Interactive Proofs

Or how I stopped worrying and learned to ask questions

Dhruva Kashyap, November 2022
Department of Computer Science and Automation, IISc, Bengaluru

What are proofs?

What are proofs?

- A proof tries to assert the correctness(or incorrectness) of a *given statement*

What are proofs?

- A proof tries to assert the correctness(or incorrectness) of a *given statement*

- A short sequence of logical statements which are either axiomatic or consequences of previous statements in the sequence, which assert the truthiness of the *given statement*.

What are proofs?

- A proof tries to assert the correctness(or incorrectness) of a *given statement*

- A short sequence of logical statements which are either axiomatic or consequences of previous statements in the sequence, which assert the truthiness of the *given statement*.

- If there is a correct proof, then the *given statement* is true

What are proofs?

- A proof tries to assert the correctness(or incorrectness) of a *given statement*
- A short sequence of logical statements which are either axiomatic or consequences of previous statements in the sequence, which assert the truthiness of the *given statement*.
- If there is a correct proof, then the *given statement* is true
- If there is no proof, then the *given statement* must be false

What are interactive proofs?

What are interactive proofs?

- We can model the concept of a proof as an interaction between a “prover” and a “verifier”.

What are interactive proofs?

- We can model the concept of a proof as an interaction between a “prover” and a “verifier”.

- The goal of a verifier is to assert the correctness of a statement.

What are interactive proofs?

- We can model the concept of a proof as an interaction between a “prover” and a “verifier”.

- The goal of a verifier is to assert the correctness of a statement.

- A verifier interrogates the prover with questions related to a statement, prover answers with intent to convince the verifier of the correctness of the statement

What are interactive proofs?

- We can model the concept of a proof as an interaction between a “prover” and a “verifier”.

How good is the prover?

- The goal of a verifier is to assert the correctness of a statement.
- A verifier interrogates the prover with questions related to a statement, prover answers with intent to convince the verifier of the correctness of the statement

What are interactive proofs?

- We can model the concept of a proof as an interaction between a “prover” and a “verifier”.

How good is the prover?

??

- The goal of a verifier is to assert the correctness of a statement.

- A verifier interrogates the prover with questions related to a statement, prover answers with intent to convince the verifier of the correctness of the statement

What are interactive proofs?

- We can model the concept of a proof as an interaction between a “prover” and a “verifier”.

How good is the prover?

??

- The goal of a verifier is to assert the correctness of a statement.

How much interaction?

- A verifier interrogates the prover with questions related to a statement, prover answers with intent to convince the verifier of the correctness of the statement

Interactive Proofs and Complexity Theory: Hiding behind ∃

Interactive Proofs and Complexity Theory: Hiding behind ∃

- Can we capture NP using an interactive proof? Yes!

Interactive Proofs and Complexity Theory: Hiding behind ∃

- Can we capture NP using an interactive proof? Yes!

- A poly-time machine asking a “machine” to provide a certificate.

Interactive Proofs and Complexity Theory: Hiding behind ∃

- Can we capture NP using an interactive proof? Yes!

- A poly-time machine asking a “machine” to provide a certificate.

Verifier

Interactive Proofs and Complexity Theory: Hiding behind ∃

- Can we capture NP using an interactive proof? Yes!

- A poly-time machine asking a “machine” to provide a certificate.

Verifier

Prover

Interactive Proofs and Complexity Theory: Hiding behind ∃

- Can we capture NP using an interactive proof? Yes!

- A poly-time machine asking a “machine” to provide a certificate.

  Verifier Prover

- Here, the verifier V, is a polynomial time Turing machine which takes strings of a language L and outputs 1 if the string is in L or 0 otherwise.

Interactive Proofs and Complexity Theory: Hiding behind ∃

- Can we capture NP using an interactive proof? Yes!

- A poly-time machine asking a "machine" to provide a certificate.

Verifier

Prover

- Here, the verifier V, is a polynomial time Turing machine which takes strings of a language L and outputs 1 if the string is in L or 0 otherwise.

- The Prover P, is a <u>function</u> that maps strings to a certificate or "Sorry, not in the language".

Interactive Proofs and Complexity Theory: Trusting Strangers

Interactive Proofs and Complexity Theory: Trusting Strangers

- The verifier still has to verify the certificate!

Interactive Proofs and Complexity Theory: Trusting Strangers

- The verifier still has to verify the certificate!

- Provers are always trying to prove correctness, even if a statement is not correct.

Interactive Proofs and Complexity Theory: Trusting Strangers

- The verifier still has to verify the certificate!

- Provers are always trying to prove correctness, even if a statement is not correct.

- Even if the prover diligently says that there is no proof, the verifier cannot be sure unless the verifier knows that the prover is **all powerful**.

Interactive Proof systems: The Protocol

Definition: Let $f, g : \{0, 1\}^* \rightarrow \{0, 1\}^*$ be functions. A k-round interaction of f and g on input $x \in \{0, 1\}^*$, denoted by $\langle f, g\rangle(x)$ is the sequence of the following strings $a_1, \ldots, a_k \in \{0, 1\}^*$ defined as follows:

$$a_1 = f(x)$$

$$a_2 = g(x, a_1)$$

$$\ldots$$

$$a_{2i+1} = f(x, a_1, \ldots, a_{2i})$$

$$a_{2i+2} = g(x, a_1, \ldots, a_{2i+1})$$

The output of f at the end of the interaction, $out_f\langle f, g\rangle(x)$, is defined to be

$f(x, a_1, \ldots, a_k)$

Interactive Proof systems: The Protocol

Definition: Let f, g : $\{0, 1\}^* \to \{0, 1\}^*$ be functions. A k-round interaction of f and g on input $x \in \{0, 1\}^*$, denoted by $<f, g>(x)$ is the sequence of the following strings $a_1, \ldots, a_k \in \{0, 1\}^*$ defined as follows:

$$a_1 = f(x)$$

$$a_2 = g(x, a_1)$$

$$\ldots$$

$$a_{2i+1} = f(x, a_1, \ldots, a_{2i})$$

$$a_{2i+2} = g(x, a_1, \ldots, a_{2i+1})$$

Transcript

The output of f at the end of the interaction, $out_f<f, g>(x)$, is defined to be

$f(x, a_1, \ldots, a_k)$

Interactive Proof systems

Interactive Proof systems

Obs 1: If the prover(P) is g and the verifier(V) f

Interactive Proof systems

Obs 1: If the prover(P) is g and the verifier(V) f

- The verifier starts

Interactive Proof systems

Obs 1: If the prover(P) is g and the verifier(V) f

- The verifier starts

Obs 2: The prover <u>must</u> be all powerful

Interactive Proof systems

Obs 1: If the prover(P) is g and the verifier(V) f

- The verifier starts

Obs 2: The prover <u>must</u> be all powerful

Obs 3: The verifier should be “efficient”

Interactive Proof systems

Obs 1: If the prover(P) is g and the verifier(V) f

- The verifier starts

Obs 2: The prover <u>must</u> be all powerful

Obs 3: The verifier should be “efficient”

- Some sort of “efficient” TM

Interactive Proof systems

Obs 1: If the prover(P) is g and the verifier(V) f

- The verifier starts

Obs 2: The prover <u>must</u> be all powerful

Obs 3: The verifier should be “efficient”

- Some sort of “efficient” TM

Obs 4: The transcript must be “short”

Interactive Proof systems

Obs 1: If the prover(P) is g and the verifier(V) f

- The verifier starts

Obs 2: The prover <u>must</u> be all powerful

Obs 3: The verifier should be “efficient”

- Some sort of “efficient” TM

Obs 4: The transcript must be “short”

- From Obs 3, if the transcript is not short, the verifier cannot be efficient.

Interactive Proof systems

Obs 1: If the prover(P) is g and the verifier(V) f

- The verifier starts

Obs 2: The prover must be all powerful

Obs 3: The verifier should be "efficient"

- Some sort of "efficient" TM

Obs 4: The transcript must be "short"

- From Obs 3, if the transcript is not short, the verifier cannot be efficient.

Obs 5: Both V and P have access to the input x

Interactive Proof systems

Obs 1: If the prover(P) is g and the verifier(V) f

- The verifier starts

Obs 2: The prover <u>must</u> be all powerful

Obs 3: The verifier should be "efficient"

- Some sort of "efficient" TM

Deterministic poly-time?

Obs 4: The transcript must be "short"

- From Obs 3, if the transcript is not short, the verifier cannot be efficient.

Obs 5: Both V and P have access to the input x

Interactive Proof systems

Definition: Deterministic proof systems

For $k \geq 1$, We say that a language L has a k-round deterministic interactive proof system if there's a deterministic poly-time TM V that on input $x, a_1, \ldots, a_i$ runs in time polynomial in $|x|$, satisfying:

$$x \in L \Rightarrow \exists P : \{0,1\}^* \rightarrow \{0,1\}^* \ out_V \langle V, P\rangle(x) = 1 \text{ (Completeness)}$$

$$x \notin L \Rightarrow \forall P : \{0,1\}^* \rightarrow \{0,1\}^* \ out_V \langle V, P\rangle(x) = 0 \text{ (Soundness)}$$

Interactive Proof systems

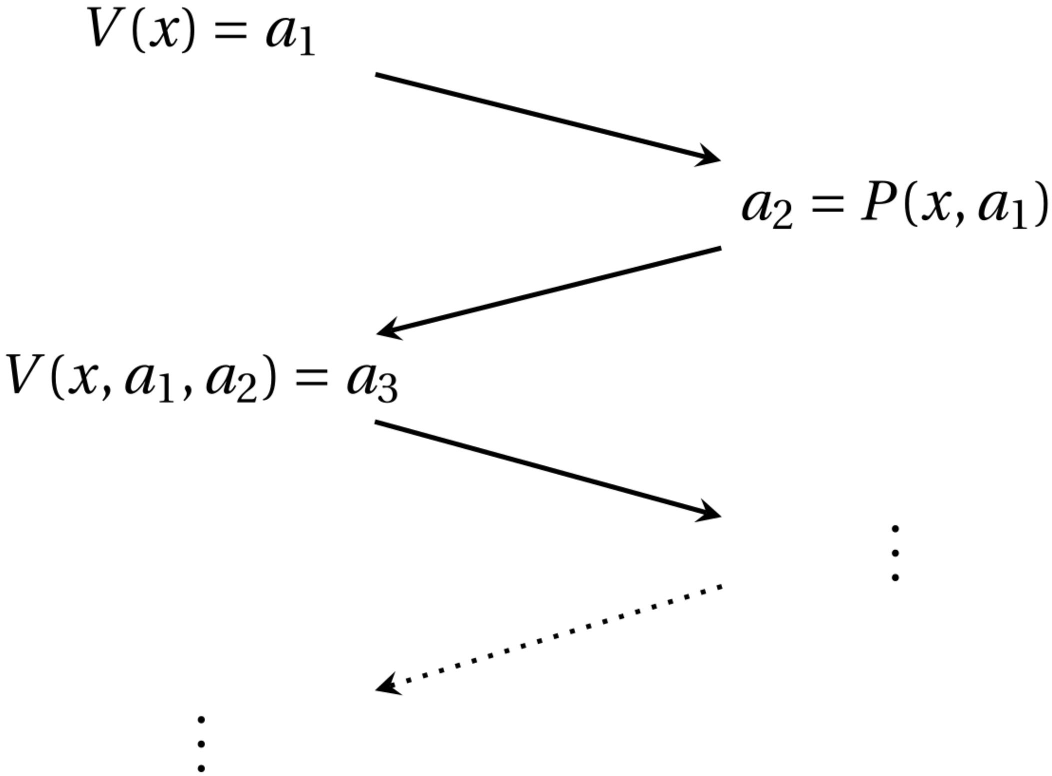


Image Credits: ProgVal, CC0, via Wikimedia Commons

Interactive Proof systems: dIP

Interactive Proof systems: dIP

- Obs: Since the verifier is poly-time, the transcript must be poly-size. Which means the number of interactions can be at most poly-size.

Interactive Proof systems: dIP

- Obs: Since the verifier is poly-time, the transcript must be poly-size. Which means the number of interactions can be at most poly-size.

- **dIP** is the set of all languages with poly(n)-round deterministic interactive proof system.

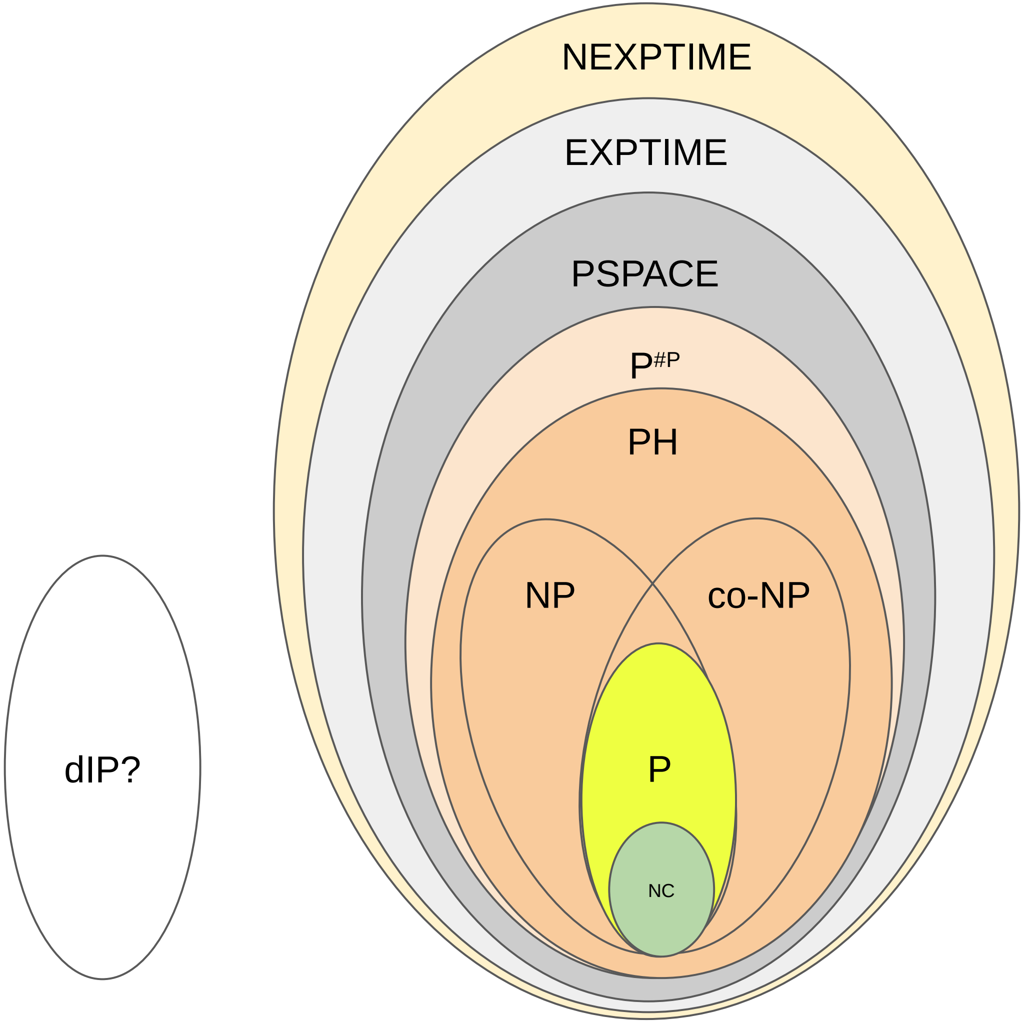
Interactive Proof systems: dIP

- Obs: Since the verifier is poly-time, the transcript must be poly-size. Which means the number of interactions can be at most poly-size.

- **dIP** is the set of all languages with poly(n)-round deterministic interactive proof system.

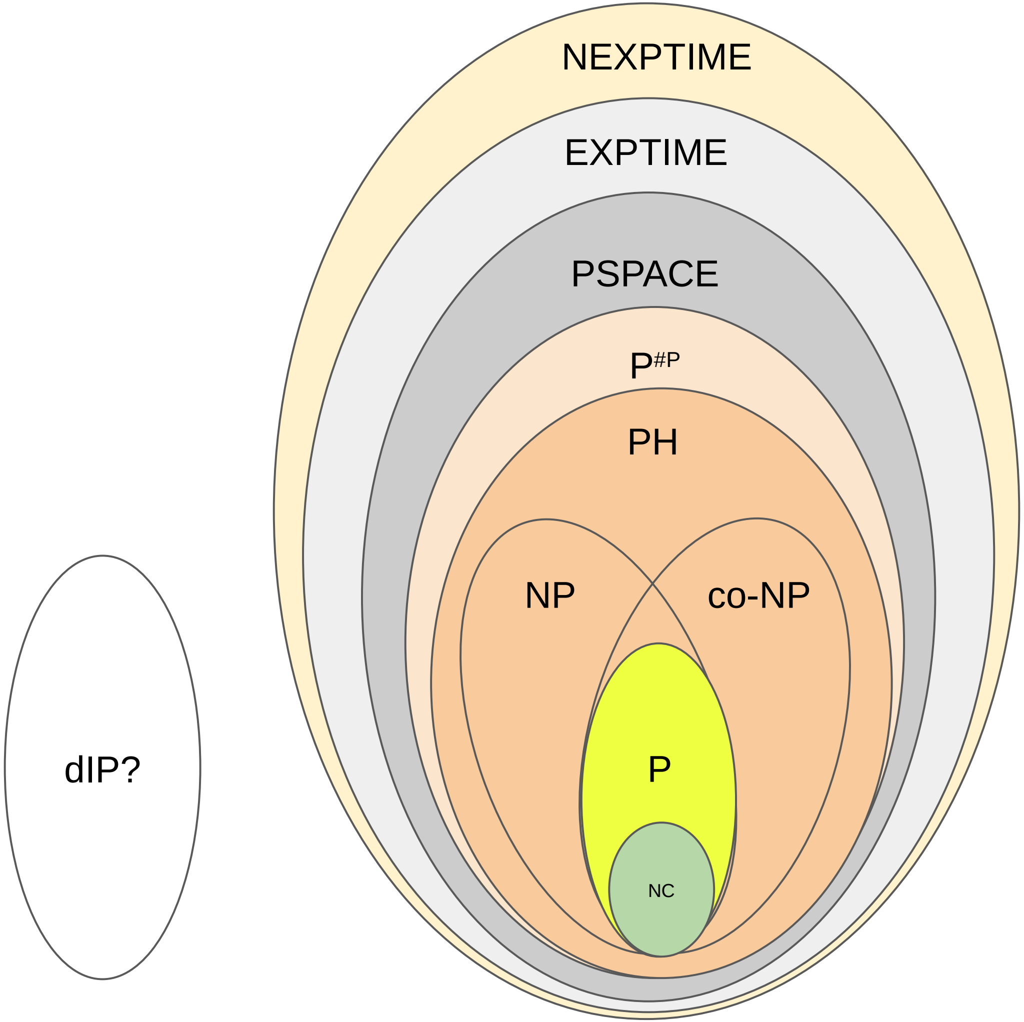
- Can't we define a class of constant round deterministic interactive proof systems?

