Interactive Proof Systems

IPs, AMs & PCPs

Antonis Antonopoulos

Theoretical Computer Science II: Structural Complexity

Computation and Reasoning Laboratory
National Technical University of Athens

March 2012

Introduction

"Maybe Fermat had a proof! But an important party was certainly missing to make the proof complete: the verifier. Each time rumor gets around that a student somewhere proved **P** = **NP**, *people ask "Has Karp seen the proof?" (they hardly even ask the student's name). Perhaps the verifier is most important that the prover." (from [BM88])*

- The notion of a mathematical proof is related to the certificate definition of **NP**.
- We enrich this scenario by introducing **interaction** in the basic scheme:
  The person (or TM) who verifies the proof asks the person who provides the proof a series of "queries", before he is convinced, and if he is, he provide the certificate.

Introduction

- The first person will be called **Verifier**, and the second **Prover**.
- In our model of computation, Prover and Verifier are interacting Turing Machines.
- We will categorize the various proof systems created by using:
  - various TMs (nondeterministic, probabilistic etc)
  - the information exchanged (private/public coins etc)
  - the number of TMs (IPs, MIPs,...)

Warmup: Interactive Proofs with deterministic Verifier

Definition (Deterministic Proof Systems)

We say that a language L has a k-round deterministic interactive proof system if there is a deterministic Turing Machine V that on input $x, \alpha_1, \alpha_2, \ldots, \alpha_i$ runs in time polynomial in $|x|$, and can have a k-round interaction with any TM P such that:

- $x \in L \Rightarrow \exists P: \langle V, P\rangle(x) = 1$ (*Completeness*)
- $x \notin L \Rightarrow \forall P: \langle V, P\rangle(x) = 0$ (*Soundness*)

The class **dIP** contains all languages that have a k-round deterministic interactive proof system, where p is polynomial in the input length.

- $\langle V, P\rangle(x)$ denotes the output of V at the end of the interaction with P on input x, and α_i the exchanged strings.
- The above definition does not place limits on the computational power of the Prover!

Warmup: Interactive Proofs with deterministic Verifier

- But...

Theorem

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

Proof: Trivially, $\mathbf{NP} \subseteq \mathbf{dIP}$. ✓

Let $L \in \mathbf{dIP}$:

- A certificate is a transcript $(\alpha_1, \ldots, \alpha_k)$ causing V to accept, i.e. $V(x, \alpha_1, \ldots, \alpha_k) = 1$.
- We can efficiently check if $V(x) = \alpha_1$, $V(x, \alpha_1, \alpha_2) = \alpha_3$ etc...
  - If $x \in L$ such a transcript exists!
  - Conversely, if a transcript exists, we can define define a proper P to satisfy: $P(x, \alpha_1) = \alpha_2$, $P(x, \alpha_1, \alpha_2, \alpha_3) = \alpha_4$ etc., so that $\langle V, P\rangle(x) = 1$, so $x \in L$.
- So $L \in \mathbf{NP}$! $\square$

Probabilistic Verifier: The Class IP

- We saw that if the verifier is a simple deterministic TM, then the interactive proof system is described precisely by the class **NP**.
- Now, we let the *verifier* be probabilistic, i.e. the verifier's queries will be computed using a probabilistic TM:

Definition (Goldwasser-Micali-Rackoff)

For an integer $k \geq 1$ (that may depend on the input length), a language L is in $\mathbf{IP}[k]$ if there is a probabilistic polynomial-time T.M. V that can have a k-round interaction with a T.M. P such that:

- $x \in L \Rightarrow \exists P : Pr[\langle V, P\rangle(x) = 1] \geq \frac{2}{3}$ (*Completeness*)
- $x \notin L \Rightarrow \forall P : Pr[\langle V, P\rangle(x) = 1] \leq \frac{1}{3}$ (*Soundness*)

Probabilistic Verifier: The Class IP

Definition

We also define:

$$\mathbf{IP} = \bigcup_{c \in \mathbb{N}} \mathbf{IP}[n^c]$$

- The "output" $\langle V, P \rangle(x)$ is a random variable.
- We'll see that **IP** is a very large class! ($\supseteq$ **PH**)
- As usual, we can replace the completeness parameter $2/3$ with $1 - 2^{-n^s}$ and the soundness parameter $1/3$ by 2^{-n^s}, without changing the class for any fixed constant $s > 0$.
- We can also replace the completeness constant $2/3$ with 1 (perfect completeness), without changing the class, but replacing the soundness constant $1/3$ with 0, is equivalent with a *deterministic verifier*, so class **IP** collapses to **NP**.

