlin}) = 0$

$AM \subseteq \Pi_2^P$

- Consider any L with an AM protocol
- By perfect completeness:
  - $x \in L \Rightarrow \forall\ y_{Arthur}\ \exists\ z_{Merlin}\ R(x, y_{Arthur}, z_{Merlin}) = 1$
- And by (any positive) soundness:
  - $x \notin L \Rightarrow \exists\ y_{Arthur}\ \forall\ z_{Merlin}\ R(x, y_{Arthur}, z_{Merlin}) = 0$
- i.e., $x \in L \Leftrightarrow \forall y\ \exists z\ R(x,y,z) = 1$

$AM \subseteq \Pi_2^P$

- Consider any L with an AM protocol
- By perfect completeness:
  - $x \in L \Rightarrow \forall\ y_{Arthur}\ \exists\ z_{Merlin}\ R(x, y_{Arthur}, z_{Merlin}) = 1$
- And by (any positive) soundness:
  - $x \notin L \Rightarrow \exists\ y_{Arthur}\ \forall\ z_{Merlin}\ R(x, y_{Arthur}, z_{Merlin}) = 0$
- i.e., $x \in L \Leftrightarrow \forall y\ \exists z\ R(x,y,z) = 1$
- Similarly, $MA \subseteq \Sigma_2^P$

AM and coNP

AM and coNP

- If coNP $\subseteq$ AM, then PH collapses to level 2

AM and coNP

- If coNP ⊆ AM, then PH collapses to level 2
  - Will show coNP ⊆ AM ⇒ $\Sigma_2^P \subseteq AM \subseteq \Pi_2^P$

AM and coNP

- If coNP $\subseteq$ AM, then PH collapses to level 2
  - Will show coNP $\subseteq$ AM $\Rightarrow$ $\Sigma_2^P \subseteq$ AM $\subseteq \Pi_2^P$
  - L $\in \Sigma_2^P$: { x| $\exists$y (x,y) $\in$ L'} where L' $\in$ coNP

AM and coNP

- If coNP $\subseteq$ AM, then PH collapses to level 2
  - Will show coNP $\subseteq$ AM $\Rightarrow$ $\Sigma_2^P \subseteq$ AM $\subseteq \Pi_2^P$
  - $L \in \Sigma_2^P$: $\{ x \mid \exists y\ (x,y) \in L' \}$ where $L' \in$ coNP
  - MAM protocol for L: Merlin sends y, and then they run an AM protocol for $(x,y) \in L'$

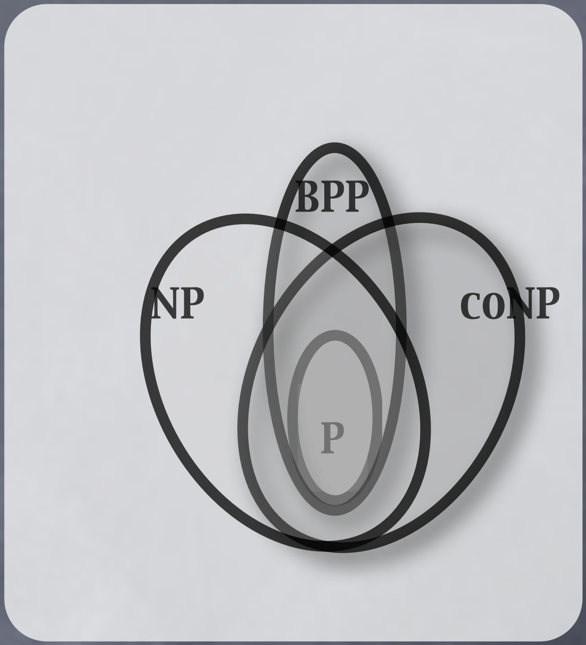
AM and coNP

- If coNP $\subseteq$ AM, then PH collapses to level 2
  - Will show coNP $\subseteq$ AM $\Rightarrow$ $\Sigma_2^P \subseteq$ AM $\subseteq \Pi_2^P$
  - L $\in \Sigma_2^P$: { x| $\exists$y (x,y) $\in$ L'} where L' $\in$ coNP
  - MAM protocol for L: Merlin sends y, and then they run an AM protocol for (x,y) $\in$ L'
    - But MAM = AM

AM and coNP

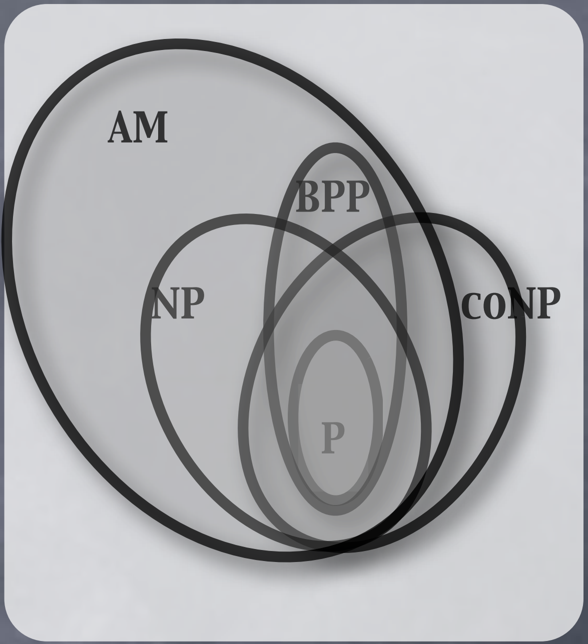
- If coNP $\subseteq$ AM, then PH collapses to level 2
  - Will show coNP $\subseteq$ AM $\Rightarrow \Sigma_2^P \subseteq$ AM $\subseteq \Pi_2^P$
  - L $\in \Sigma_2^P$: $\{ x | \exists y\ (x,y) \in L' \}$ where $L' \in$ coNP
  - MAM protocol for L: Merlin sends y, and then they run an AM protocol for $(x,y) \in L'$
    - But MAM = AM
- Corollary: If GI is NP-complete, PH collapses (recall GNI $\in$ AM)

AM and coNP



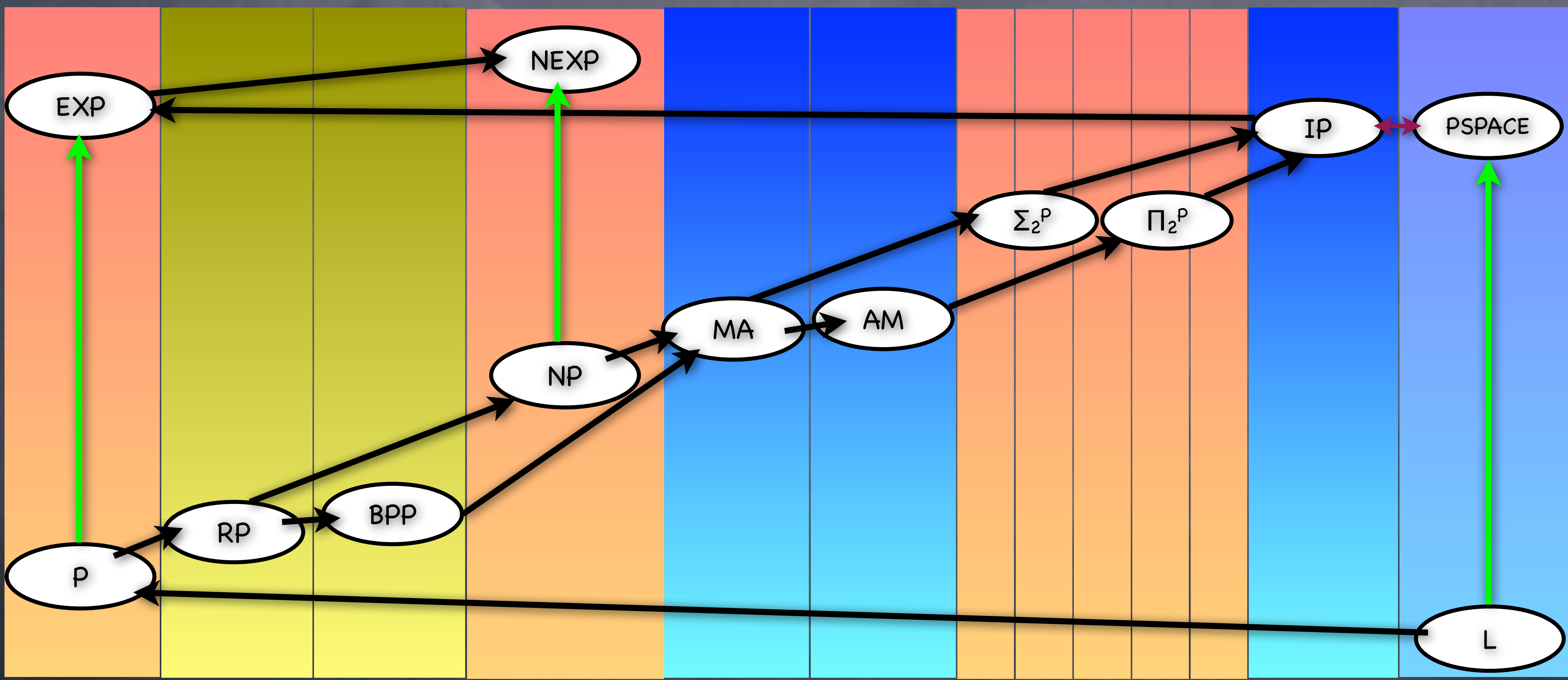
- If coNP ⊆ AM, then PH collapses to level 2
  - Will show coNP ⊆ AM ⇒ $\Sigma_2^P \subseteq AM \subseteq \Pi_2^P$
  - $L \in \Sigma_2^P$: { x| ∃y (x,y) ∈ L'} where L' ∈ coNP
  - MAM protocol for L: Merlin sends y, and then they run an AM protocol for (x,y) ∈ L'
    - But MAM = AM
- Corollary: If GI is NP-complete, PH collapses (recall GNI ∈ AM)

AM and coNP



- If coNP ⊆ AM, then PH collapses to level 2
  - Will show coNP ⊆ AM ⇒ $\Sigma_2^P \subseteq AM \subseteq \Pi_2^P$
  - $L \in \Sigma_2^P$: { x| ∃y (x,y) ∈ L'} where L' ∈ coNP
  - MAM protocol for L: Merlin sends y, and then they run an AM protocol for (x,y) ∈ L'
    - But MAM = AM
- Corollary: If GI is NP-complete, PH collapses (recall GNI ∈ AM)

Zoo



Program Checking

Program Checking

- Suppose a special computer (using nano-bio-quantum technology!) is being sold for solving Graph Non-Isomorphism (GNI) efficiently

Program Checking

- Suppose a special computer (using nano-bio-quantum technology!) is being sold for solving Graph Non-Isomorphism (GNI) efficiently
  - How do we trust this?