Where is dIP?



Where is dIP?

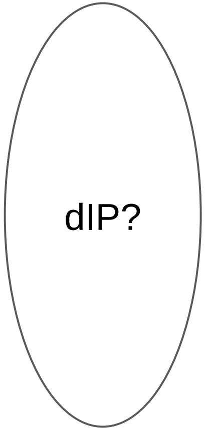
- Claim: **NP $\subseteq$ dIP**

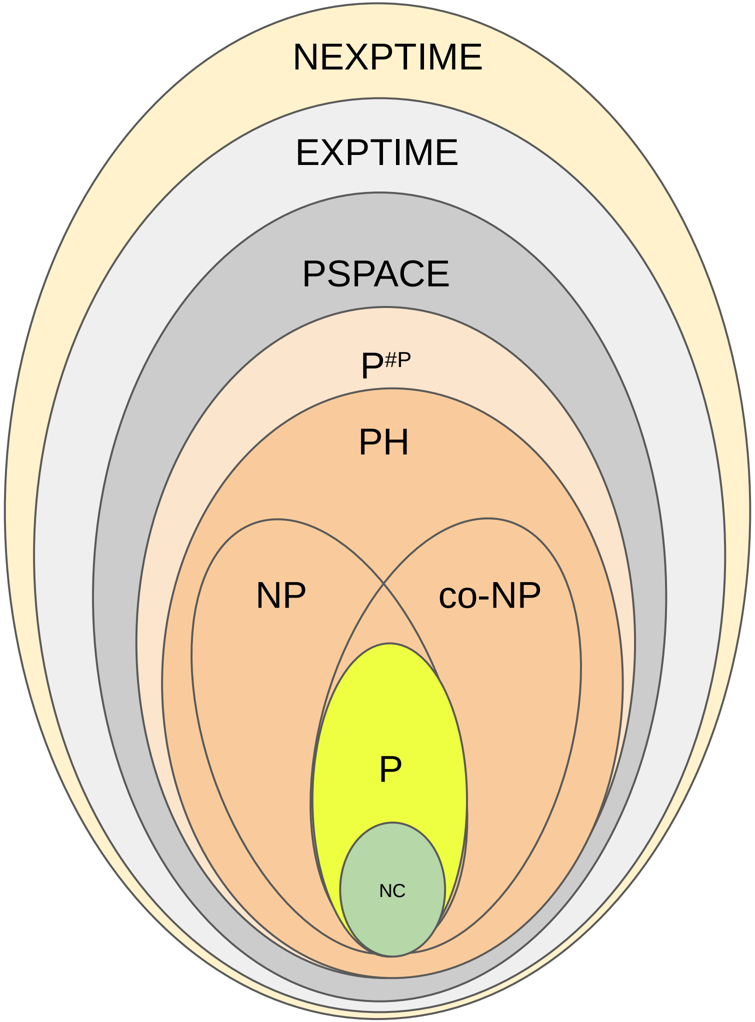


Where is dIP?

- Claim: **NP $\subseteq$ dIP**

Proof: One round protocol for 3SAT, where a prover returns a satisfying assignment for the input if it exists.





Where is dIP?

Where is dIP?

- Claim: **dIP** $\subseteq$ **NP**

Where is dIP?

- Claim: **dIP** $\subseteq$ **NP**

Proof: Consider a dIP system with P,V. Consider a poly-time verifier M, the entire transcript of a deterministic interaction is a certificate.

Where is dIP?

- Claim: **dIP $\subseteq$ NP**

Proof: Consider a dIP system with P,V. Consider a poly-time verifier M, the entire transcript of a deterministic interaction is a certificate.

M verifies that the output of each round from the verifier matches that in the transcript by simulating V.

Where is dIP?

- Claim: **dIP $\subseteq$ NP**

Proof: Consider a dIP system with P,V. Consider a poly-time verifier M, the entire transcript of a deterministic interaction is a certificate.

M verifies that the output of each round from the verifier matches that in the transcript by simulating V.

It does not need to simulate P, as if a certificate exists, the string must be in the language and a prover must exist which outputs matching values in the transcript.

Where is dIP?

- Claim: **dIP $\subseteq$ NP**

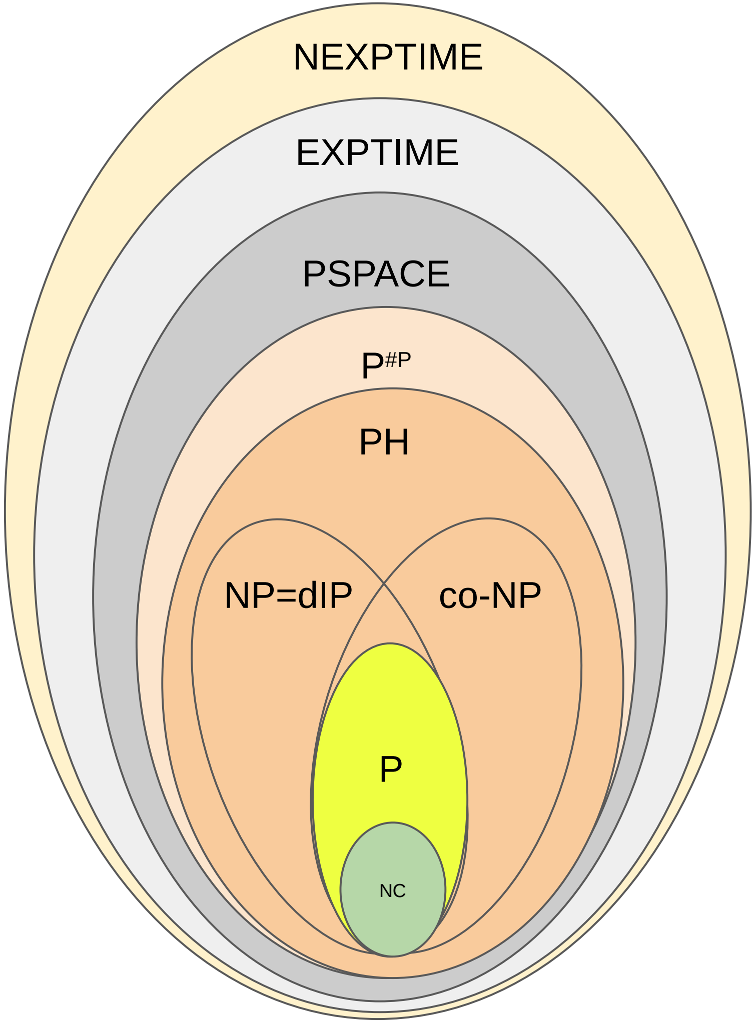
Proof: Consider a dIP system with P,V. Consider a poly-time verifier M, the entire transcript of a deterministic interaction is a certificate.

M verifies that the output of each round from the verifier matches that in the transcript by simulating V.

It does not need to simulate P, as if a certificate exists, the string must be in the language and a prover must exist which outputs matching values in the transcript.

- Lemma: **dIP = NP**

Where is dIP?



Interactive Proof systems

Obs 1: If the prover(P) is g and the verifier(V) f

- The verifier starts

Obs 2: The prover <u>must</u> be all powerful

Obs 3: The verifier should be “efficient”

- Some sort of “efficient” TM

Obs 4: The transcript must be “short”

- From Obs 3, if the transcript is not short, the verifier cannot be efficient.

Obs 5: Both V and P have access to the input x

Interactive Proof systems

Obs 1: If the prover(P) is g and the verifier(V) f

- The verifier starts

Obs 2: The prover <u>must</u> be all powerful

Obs 3: The verifier should be "efficient"

Probabilistic poly-time?

- Some sort of "efficient" TM

Obs 4: The transcript must be "short"

- From Obs 3, if the transcript is not short, the verifier cannot be efficient.

Obs 5: Both V and P have access to the input x

IP: Probabilistic Verifier

Definition [GMR '89]: Probabilistic Verifiers and **IP**

For $k \geq 1$, we say that a language L has in **IPTIME**[k] if there's a probabilistic poly-time TM V that has a k-round interaction with P: $\{0,1\}^* \to \{0,1\}^*$ that on input x

$$x \in L \Rightarrow \exists P \; \Pr_r[\mathrm{out}_V \langle V, P\rangle(x) = 1] \geq 2/3 \text{ (Completeness)}$$

$$x \notin L \Rightarrow \forall P \; \Pr_r[\mathrm{out}_V \langle V, P\rangle(x) = 1] \leq 1/3 \text{ (Soundness)}$$

The probabilities over the random bits r of V.

The class IP is defined as **IP** = $\bigcup_{c>0}$ **IPTIME**$[n^c]$

P:BPP::NP:IP

P:BPP::NP:IP

- Lemma: We can boost the completeness and soundness probability by

P:BPP::NP:IP

- Lemma: We can boost the completeness and soundness probability by $1 - 2^{-n^c}$ and 2^{-n^c} respectively for some constant c.

P:BPP::NP:IP

- Lemma: We can boost the completeness and soundness probability by

  $1 - 2^{-n^c}$ and 2^{-n^c} respectively for some constant c.

  Proof: Similar to boosting a BPP machine. Polynomially(n^c) many independent repetitions of protocol.

P:BPP::NP:IP

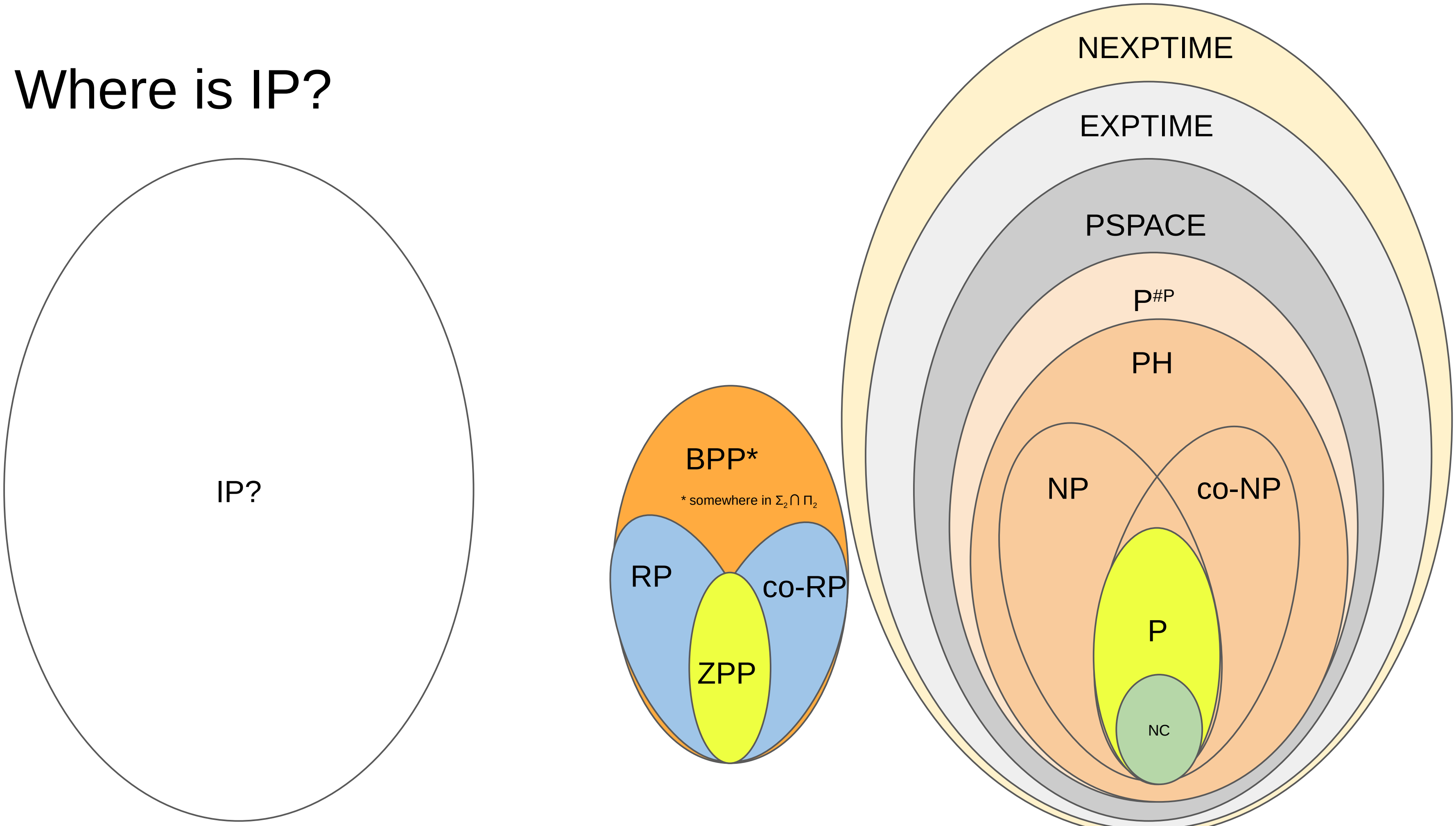
- Lemma: We can boost the completeness and soundness probability by

  $1 - 2^{-n^c}$ and 2^{-n^c} respectively for some constant c.

  Proof: Similar to boosting a BPP machine. Polynomially(n^c) many independent repetitions of protocol.

  Additionally, we can also do all repetitions in parallel by asking multiple questions in each round, thereby decreasing the number of rounds.

Where is IP?



IP?
BPP*
* somewhere in Σ2 ∩ Π2
RP
co-RP
ZPP
NEXPTIME
EXPTIME
PSPACE
P#P
PH
NP
co-NP
P
NC

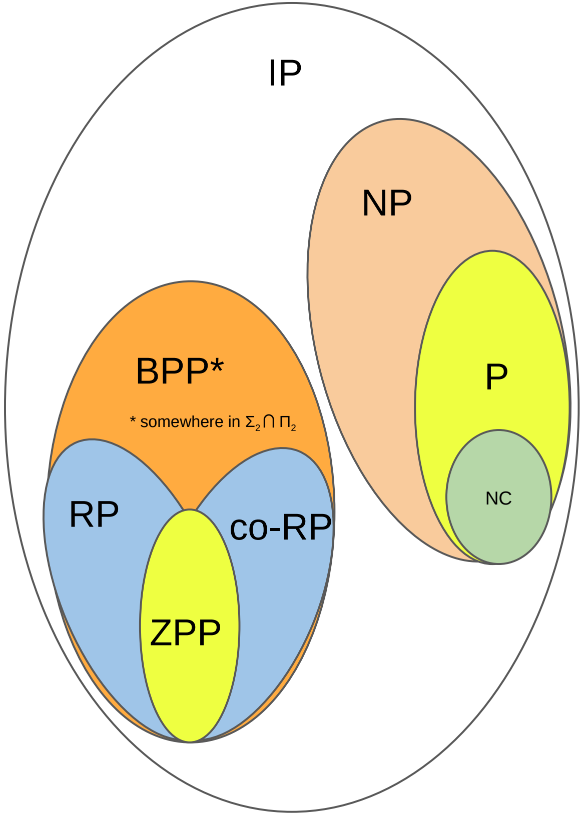
What's in IP?

- Clearly, NP is also in IP.

As dIP is in IP

- So is BPP

The verifier is a BPP machine that ignores the prover



How Big is IP?

How Big is IP?

- Graph isomorphism known to be in NP, hence in IP. Unclear whether non-isomorphism is in NP, but an interactive proof exists.

How Big is IP?

- Graph isomorphism known to be in NP, hence in IP. Unclear whether non-isomorphism is in NP, but an interactive proof exists.

- Graph non-isomorphism is defined as the following language

How Big is IP?

- Graph isomorphism known to be in NP, hence in IP. Unclear whether non-isomorphism is in NP, but an interactive proof exists.

- Graph non-isomorphism is defined as the following language

$$\text{NONISO} = \{(G_1,G_2) \mid G_1 \text{ is not isomorphic to } G_2\}$$

How Big is IP?

- Graph isomorphism known to be in NP, hence in IP. Unclear whether non-isomorphism is in NP, but an interactive proof exists.

- Graph non-isomorphism is defined as the following language

$$\text{NONISO} = \{(G_1,G_2) \mid G_1 \text{ is not isomorphic to } G_2\}$$

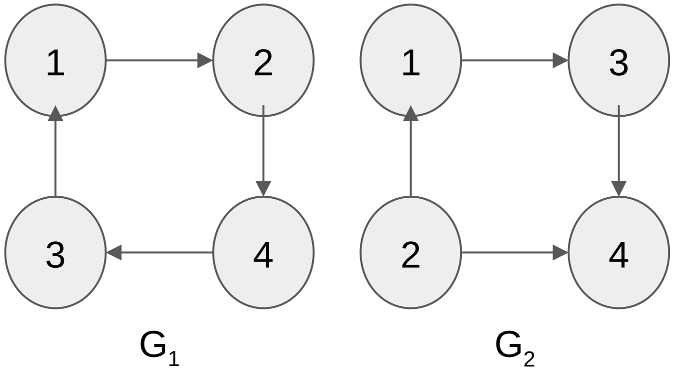
- Lemma: NONISO $\in$ IP [GMW '91]

NONISO in IP: Private Coin Protocol

Private Coin Protocol

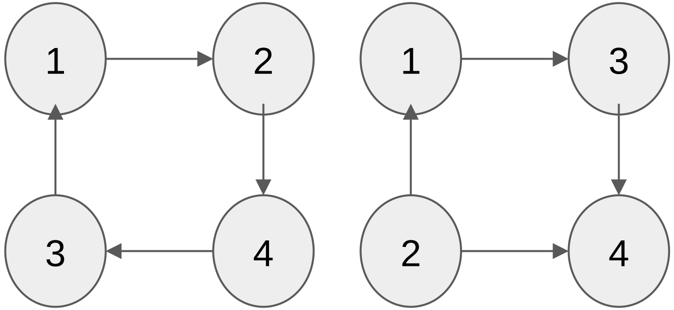
1. V randomly picks a graph between G_1 and G_2, say G_i. Randomly permute vertices of G_i to make H. Send H to P and asks if H is isomorphic to G_1 or G_2
2. Prover tries to figure out whether H is isomorphic to G_1 or G_2, sends $j \in \{1,2\}$ to V
3. V accepts if j==i.