Interactive Proof for Graph Non-Isomorphism

Definition

Two graphs G_1 and G_2 are *isomorphic*, if there exists a permutation π of the labels of the nodes of G_1, such that $\pi(G_1) = G_2$. If G_1 and G_2 are isomorphic, we write $G_1 \cong G_2$.

- $\texttt{GI}$: Given two graphs G_1, G_2, decide if they are isomorphic.
- $\texttt{GNI}$: Given two graphs G_1, G_2, decide if they are *not* isomorphic.

- Obviously, $\texttt{GI} \in \mathbf{NP}$ and $\texttt{GNI} \in co\mathbf{NP}$.
- This proof system relies on the Verifier's access to a *private* random source which cannot be seen by the Prover, so we confirm the crucial role the private coins play.

Interactive Proof for Graph Non-Isomorphism

Verifier: Picks $i \in \{1, 2\}$ uniformly at random.
Then, it permutes randomly the vertices of G_i to get a new graph H. Is sends H to the Prover.
Prover: Identifies which of G_1, G_2 was used to produce H.
Let G_j be the graph. Sends j to V.
Verifier: Accept if $i = j$. Reject otherwise.

Interactive Proof for Graph Non-Isomorphism

Verifier: Picks $i \in \{1, 2\}$ uniformly at random.
Then, it permutes randomly the vertices of G_i to get a new graph H. Is sends H to the Prover.
Prover: Identifies which of G_1, G_2 was used to produce H. Let G_j be the graph. Sends j to V.
Verifier: Accept if $i = j$. Reject otherwise.

- If $G_1 \ncong G_2$, then the powerfull prover can (nondeterministivally) guess which one of the two graphs is isomprphic to H, and so the Verifier accepts with probability 1.
- If $G_1 \cong G_2$, the prover can't distinguish the two graphs, since a random permutation of G_1 looks exactly like a random permutation of G_2. So, the best he can do is guess randomly one, and the Verifier accepts with probability (at most) $1/2$, which can be reduced by additional repetitions.

Babai's Arthur-Merlin Games

Definition (Extended (FGMSZ89))

An Arhur-Merlin Game is a pair of interactive TMs A and M, and a predicate R such that:

- On input x, exactly $2q(|x|)$ messages of length $m(|x|)$ are exchanged, $q, m \in poly(|x|)$.
- A goes first, and at iteration $1 \leq i \leq q(|x|)$ chooses u.a.r. a string r_i of length $m(|x|)$.
- M's reply in the i^{th} iteration is $y_i = M(x, r_1, \ldots, r_i)$ (M's strategy).
- For every M', a **conversation** between A and M' on input x is $r_1 y_1 r_2 y_2 \cdots r_{q(|x|)} y_{q(|x|)}$.
- The set of all conversations is denoted by $CONV_x^{M'}$, $|CONV_x^{M'}| = 2^{q(|x|)m(|x|)}$.

Babai's Arthur-Merlin Games

Definition (*cont'd*)

- The predicate R maps the input x and a conversation to a Boolean value.
- The set of accepting conversations is denoted by $ACC_x^{R,M}$, and is the set:

$$\{r_1 \cdots r_q | \exists y_1 \cdots y_q \ s.t. \ r_1 y_1 \cdots r_q y_q \in CONV_x^M \wedge R(r_1 y_1 \cdots r_q y_q) = 1\}$$

- A language L has an Arthur-Merlin proof system if:
  - **There exists** a strategy for M, such that for all $x \in L$: $\frac{ACC_x^{R,M}}{CONV_x^M} \geq \frac{2}{3}$ (*Completeness*)
  - **For every** strategy for M, and for every $x \notin L$: $\frac{ACC_x^{R,M}}{CONV_x^M} \leq \frac{1}{3}$ (*Soundness*)

Definitions

- So, with respect to the previous **IP** definition:

Definition

For every k, the complexity class $\mathbf{AM}[k]$ is defined as a subset to $\mathbf{IP}[k]$ obtained when we restrict the verifier's messages to be *random bits*, and not allowing it to use any other random bits that are not contained in these messages.
We denote $\mathbf{AM} \equiv \mathbf{AM}[2]$.