Program Checking

- Suppose a special computer (using nano-bio-quantum technology!) is being sold for solving Graph Non-Isomorphism (GNI) efficiently
  - How do we trust this?
- Vendor: Trust me, this always works

Program Checking

- Suppose a special computer (using nano-bio-quantum technology!) is being sold for solving Graph Non-Isomorphism (GNI) efficiently
  - How do we trust this?
- Vendor: Trust me, this always works
- User: In fact I just care if it works correctly on the inputs I want to solve. Maybe for each input I have, your machine could prove correctness using an IP protocol?

Program Checking

- Suppose a special computer (using nano-bio-quantum technology!) is being sold for solving Graph Non-Isomorphism (GNI) efficiently
  - How do we trust this?
- Vendor: Trust me, this always works
- User: In fact I just care if it works correctly on the inputs I want to solve. Maybe for each input I have, your machine could prove correctness using an IP protocol?
- Vendor: But I don't have a (nano-bio-quantum) implementation of the prover's program...

Program Checking

Program Checking

- Program checker

Program Checking

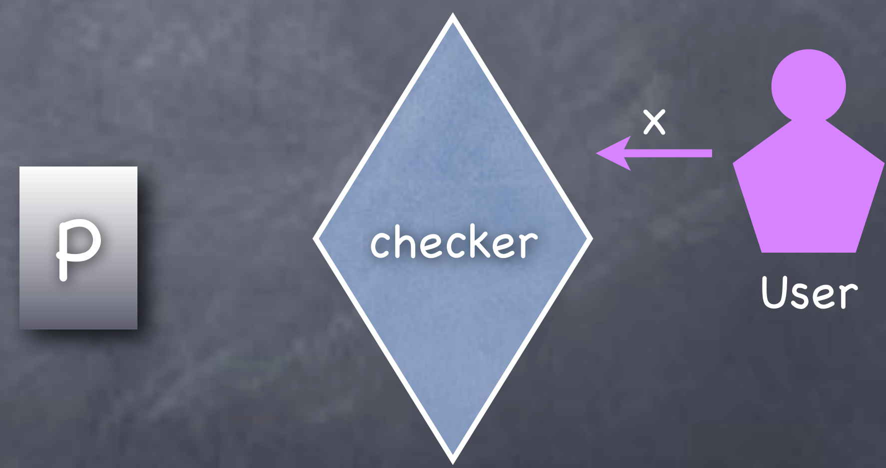
- Program checker

Program Checking

- Program checker

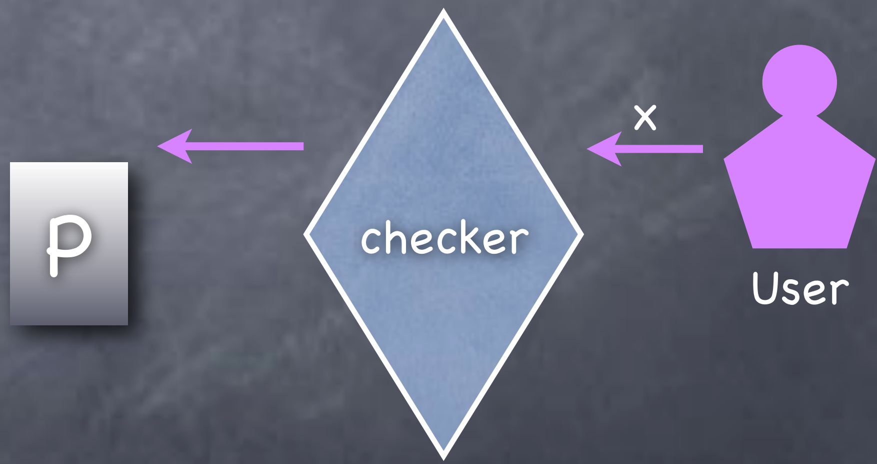
Program Checking

- Program checker



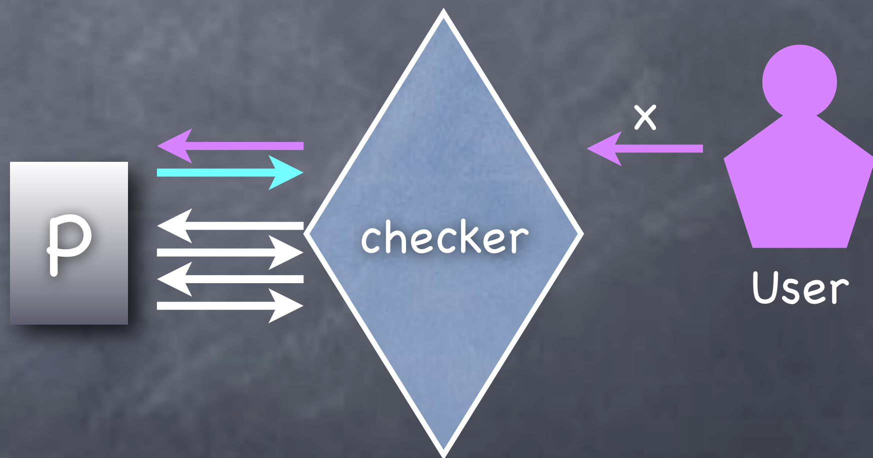
Program Checking

- Program checker



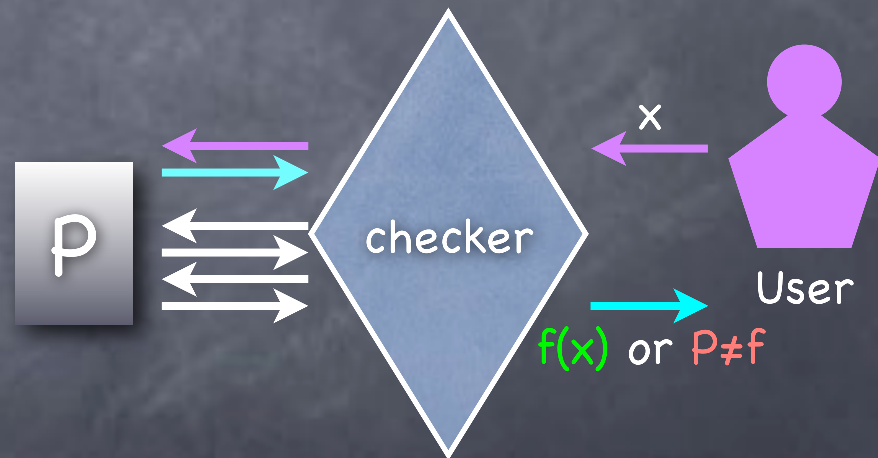
Program Checking

- Program checker



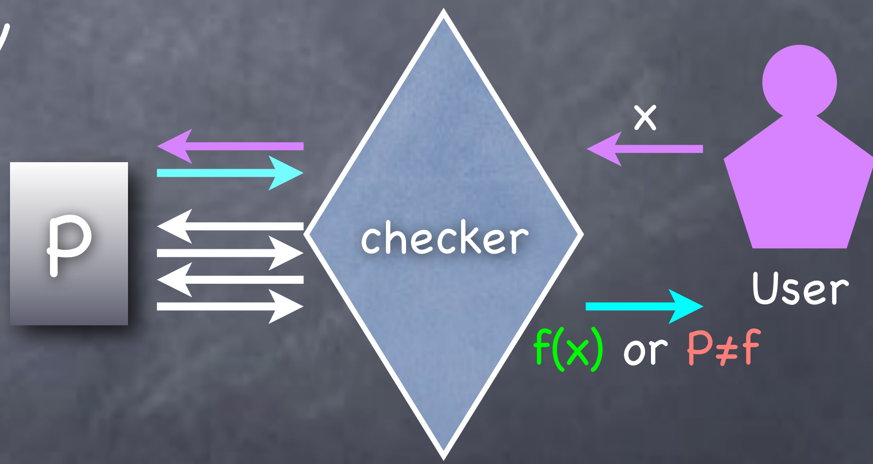
Program Checking

- Program checker



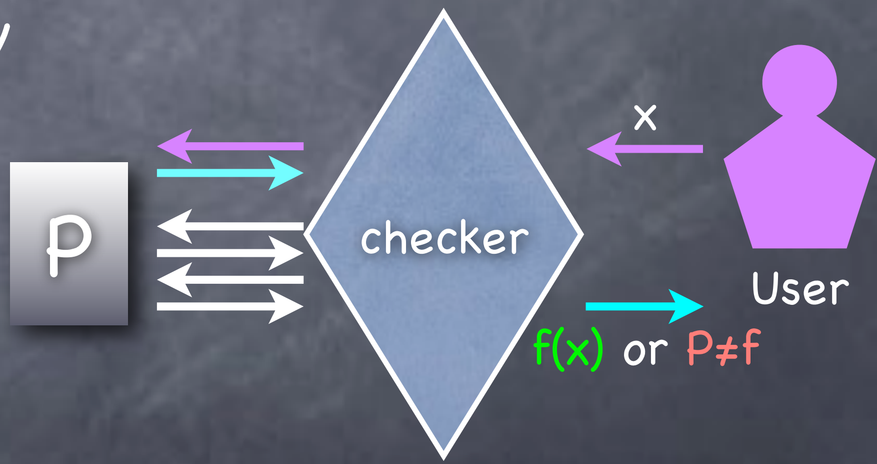
Program Checking

- Program checker
  - On each input, either ensures (w.h.p) that P's output is correct, or finds out that P≠f, efficiently