NONISO in IP: Private Coin Protocol



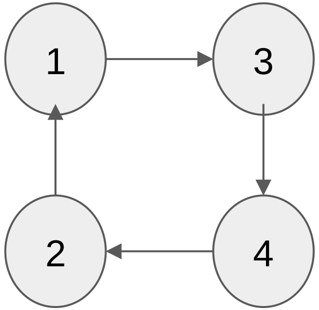
NONISO in IP: Private Coin Protocol

V



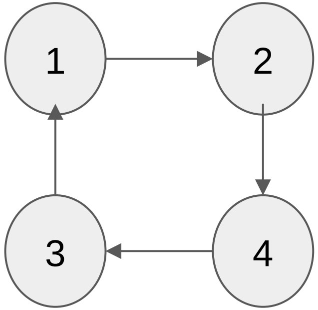
G_1 G_2

NONISO in IP: Private Coin Protocol

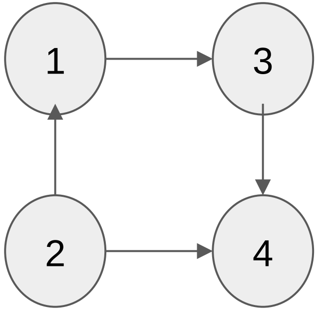


H

V

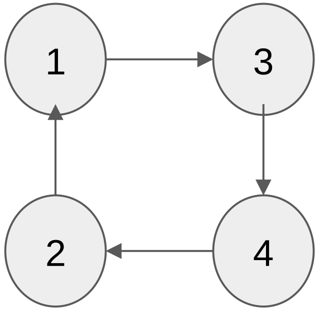


G_1

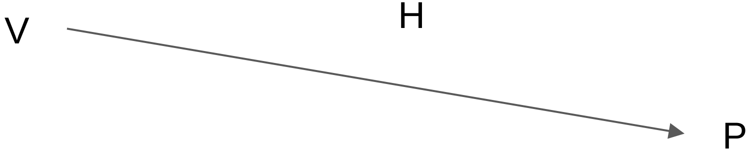


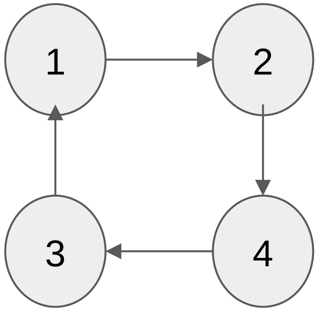
G_2

NONISO in IP: Private Coin Protocol

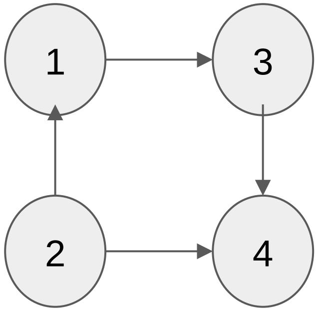


H



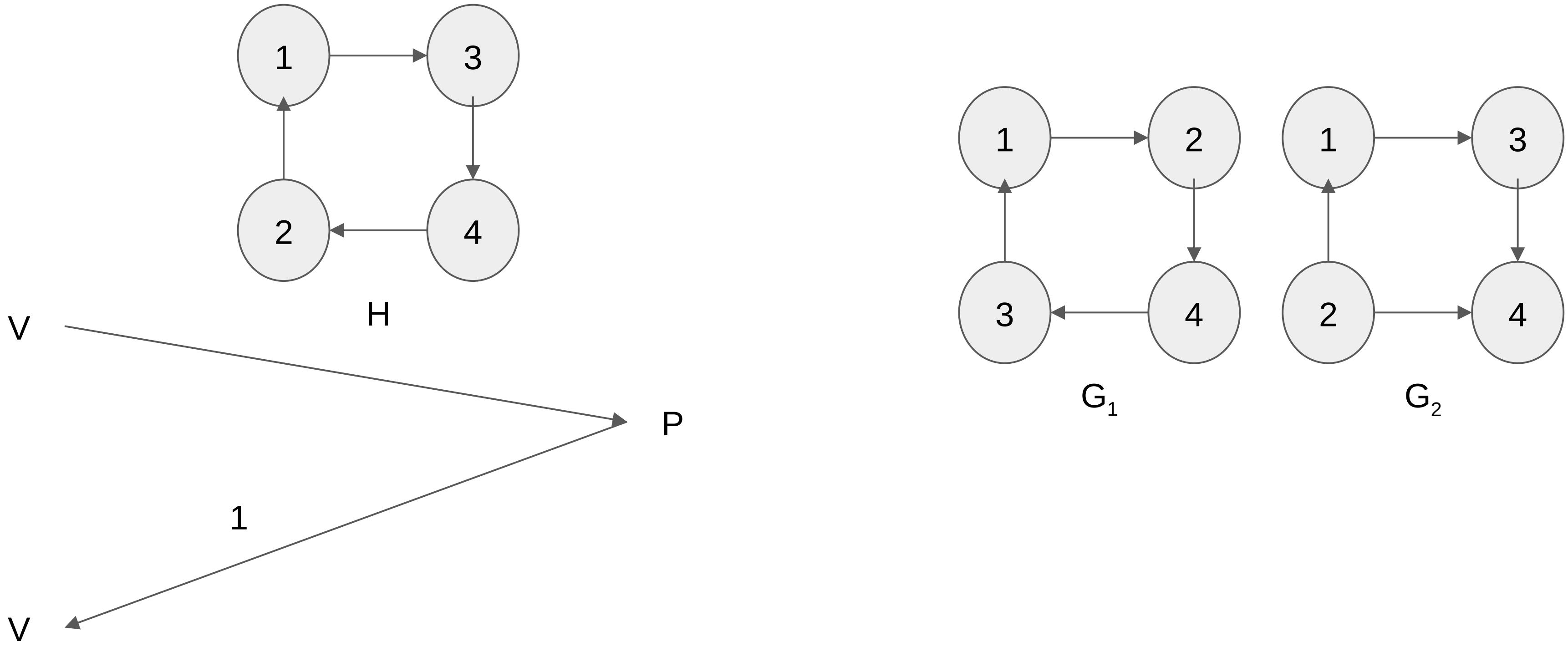


G_1

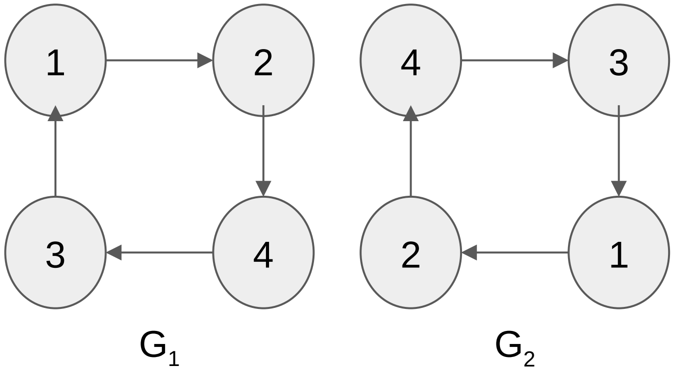


G_2

NONISO in IP: Private Coin Protocol

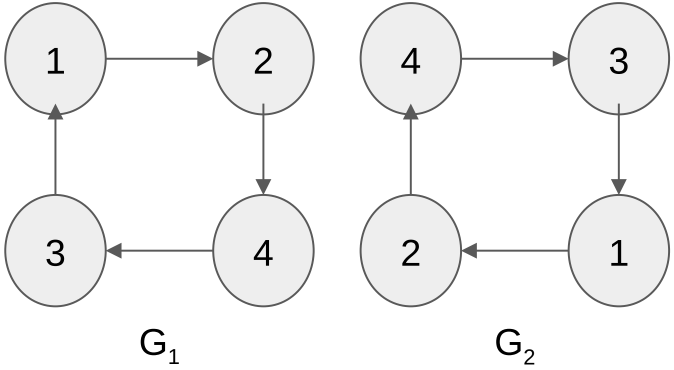


NONISO in IP: Private Coin Protocol



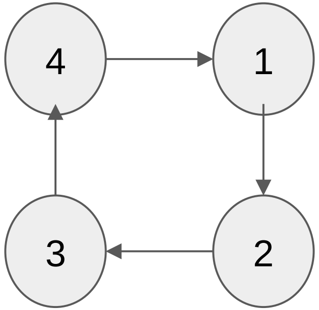
NONISO in IP: Private Coin Protocol

V

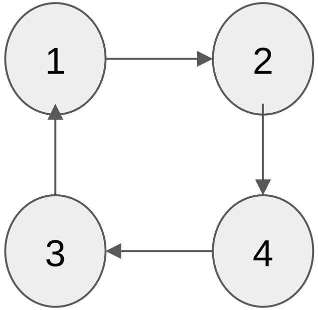


NONISO in IP: Private Coin Protocol

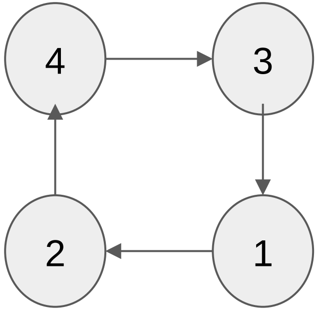
V



H

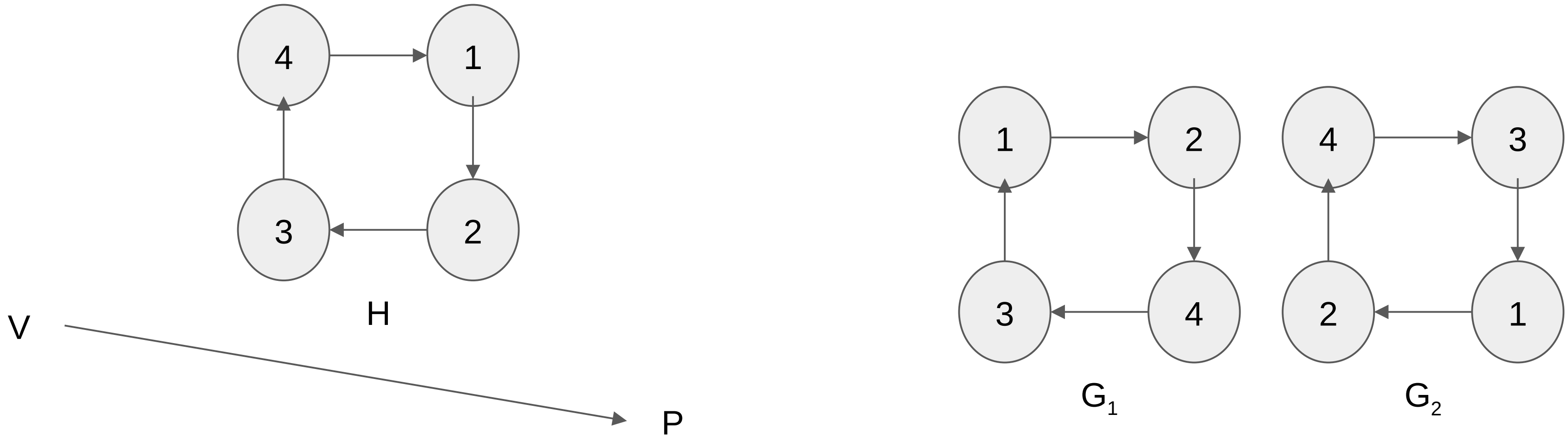


G_1

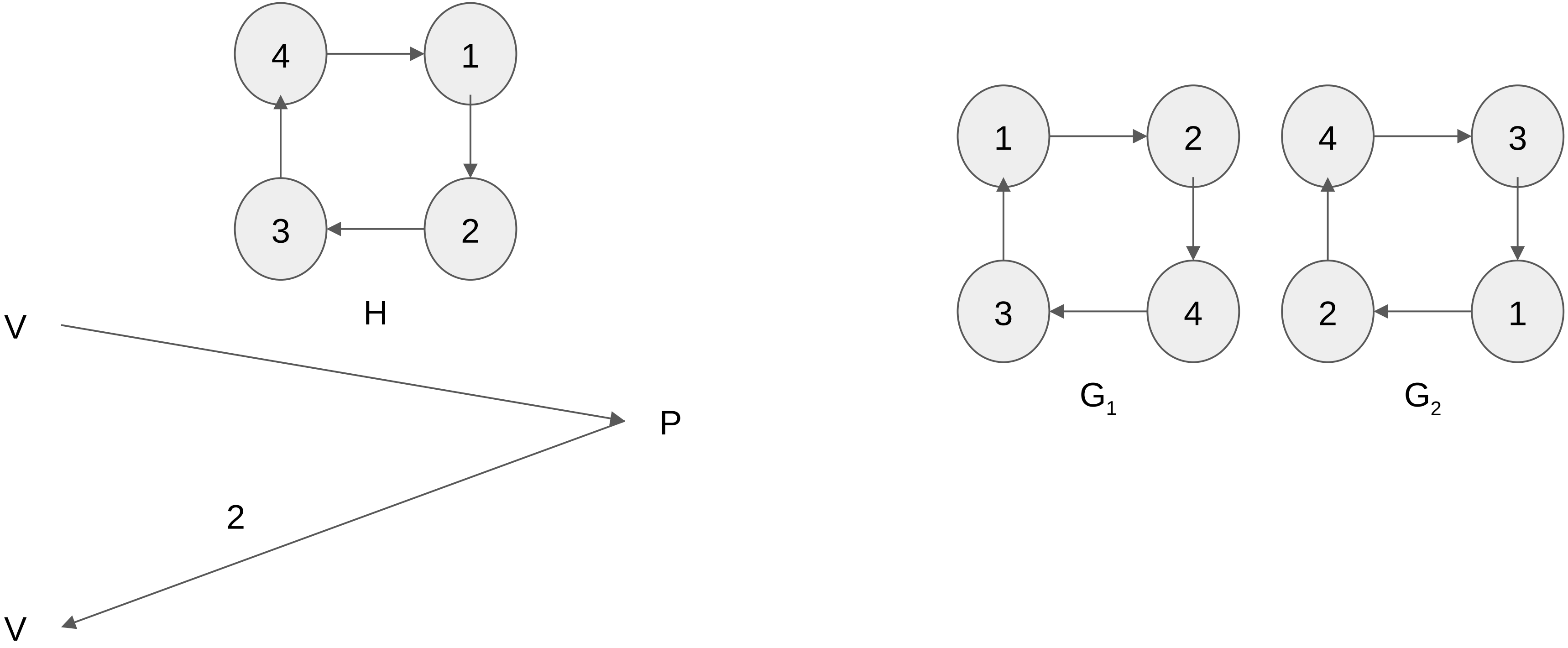


G_2

NONISO in IP: Private Coin Protocol



NONISO in IP: Private Coin Protocol



NONISO in IP: Private Coin Protocol

NONISO in IP: Private Coin Protocol

- If G_1 and G_2 are not isomorphic, then the prover should be able to compare every permutation of H with G_1 and G_2 to be able to answer correctly.

NONISO in IP: Private Coin Protocol

- If G_1 and G_2 are not isomorphic, then the prover should be able to compare every permutation of H with G_1 and G_2 to be able to answer correctly.

- The probability of acceptance when the string is in the language is 1. (Perfect Completeness)

NONISO in IP: Private Coin Protocol

- If G_1 and G_2 are not isomorphic, then the prover should be able to compare every permutation of H with G_1 and G_2 to be able to answer correctly.

- The probability of acceptance when the string is in the language is 1. (Perfect Completeness)

- If they are not isomorphic, the best the prover can do is to guess at random. So the probability of acceptance when it isn't in the language is ½. We can decrease this be multiple repetitions.

What's in IP?

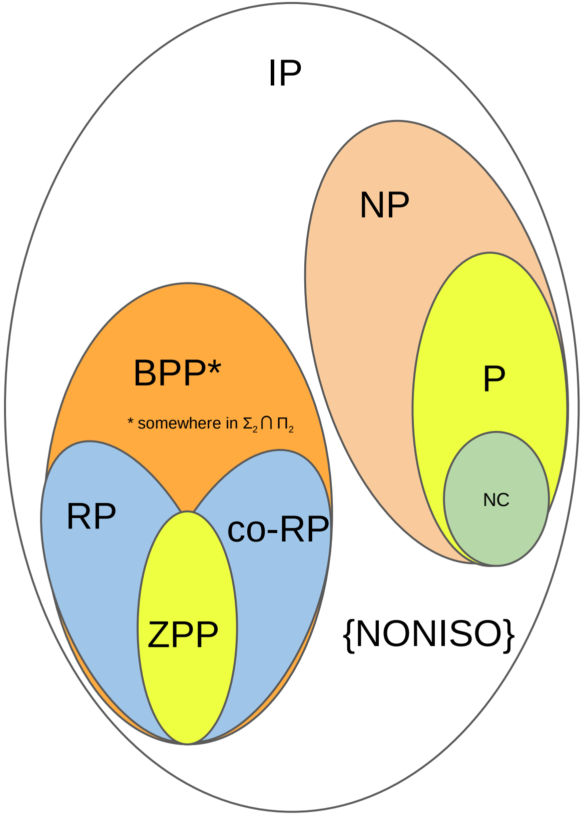
- Clearly, NP is also in IP.

As dIP is in IP

- So is BPP

The verifier is a BPP machine that ignores the prover

- NONISO in IP



Interactive Proof systems

Obs 1: If the prover(P) is g and the verifier(V) f

Obs 2: The prover <u>must</u> be all powerful

Obs 3: The verifier should be “efficient”

Probabilistic poly-time.

Obs 4: The transcript must be “short”

Obs 5: Both V and P have access to the input x

Interactive Proof systems

Obs 1: If the prover(P) is g and the verifier(V) f

Who starts?

Obs 2: The prover <u>must</u> be all powerful

Obs 3: The verifier should be “efficient”

Probabilistic poly-time.

Obs 4: The transcript must be “short”

Obs 5: Both V and P have access to the input x

Interactive Proof systems

Obs 1: If the prover(P) is g and the verifier(V) f

Who starts?

Probabilistic?

Obs 2: The prover must be all powerful

Obs 3: The verifier should be "efficient"

Probabilistic poly-time.

Obs 4: The transcript must be "short"

Obs 5: Both V and P have access to the input x

Interactive Proof systems

Obs 1: If the prover(P) is g and the verifier(V) f

Who starts?

Probabilistic?

Obs 2: The prover <u>must</u> be all powerful

Obs 3: The verifier should be "efficient"

Probabilistic poly-time.

Obs 4: The transcript must be "short"

Obs 5: Both V and P have access to the input x

What about random bits of V?

AM and MA: Public and Private coins

AM and MA: Public and Private coins

Constant round interactive proofs with public coins: **AM** and **MA.**

AM and MA: Public and Private coins

Verifier starts

Constant round interactive proofs with public coins: **AM** and **MA.**

Prover starts

AM and MA: Public and Private coins

Verifier starts

Constant round interactive proofs with public coins: **AM** and **MA**.

Prover starts

We have already seen the set lower bound protocol, which was used to show that graph non-isomorphism is in BP.NP

AM and MA: Public and Private coins

Verifier starts

Constant round interactive proofs with public coins: **AM** and **MA.**

Prover starts

We have already seen the set lower bound protocol, which was used to show that graph non-isomorphism is in BP.NP

- Theorem: **BP.NP** = **AM** $\subseteq \boldsymbol{\Sigma_3}$

AM and MA: Public and Private coins

Verifier starts

Constant round interactive proofs with public coins: **AM** and **MA**.