Definitions

- So, with respect to the previous **IP** definition:

Definition

For every k, the complexity class $\mathbf{AM}[k]$ is defined as a subset to $\mathbf{IP}[k]$ obtained when we restrict the verifier's messages to be *random bits*, and not allowing it to use any other random bits that are not contained in these messages.
We denote $\mathbf{AM} \equiv \mathbf{AM}[2]$.

- **Merlin** $\rightarrow$ **Prover**
- **Arthur** $\rightarrow$ **Verifier**

Definitions

- So, with respect to the previous **IP** definition:

Definition

For every k, the complexity class $\mathbf{AM}[k]$ is defined as a subset to $\mathbf{IP}[k]$ obtained when we restrict the verifier's messages to be *random bits*, and not allowing it to use any other random bits that are not contained in these messages.
We denote $\mathbf{AM} \equiv \mathbf{AM}[2]$.

- **Merlin** $\rightarrow$ **Prover**
- **Arthur** $\rightarrow$ **Verifier**
- Also, the class **MA** consists of all languages L, where there's an interactive proof for L in which the prover first sending a message, and then the verifier is "tossing coins" and computing its decision by doing a deterministic polynomial-time computation involving the input, the message and the random output.

Public vs. Private Coins

Theorem

$$\texttt{GNI} \in \mathbf{AM}[2]$$

Theorem

For every $p \in poly(n)$:

$$\mathbf{IP}\,(p(n)) = \mathbf{AM}(p(n) + 2)$$

- So,

$$\mathbf{IP}[poly] = \mathbf{AM}[poly]$$

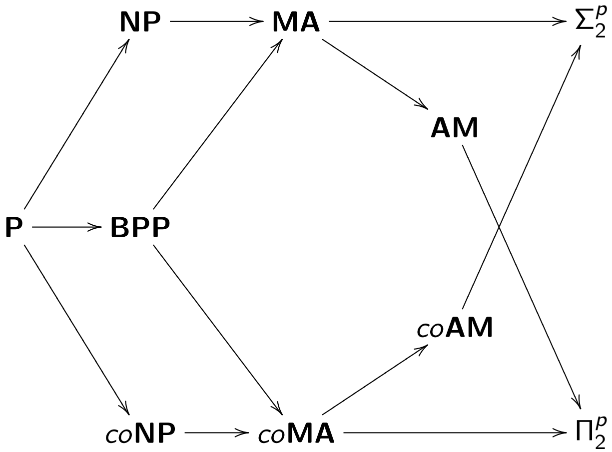
Properties of Arthur-Merlin Games

- $\mathbf{MA} \subseteq \mathbf{AM}$
- $\mathbf{MA}[1] = \mathbf{NP}$, $\mathbf{AM}[1] = \mathbf{BPP}$
- **AM** could be intuitively approached as the probabilistic version of **NP** (usually denoted as $\mathbf{AM} = \mathcal{BP} \cdot \mathbf{NP}$).
- $\mathbf{AM} \subseteq \Pi_2^p$ and $\mathbf{MA} \subseteq \Sigma_2^p \cap \Pi_2^p$.
- $\mathbf{NP}^{\mathbf{BPP}} \subseteq \mathbf{MA}$, $\mathbf{MA}^{\mathbf{BPP}} = \mathbf{MA}$, $\mathbf{AM}^{\mathbf{BPP}} = \mathbf{AM}$ and $\mathbf{AM}^{\Delta\Sigma_1^p} = \mathbf{AM}^{\mathbf{NP} \cap co\mathbf{NP}} = \mathbf{AM}$
- If we consider the complexity classes $\mathbf{AM}[k]$ (the languages that have Arthur-Merlin proof systems of a bounded number of rounds, they form an hierarchy:

$$\mathbf{AM}[0] \subseteq \mathbf{AM}[1] \subseteq \cdots \subseteq \mathbf{AM}[k] \subseteq \mathbf{AM}[k+1] \subseteq \cdots$$

- Are these inclusions proper ? ? ?