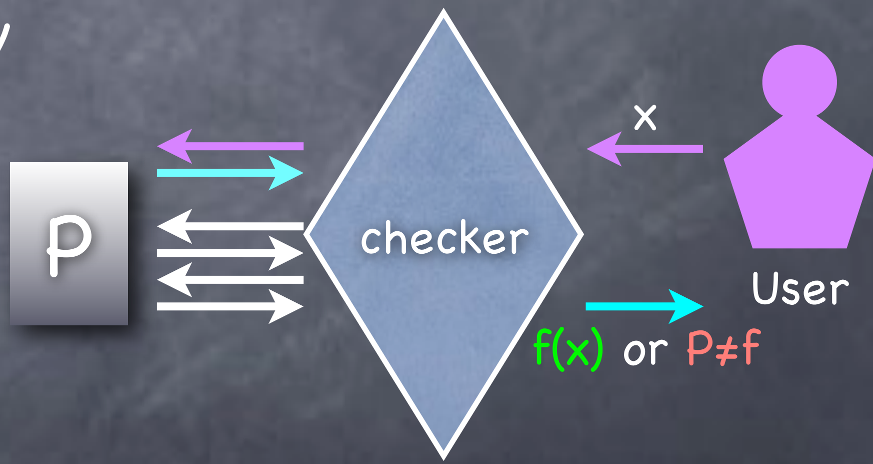
Program Checking

- Program checker
  - On each input, either ensures (w.h.p) that P's output is correct, or finds out that P≠f, efficiently
- Completeness: Vendor need not fear being falsely accused



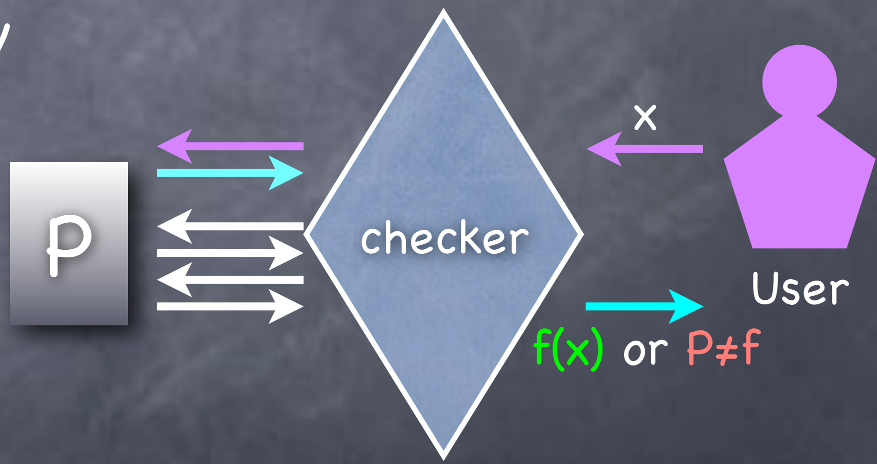
Program Checking

- Program checker
  - On each input, either ensures (w.h.p) that P's output is correct, or finds out that P≠f, efficiently
- Completeness: Vendor need not fear being falsely accused
- Soundness: User need not fear using a wrong value as f(x)

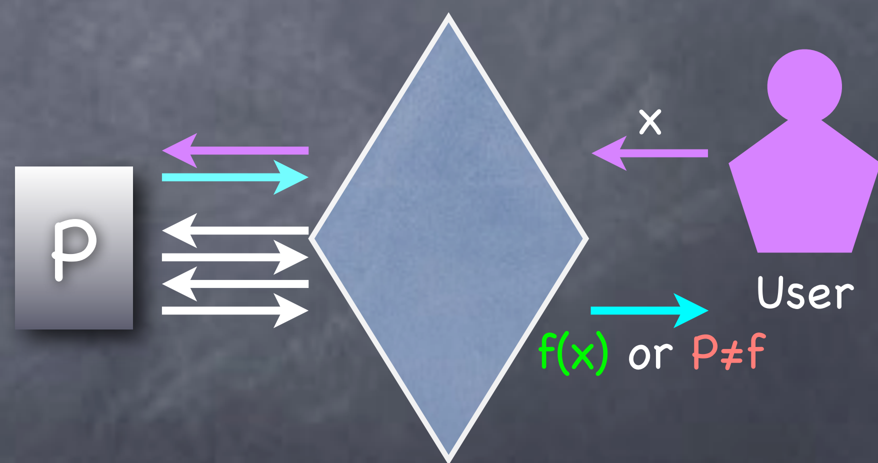


Program Checking

- Program checker
  - On each input, either ensures (w.h.p) that P's output is correct, or finds out that P≠f, efficiently
- Completeness: Vendor need not fear being falsely accused
- Soundness: User need not fear using a wrong value as f(x)
- Will consider boolean f (i.e., a language L)

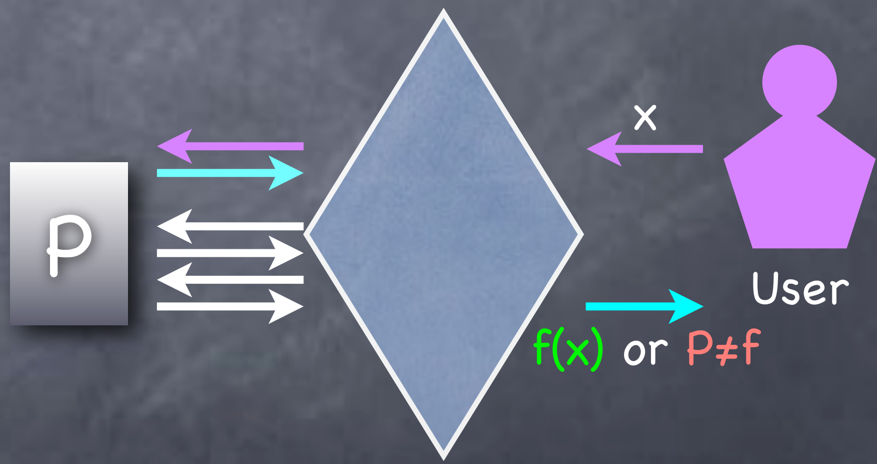


Program Checking and IP



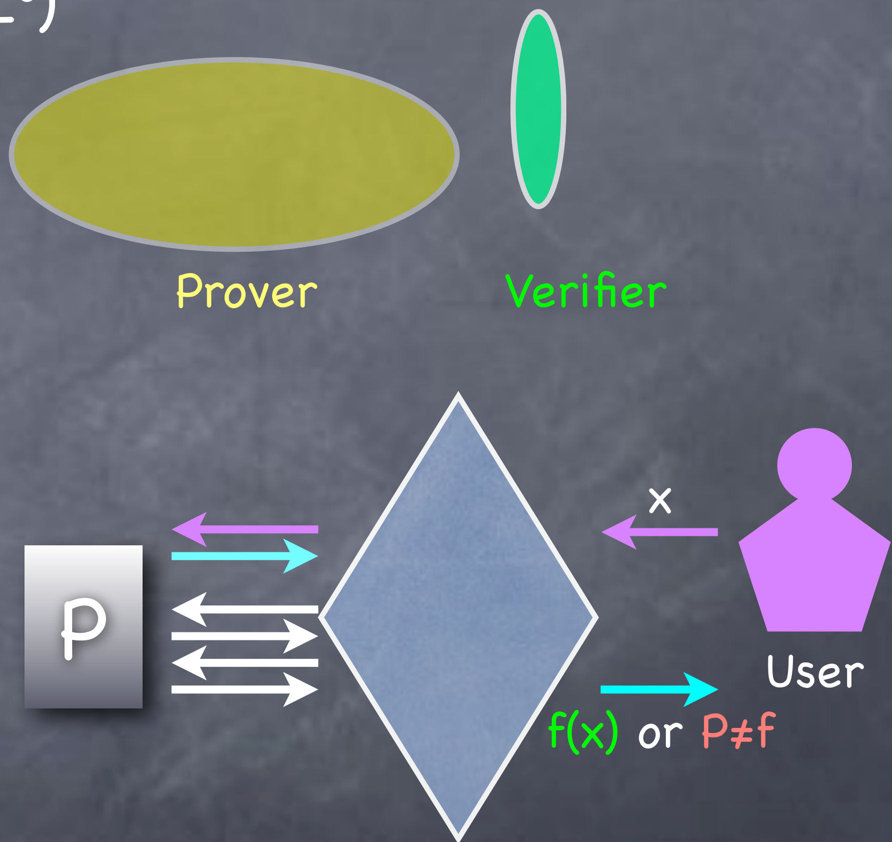
Program Checking and IP

- PC for L from IP protocols (for L and L^c)