Prover starts

We have already seen the set lower bound protocol, which was used to show that graph non-isomorphism is in BP.NP

- Theorem: **BP.NP** = **AM** $\subseteq \boldsymbol{\Sigma_3}$
- Theorem [Babai ‘88]: **AM[k]** = **AM[2]** for constant k

AM and MA: Public and Private coins

Verifier starts

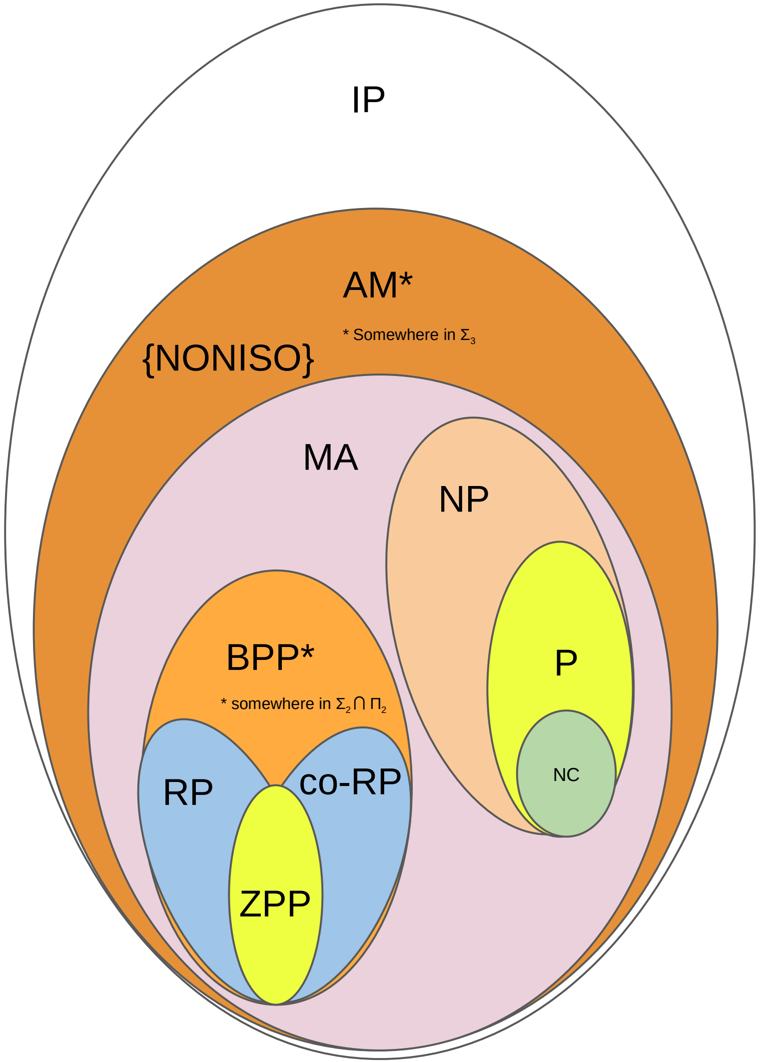
Constant round interactive proofs with public coins: **AM** and **MA.**

Prover starts

We have already seen the set lower bound protocol, which was used to show that graph non-isomorphism is in BP.NP

- Theorem: **BP.NP** = **AM** $\subseteq \boldsymbol{\Sigma_3}$
- Theorem [Babai '88]: **AM[k]** = **AM[2]** for constant k
- Theorem [GS '86]: **AM[k]** $\subseteq$ **IP[k]** $\subseteq$ **AM[k+2]** for polynomial k.

What’s in IP?


IP
AM*
* Somewhere in Σ_3
{NONISO}
MA
NP
BPP*
* somewhere in $\Sigma_2 \cap \Pi_2$
P
NC
RP
co-RP
ZPP

Theorem: IP $\subseteq$ PSPACE

Theorem: IP $\subseteq$ PSPACE

Proof Idea: Since we restrict certificates to be poly-size, it's easy to see that one can use a PSPACE machine to run through all possible transcripts to simulate a prover and calculate exactly the acceptance probability.

Theorem: IP $\subseteq$ PSPACE

Proof Idea: Since we restrict certificates to be poly-size, it's easy to see that one can use a PSPACE machine to run through all possible transcripts to simulate a prover and calculate exactly the acceptance probability.

Proof: Consider a language A in IP with a verifier V. Let the transcript be exactly of size $p = \text{poly}(n)$ for all inputs x of size n. We will construct a PSPACE machine M which decides A.

Theorem: IP $\subseteq$ PSPACE

Theorem: IP $\subseteq$ PSPACE

Definition: For any string x, we define

$$\Pr[\text{ V accepts x }] = \max_P \Pr[\text{ <V,P> accepts x }]$$

If x is in A, then it is at least ⅔ and at most ⅓ if it is not.

Theorem: IP $\subseteq$ PSPACE

Definition: For any string x, we define

$$\Pr[\text{ V accepts x }] = \max_P \Pr[\text{ <V,P> accepts x }]$$

If x is in A, then it is at least ⅔ and at most ⅓ if it is not.

Definition: $M_j = m_1,\ldots,m_j$ is the partial transcript upto length j. m_i represents the i^{th} message.

Theorem: IP $\subseteq$ PSPACE

Definition: For any string x, we define

$$\Pr[\text{ V accepts x }] = \max_P \Pr[\text{ <V,P> accepts x }]$$

If x is in A, then it is at least ⅔ and at most ⅓ if it is not.

Definition: $M_j = m_1,\ldots,m_j$ is the partial transcript upto length j. m_i represents the i^{th} message.

Definition: $\langle V,P\rangle(x,r,M_j)$ = accept, for a random string r of length p, if there exists $m_{j+1},..m_p$ such that

1. For $j \le i < p$ and i is even $V(x,r,M_i) = m_{i+1}$
2. For $j \le i < p$ and i is odd $P(x,M_i) = m_{i+1}$
3. m_p is accept

Theorem: IP $\subseteq$ PSPACE

Theorem: IP $\subseteq$ PSPACE

Obs: Using previous definitions,

$\Pr[\ \langle V,P\rangle$ accepts x starting at $M_j\ ] = \Pr[\ \langle V,P\rangle(x,r,M_j) = \text{accept}\ ]$ (1)

$\Pr[\ V$ accepts x starting at $M_j\ ] = \max_P \Pr[\ \langle V,P\rangle$ accepts x starting at $M_j\ ]$ (2)

Theorem: IP $\subseteq$ PSPACE

Obs: Using previous definitions,

Pr[<V,P> accepts x starting at M_j] = Pr[<V,P>(x,r,M_j) = accept] (1)

Pr[V accepts x starting at M_j] = $\max_P$ Pr[<V,P> accepts x starting at M_j] (2)

The goal is now to compute the probability of V accepting x starting from M_0. If this is greater than ⅔ then x must be in A, if it less than ⅓ then it must not be in A. We do this recursively.

Theorem: IP $\subseteq$ PSPACE

$N_{M\_j} = 0$ if $j = p$ and m_p = reject

$= 1$ if $j = p$ and m_p = accept

$= \max_{m\_\{j+1\}} N_{M\_\{j+1\}}$ odd $j < p$

$=$ wt-avg$_{m_\{j+1\}}$ $N_{M\_\{j+1\}}$ even $j<p$

wt-avg$_{m_\{j+1\}}$ $N_{M\_\{j+1\}} = \sum_{m\_\{j+1\}} ((\Pr[V(w,r,M_j)=m_{j+1}]) \cdot N_{M\_\{j+1\}})$

Theorem: IP $\subseteq$ PSPACE

Theorem: IP $\subseteq$ PSPACE

Claim 1: N_{M_j} = Pr[V accepts x starting at M_j]

Theorem: IP $\subseteq$ PSPACE

Claim 1: N_{M_j} = Pr[V accepts x starting at M_j]

Claim 2: N_{M_j} can be calculated in PSPACE

Theorem: IP $\subseteq$ PSPACE

Claim 1: N_{M_j} = Pr[V accepts x starting at M_j]

Claim 2: N_{M_j} can be calculated in PSPACE

We need to prove the following 2 claims, with that the proof is complete.

Theorem: IP $\subseteq$ PSPACE

Theorem: IP $\subseteq$ PSPACE

Claim 1: N_{M_j} = Pr[V accepts x starting at M_j]

Theorem: IP $\subseteq$ PSPACE

Claim 1: N_{M_j} = Pr[V accepts x starting at M_j]

Proof: We prove by top down induction.

Theorem: IP $\subseteq$ PSPACE

Claim 1: N_{M_j} = Pr[V accepts x starting at M_j]

Proof: We prove by top down induction.

Base case: j = p. The last message must be *accept* or *reject.* Hence, the probability of acceptance when the last message is *reject* is 0 and when the last message is *accept*, it is 1. This is exactly how N_{M_j} is defined.

Theorem: IP $\subseteq$ PSPACE

Claim 1: N_{M_j} = Pr[V accepts x starting at M_j]

Proof: We prove by top down induction.

Base case: j = p. The last message must be *accept* or *reject.* Hence, the probability of acceptance when the last message is *reject* is 0 and when the last message is *accept*, it is 1. This is exactly how N_{M_j} is defined.

Inductive step: Assume the claim to be true for some $j+1 \leq p$. We have 2 cases, one when j is even and when j is odd.

Theorem: IP $\subseteq$ PSPACE

Claim 1: N_{M_j} = Pr[V accepts x starting at M_j]

Proof: We prove by top down induction.

Base case: j = p. The last message must be *accept* or *reject.* Hence, the probability of acceptance when the last message is *reject* is 0 and when the last message is *accept*, it is 1. This is exactly how N_{M_j} is defined.

Inductive step: Assume the claim to be true for some j+1 ≤ p. We have 2 cases, one when j is even and when j is odd.

IH: $N_{M\_\{j+1\}}$ = Pr[V accepts x starting at M_{j+1}]

Theorem: IP $\subseteq$ PSPACE

Theorem: IP $\subseteq$ PSPACE

Claim 1: N_{M_j} = Pr[V accepts x starting at M_j]

Theorem: IP $\subseteq$ PSPACE

Claim 1: N_{M_j} = Pr[V accepts x starting at M_j]

When j is even, the message m_{j+1} is from V to P. From the definition of N_{M_j}

Theorem: IP $\subseteq$ PSPACE

Claim 1: N_{M_j} = Pr[V accepts x starting at M_j]

When j is even, the message m_{j+1} is from V to P. From the definition of N_{M_j}

$N_{M\_j} = \sum_{m\_\{j+1\}} ((Pr[V(w,r,M_j) = m_{j+1}]) \cdot N_{M\_\{j+1\}})$

Theorem: IP $\subseteq$ PSPACE

Claim 1: N_{M_j} = Pr[V accepts x starting at M_j]

When j is even, the message m_{j+1} is from V to P. From the definition of N_{M_j}

$N_{M\_j} = \sum_{m\_\{j+1\}} ((Pr[V(w,r,M_j) = m_{j+1}]) \cdot N_{M\_\{j+1\}})$

From the Induction hypothesis, we can conclude

Theorem: IP $\subseteq$ PSPACE

Claim 1: N_{M_j} = Pr[V accepts x starting at M_j]

When j is even, the message m_{j+1} is from V to P. From the definition of N_{M_j}

$N_{M\_j} = \sum_{m\_\{j+1\}} ((Pr[V(w,r,M_j) = m_{j+1}]) \quad \bullet \quad N_{M\_\{j+1\}})$

From the Induction hypothesis, we can conclude

$N_{M\_j} = \sum_{m\_\{j+1\}} ((Pr[V(w,r,M_j) = m_{j+1}]) \quad \bullet \quad Pr[\text{ V accepts x starting at } M_{j+1}])$

Theorem: IP $\subseteq$ PSPACE

Claim 1: N_{M_j} = Pr[V accepts x starting at M_j]

When j is even, the message m_{j+1} is from V to P. From the definition of N_{M_j}

$N_{M\_j} = \sum_{m\_\{j+1\}} ((Pr[V(w,r,M_j) = m_{j+1}]) \bullet N_{M\_\{j+1\}})$

From the Induction hypothesis, we can conclude

$N_{M\_j} = \sum_{m\_\{j+1\}} ((Pr[V(w,r,M_j) = m_{j+1}]) \bullet$ Pr[V accepts x starting at M_{j+1}])

This is the total probability partitioned over all possible messages m_{j+1}. Hence,

Theorem: IP $\subseteq$ PSPACE

Claim 1: N_{M_j} = Pr[V accepts x starting at M_j]

When j is even, the message m_{j+1} is from V to P. From the definition of N_{M_j}

$N_{M\_j} = \sum_{m\_\{j+1\}} ((Pr[V(w,r,M_j) = m_{j+1}]) \bullet N_{M\_\{j+1\}})$

From the Induction hypothesis, we can conclude

$N_{M\_j} = \sum_{m\_\{j+1\}} ((Pr[V(w,r,M_j) = m_{j+1}]) \bullet$ Pr[V accepts x starting at M_{j+1}])

This is the total probability partitioned over all possible messages m_{j+1}. Hence,

N_{M_j} = Pr[V accepts x starting at M_j]

Theorem: IP $\subseteq$ PSPACE

When j is odd, the message m_{j+1} is from P to V. From the definition of N_{M_j}

$N_{M\_j} = \max_{m\_\{j+1\}} N_{M\_\{j+1\}}$

$N_{M\_j} = \max_{m\_\{j+1\}}$ Pr[V accepts x starting at M_{j+1}]

$= \max_{m\_\{j+1\}} \max_{P'}$ Pr[<V,P'>(x,r,M_{j+1}) = accept]

1… $\leq \max_P$ Pr[<V,P> accepts x starting at M_j], P can send the maximizing m^*_{j+1}

2… $\geq \max_P$ Pr[<V,P> accepts x starting at M_j], P cannot be better than P'

Therefore,

N_{M_j} = Pr[V accepts x starting at M_j]

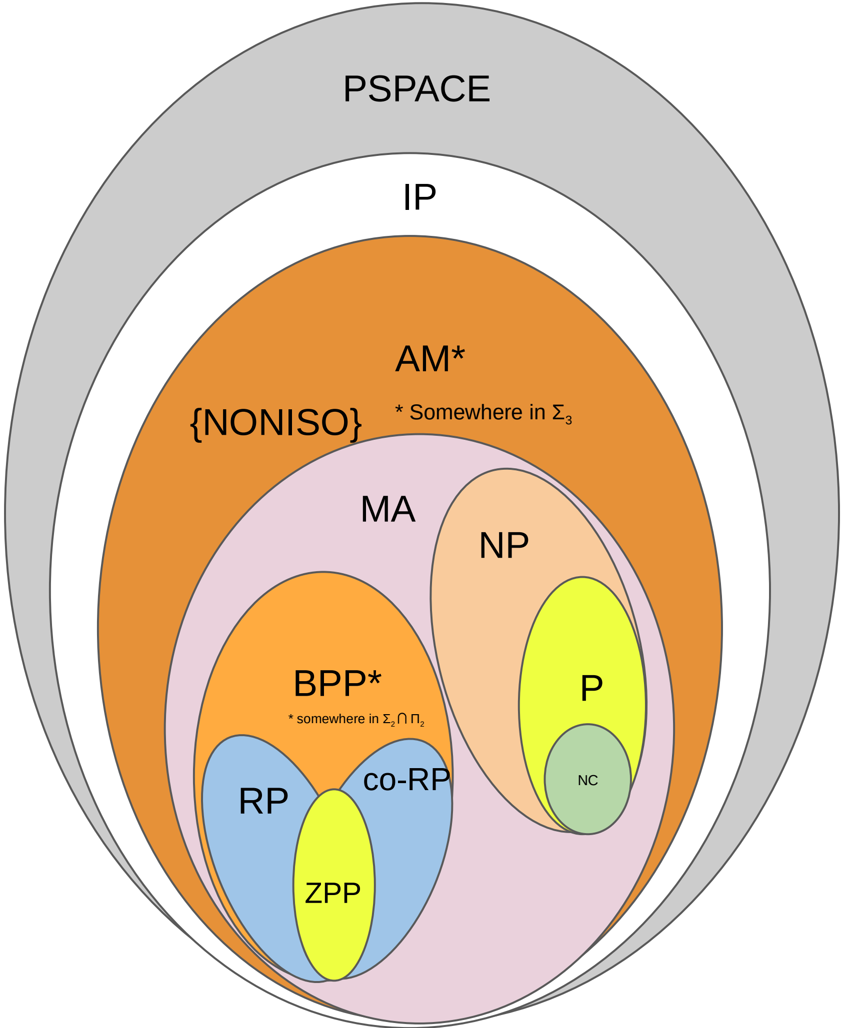
Theorem: IP $\subseteq$ PSPACE

Claim 2: N_{M_j} can be calculated in PSPACE

From the above proof, it also clear that these values can be calculated in PSPACE recursively. The depth of the recursion would be p. M calculates N_{M_j} for every j and M_j.

$\square$

Where is IP?



PSPACE
IP
AM*
* Somewhere in Σ3
{NONISO}
MA
NP
BPP*
* somewhere in Σ2 ∩ Π2
P
NC
RP
co-RP
ZPP

co-NP $\subseteq$ IP?

co-NP $\subseteq$ IP?

- One way to show PH $\subseteq$ IP, is to show a PH-complete problem is in IP

co-NP $\subseteq$ IP?

- One way to show PH $\subseteq$ IP, is to show a PH-complete problem is in IP

Doesn't exit(as far as we know)

co-NP $\subseteq$ IP?

- One way to show PH $\subseteq$ IP, is to show a PH-complete problem is in IP

Doesn't exit(as far as we know)

- Or, show every Σ_i-SAT is in IP

co-NP $\subseteq$ IP?

- One way to show PH $\subseteq$ IP, is to show a PH-complete problem is in IP

Doesn't exit(as far as we know)

- Or, show every Σ_i-SAT is in IP

- We can prove $P^{\#P} \subseteq IP$ if #3SAT is in IP, would automatically imply PH is in IP by Toda's theorem

co-NP $\subseteq$ IP?