Properties of Arthur-Merlin Games



Properties of Arthur-Merlin Games

- Proper formalism (*Zachos et al.*):

Definition (Majority Quantifier)

Let $R : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}$ be a predicate, and ε a rational number, such that $\varepsilon \in \left(0, \frac{1}{2}\right)$. We denote by $(\exists^+ y, |y| = k) R(x, y)$ the following predicate:

> *"There exist at least $\left(\frac{1}{2} + \varepsilon\right) \cdot 2^k$ strings y of length m for which $R(x, y)$ holds."*

We call $\exists^+$ the *overwhelming majority* quantifier.

- $\exists^+_r$ means that the fraction r of the possible certificates of a certain length satisfy the predicate for the certain input.
- Obviously, $\exists^+ = \exists^+_{1/2+\varepsilon} = \exists^+_{2/3} = \exists^+_{3/4} = \exists^+_{0.99} = \exists^+_{1-2^{-p(|x|)}}$

Properties of Arthur-Merlin Games

Definition

We denote as $\mathcal{C} = (Q_1/Q_2)$, where $Q_1, Q_2 \in \{\exists, \forall, \exists^+\}$, the class $\mathcal{C}$ of languages L satisfying:

- $x \in L \Rightarrow Q_1 y \ R(x, y)$
- $x \notin L \Rightarrow Q_2 y \ \neg R(x, y)$

- So: $\mathbf{P} = (\forall/\forall)$, $\mathbf{NP} = (\exists/\forall)$, $co\mathbf{NP} = (\forall/\exists)$
  $\mathbf{BPP} = (\exists^+/\exists^+)$, $\mathbf{RP} = (\exists^+/\forall)$, $co\mathbf{RP} = (\forall/\exists^+)$

Properties of Arthur-Merlin Games

Definition

We denote as $\mathcal{C} = (Q_1/Q_2)$, where $Q_1, Q_2 \in \{\exists, \forall, \exists^+\}$, the class $\mathcal{C}$ of languages L satisfying:

- $x \in L \Rightarrow Q_1 y \ R(x,y)$
- $x \notin L \Rightarrow Q_2 y \ \neg R(x,y)$

- So: $\mathbf{P} = (\forall/\forall)$, $\mathbf{NP} = (\exists/\forall)$, $co\mathbf{NP} = (\forall/\exists)$
  $\mathbf{BPP} = (\exists^+/\exists^+)$, $\mathbf{RP} = (\exists^+/\forall)$, $co\mathbf{RP} = (\forall/\exists^+)$

Arthur-Merlin Games

$$\mathbf{AM} = \mathbf{BP} \cdot \mathbf{NP} = (\exists^+\exists/\exists^+\forall)$$

$$\mathbf{MA} = \mathbf{N} \cdot \mathbf{BPP} = (\exists\exists^+/\forall\exists^+)$$

- Similarly: $\mathbf{AMA} = (\exists^+\exists\exists^+/\exists^+\forall\exists^+)$ etc.

Properties of Arthur-Merlin Games

Theorem

1. $\mathbf{MA} = (\exists\forall/\forall\exists^+)$
2. $\mathbf{AM} = (\forall\exists/\exists^+\forall)$

Proof:

Lemma

- $\mathbf{BPP} = (\exists^+/\exists^+) = (\exists^+\forall/\forall\exists^+) = (\forall\exists^+/\exists^+\forall)$ **(1)** (BPP-Theorem)
- $(\exists\forall/\forall\exists^+) \subseteq (\forall\exists/\exists^+\forall)$ **(2)**

i) $\mathbf{MA} = \mathbf{N}\cdot\mathbf{BPP} = (\exists\exists^+/\forall\exists^+) \overset{(1)}{=} (\exists\exists^+\forall/\forall\forall\exists^+) \subseteq (\exists\forall/\forall\exists^+)$ (*the last inclusion holds by quantifier contraction*). Also, $(\exists\forall/\forall\exists^+) \subseteq (\exists\exists^+/\forall\exists^+) = \mathbf{MA}$.

ii) Similarly,
$\mathbf{AM} = \mathbf{BP}\cdot\mathbf{NP} = (\exists^+\exists/\exists^+\forall) = (\forall\exists^+\exists/\exists^+\forall\forall) \subseteq (\forall\exists/\exists^+\forall)$.
Also, $(\forall\exists/\exists^+\forall) \subseteq (\exists^+\exists/\exists^+\forall) = \mathbf{AM}$.