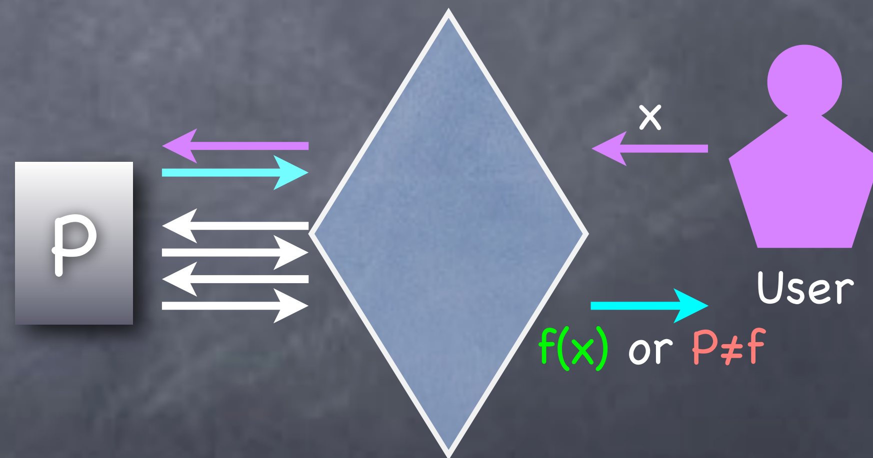
Program Checking and IP

- PC for L from IP protocols (for L and L^c)



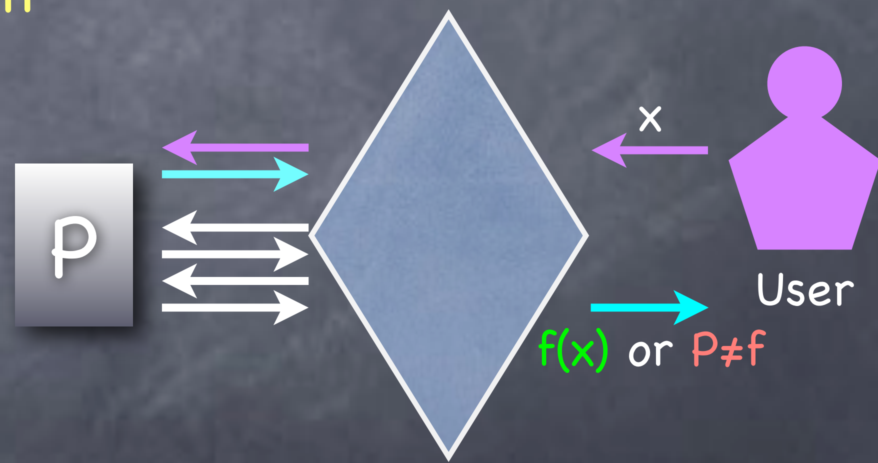
Program Checking and IP

- PC for L from IP protocols (for L and L^c)
  - PC must be efficient. Provers may not be



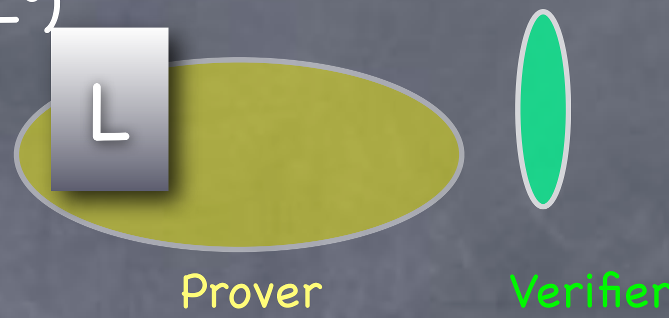
Program Checking and IP

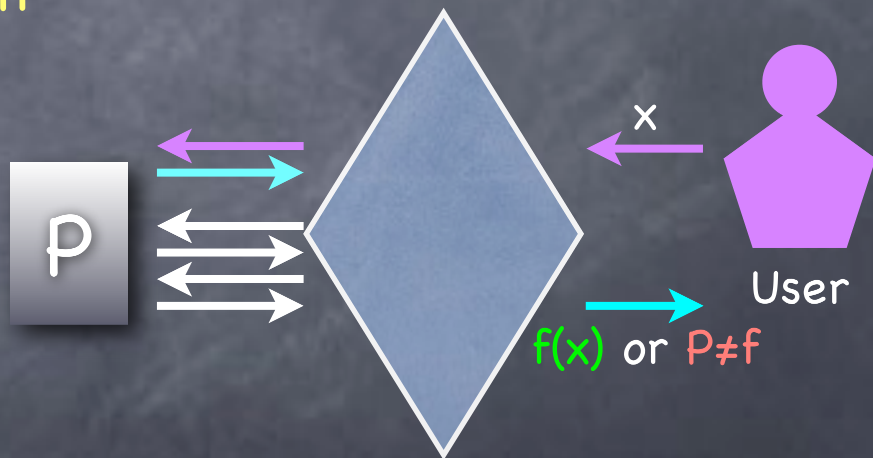
- PC for L from IP protocols (for L and L^c)
  - PC must be efficient. Provers may not be
  - If provers (for L and L^c) are efficient given L-oracle, can construct PC!



Program Checking and IP

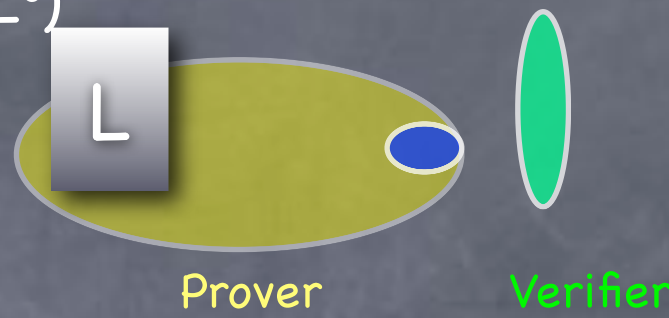
- PC for L from IP protocols (for L and L^c)
  - PC must be efficient. Provers may not be
  - If provers (for L and L^c) are efficient given L-oracle, can construct PC!

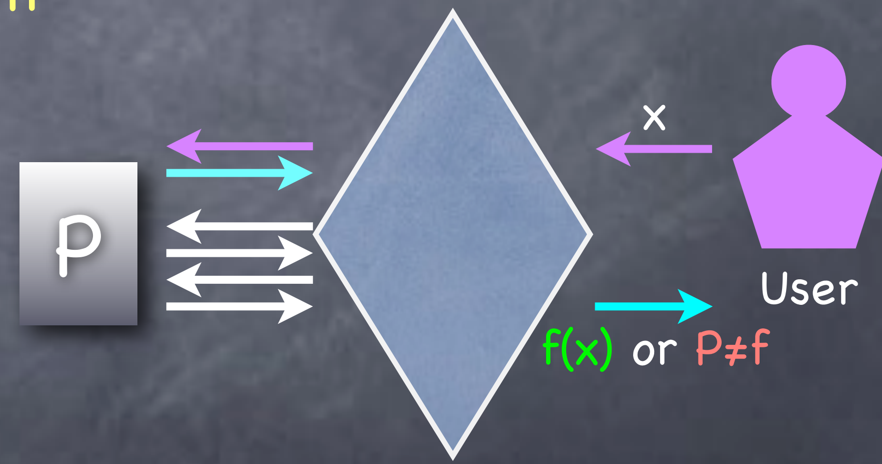




Program Checking and IP

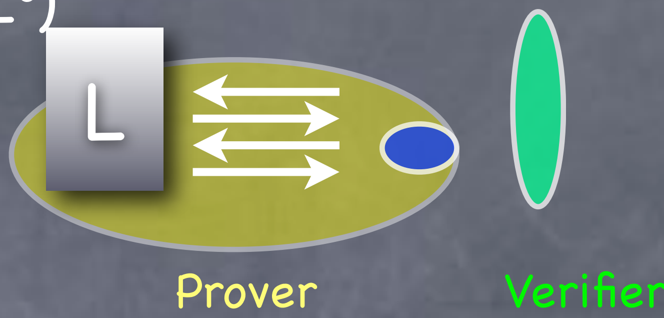
- PC for L from IP protocols (for L and L^c)
    - PC must be efficient. Provers may not be
    - If provers (for L and L^c) are efficient given L-oracle, can construct PC!

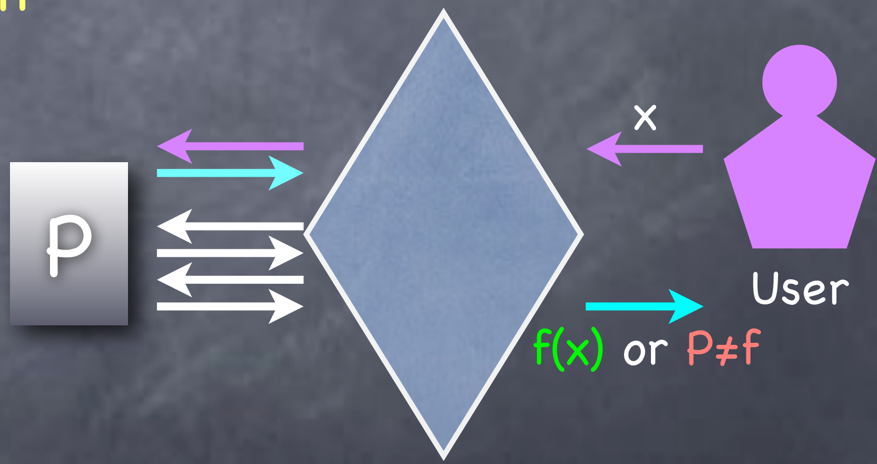




Program Checking and IP

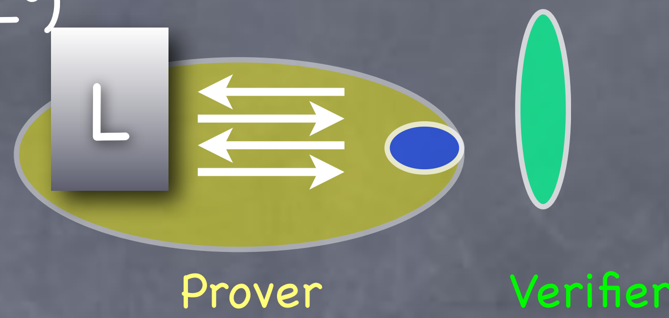
- PC for L from IP protocols (for L and L^c)
  - PC must be efficient. Provers may not be
  - If provers (for L and L^c) are efficient given L-oracle, can construct PC!