- One way to show PH $\subseteq$ IP, is to show a PH-complete problem is in IP

Doesn't exit(as far as we know)

- Or, show every Σ_i-SAT is in IP

- We can prove $P^{\#P} \subseteq$ IP if #3SAT is in IP, would automatically imply PH is in IP by Toda's theorem

- Proven by [LFKN '92]

#3SAT Prerequisites

- Definition: #3SAT

#3SAT = {(ϕ,k)| where ϕ is a 3CNF with exactly k satisfying assignments}

#ϕ is the number of satisfying assignments of 3CNF ϕ

Say $\phi(x_1,...x_n)$, then

$$\#\phi = \Sigma_{b\_1 \in \{0,1\}}\Sigma_{b\_2 \in \{0,1\}}...\Sigma_{b\_n \in \{0,1\}}\ \phi(b_1,...b_n)$$

$\phi(b_1,...b_n) = 1$ if $b_1 \ldots b_n$ is a satisfying assignment, 0 otherwise

We define $\#\phi(a_1,...a_{i-1})$ as

$\#\phi(a_1,...a_{i-1}) = \Sigma_{b\_i \in \{0,1\}}...\Sigma_{b\_n \in \{0,1\}}\phi(a_1,...,a_{i-1},b_i,...,b_n)$

#3SAT Prerequisites

Observation*: $\#\phi(a_1,...a_{i-1}) = \#\phi(a_1,...a_{i-1},0) + \#\phi(a_1,...a_{i-1},1)$

$\#\phi(a_1,...a_{i-1}) = \Sigma_{b\_i \in \{0,1\}}...\Sigma_{b\_n \in \{0,1\}}\phi(a_1,...,a_{i-1},b_i,...,b_n)$

$= \Sigma_{b\_\{i+1\} \in \{0,1\}}...\Sigma_{b\_n \in \{0,1\}}\phi(a_1,...,a_{i-1},0,...,b_n) + \Sigma_{b\_\{i+1\} \in \{0,1\}}...\Sigma_{b\_n \in \{0,1\}}\phi(a_1,...,a_{i-1},1,...,b_n)$

$= \#\phi(a_1,...a_{i-1},0) + \#\phi(a_1,...a_{i-1},1)$

#3SAT $\subseteq$ IP? [Attempt 1]

#3SAT $\subseteq$ IP? [Attempt 1]

Say the input is (ϕ,K). The verifier has to check whether ϕ indeed has K satisfying assignments. Try to verify observation*

#3SAT $\subseteq$ IP? [Attempt 1]

Say the input is (ϕ,K). The verifier has to check whether ϕ indeed has K satisfying assignments. Try to verify observation*

1. Step 0: Verifier sends ϕ to the Prover and asks for number of satisfying assignments to ϕ

#3SAT $\subseteq$ IP? [Attempt 1]

Say the input is (ϕ,K). The verifier has to check whether ϕ indeed has K satisfying assignments. Try to verify observation*

1. Step 0: Verifier sends ϕ to the Prover and asks for number of satisfying assignments to ϕ
2. Step 1: Prover sends K

#3SAT $\subseteq$ IP? [Attempt 1]

Say the input is (ϕ,K). The verifier has to check whether ϕ indeed has K satisfying assignments. Try to verify observation*

1. Step 0: Verifier sends ϕ to the Prover and asks for number of satisfying assignments to ϕ
2. Step 1: Prover sends K
3. Step 2: Verifier sets x_1 to 0 in ϕ (ϕ_1) and x_1 to 1(ϕ_2) and evaluates ϕ_1 and ϕ_2 and asks the verifier for $\#\phi_1$ and $\#\phi_2$

#3SAT $\subseteq$ IP? [Attempt 1]

Say the input is (ϕ, K). The verifier has to check whether ϕ indeed has K satisfying assignments. Try to verify observation*

1. Step 0: Verifier sends ϕ to the Prover and asks for number of satisfying assignments to ϕ
2. Step 1: Prover sends K
3. Step 2: Verifier sets x_1 to 0 in ϕ (ϕ_1) and x_1 to 1(ϕ_2) and evaluates ϕ_1 and ϕ_2 and asks the verifier for $\#\phi_1$ and $\#\phi_2$
4. Step 3: Prover sends k_1 and k_2

#3SAT $\subseteq$ IP? [Attempt 1]

Say the input is (ϕ,K). The verifier has to check whether ϕ indeed has K satisfying assignments. Try to verify observation*

1. Step 0: Verifier sends ϕ to the Prover and asks for number of satisfying assignments to ϕ
2. Step 1: Prover sends K
3. Step 2: Verifier sets x_1 to 0 in ϕ (ϕ_1) and x_1 to 1(ϕ_2) and evaluates ϕ_1 and ϕ_2 and asks the verifier for $\#\phi_1$ and $\#\phi_2$
4. Step 3: Prover sends k_1 and k_2
5. Step 4: Verifier verifies that $K = k_1 + k_2$

#3SAT $\subseteq$ IP? [Attempt 1]

Say the input is (ϕ,K). The verifier has to check whether ϕ indeed has K satisfying assignments. Try to verify observation*

1. Step 0: Verifier sends ϕ to the Prover and asks for number of satisfying assignments to ϕ
2. Step 1: Prover sends K
3. Step 2: Verifier sets x_1 to 0 in ϕ (ϕ_1) and x_1 to 1(ϕ_2) and evaluates ϕ_1 and ϕ_2 and asks the verifier for $\#\phi_1$ and $\#\phi_2$
4. Step 3: Prover sends k_1 and k_2
5. Step 4: Verifier verifies that $K = k_1 + k_2$
6. Repeat by setting each variable x_i to 0 and 1 and verifying

#3SAT $\subseteq$ IP? [Attempt 1]

Say the input is (ϕ,K). The verifier has to check whether ϕ indeed has K satisfying assignments. Try to verify observation*

1. Step 0: Verifier sends ϕ to the Prover and asks for number of satisfying assignments to ϕ
2. Step 1: Prover sends K
3. Step 2: Verifier sets x_1 to 0 in ϕ (ϕ_1) and x_1 to 1(ϕ_2) and evaluates ϕ_1 and ϕ_2 and asks the verifier for $\#\phi_1$ and $\#\phi_2$
4. Step 3: Prover sends k_1 and k_2
5. Step 4: Verifier verifies that $K = k_1 + k_2$
6. Repeat by setting each variable x_i to 0 and 1 and verifying
7. Step ??: Once all variables have been set, Verifier asks the prover the number of satisfying assignments and also verifies the answer by itself.

#3SAT $\subseteq$ IP? [Attempt 1]

#3SAT $\subseteq$ IP? [Attempt 1]

$\phi(x_1,..x_n)$

#3SAT $\subseteq$ IP? [Attempt 1]

$\phi(x_1,..x_n)$ K

#3SAT $\subseteq$ IP? [Attempt 1]

$\phi(x_1,..x_n)$ K

$\phi(x_1,..x_n) \longrightarrow \phi(0,..x_n)$

#3SAT $\subseteq$ IP? [Attempt 1]

$\phi(x_1,..x_n)$ K

$\phi(0,..x_n)$ k1

#3SAT $\subseteq$ IP? [Attempt 1]

$\phi(x_1,..x_n)$ K

$\phi(0,..x_n)$ k1

$\phi(1,..x_n)$

#3SAT $\subseteq$ IP? [Attempt 1]

$\phi(x_1,..x_n)$ K

$\phi(0,..x_n)$ k1

$\phi(1,..x_n)$ k2

#3SAT $\subseteq$ IP? [Attempt 1]

$\phi(x_1,..x_n)$ K

$\phi(0,..x_n)$ k1

$\phi(1,..x_n)$ k2

K=k1+k2?

#3SAT $\subseteq$ IP? [Attempt 1]

$\phi(x_1,..x_n)$ K

$\phi(0,..x_n)$ k1

K=k1+k2?

$\phi(1,..x_n)$ k2

$\phi(0,0,..x_n)$

$\phi(0,1,..x_n)$

#3SAT $\subseteq$ IP? [Attempt 1]

$\phi(x_1,..x_n)$ K

$\phi(0,..x_n)$ k1

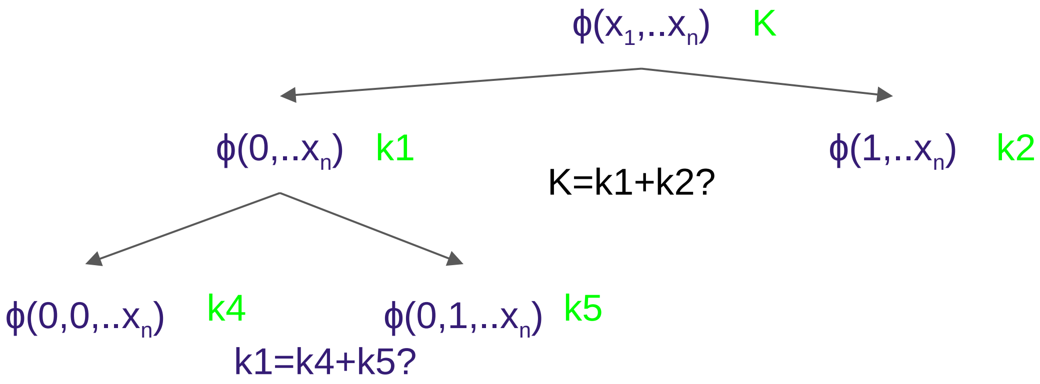
$\phi(1,..x_n)$ k2

K=k1+k2?

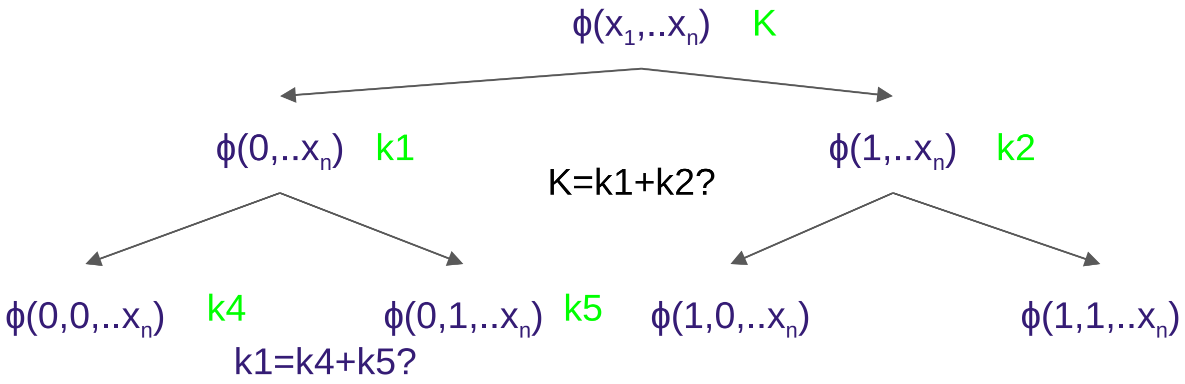
$\phi(0,0,..x_n)$ k4

$\phi(0,1,..x_n)$ k5

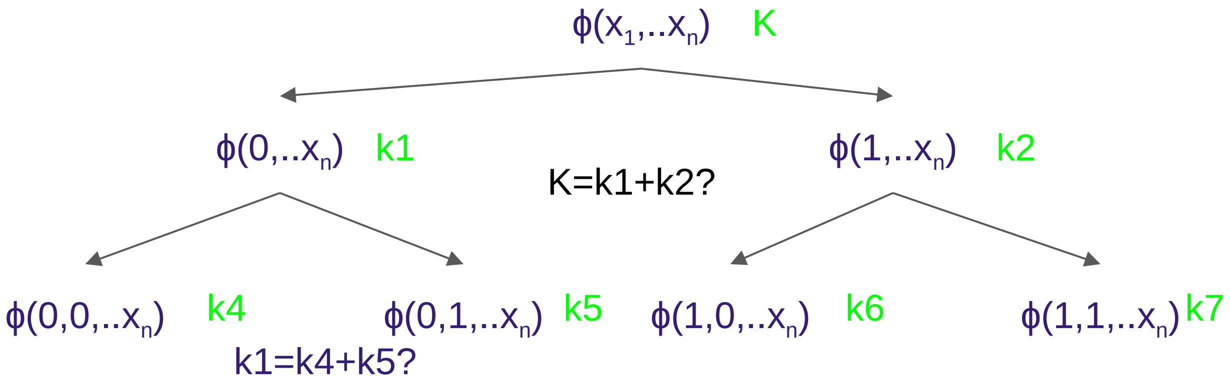
#3SAT $\subseteq$ IP? [Attempt 1]



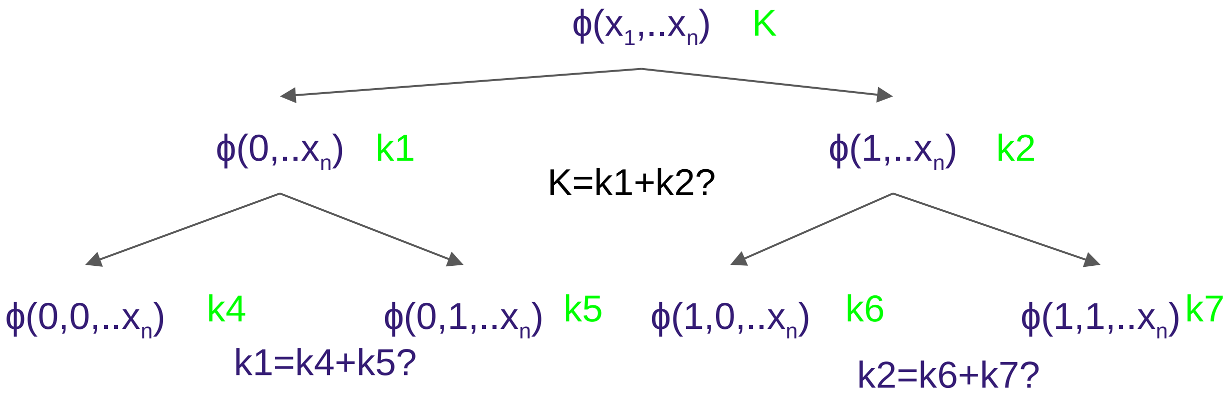
#3SAT $\subseteq$ IP? [Attempt 1]



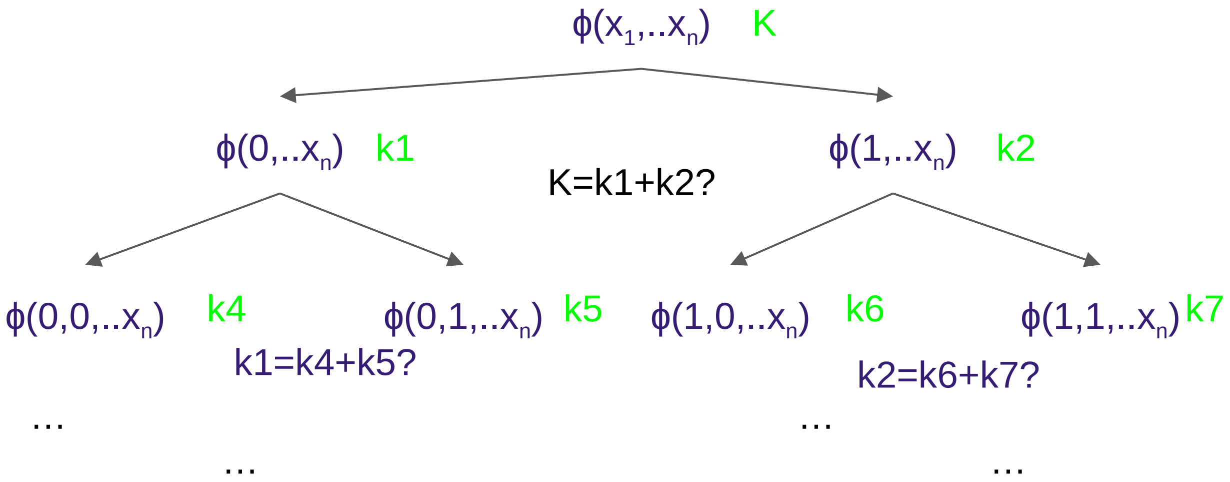
#3SAT $\subseteq$ IP? [Attempt 1]



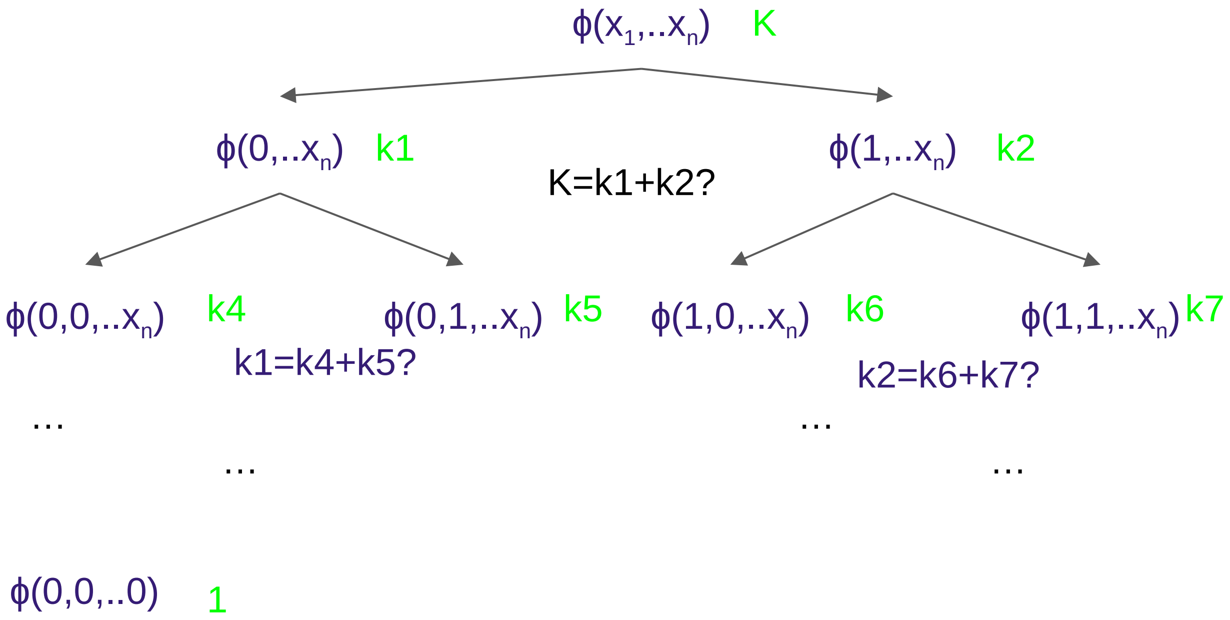
#3SAT $\subseteq$ IP? [Attempt 1]



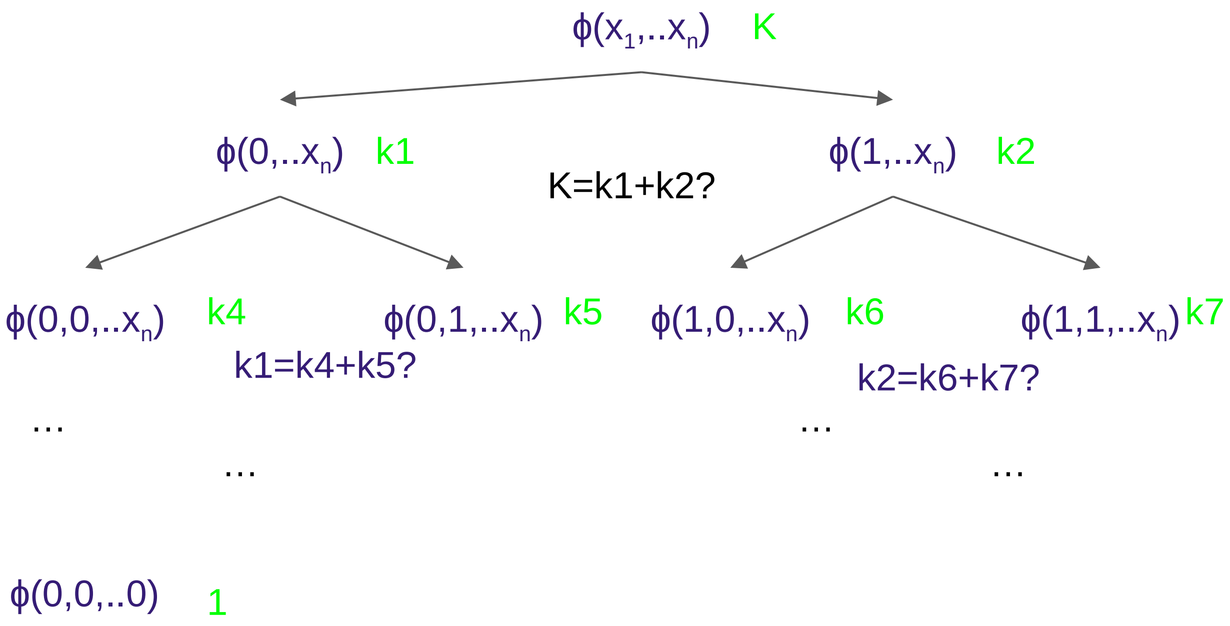
#3SAT $\subseteq$ IP? [Attempt 1]



#3SAT $\subseteq$ IP? [Attempt 1]



#3SAT $\subseteq$ IP? [Attempt 1]



$\phi(0,0,..0)$=1 == 1

#3SAT $\subseteq$ IP? [Attempt 1]

$\phi(x_1,..x_n)$ K

$\phi(0,..x_n)$ k1

K=k1+k2?