Properties of Arthur-Merlin Games

Theorem

1. $\mathbf{MA} = (\exists\forall/\forall\exists^+)$
2. $\mathbf{AM} = (\forall\exists/\exists^+\forall)$

Proof:

Lemma

- $\mathbf{BPP} = (\exists^+/\exists^+) = (\exists^+\forall/\forall\exists^+) = (\forall\exists^+/\exists^+\forall)$ **(1)** (BPP-Theorem)
- $(\exists\forall/\forall\exists^+) \subseteq (\forall\exists/\exists^+\forall)$ **(2)**

i) $\mathbf{MA} = \mathbf{N}\cdot\mathbf{BPP} = (\exists\exists^+/\forall\exists^+) \overset{(1)}{=} (\exists\exists^+\forall/\forall\forall\exists^+) \subseteq (\exists\forall/\forall\exists^+)$ (*the last inclusion holds by quantifier contraction*). Also, $(\exists\forall/\forall\exists^+) \subseteq (\exists\exists^+/\forall\exists^+) = \mathbf{MA}$.

ii) Similarly, $\mathbf{AM} = \mathbf{BP}\cdot\mathbf{NP} = (\exists^+\exists/\exists^+\forall) = (\forall\exists^+\exists/\exists^+\forall\forall) \subseteq (\forall\exists/\exists^+\forall)$. Also, $(\forall\exists/\exists^+\forall) \subseteq (\exists^+\exists/\exists^+\forall) = \mathbf{AM}$.

Properties of Arthur-Merlin Games

Theorem

1. **MA** $= (\exists\forall/\forall\exists^+)$
2. **AM** $= (\forall\exists/\exists^+\forall)$

Proof:

Lemma

- **BPP** $= (\exists^+/\exists^+) = (\exists^+\forall/\forall\exists^+) = (\forall\exists^+/\exists^+\forall)$ **(1)** (BPP-Theorem)
- $(\exists\forall/\forall\exists^+) \subseteq (\forall\exists/\exists^+\forall)$ **(2)**

i) **MA** $=$ **N** $\cdot$ **BPP** $= (\exists\exists^+/\forall\exists^+) \overset{(1)}{=} (\exists\exists^+\forall/\forall\forall\exists^+) \subseteq (\exists\forall/\forall\exists^+)$
(*the last inclusion holds by quantifier contraction*). Also,
$(\exists\forall/\forall\exists^+) \subseteq (\exists\exists^+/\forall\exists^+) =$ **MA**.
ii) Similarly,
AM $=$ **BP** $\cdot$ **NP** $= (\exists^+\exists/\exists^+\forall) = (\forall\exists^+\exists/\exists^+\forall\forall) \subseteq (\forall\exists/\exists^+\forall)$.
Also, $(\forall\exists/\exists^+\forall) \subseteq (\exists^+\exists/\exists^+\forall) =$ **AM**.

Properties of Arthur-Merlin Games

Theorem

1. $\mathbf{MA} = (\exists\forall / \forall\exists^+)$
2. $\mathbf{AM} = (\forall\exists / \exists^+\forall)$

Proof:

Lemma

- $\mathbf{BPP} = (\exists^+ / \exists^+) = (\exists^+\forall / \forall\exists^+) = (\forall\exists^+ / \exists^+\forall)$ (1) (BPP-Theorem)
- $(\exists\forall / \forall\exists^+) \subseteq (\forall\exists / \exists^+\forall)$ (2)

i) $\mathbf{MA} = \mathbf{N} \cdot \mathbf{BPP} = (\exists\exists^+ / \forall\exists^+) \overset{(1)}{=} (\exists\exists^+\forall / \forall\forall\exists^+) \subseteq (\exists\forall / \forall\exists^+)$
(*the last inclusion holds by quantifier contraction*). Also,
$(\exists\forall / \forall\exists^+) \subseteq (\exists\exists^+ / \forall\exists^+) = \mathbf{MA}$.
ii) Similarly,
$\mathbf{AM} = \mathbf{BP} \cdot \mathbf{NP} = (\exists^+\exists / \exists^+\forall) = (\forall\exists^+\exists / \exists^+\forall\forall) \subseteq (\forall\exists / \exists^+\forall)$.
Also, $(\forall\exists / \exists^+\forall) \subseteq (\exists^+\exists / \exists^+\forall) = \mathbf{AM}$.