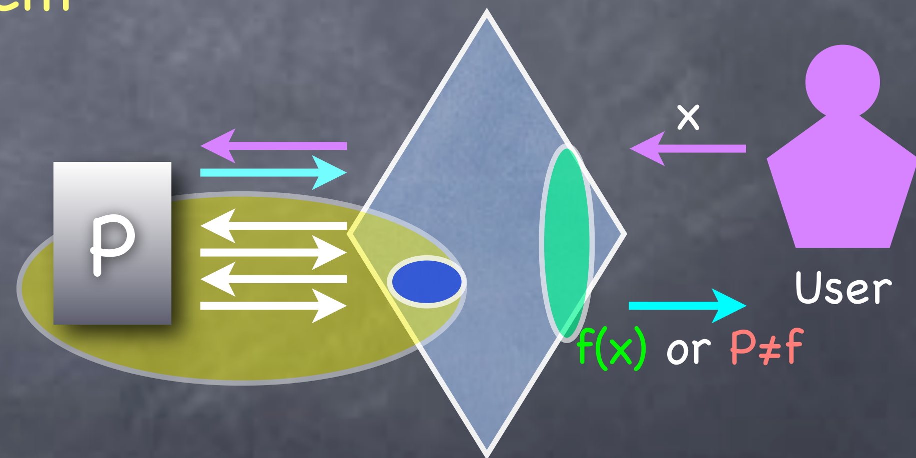




Program Checking and IP

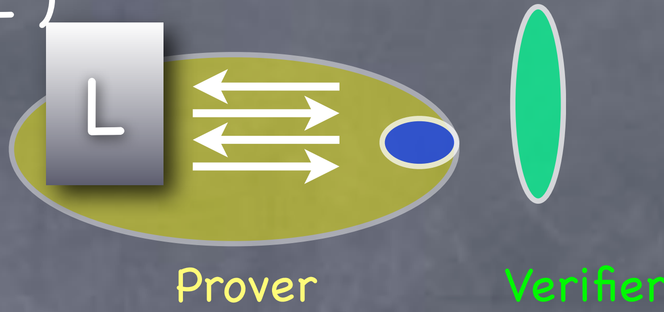
- PC for L from IP protocols (for L and L^c)
  - PC must be efficient. Provers may not be
  - If provers (for L and L^c) are efficient given L-oracle, can construct PC!

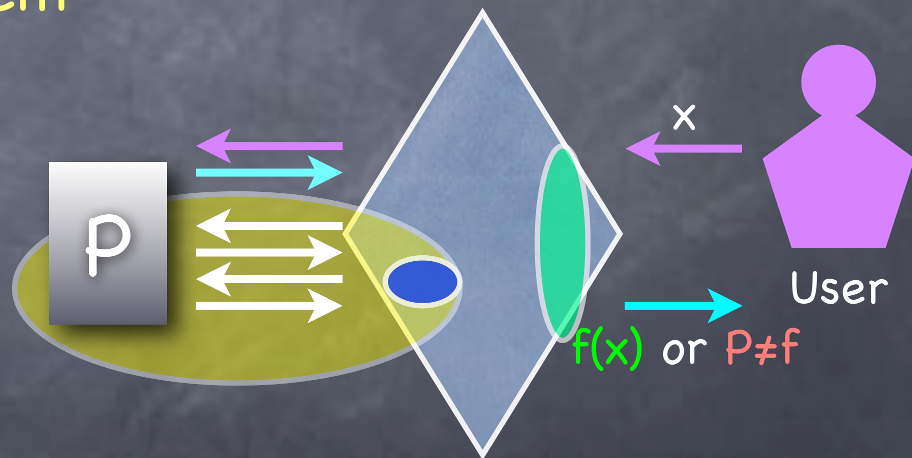




Program Checking and IP

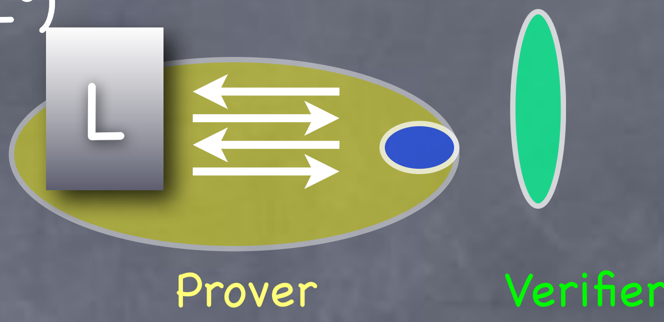
- PC for L from IP protocols (for L and L^c)
  - PC must be efficient. Provers may not be
  - If provers (for L and L^c) are efficient given L-oracle, can construct PC!
    - Retains completeness and soundness

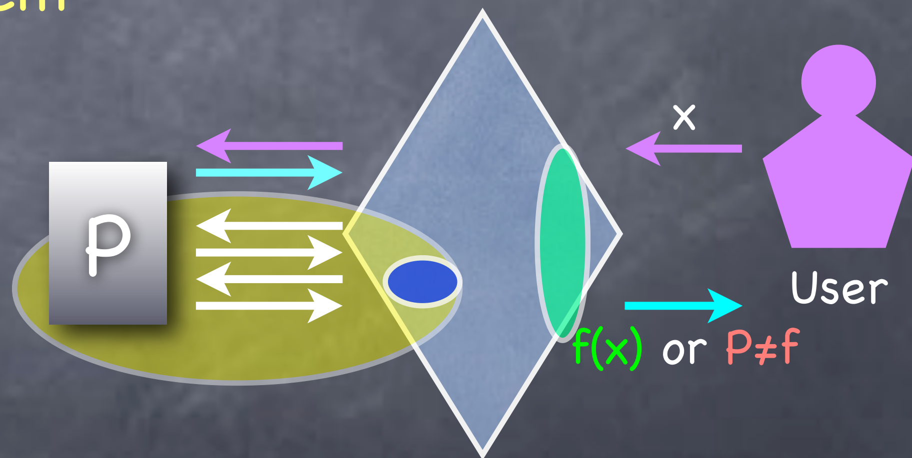




Program Checking and IP

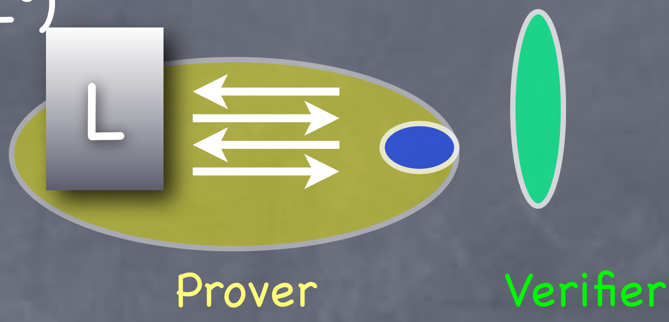
- PC for L from IP protocols (for L and L^c)
  - PC must be efficient. Provers may not be
  - If provers (for L and L^c) are efficient given L-oracle, can construct PC!
    - Retains completeness and soundness
- e.g. For PSPACE-complete L (why?)

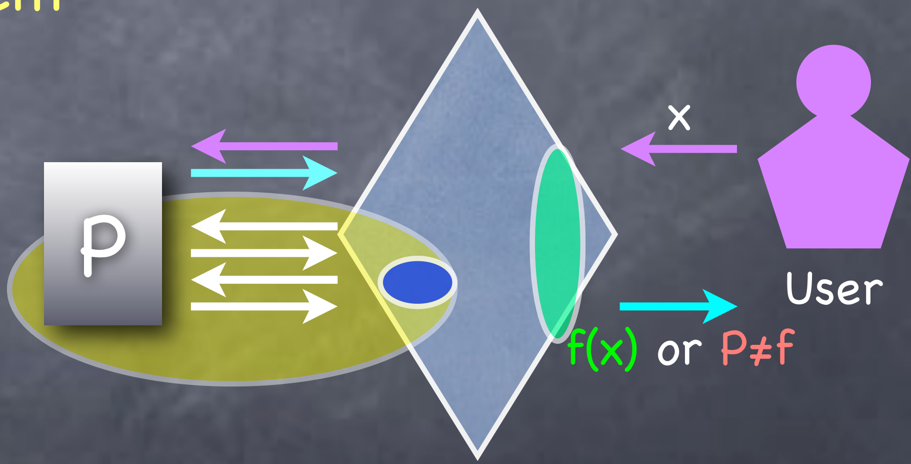




Program Checking and IP

- PC for L from IP protocols (for L and L^c)
  - PC must be efficient. Provers may not be
  - If provers (for L and L^c) are efficient given L-oracle, can construct PC!
    - Retains completeness and soundness
- e.g. For PSPACE-complete L (why?)
- How about Graph Isomorphism?