$\phi(1,..x_n)$ k2

$\phi(0,0,..x_n)$ k4

$\phi(0,1,..x_n)$ k5

$\phi(1,0,..x_n)$ k6

$\phi(1,1,..x_n)$ k7

k1=k4+k5?

k2=k6+k7?

…

…

…

…

$\phi(0,0,..0)$ 1

$\phi(0,1,..1)$ 0

$\phi(0,0,..0)$=1 == 1

#3SAT $\subseteq$ IP? [Attempt 1]

$\phi(x_1,..x_n)$ K

$\phi(0,..x_n)$ k1

$\phi(1,..x_n)$ k2

K=k1+k2?

$\phi(0,0,..x_n)$ k4

$\phi(0,1,..x_n)$ k5

$\phi(1,0,..x_n)$ k6

$\phi(1,1,..x_n)$ k7

k1=k4+k5?

k2=k6+k7?

…

…

…

…

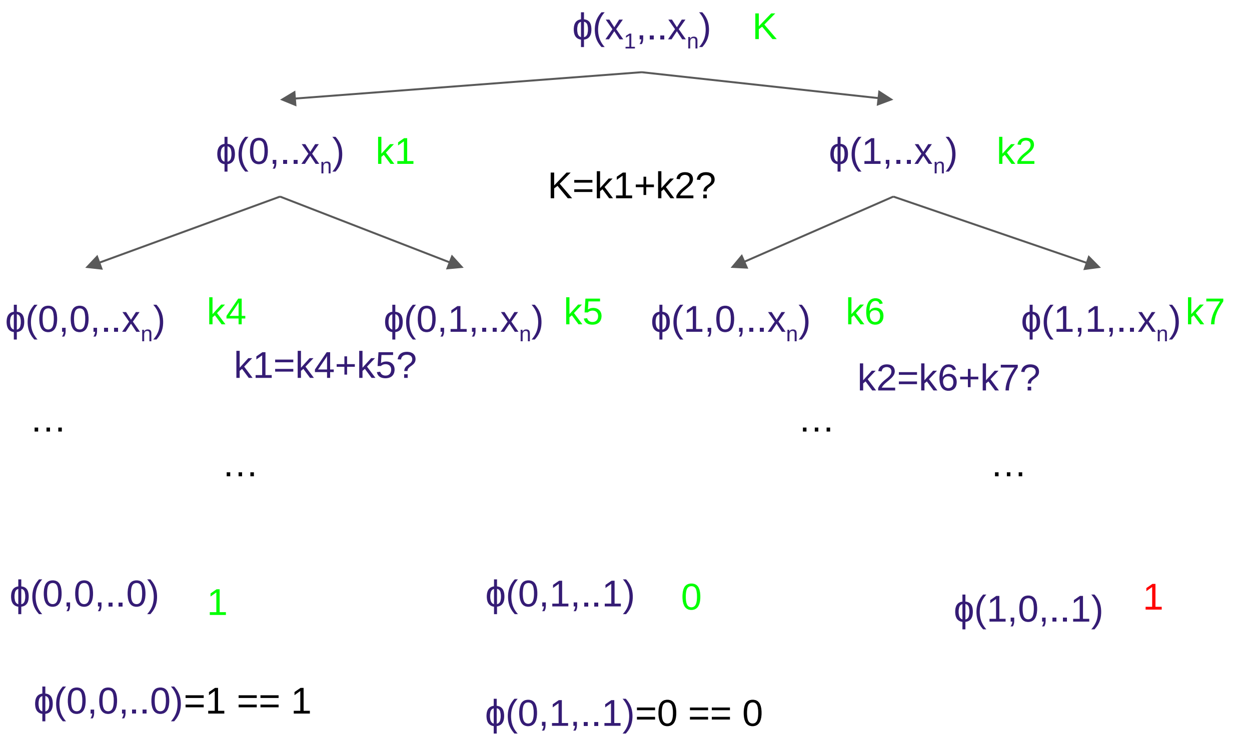
$\phi(0,0,..0)$ 1

$\phi(0,1,..1)$ 0

$\phi(0,0,..0)$=1 == 1

$\phi(0,1,..1)$=0 == 0

#3SAT ⊆ IP? [Attempt 1]



#3SAT $\subseteq$ IP? [Attempt 1]

$\phi(x_1,..x_n)$ K

$\phi(0,..x_n)$ k1

K=k1+k2?

$\phi(1,..x_n)$ k2

$\phi(0,0,..x_n)$ k4

$\phi(0,1,..x_n)$ k5

k1=k4+k5?

$\phi(1,0,..x_n)$ k6

$\phi(1,1,..x_n)$ k7

k2=k6+k7?

…

…

…

…

$\phi(0,0,..0)$ 1

$\phi(0,1,..1)$ 0

$\phi(1,0,..1)$ 1

$\phi(0,0,..0)$=1 == 1

$\phi(0,1,..1)$=0 == 0

$\phi(1,0,..1)$=0 != 1

#3SAT $\subseteq$ IP? [Attempt 1]

#3SAT $\subseteq$ IP? [Attempt 1]

Problem: Requires exponential rounds of interaction to enumerate over all assignments

#3SAT $\subseteq$ IP? [Attempt 1]

Problem: Requires exponential rounds of interaction to enumerate over all assignments

Issues: We are not using the probabilistic nature of the verifier

#3SAT $\subseteq$ IP? [Attempt 1]

Problem: Requires exponential rounds of interaction to enumerate over all assignments

Issues: We are not using the probabilistic nature of the verifier

Idea: Randomly choose a path in the tree

#3SAT $\subseteq$ IP? [Attempt 2]

#3SAT $\subseteq$ IP? [Attempt 2]

$\phi(x_1,..x_n)$

#3SAT $\subseteq$ IP? [Attempt 2]

$\phi(x_1,..x_n)$ K

#3SAT $\subseteq$ IP? [Attempt 2]

$\phi(x_1,..x_n)$ K

$\phi(0,..x_n)$

#3SAT $\subseteq$ IP? [Attempt 2]

$\phi(x_1,..x_n)$ K

$\phi(0,..x_n)$ k1

#3SAT $\subseteq$ IP? [Attempt 2]

$$\begin{tikzcd}[column sep=small] & \phi(x_1,..x_n) \quad K \arrow[dl] \arrow[dr] & \\ \phi(0,..x_n) \quad k1 & & \phi(1,..x_n) \end{tikzcd}$$

#3SAT $\subseteq$ IP? [Attempt 2]

$$\begin{array}{ccc} & \phi(x_1,..x_n) \quad \color{lime}{K} & \\ \swarrow & & \searrow \\ \phi(0,..x_n) \quad \color{lime}{k1} & & \phi(1,..x_n) \quad \color{lime}{k2} \end{array}$$

#3SAT $\subseteq$ IP? [Attempt 2]

$\phi(x_1,..x_n)$ K

$\phi(0,..x_n)$ k1

$\phi(1,..x_n)$ k2

K=k1+k2?

#3SAT $\subseteq$ IP? [Attempt 2]

$\phi(x_1,..x_n)$ K

$\phi(0,..x_n)$ k1

$\phi(1,..x_n)$ k2

K=k1+k2?

$\phi(1,0,..x_n)$

$\phi(1,1,..x_n)$

#3SAT $\subseteq$ IP? [Attempt 2]

$$\begin{array}{ccc} & \phi(x_1,..x_n) \quad K & \\ \swarrow & & \searrow \\ \phi(0,..x_n) \quad k1 & & \phi(1,..x_n) \quad k2 \end{array}$$

K=k1+k2?

$$\begin{array}{ccc} & \phi(1,..x_n) \quad k2 & \\ \swarrow & & \searrow \\ \phi(1,0,..x_n) \quad k6 & & \phi(1,1,..x_n) \quad k7 \end{array}$$

#3SAT $\subseteq$ IP? [Attempt 2]

$\phi(x_1,..x_n)$ K

$\phi(0,..x_n)$ k1

$\phi(1,..x_n)$ k2

K=k1+k2?

$\phi(1,0,..x_n)$ k6

$\phi(1,1,..x_n)$ k7

k2=k6+k7?

#3SAT $\subseteq$ IP? [Attempt 2]

$\phi(x_1,..x_n)$ K

$\phi(0,..x_n)$ k1

$\phi(1,..x_n)$ k2

K=k1+k2?

$\phi(1,0,..x_n)$ k6

$\phi(1,1,..x_n)$ k7

k2=k6+k7?

...

#3SAT $\subseteq$ IP? [Attempt 2]

$\phi(x_1,..x_n)$ K

$\phi(0,..x_n)$ k1

$\phi(1,..x_n)$ k2

K=k1+k2?

$\phi(1,0,..x_n)$ k6

$\phi(1,1,..x_n)$ k7

k2=k6+k7?

...

$\phi(1,0,..1)$ 1

#3SAT $\subseteq$ IP? [Attempt 2]

$\phi(x_1,..x_n)$ K

$\phi(0,..x_n)$ $k1$

$\phi(1,..x_n)$ $k2$

K=k1+k2?

$\phi(1,0,..x_n)$ $k6$

$\phi(1,1,..x_n)$ $k7$

k2=k6+k7?

…

$\phi(1,0,..1)$ 1

$\phi(1,0,..1)=0$!= 1

#3SAT $\subseteq$ IP? [Attempt 2]

#3SAT $\subseteq$ IP? [Attempt 2]

- Clearly, we may accidentally accept the wrong value.

#3SAT $\subseteq$ IP? [Attempt 2]

- Clearly, we may accidentally accept the wrong value.

- What if k1 was not actually the number of satisfying assignments of ϕ_1 and k2 is correct and we decide to go down k2. How lucky can the prover get?

#3SAT $\subseteq$ IP? [Attempt 2]

- Clearly, we may accidentally accept the wrong value.

- What if k1 was not actually the number of satisfying assignments of ϕ_1 and k2 is correct and we decide to go down k2. How lucky can the prover get?

#3SAT $\subseteq$ IP? [Attempt 2]

- Clearly, we may accidentally accept the wrong value.

- What if k1 was not actually the number of satisfying assignments of ϕ_1 and k2 is correct and we decide to go down k2. How lucky can the prover get?

- The probability that the prover actually gets caught is 2^{-n}. We need to catch every wrong branch at every step.

#3SAT $\subseteq$ IP? [Attempt 2]

- Clearly, we may accidentally accept the wrong value.

- What if k1 was not actually the number of satisfying assignments of ϕ_1 and k2 is correct and we decide to go down k2. How lucky can the prover get?

- The probability that the prover actually gets caught is 2^{-n}. We need to catch every wrong branch at every step.

- So, we always accept when the number of satisfying assignments are correct, but we will also accept when it is incorrect with probability $1 - 2^{-n}$.

Boolean is F_2

Boolean is F_2

Every boolean formula can be expressed as a polynomial over elements of F_2

We use the following trick:

$$a \land b \equiv ab$$

$$a \lor b \equiv 1 - (1-a)(1-b) \equiv a + b - ab$$

$$\neg a \equiv (1-a)$$

$$\text{True} \equiv 1$$

$$\text{False} \equiv 0$$

Example:

$$(x_1 \lor x_3 \lor \neg x_4) \equiv (x_1 + x_3 - x_1 x_3) + (1-x_4) - (x_1 + x_3 - x_1 x_3)(1-x_4)$$

Boolean is F_2

Boolean is F_2

- We are now able to express a boolean formula ϕ as a polynomial P_ϕ

Boolean is F_2

- We are now able to express a boolean formula ϕ as a polynomial P_ϕ

- The degree of each clause will be at most 3, as ϕ is a 3CNF, and the net degree will be at most 3m where there are m clauses in ϕ.

Boolean is F_2

- We are now able to express a boolean formula ϕ as a polynomial P_ϕ

- The degree of each clause will be at most 3, as ϕ is a 3CNF, and the net degree will be at most 3m where there are m clauses in ϕ.

- The size of the polynomial will also be bound polynomial in the size of ϕ as we don't need to expand the terms

Boolean is F_2

We can restate our equations as follows, where X_is are now formal variables

$$\#\phi = \Sigma_{b\_1 \in \{0,1\}} \Sigma_{b\_2 \in \{0,1\}} \ldots \Sigma_{b\_n \in \{0,1\}} P_\phi(b_1,\ldots b_n)$$

$$\#\phi(X_1,\ldots X_{i-1}) = \Sigma_{b\_i \in \{0,1\}} \ldots \Sigma_{b\_n \in \{0,1\}} P_\phi(X_1,\ldots,X_{i-1},b_i,\ldots,b_n)$$

$$\#\phi(X_1,\ldots X_i) = \#\phi(X_1,\ldots X_{i-1},0) + \#\phi(X_1,\ldots X_{i-1},1)$$

Theorem[LFKN ‘92]: #3SAT $\in$ IP

Theorem[LFKN '92]: #3SAT $\in$ IP

Key Idea: Arithmetization

Theorem[LFKN ‘92]: #3SAT $\in$ IP

Key Idea: Arithmetization

None of the previous definitions are impacted if we moved from F_2 to F_p as long as p is a suitably large prime

Theorem[LFKN '92]: #3SAT $\in$ IP

Key Idea: Arithmetization

None of the previous definitions are impacted if we moved from F_2 to F_p as long as p is a suitably large prime

Once we do that, we can plug in any element in F_p into our polynomial

Theorem[LFKN ‘92]: #3SAT $\in$ IP

Theorem[LFKN ‘92]: #3SAT $\in$ IP

- How large should p be?

Theorem[LFKN '92]: #3SAT $\in$ IP

- How large should p be?

The number of satisfying assignments can be at most 2^n, therefore, we can chose a prime between 2^n and 2^{2n}.

Theorem[LFKN ‘92]: #3SAT $\in$ IP

- How large should p be?

The number of satisfying assignments can be at most 2^n, therefore, we can chose a prime between 2^n and 2^{2n}.

We ask the prover to provide this prime at the start of the protocol and the verifier can verify primalty in polynomial time.

Sumcheck protocol

A generic protocol to verify equations of the form

$$K = \Sigma_{b\_1 \in \{0,1\}} \ldots \Sigma_{b\_n \in \{0,1\}}\, g(X_1,\ldots X_n) \qquad \ldots \text{ eq(1)}$$

Where g is any polynomial of small size and which can be evaluated in polynomial time.

Sumcheck protocol

A generic protocol to verify equations of the form

$$K = \Sigma_{b\_1 \in \{0,1\}} \ldots \Sigma_{b\_n \in \{0,1\}} g(X_1, \ldots X_n) \qquad \ldots \text{ eq(1)}$$

Where g is any polynomial of small size and which can be evaluated in polynomial time.

Obs: P_ϕ is a degree 3m polynomial it’s size is of the order of the size of ϕ. It can also be easily evaluated in the same way we evaluate formulas on assignments. So we can use the sumcheck protocol.

Sumcheck protocol

Sumcheck protocol

Obs: $h(X_1) = \Sigma_{b\_2 \in \{0,1\}} ... \Sigma_{b\_n \in \{0,1\}}\, g(X_1, b_2, ... b_n)$

Sumcheck protocol

Obs: $h(X_1) = \Sigma_{b\_2 \in \{0,1\}} \ldots \Sigma_{b\_n \in \{0,1\}}\, g(X_1, b_2, \ldots b_n)$

Is a univariate polynomial of degree at most m in the variable X_1.

Sumcheck protocol

Obs: $h(X_1) = \Sigma_{b\_2 \in \{0,1\}} \ldots \Sigma_{b\_n \in \{0,1\}}\, g(X_1, b_2, \ldots b_n)$

Is a univariate polynomial of degree at most m in the variable X_1.

If eq(1) is true, then $h(0) + h(1) = K$

Sumcheck protocol

Obs: $h(X_1) = \Sigma_{b\_2 \in \{0,1\}} \ldots \Sigma_{b\_n \in \{0,1\}}\, g(X_1, b_2, \ldots b_n)$

Is a univariate polynomial of degree at most m in the variable X_1.

If eq(1) is true, then $h(0) + h(1) = K$

The input to the protocol would be a polynomial $g(X_1,\ldots,X_n)$ and K.

Sumcheck protocol

Obs: $h(X_1) = \Sigma_{b\_2 \in \{0,1\}} \ldots \Sigma_{b\_n \in \{0,1\}} g(X_1, b_2, \ldots b_n)$

Is a univariate polynomial of degree at most m in the variable X_1.

If eq(1) is true, then $h(0) + h(1) = K$

The input to the protocol would be a polynomial $g(X_1,\ldots,X_n)$ and K.

Obs: g can be evaluated in polynomial time, however h cannot even be computed in polynomial time

Sumcheck protocol

Input: $g(X_1,...X_n)$, K

V: if n = 1, verify K = g(0) + g(1)

V: It asks the prover to send a polynomial h, as defined previously, a polynomial in X_1

P: sends a polynomial s

V: verify that s(0) + s(1) = K. Selects a random element from F_p, say a. It calculates s(a).

Recursively solve with the input as

$g(a,X_2,...X_n)$ and s(a).

Sumcheck protocol

Sumcheck protocol

$g(X_1,..X_n)$

Sumcheck protocol

$g(X_1,..X_n)$

Sumcheck protocol

$g(X_1,..X_n)$

$s_1(X_1)$

Sumcheck protocol

$g(X_1,..X_n)$

$s_1(X_1)$

Sumcheck protocol

$g(X_1,..X_n)$

$s_1(X_1)$

$s_1(0)+s_1(1) == K$

Sumcheck protocol

$g(X_1,..X_n)$

$s_1(X_1)$

$s_1(0)+s_1(1) == K$

$g(a_1,X_2,...X_n)$

Sumcheck protocol

$g(X_1,..X_n)$

$s_1(X_1)$

$s_1(0)+s_1(1) == K$

$g(a_1,X_2,...X_n)$

Sumcheck protocol

$g(X_1,...X_n)$

$s_1(X_1)$

$s_1(0)+s_1(1) == K$

$g(a_1,X_2,...X_n)$

$s_2(X_2)$

Sumcheck protocol

$g(X_1,..X_n)$

$s_1(X_1)$

$s_1(0)+s_1(1) == K$

$g(a_1,X_2,...X_n)$

$s_2(X_2)$

Sumcheck protocol

$g(X_1,...X_n)$

$s_1(X_1)$

$s_1(0)+s_1(1) == K$

$g(a_1,X_2,...X_n)$

$s_2(X_2)$

$s_2(0)+s_2(1) == s_1(a_1)$

Sumcheck protocol

$g(X_1,...X_n)$

$s_1(X_1)$

$s_1(0)+s_1(1) == K$

$g(a_1,X_2,...X_n)$

$s_2(X_2)$

$s_2(0)+s_2(1) == s_1(a_1)$

$g(a_1,a_2,...X_n)$

Sumcheck protocol

$g(X_1,...X_n)$

$s_1(X_1)$

$s_1(0)+s_1(1) == K$

$g(a_1,X_2,...X_n)$

$s_2(X_2)$

$s_2(0)+s_2(1) == s_1(a_1)$

$g(a_1,a_2,...X_n)$

...

Sumcheck protocol

$g(X_1,\ldots X_n)$

$s_1(X_1)$

$s_1(0)+s_1(1) == K$

$g(a_1,X_2,\ldots X_n)$

$s_2(X_2)$

$s_2(0)+s_2(1) == s_1(a_1)$

$g(a_1,a_2,\ldots X_n)$

…

$s_n(X_n)$

Sumcheck protocol

$g(X_1,\ldots X_n)$

$s_1(X_1)$

$s_1(0)+s_1(1) == K$

$g(a_1,X_2,\ldots X_n)$

$s_2(X_2)$

$s_2(0)+s_2(1) == s_1(a_1)$

$g(a_1,a_2,\ldots X_n)$

...

$s_n(X_n)$

Sumcheck protocol

$g(X_1,\ldots X_n)$

$s_1(X_1)$

$s_1(0)+s_1(1) == K$

$g(a_1,X_2,\ldots X_n)$

$s_2(X_2)$

$s_2(0)+s_2(1) == s_1(a_1)$

$g(a_1,a_2,\ldots X_n)$

...

$s_n(X_n)$

$s_n(0)+s_n(1) == s_{n-1}(a_{n-1})$

Sumcheck protocol

$g(X_1,...X_n)$

$s_1(X_1)$

$s_1(0)+s_1(1) == K$

$g(a_1,X_2,...X_n)$

$s_2(X_2)$

$s_2(0)+s_2(1) == s_1(a_1)$

$g(a_1,a_2,...X_n)$

...

$s_n(X_n)$

$s_n(0)+s_n(1) == s_{n-1}(a_{n-1})$ $g(a_1,a_2,...a_n) == s_n(a_n)$

Analysis of protocol

Analysis of protocol

- Sending univariate polynomials is sending d numbers where d is the degree of the polynomial.

Analysis of protocol

- Sending univariate polynomials is sending d numbers where d is the degree of the polynomial.

- If eq(1) is true, then the prover sends the correct polynomial h in the first round, ie, $s_1 = h$. So we will never reject a correct string. (Perfect completeness)

Analysis of protocol

- Sending univariate polynomials is sending d numbers where d is the degree of the polynomial.

- If eq(1) is true, then the prover sends the correct polynomial h in the first round, ie, $s_1 = h$. So we will never reject a correct string. (Perfect completeness)

- How lucky does the prover need to be for the verifier to accept an incorrect string?

Analysis of Error bound

Analysis of Error bound

What is the probability over a that s(a)=h(a) for 2 univariate polynomials s and h?

Analysis of Error bound

What is the probability over a that s(a)=h(a) for 2 univariate polynomials s and h?

From the Schwartz-Zippel lemma, we have a bound on this number

Analysis of Error bound

What is the probability over a that s(a)=h(a) for 2 univariate polynomials s and h?

From the Schwartz-Zippel lemma, we have a bound on this number

$$Pr_a[s(a)-h(a)=0] \leq d/p$$

Analysis of Error bound

What is the probability over a that s(a)=h(a) for 2 univariate polynomials s and h?

From the Schwartz-Zippel lemma, we have a bound on this number

$$Pr_a[s(a)-h(a)=0] \leq d/p$$

Where d is the degree of the difference polynomial and p is the size of the field.

Analysis of Error bound

What is the probability over a that $s(a)=h(a)$ for 2 univariate polynomials s and h?

From the Schwartz-Zippel lemma, we have a bound on this number

$$Pr_a[s(a)-h(a)=0] \leq d/p$$

Where d is the degree of the difference polynomial and p is the size of the field.

Thus, the probability that at any step, the prover is caught is at least 1-d/p. Therefore, applying the union bound, the probability that the prover is never caught is (d*n/p)

Analysis of Error bound

What is the probability over a that s(a)=h(a) for 2 univariate polynomials s and h?

From the Schwartz-Zippel lemma, we have a bound on this number

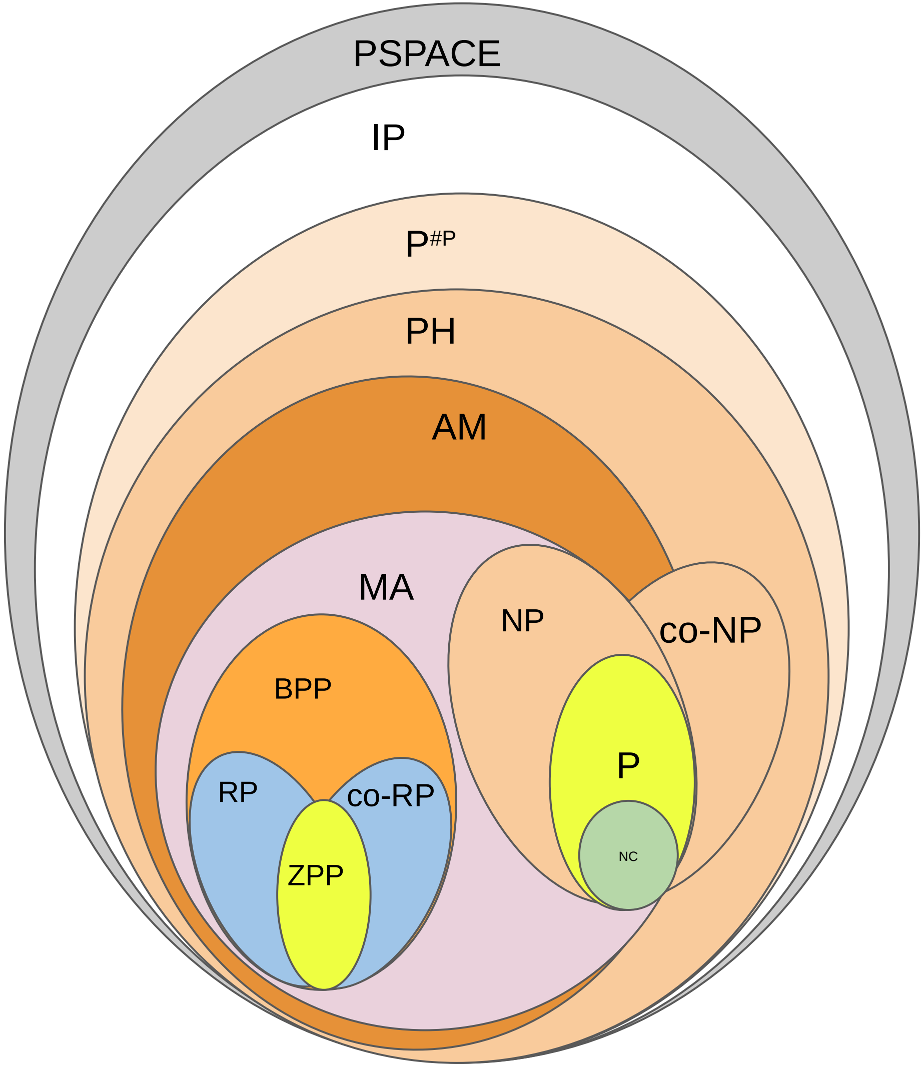
$$Pr_a[s(a)-h(a)=0] \leq d/p$$

Where d is the degree of the difference polynomial and p is the size of the field.

Thus, the probability that at any step, the prover is caught is at least 1-d/p. Therefore, applying the union bound, the probability that the prover is never caught is (d*n/p)

Therefore the error probability is less than $3n^2/2^n$ which is less than ⅓ for n>9

What's in IP?



TQBF $\subseteq$ IP?

Definition: TQBF

$$\text{TQBF} = \{\ \Psi = Q_1x_1 \ldots Q_nx_n\ \phi(x_1,\ldots x_n) \mid \Psi = \text{True},\ Q_i \text{ in } \{\exists,\forall\}, \text{ boolean formula } \phi\}$$

$\Psi = \forall x_1,\exists x_2,\forall x_3 \ldots \exists x_n\ \phi(x_1,\ldots x_n) \in \text{TQBF}$ iff

$$\Pi_{b\_1 \in \{0,1\}} \Sigma_{b\_2 \in \{0,1\}} \Pi_{b\_3 \in \{0,1\}} \ldots \Sigma_{b\_n \in \{0,1\}} P_\phi(b_1,\ldots b_n) = 1$$

Where P_ϕ is the polynomial as defined before over F_2

Sumcheck Protocol?

Sumcheck Protocol?

- How do we modify the sumcheck protocol for TQBF?

Sumcheck Protocol?

- How do we modify the sumcheck protocol for TQBF?

Obs 1: Add over ∃

Sumcheck Protocol?

- How do we modify the sumcheck protocol for TQBF?

Obs 1: Add over ∃

As for 3SAT, when we need a univariate polynomial over a variable quantified by ∃, we must check the additivity, i.e, $s(0)+s(1) = K$

Sumcheck Protocol?

- How do we modify the sumcheck protocol for TQBF?

Obs 1: Add over ∃

As for 3SAT, when we need a univariate polynomial over a variable quantified by ∃, we must check the additivity, i.e, $s(0)+s(1) = K$

Obs 2: Multiply over ∀

Sumcheck Protocol?

- How do we modify the sumcheck protocol for TQBF?

Obs 1: Add over ∃

As for 3SAT, when we need a univariate polynomial over a variable quantified by ∃, we must check the additivity, i.e, $s(0)+s(1) = K$

Obs 2: Multiply over ∀

When we have a univariate polynomial over a variable quantified by ∀, we must check multiplicity, i.e, $s(0) \cdot s(1) = K$

Sumcheck Protocol?

- Unlike adding polynomials, multiplying polynomials increase the degree

- If we define $h(X_1)$ as defined previously:

$$h(X_1) = \Sigma_{b\_2 \in \{0,1\}} \Pi_{b\_3=\{0,1\}} \ldots \Sigma_{b\_n \in \{0,1\}} P_{\phi}(X_1, \ldots b_n)$$

This can have degree at most 2^n. Which cannot be sent from the prover to the verifier.

Obs:

$$x^k = x \text{ in } F_2 \text{ for any } k > 0$$

Linearization

Obs:

$$x^k = x \text{ in } F_2 \text{ for any } k > 0$$

Linearization

Obs:

$$x^k = x \text{ in } F_2 \text{ for any } k > 0$$

Any polynomial $p(X_1,...X_n)$ can be converted to a *multilinear* polynomial $q(X_1,...X_n)$ where

1. The degree of any variable in any term of q is at most 1
2. $p(a_1,...a_n) = q(a_1,...a_n)$ for any $a_1 \ldots a_n \in \{0,1\}$

Linearization

Definition: Linearization operator L

$L_i(p) = X_i \cdot p(X_1,\ldots,X_{i-1},1,X_{i+1},\ldots X_n) + (1-X_i) \cdot p(X_1,\ldots,X_{i-1},0,X_{i+1},\ldots X_n)$

Defines a new polynomial such that

1. Degree of X_i in $L_i(p)$ is at most 1
2. $L_i(p)$ gives the same values as p for all binary inputs

Obs: $q = L_1(L_2(\ldots L_n(p)\ldots)))$

Linearization

Definition: ∀ operator for polynomials

$$\forall_i\, p(X_1,\ldots X_n) = p(X_1,\ldots,X_{i-1},0,X_{i+1},\ldots X_n) \cdot p(X_1,\ldots,X_{i-1},1,X_{i+1},\ldots X_n)$$

Definition: ∃ operator for polynomials

$$\exists_i\, p(X_1,\ldots X_n) = p(X_1,\ldots,X_{i-1},0,X_{i+1},\ldots X_n) + p(X_1,\ldots,X_{i-1},1,X_{i+1},\ldots X_n)$$

Linearization

Linearization

Original polynomial:

Linearization

Original polynomial:

$$\Pi_{b\_1 \in \{0,1\}} \Sigma_{b\_2 \in \{0,1\}} \Pi_{b\_3 \in \{0,1\}} \ldots \Sigma_{b\_n \in \{0,1\}} P_\phi(b_1,\ldots b_n) = 1$$

Linearization

Original polynomial:

$$\Pi_{b\_1 \in \{0,1\}}\Sigma_{b\_2 \in \{0,1\}}\Pi_{b\_3 \in \{0,1\}}...\Sigma_{b\_n \in \{0,1\}} P_{\phi}(b_1,...b_n) = 1$$

Can be equivalently rewritten as

Linearization

Original polynomial:

$$\Pi_{b\_1 \in \{0,1\}} \Sigma_{b\_2 \in \{0,1\}} \Pi_{b\_3 \in \{0,1\}} \ldots \Sigma_{b\_n \in \{0,1\}} P_{\phi}(b_1,\ldots b_n) = 1$$

Can be equivalently rewritten as

$$\forall_1 \exists_2 \forall_3 \ldots \exists_n P_{\phi}(X_1,\ldots X_n) = 1$$

Linearization

Original polynomial:

$$\Pi_{b\_1 \in \{0,1\}} \Sigma_{b\_2 \in \{0,1\}} \Pi_{b\_3 \in \{0,1\}} \ldots \Sigma_{b\_n \in \{0,1\}} P_{\phi}(b_1,\ldots b_n) = 1$$

Can be equivalently rewritten as

$$\forall_1 \exists_2 \forall_3 \ldots \exists_n P_{\phi}(X_1,\ldots X_n) = 1$$

Since we only care about using {0,1} to $P_{\phi}(X_1,\ldots X_n)$, we do not lose semantics by adding linearization operators in between,

Linearization

Original polynomial:

$$\Pi_{b\_1 \in \{0,1\}} \Sigma_{b\_2 \in \{0,1\}} \Pi_{b\_3 \in \{0,1\}} \ldots \Sigma_{b\_n \in \{0,1\}} P_\phi(b_1,\ldots b_n) = 1$$

Can be equivalently rewritten as

$$\forall_1 \exists_2 \forall_3 \ldots \exists_n P_\phi(X_1,\ldots X_n) = 1$$

Since we only care about using {0,1} to $P_\phi(X_1,\ldots X_n)$, we do not lose semantics by adding linearization operators in between,

$$\forall_1 L_1\ \exists_2 L_1 L_2\ \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n P_\phi(X_1,\ldots X_n) = 1$$

Linearization

Original polynomial:

$$\Pi_{b\_1 \in \{0,1\}}\Sigma_{b\_2 \in \{0,1\}}\Pi_{b\_3 \in \{0,1\}}...\Sigma_{b\_n \in \{0,1\}} P_\phi(b_1,...b_n) = 1$$

Can be equivalently rewritten as

$$\forall_1\exists_2\forall_3\ldots\exists_n P_\phi(X_1,...X_n) = 1$$

Since we only care about using {0,1} to $P_\phi(X_1,...X_n)$, we do not lose semantics by adding linearization operators in between,

$$\forall_1 L_1 \ \exists_2 L_1 L_2 \ \forall_3\ldots\exists_n L_1 L_2\ldots L_n P_\phi(X_1,...X_n) = 1$$

The size of this expression is increased due to the addition of the linearization operator. The size will then be $O(n+1+2+...+n+|P_\phi|)$, which is still poly-size

Modified Sumcheck protocol

Modified Sumcheck protocol

Consider a polynomial $g(X_1,...X_n)$, we need to check whether

Modified Sumcheck protocol

Consider a polynomial $g(X_1,...X_n)$, we need to check whether

$$\forall_1 L_1\ \exists_2 L_1 L_2\ \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(X_1,...X_n) = 1$$

Modified Sumcheck protocol

Consider a polynomial $g(X_1,...X_n)$, we need to check whether

$$\forall_1 L_1 \exists_2 L_1 L_2 \forall_3 \dots \exists_n L_1 L_2 \dots L_n g(X_1,...X_n) = 1$$

Input: $R_1 R_2 \dots R_t g(X_1,...X_n)$ where R represents one of the 3 operators, t is poly(n) and a claim C.

Modified Sumcheck protocol

Consider a polynomial $g(X_1,...X_n)$, we need to check whether

$$\forall_1 L_1\ \exists_2 L_1 L_2\ \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(X_1,...X_n) = 1$$

Input: $R_1 R_2 \ldots R_t g(X_1,...X_n)$ where R represents one of the 3 operators, t is poly(n) and a claim C.