Program Checking for GI

Program Checking for GI

- If $P(G_0,G_1)$ says $G_0 \equiv G_1$, try to extract the isomorphism

Program Checking for GI

- If $P(G_0,G_1)$ says $G_0 \equiv G_1$, try to extract the isomorphism
  - Pick a node v in G_0. For each node u in G_1 and ask for isomorphism of $(G_0\backslash v, G_1\backslash u)$

Program Checking for GI

- If $P(G_0,G_1)$ says $G_0 \equiv G_1$, try to extract the isomorphism
  - Pick a node v in G_0. For each node u in G_1 and ask for isomorphism of $(G_0\backslash v, G_1\backslash u)$
  - If P says no for all u in G_1, report "P bad"

Program Checking for GI

- If $P(G_0,G_1)$ says $G_0 \equiv G_1$, try to extract the isomorphism
  - Pick a node v in G_0. For each node u in G_1 and ask for isomorphism of $(G_0\backslash v,\ G_1\backslash u)$
  - If P says no for all u in G_1, report "P bad"
  - Else remember $v \mapsto u$, and recurse on $(G_0\backslash v,\ G_1\backslash u)$

Program Checking for GI

- If $P(G_0,G_1)$ says $G_0 \equiv G_1$, try to extract the isomorphism
  - Pick a node v in G_0. For each node u in G_1 and ask for isomorphism of $(G_0\backslash v, G_1\backslash u)$
  - If P says no for all u in G_1, report "P bad"
  - Else remember $v \mapsto u$, and recurse on $(G_0\backslash v, G_1\backslash u)$
  - On finding isomorphism, verify and output $G_0 \equiv G_1$

Program Checking for GI

- If $P(G_0,G_1)$ says $G_0 \equiv G_1$, try to extract the isomorphism
  - Pick a node v in G_0. For each node u in G_1 and ask for isomorphism of $(G_0\backslash v,\ G_1\backslash u)$
  - If P says no for all u in G_1, report "P bad"
  - Else remember $v \mapsto u$, and recurse on $(G_0\backslash v,\ G_1\backslash u)$
  - On finding isomorphism, verify and output $G_0 \equiv G_1$
- Note: An IP protocol (i.e., an NP proof) for GI, where prover is in P^{GI}

Program Checking for GI

Program Checking for GI

- If $P(G_0,G_1)$ says $G_0 \not\equiv G_1$, test P similar to in IP protocol for GNI (coke from can/bottle)

Program Checking for GI

- If $P(G_0,G_1)$ says $G_0 \not\equiv G_1$, test P similar to in IP protocol for GNI (coke from can/bottle)
  - Let $H = \pi(G_b)$ where π is a random permutation and $b = 0$ or 1 at random

Program Checking for GI

- If $P(G_0,G_1)$ says $G_0 \not\cong G_1$, test P similar to in IP protocol for GNI (coke from can/bottle)
  - Let $H = \pi(G_b)$ where π is a random permutation and $b = 0$ or 1 at random
  - Run $P(G_0,H)$ many times

Program Checking for GI

- If $P(G_0,G_1)$ says $G_0 \not\equiv G_1$, test P similar to in IP protocol for GNI (coke from can/bottle)
  - Let $H = \pi(G_b)$ where π is a random permutation and $b = 0$ or 1 at random
  - Run $P(G_0,H)$ many times
  - If P says $G_0 \equiv H$ exactly whenever $b=0$, output $G_0 \not\equiv G_1$

Program Checking for GI

- If $P(G_0,G_1)$ says $G_0 \not\equiv G_1$, test P similar to in IP protocol for GNI (coke from can/bottle)
  - Let $H = \pi(G_b)$ where π is a random permutation and $b = 0$ or 1 at random
  - Run $P(G_0,H)$ many times
  - If P says $G_0 \equiv H$ exactly whenever $b=0$, output $G_0 \not\equiv G_1$
  - Else output "Bad P"

Program Checking for GI

- If $P(G_0,G_1)$ says $G_0 \not\equiv G_1$, test P similar to in IP protocol for GNI (coke from can/bottle)
  - Let $H = \pi(G_b)$ where π is a random permutation and $b = 0$ or 1 at random
  - Run $P(G_0,H)$ many times
  - If P says $G_0 \equiv H$ exactly whenever $b=0$, output $G_0 \not\equiv G_1$
  - Else output "Bad P"
- Note: Prover in the IP protocol for GNI is in P^{GI}

Multi-Prover Interactive Proofs

Multi-Prover Interactive Proofs

- Interrogate multiple provers separately

Multi-Prover Interactive Proofs

- Interrogate multiple provers separately
  - Provers can't talk to each other during the interrogation (but can agree on a strategy a priori)

Multi-Prover Interactive Proofs

- Interrogate multiple provers separately
  - Provers can't talk to each other during the interrogation (but can agree on a strategy a priori)
  - Verifier cross-checks answers from the provers

Multi-Prover Interactive Proofs

- Interrogate multiple provers separately
  - Provers can't talk to each other during the interrogation (but can agree on a strategy a priori)
  - Verifier cross-checks answers from the provers
  - 2 provers as good as k provers

Multi-Prover Interactive Proofs

- Interrogate multiple provers separately
  - Provers can't talk to each other during the interrogation (but can agree on a strategy a priori)
  - Verifier cross-checks answers from the provers
  - 2 provers as good as k provers
  - MIP = NEXP

Multi-Prover Interactive Proofs

- Interrogate multiple provers separately
  - Provers can't talk to each other during the interrogation (but can agree on a strategy a priori)
  - Verifier cross-checks answers from the provers
  - 2 provers as good as k provers
  - MIP = NEXP
  - Parallel repetition theorem highly non-trivial!

Probabilistically Checkable Proofs (PCPs)

Probabilistically Checkable Proofs (PCPs)

- Prover submits a (very long) written proof

Probabilistically Checkable Proofs (PCPs)

- Prover submits a (very long) written proof
  - Verifier reads some positions (probabilistically chosen) from the proof and decides to accept or reject

Probabilistically Checkable Proofs (PCPs)

- Prover submits a (very long) written proof
  - Verifier reads some positions (probabilistically chosen) from the proof and decides to accept or reject
- PCP[r,q]: length of proof 2^r, number of queries q

Probabilistically Checkable Proofs (PCPs)

- Prover submits a (very long) written proof
  - Verifier reads some positions (probabilistically chosen) from the proof and decides to accept or reject
- PCP[r,q]: length of proof 2^r, number of queries q
- Intuitively, in MIP, the provers cannot change their strategy (because one does not know what the other sees), so must stick to a prior agreed up on strategy

Probabilistically Checkable Proofs (PCPs)

- Prover submits a (very long) written proof
  - Verifier reads some positions (probabilistically chosen) from the proof and decides to accept or reject
- PCP[r,q]: length of proof 2^r, number of queries q
- Intuitively, in MIP, the provers cannot change their strategy (because one does not know what the other sees), so must stick to a prior agreed up on strategy
  - Which will be the written proof

Probabilistically Checkable Proofs (PCPs)

- Prover submits a (very long) written proof
  - Verifier reads some positions (probabilistically chosen) from the proof and decides to accept or reject
- PCP[r,q]: length of proof 2^r, number of queries q
- Intuitively, in MIP, the provers cannot change their strategy (because one does not know what the other sees), so must stick to a prior agreed up on strategy
  - Which will be the written proof
  - PCP[poly,poly] = MIP = NEXP

PCP Theorem

PCP Theorem

- NP = PCP[log,const]

PCP Theorem

- **NP = PCP[log,const]**
  - PCP is only poly long (just like usual NP certificate)

PCP Theorem

- **NP = PCP[log,const]**
  - PCP is only poly long (just like usual NP certificate)
  - But verifier reads only constantly many bits!

PCP Theorem

- **NP = PCP[log,const]**
  - PCP is only poly long (just like usual NP certificate)
  - But verifier reads only constantly many bits!
  - Extensively useful in proving "hardness of approximation" results for optimization problems

PCP Theorem

- **NP = PCP[log,const]**
  - PCP is only poly long (just like usual NP certificate)
  - But verifier reads only constantly many bits!
  - Extensively useful in proving "hardness of approximation" results for optimization problems
    - Also useful in certain cryptographic protocols

Zero-Knowledge Proofs

Zero-Knowledge Proofs

- Interactive Proof for membership in L

Zero-Knowledge Proofs

- Interactive Proof for membership in L
  - Complete and Sound

Zero-Knowledge Proofs

- Interactive Proof for membership in L
  - Complete and Sound
- ZK Property: Verifier "learns nothing" except that x is in L

Zero-Knowledge Proofs

- Interactive Proof for membership in L
  - Complete and Sound
- ZK Property: Verifier "learns nothing" except that x is in L

Zero-Knowledge Proofs

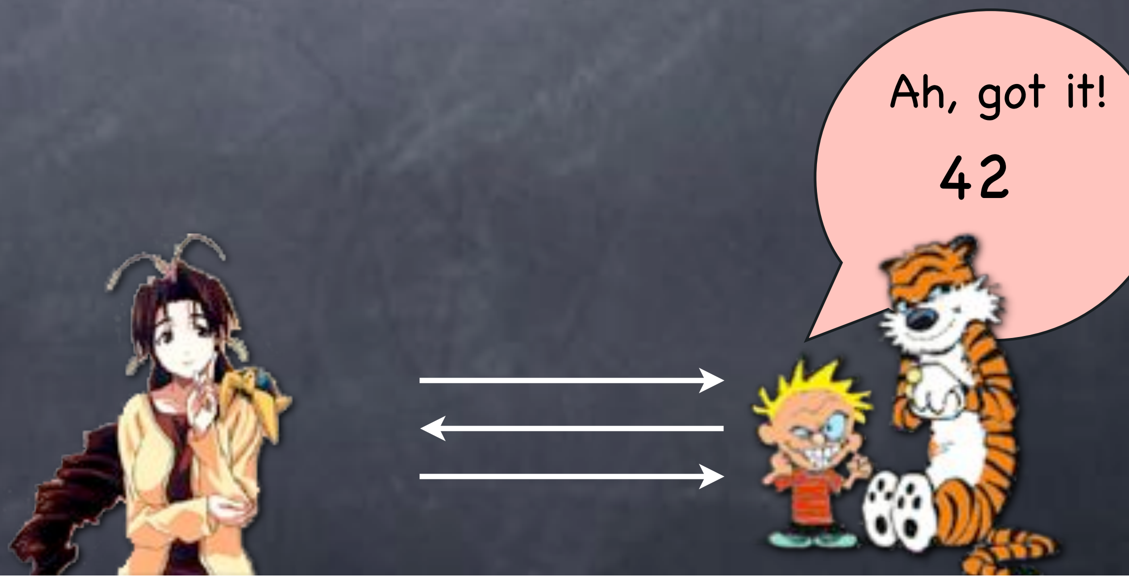
- Interactive Proof for membership in L
  - Complete and Sound
- ZK Property: Verifier "learns nothing" except that x is in L