TQBF: g would be P_ϕ, t would be $o(n^3)$, and C would be 1

Modified Sumcheck protocol

V: provide a polynomial equal to $R_2 \ldots R_t g(X_1, \ldots X_n)$

P: returns a polynomial $s(X_1)$

V: 1) If $R_1 = \exists_1$ verify that $s(0) + s(1) = C$

2) If $R_1 = \forall_1$ verify that $s(0) \cdot s(1) = C$

3) If $R_1 = L_1$ and verify that $a \cdot s(1) + (1-a) \cdot s(0) = s(a)$

If all checks pass, pick a random element a, recursively prove that the polynomial $R_2 \ldots R_t g(a, \ldots X_n) = s(a)$

Modified Sumcheck protocol

$\forall_1 L_1\ \exists_2 L_1 L_2\ \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(X_1, \ldots X_n)$

Modified Sumcheck protocol

$\forall_1 L_1 \exists_2 L_1 L_2 \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(X_1,\ldots X_n)$

Modified Sumcheck protocol

$\forall_1 L_1 \exists_2 L_1 L_2 \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(X_1,\ldots X_n)$ $\longrightarrow$ $s_1(X_1)$

Modified Sumcheck protocol

$\forall_1 L_1 \; \exists_2 L_1 L_2 \; \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(X_1, \ldots X_n)$

$s_1(X_1)$

Modified Sumcheck protocol

$\forall_1 L_1 \; \exists_2 L_1 L_2 \; \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(X_1, \ldots X_n)$

$s_1(X_1)$

$s_1(0) \cdot s_1(1) == C$

Modified Sumcheck protocol

$\forall_1 L_1\ \exists_2 L_1 L_2\ \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(X_1,\ldots X_n)$

$s_1(X_1)$

$s_1(0) \cdot s_1(1) == C$

$L_1\ \exists_2 L_1 L_2\ \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(a_1, X_2,\ldots X_n)$

Modified Sumcheck protocol

$\forall_1 L_1\ \exists_2 L_1 L_2\ \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(X_1,\ldots X_n)$

$s_1(X_1)$

$s_1(0) \cdot s_1(1) == C$

$L_1\ \exists_2 L_1 L_2\ \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(a_1, X_2,\ldots X_n)$

Modified Sumcheck protocol

$\forall_1 L_1 \; \exists_2 L_1 L_2 \; \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(X_1, \ldots X_n)$ $\longrightarrow$ $s_1(X_1)$

$s_1(0) \cdot s_1(1) == C$ $\longleftarrow$

$L_1 \; \exists_2 L_1 L_2 \; \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(a_1, X_2, \ldots X_n)$ $\longrightarrow$ $s_2(X_1)$

Modified Sumcheck protocol

$\forall_1 L_1\ \exists_2 L_1 L_2\ \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(X_1,\ldots X_n)$

$s_1(X_1)$

$s_1(0) \cdot s_1(1) == C$

$L_1\ \exists_2 L_1 L_2\ \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(a_1, X_2, \ldots X_n)$

$s_2(X_1)$

Modified Sumcheck protocol

$\forall_1 L_1 \exists_2 L_1 L_2 \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(X_1,\ldots X_n)$

$s_1(X_1)$

$s_1(0) \cdot s_1(1) == C$

$L_1 \exists_2 L_1 L_2 \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(a_1,X_2,\ldots X_n)$

$s_2(X_1)$

$(1 - a_1) \cdot s_2(0) + a_1 \cdot s_2(1) == s_2(a_1)$

Modified Sumcheck protocol

$$\forall_1 L_1\ \exists_2 L_1 L_2\ \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(X_1,...X_n)$$

$$s_1(X_1)$$

$$s_1(0) \cdot s_1(1) == C$$

$$L_1\ \exists_2 L_1 L_2\ \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(a_1,X_2,...X_n)$$

$$s_2(X_1)$$

$$(1 - a_1) \cdot s_2(0) + a_1 \cdot s_2(1) == s_2(a_1)$$

$$\exists_2 L_1 L_2\ \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(a_1,a_2,...X_n)$$

Modified Sumcheck protocol

$\forall_1 L_1\ \exists_2 L_1 L_2\ \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(X_1,\ldots X_n)$

$s_1(X_1)$

$s_1(0) \cdot s_1(1) == C$

$L_1\ \exists_2 L_1 L_2\ \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(a_1, X_2,\ldots X_n)$

$s_2(X_1)$

$(1 - a_1) \cdot s_2(0) + a_1 \cdot s_2(1) == s_2(a_1)$

$\exists_2 L_1 L_2\ \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(a_1, a_2,\ldots X_n)$

Modified Sumcheck protocol

$\forall_1 L_1\ \exists_2 L_1 L_2\ \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(X_1,...X_n)$

$s_1(X_1)$

$s_1(0) \cdot s_1(1) == C$

$L_1\ \exists_2 L_1 L_2\ \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(a_1,X_2,...X_n)$

$s_2(X_1)$

$(1 - a_1) \cdot s_2(0) + a_1 \cdot s_2(1) == s_2(a_1)$

$\exists_2 L_1 L_2\ \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(a_1,a_2,...X_n)$

$s_3(X_2)$

Modified Sumcheck protocol

$\forall_1 L_1 \ \exists_2 L_1 L_2 \ \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(X_1,\ldots X_n)$

$s_1(X_1)$

$s_1(0) \cdot s_1(1) == C$

$L_1 \ \exists_2 L_1 L_2 \ \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(a_1, X_2, \ldots X_n)$

$s_2(X_1)$

$(1 - a_1) \cdot s_2(0) + a_1 \cdot s_2(1) == s_2(a_1)$

$\exists_2 L_1 L_2 \ \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(a_1, a_2, \ldots X_n)$

$s_3(X_2)$

$s_3(0) + s_3(1) == s_2(a_2)$

Modified Sumcheck protocol

$\forall_1 L_1\ \exists_2 L_1 L_2\ \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(X_1,\ldots X_n)$

$s_1(X_1)$

$s_1(0) \cdot s_1(1) == C$

$L_1\ \exists_2 L_1 L_2\ \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(a_1, X_2,\ldots X_n)$

$s_2(X_1)$

$(1 - a_1) \cdot s_2(0) + a_1 \cdot s_2(1) == s_2(a_1)$

$\exists_2 L_1 L_2\ \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(a_1, a_2,\ldots X_n)$

$s_3(X_2)$

$s_3(0) + s_3(1) == s_2(a_2)$

$L_1 L_2\ \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(a_1, a_2, a_3,\ldots X_n)$

Modified Sumcheck protocol

$\forall_1 L_1 \; \exists_2 L_1 L_2 \; \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(X_1,\ldots X_n)$

$s_1(X_1)$

$s_1(0) \cdot s_1(1) == C$

$L_1 \; \exists_2 L_1 L_2 \; \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(a_1,X_2,\ldots X_n)$

$s_2(X_1)$

$(1 - a_1) \cdot s_2(0) + a_1 \cdot s_2(1) == s_2(a_1)$

$\exists_2 L_1 L_2 \; \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(a_1,a_2,\ldots X_n)$

$s_3(X_2)$

$s_3(0) + s_3(1) == s_2(a_2)$

$L_1 L_2 \; \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(a_1,a_2,a_3,\ldots X_n)$

…

Modified Sumcheck protocol

$\forall_1 L_1 \exists_2 L_1 L_2 \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(X_1,\ldots X_n)$

$s_1(X_1)$

$s_1(0) \cdot s_1(1) == C$

$L_1 \exists_2 L_1 L_2 \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(a_1,X_2,\ldots X_n)$

$s_2(X_1)$

$(1 - a_1) \cdot s_2(0) + a_1 \cdot s_2(1) == s_2(a_1)$

$\exists_2 L_1 L_2 \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(a_1,a_2,\ldots X_n)$

$s_3(X_2)$

$s_3(0) + s_3(1) == s_2(a_2)$

$L_1 L_2 \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(a_1,a_2,a_3,\ldots X_n)$

…

$s_t(X_n)$

Modified Sumcheck protocol

$\forall_1 L_1\ \exists_2 L_1 L_2\ \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(X_1, \ldots X_n)$

$s_1(X_1)$

$s_1(0) \cdot s_1(1) == C$

$L_1\ \exists_2 L_1 L_2\ \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(a_1, X_2, \ldots X_n)$

$s_2(X_1)$

$(1 - a_1) \cdot s_2(0) + a_1 \cdot s_2(1) == s_2(a_1)$

$\exists_2 L_1 L_2\ \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(a_1, a_2, \ldots X_n)$

$s_3(X_2)$

$s_3(0) + s_3(1) == s_2(a_2)$

$L_1 L_2\ \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(a_1, a_2, a_3, \ldots X_n)$

…

$s_t(X_n)$

Modified Sumcheck protocol

$\forall_1 L_1 \exists_2 L_1 L_2 \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(X_1,\ldots X_n)$

$s_1(X_1)$

$s_1(0) \cdot s_1(1) == C$

$L_1 \exists_2 L_1 L_2 \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(a_1,X_2,\ldots X_n)$

$s_2(X_1)$

$(1 - a_1) \cdot s_2(0) + a_1 \cdot s_2(1) == s_2(a_1)$

$\exists_2 L_1 L_2 \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(a_1,a_2,\ldots X_n)$

$s_3(X_2)$

$s_3(0) + s_3(1) == s_2(a_2)$

$L_1 L_2 \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(a_1,a_2,a_3,\ldots X_n)$

…

$s_t(X_n)$

$(1 - a_{n-1}) \cdot s_t(0) + a_{n-1} \cdot s_t(1) == s_t(a_{n-1})$

Modified Sumcheck protocol

$\forall_1 L_1\ \exists_2 L_1 L_2\ \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(X_1,\ldots X_n)$

$s_1(X_1)$

$s_1(0) \cdot s_1(1) == C$

$L_1\ \exists_2 L_1 L_2\ \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(a_1, X_2,\ldots X_n)$

$s_2(X_1)$

$(1 - a_1) \cdot s_2(0) + a_1 \cdot s_2(1) == s_2(a_1)$

$\exists_2 L_1 L_2\ \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(a_1, a_2,\ldots X_n)$

$s_3(X_2)$

$s_3(0) + s_3(1) == s_2(a_2)$

$L_1 L_2\ \forall_3 \ldots \exists_n L_1 L_2 \ldots L_n g(a_1, a_2, a_3,\ldots X_n)$

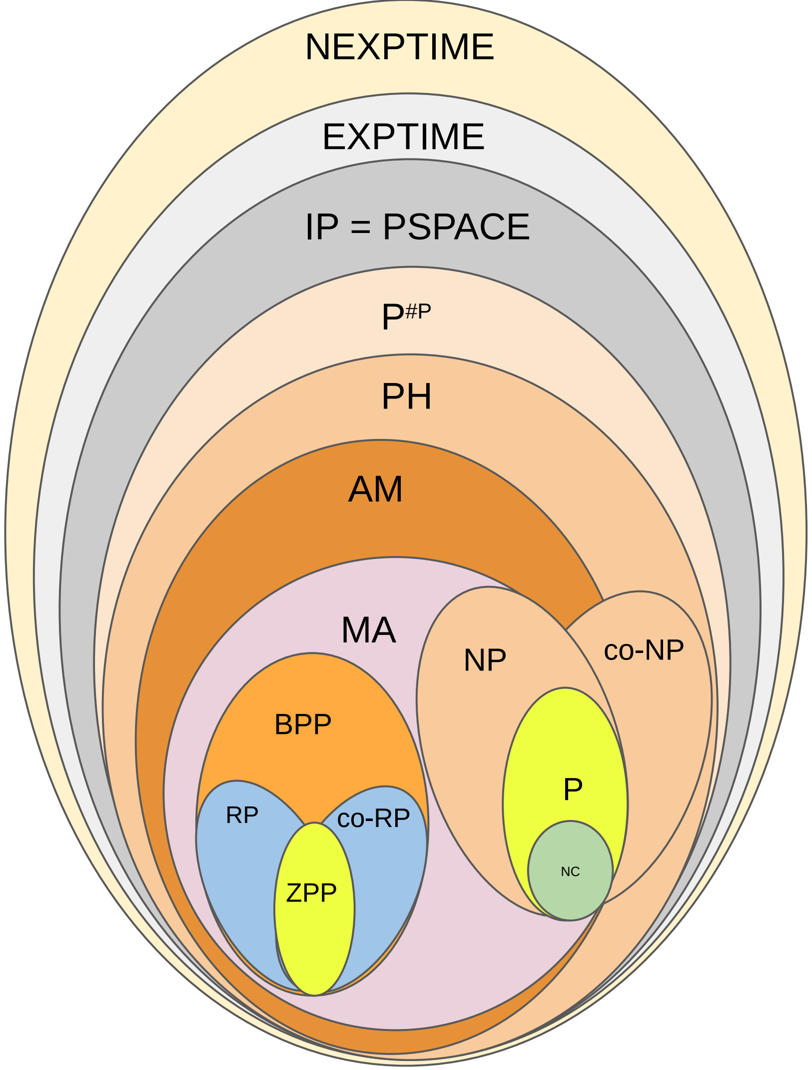
…

$s_t(X_n)$

$(1 - a_{n-1}) \cdot s_t(0) + a_{n-1} \cdot s_t(1) == s_t(a_{n-1})$

$g(a_1, a_2,\ldots a_n) == s_t(a_n)$

Where's IP?


NEXPTIME
EXPTIME
IP = PSPACE
P#P
PH
AM
MA
NP
co-NP
BPP
RP
co-RP
ZPP
P
NC

MIP

MIP

- We don’t need to restrict ourselves to one prover. If we could interact with multiple provers, we would get the class **MIP[BGK ‘88]**

MIP

- We don’t need to restrict ourselves to one prover. If we could interact with multiple provers, we would get the class **MIP[BGK ‘88]**

- Note: Provers cannot talk to each other, they communicate only to the verifier on the transcript which everyone can see.

MIP

- We don’t need to restrict ourselves to one prover. If we could interact with multiple provers, we would get the class **MIP[BGK ‘88]**

- Note: Provers cannot talk to each other, they communicate only to the verifier on the transcript which everyone can see.

- What power does each prover give? More Provers => More Power?

MIP

- We don’t need to restrict ourselves to one prover. If we could interact with multiple provers, we would get the class **MIP[BGK ‘88]**

- Note: Provers cannot talk to each other, they communicate only to the verifier on the transcript which everyone can see.

- What power does each prover give? More Provers => More Power?

  No.

MIP

- We don’t need to restrict ourselves to one prover. If we could interact with multiple provers, we would get the class **MIP[BGK ‘88]**

- Note: Provers cannot talk to each other, they communicate only to the verifier on the transcript which everyone can see.

- What power does each prover give? More Provers => More Power?

  No.

- Theorem[BFL ‘91]: **MIP** = **MIP[2]** = **NEXPTIME**

QIP, MIP*

QIP, MIP*

- Replacing the **BPP** verifier with a **BQP** verifier in **IP** gives **QIP[Wat ‘99]**

QIP, MIP*

- Replacing the **BPP** verifier with a **BQP** verifier in **IP** gives **QIP[Wat ‘99]**
- Theorem [JJUW ‘09]: **QIP** = **PSPACE**

QIP, MIP*

- Replacing the **BPP** verifier with a **BQP** verifier in **IP** gives **QIP[Wat ‘99]**
- Theorem [JJUW ‘09]: **QIP** = **PSPACE**

- What if we allowed provers to converse in **MIP**? Suppose, through arbitrary length quantum entangled qubits. We would get the class **MIP*[CHT ‘04]**

QIP, MIP*

- Replacing the **BPP** verifier with a **BQP** verifier in **IP** gives **QIP[Wat ‘99]**
- Theorem [JJUW ‘09]: **QIP** = **PSPACE**

- What if we allowed provers to converse in **MIP**? Suppose, through arbitrary length quantum entangled qubits. We would get the class **MIP*[CHT ‘04]**

- Theorem[JNVWY ‘20]: **MIP*** = **RE**

QIP, MIP*

- Replacing the **BPP** verifier with a **BQP** verifier in **IP** gives **QIP[Wat ‘99]**
- Theorem [JJUW ‘09]: **QIP** = **PSPACE**

- What if we allowed provers to converse in **MIP**? Suppose, through arbitrary length quantum entangled qubits. We would get the class **MIP*[CHT ‘04]**

- Theorem[JNVWY ‘20]: **MIP*** = **RE**

- We would be able to solve undecidable problems like the halting problem

IP = PSPACE Timeline

1985: AM, MA defined by Babai

1986: Goldwasser and Sipser show public coin private coin equivalence

1988: AM=AM[2] by BM, MIP is defined by BGKW

1989: IP is defined by GMR

1991: ZKP(NONISO in IP) by GMW, MIP=NEXP by BFL

1992: #3SAT in IP by LFKN, IP=PSPACE by Shamir, Simpler proof by Shen

References

[GMR ‘89] S. Goldwasser, S. Micali, and C. Rackoff. 1989. The knowledge complexity of interactive proof systems. SIAM J. Comput. 18, 1 (Feb. 1989), 186–208. https://doi.org/10.1137/0218012

[GMW ‘91] Oded Goldreich, Silvio Micali, and Avi Wigderson. 1991. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. J. ACM 38, 3 (July 1991), 690–728. https://doi.org/10.1145/116825.116852

[Babai ‘85] L Babai. 1985. Trading group theory for randomness. In Proceedings of the seventeenth annual ACM symposium on Theory of computing (STOC '85). Association for Computing Machinery, New York, NY, USA, 421–429. https://doi.org/10.1145/22145.22192

[GS ‘86] S Goldwasser and M Sipser. 1986. Private coins versus public coins in interactive proof systems. In Proceedings of the eighteenth annual ACM symposium on Theory of computing (STOC '86). Association for Computing Machinery, New York, NY, USA, 59–68. https://doi.org/10.1145/12130.12137

[BM ‘88] László Babai and Shlomo Moran. 1988. Arthur-Merlin games: a randomized proof system, and a hierarchy of complexity class. J. Comput. Syst. Sci. 36, 2 (April 1988), 254–276. https://doi.org/10.1016/0022-0000(88)90028-1

References

[LFKN ‘92] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. 1992. Algebraic methods for interactive proof systems. J. ACM 39, 4 (Oct. 1992), 859–868. https://doi.org/10.1145/146585.146605

[Shamir ‘92] Adi Shamir. 1992. IP = PSPACE. J. ACM 39, 4 (Oct. 1992), 869–877. https://doi.org/10.1145/146585.146609

[Shen ‘92] A. Shen. 1992. IP = SPACE: simplified proof. J. ACM 39, 4 (Oct. 1992), 878–880. https://doi.org/10.1145/146585.146613

[BGKW ‘88] Michael Ben-Or, Shafi Goldwasser, Joe Kilian, and Avi Wigderson. 1988. Multi-prover interactive proofs: how to remove intractability assumptions. In Proceedings of the twentieth annual ACM symposium on Theory of computing (STOC '88). Association for Computing Machinery, New York, NY, USA, 113–131. https://doi.org/10.1145/62212.62223

[BFL ‘91] Babai, L., Fortnow, L. & Lund, C. Non-deterministic exponential time has two-prover interactive protocols. Comput Complexity 1, 3–40 (1991). https://doi.org/10.1007/BF01200056

References

[Wat ‘99] J. Watrous. PSPACE has constant-round quantum interactive proof systems, Proceedings of IEEE FOCS'99, pp. 112-119, 1999. arXiv:cs.CC/9901015

[JJUW ‘09] R. Jain, Z. Ji, S. Upadhyay, and J. Watrous. QIP = PSPACE, J. ACM 58(6):1-27, 2011. doi:10.1145/2049697.2049704 arXiv:0907.4737.

[CHT ‘04] Cleve, R., Hoyer, P., Toner, B., & Watrous, J. (2004). Consequences and Limits of Nonlocal Strategies. arXiv. https://doi.org/10.48550/arXiv.quant-ph/0404076

[JNVWY ‘20] Ji, Z., Natarajan, A., Vidick, T., Wright, J., & Yuen, H. (2020). MIP*=RE. arXiv. https://doi.org/10.48550/arXiv.2001.04383

Additional References

- Introduction to the Theory of Computation, by Michael Sipser
- Computational Complexity: A Modern Approach, by Sanjeev Arora and Boaz Barak. https://theory.cs.princeton.edu/complexity/book.pdf

TL; DR

- Randomness+Interaction is the key, alone they are “weak”
- Supreme power is useless unless succinct
- Mapping to polynomials is a very powerful technique