Zero-Knowledge Proofs

- Interactive Proof for membership in L
  - Complete and Sound
- ZK Property: Verifier "learns nothing" except that x is in L

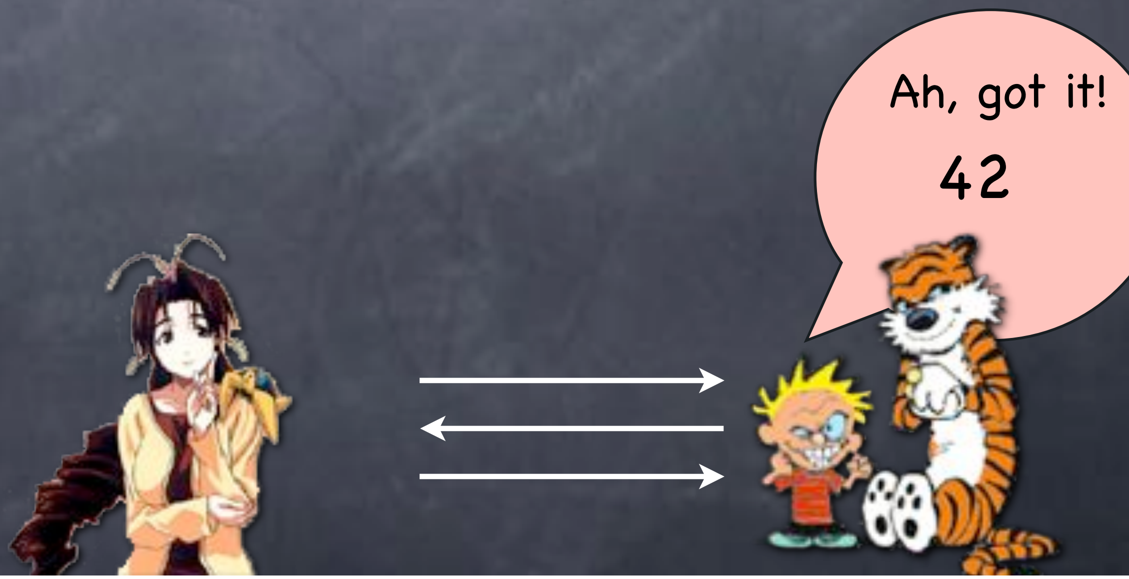
Zero-Knowledge Proofs

- Interactive Proof for membership in L
    - Complete and Sound
- ZK Property: Verifier "learns nothing" except that x is in L



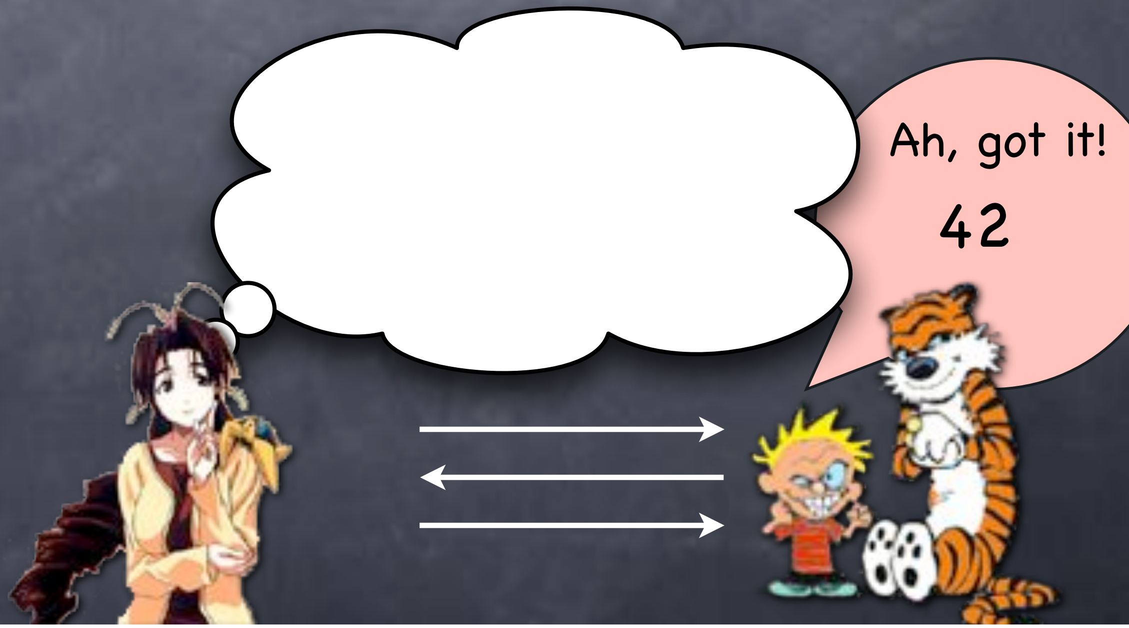
Zero-Knowledge Proofs

- Interactive Proof for membership in L
  - Complete and Sound
- ZK Property: Verifier "learns nothing" except that x is in L
  - Verifier's view could have been "simulated"



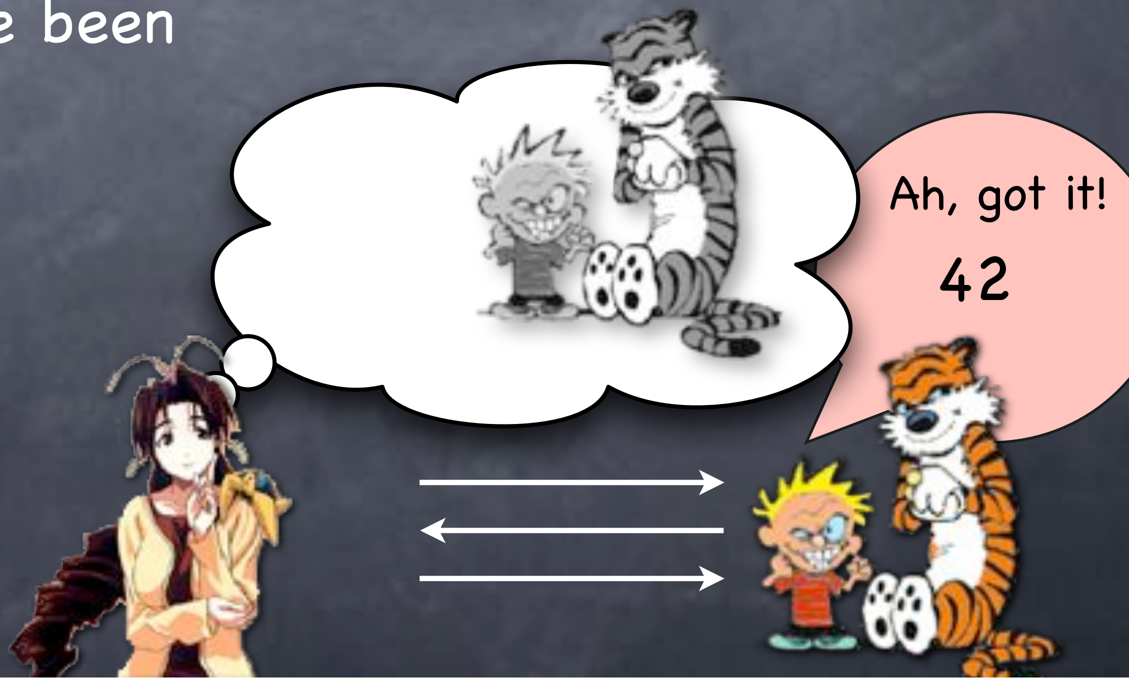
Zero-Knowledge Proofs

- Interactive Proof for membership in L
  - Complete and Sound
- ZK Property: Verifier "learns nothing" except that x is in L
  - Verifier's view could have been "simulated"



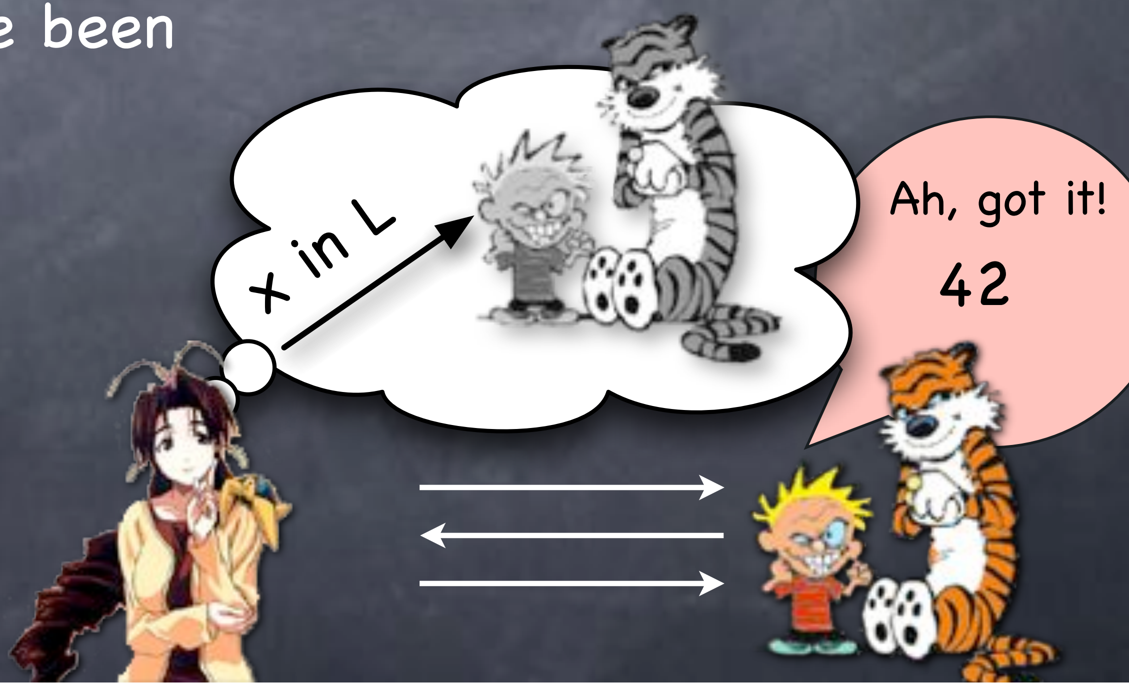
Zero-Knowledge Proofs

- Interactive Proof for membership in L
  - Complete and Sound
- ZK Property: Verifier "learns nothing" except that x is in L
  - Verifier's view could have been "simulated"



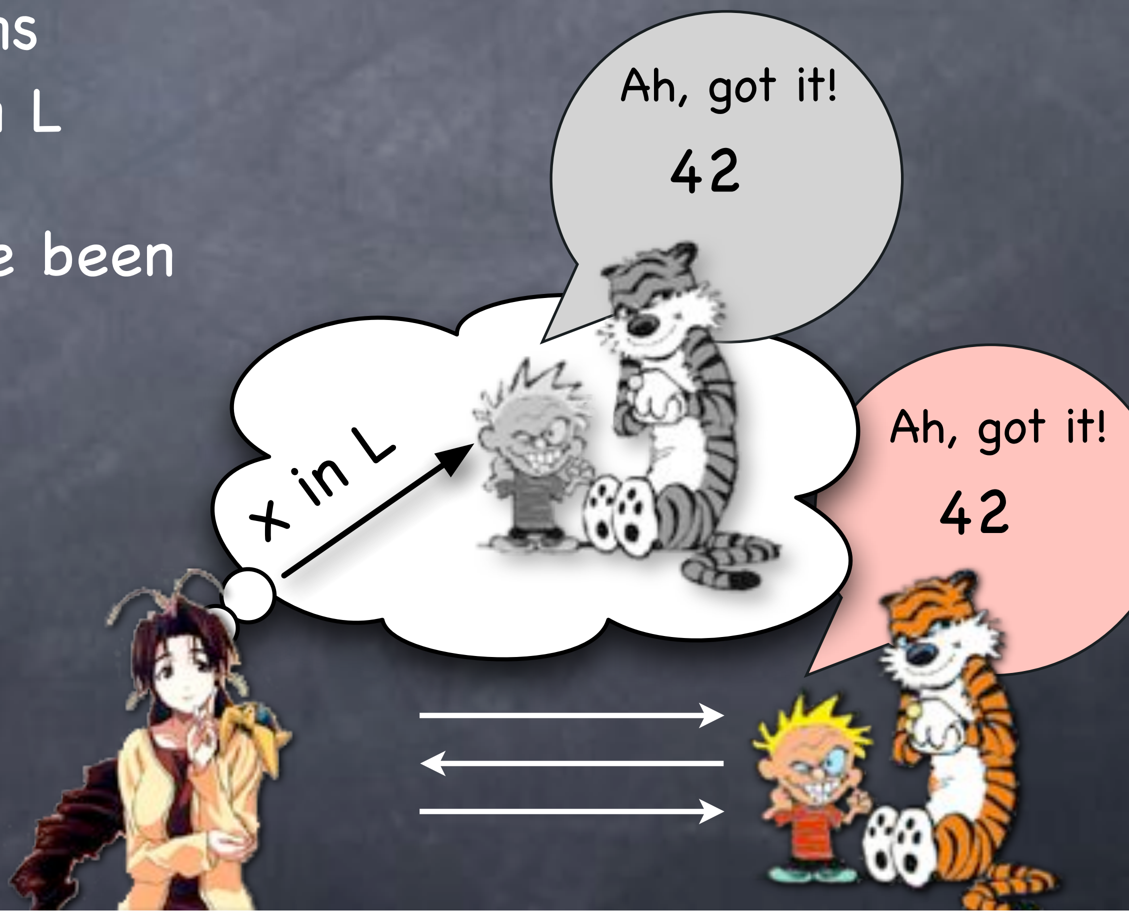
Zero-Knowledge Proofs

- Interactive Proof for membership in L
  - Complete and Sound
- ZK Property: Verifier "learns nothing" except that x is in L
  - Verifier's view could have been "simulated"



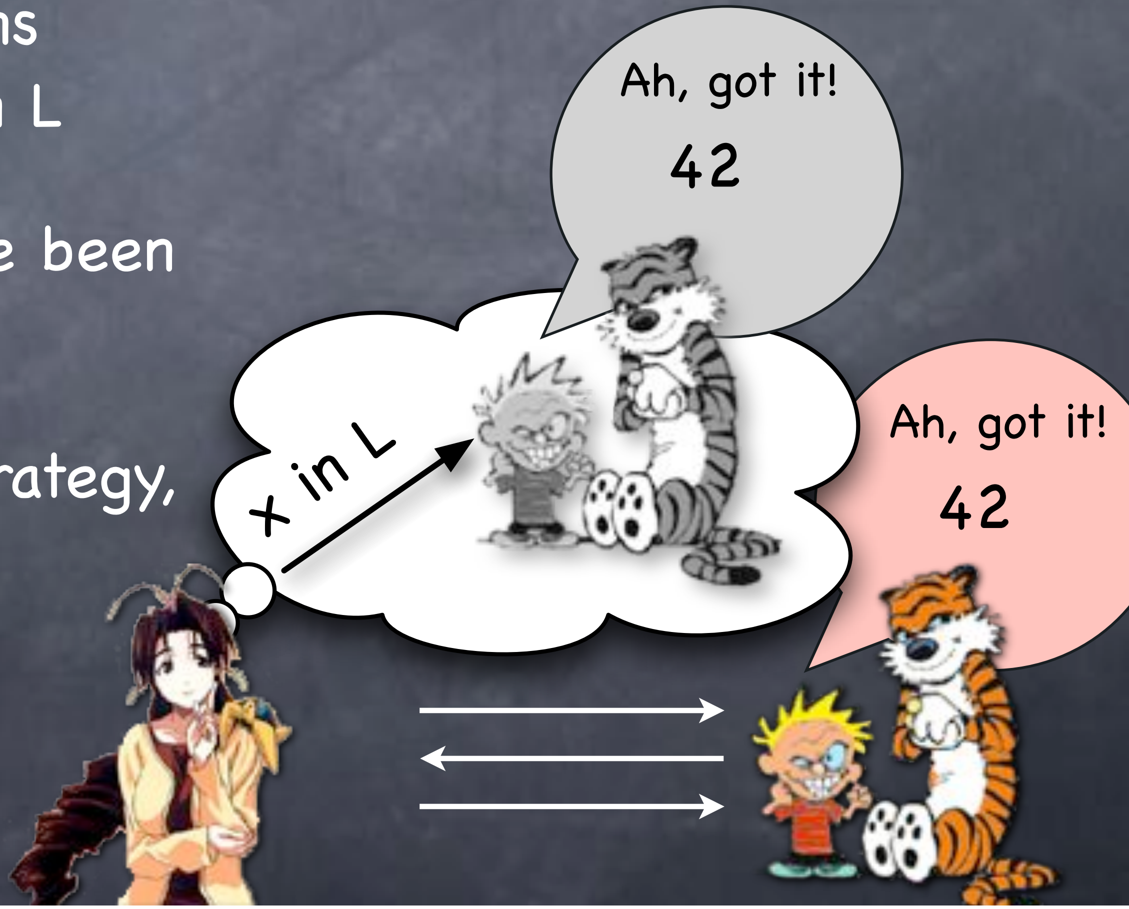
Zero-Knowledge Proofs

- Interactive Proof for membership in L
  - Complete and Sound
- ZK Property: Verifier "learns nothing" except that x is in L
  - Verifier's view could have been "simulated"



Zero-Knowledge Proofs

- Interactive Proof for membership in L
  - Complete and Sound
- ZK Property: Verifier "learns nothing" except that x is in L
  - Verifier's view could have been "simulated"
  - For every adversarial strategy, there exists a simulation strategy



Summary

Summary

- Interactive Protocols

Summary

- Interactive Protocols
  - Public coins, ATTMs, collapse of AM[k], arithmetization, set lower-bound, perfect completeness

Summary

- Interactive Protocols
  - Public coins, ATTMs, collapse of AM[k], arithmetization, set lower-bound, perfect completeness
  - Zoo: MA and AM, between 1st and 2nd levels of PH

Summary

- Interactive Protocols
  - Public coins, ATTMs, collapse of AM[k], arithmetization, set lower-bound, perfect completeness
  - Zoo: MA and AM, between 1st and 2nd levels of PH
- Other related concepts

Summary

- Interactive Protocols
  - Public coins, ATTMs, collapse of AM[k], arithmetization, set lower-bound, perfect completeness
  - Zoo: MA and AM, between 1st and 2nd levels of PH
- Other related concepts
  - MIP, PCP, ZK proofs

Summary

- Interactive Protocols
  - Public coins, ATTMs, collapse of AM[k], arithmetization, set lower-bound, perfect completeness
  - Zoo: MA and AM, between 1st and 2nd levels of PH
- Other related concepts
  - MIP, PCP, ZK proofs
- Understanding power of interaction/non-determinism and randomness

Summary

- Interactive Protocols
  - Public coins, ATTMs, collapse of AM[k], arithmetization, set lower-bound, perfect completeness
  - Zoo: MA and AM, between 1st and 2nd levels of PH
- Other related concepts
  - MIP, PCP, ZK proofs
- Understanding power of interaction/non-determinism and randomness
  - Useful in "hardness of approximation", in cryptography, ...