Properties of Arthur-Merlin Games

Theorem

$$\mathbf{MA} \subseteq \mathbf{AM}$$

Proof:
Obvious from (2): $(\exists\forall/\forall\exists^+) \subseteq (\forall\exists/\exists^+\forall)$. $\square$

Theorem

1. $\mathbf{AM} \subseteq \Pi_2^p$
2. $\mathbf{MA} \subseteq \Sigma_2^p \cap \Pi_2^p$

Proof:
i) $\mathbf{AM} = (\forall\exists/\exists^+\forall) \subseteq (\forall\exists/\exists\forall) = \Pi_2^p$
ii) $\mathbf{MA} = (\exists\forall/\forall\exists^+) \subseteq (\exists\forall/\forall\exists) = \Sigma_2^p$, and
$\mathbf{MA} \subseteq \mathbf{AM} \Rightarrow \mathbf{MA} \subseteq \Pi_2^p$. So, $\mathbf{MA} \subseteq \Sigma_2^p \cap \Pi_2^p$. $\square$

Properties of Arthur-Merlin Games

Theorem (Speedup Theorem)

For $t(n) \geq 2$:

$$\mathbf{AM}[2t(n)] = \mathbf{AM}[t(n)]$$

- The Arthur-Merlin Hierarchy collapses at its second level:

Theorem (Collapse Theorem)

For every $k \geq 2$:

$$\mathbf{AM} = \mathbf{AM}[k] = \mathbf{MA}[k+1]$$

Example

$$\mathbf{MAM} = (\exists\exists^+\exists/\forall\exists^+\forall) \overset{(1)}{\subseteq} (\exists\exists^+\forall\exists/\forall\forall\exists^+\forall) \subseteq (\exists\forall\exists/\forall\exists^+\forall) \overset{(2)}{\subseteq}$$
$$\subseteq (\forall\exists\exists/\exists^+\forall\forall) \subseteq (\forall\exists/\exists^+\forall) = \mathbf{AM}$$

Properties of Arthur-Merlin Games

Theorem (Speedup Theorem)

For $t(n) \geq 2$:

$$\mathbf{AM}[2t(n)] = \mathbf{AM}[t(n)]$$

- The Arthur-Merlin Hierarchy collapses at its second level:

Theorem (Collapse Theorem)

For every $k \geq 2$:

$$\mathbf{AM} = \mathbf{AM}[k] = \mathbf{MA}[k+1]$$

Example

$$\mathbf{MAM} = (\exists\exists^+\exists/\forall\exists^+\forall) \overset{(1)}{\subseteq} (\exists\exists^+\forall\exists/\forall\forall\exists^+\forall) \subseteq (\exists\forall\exists/\forall\exists^+\forall) \overset{(2)}{\subseteq}$$
$$\subseteq (\forall\exists\exists/\exists^+\forall\forall) \subseteq (\forall\exists/\exists^+\forall) = \mathbf{AM}$$

Properties of Arthur-Merlin Games

Theorem (Speedup Theorem)

For $t(n) \geq 2$:

$$\mathbf{AM}[2t(n)] = \mathbf{AM}[t(n)]$$

- The Arthur-Merlin Hierarchy collapses at its second level:

Theorem (Collapse Theorem)

For every $k \geq 2$:

$$\mathbf{AM} = \mathbf{AM}[k] = \mathbf{MA}[k+1]$$

Example

$$\mathbf{MAM} = (\exists\exists^{+}\exists/\forall\exists^{+}\forall) \overset{(1)}{\subseteq} (\exists\exists^{+}\forall\exists/\forall\forall\exists^{+}\forall) \subseteq (\exists\forall\exists/\forall\exists^{+}\forall) \overset{(2)}{\subseteq}$$
$$\subseteq (\forall\exists\exists/\exists^{+}\forall\forall) \subseteq (\forall\exists/\exists^{+}\forall) = \mathbf{AM}$$

Properties of Arthur-Merlin Games

Theorem (Speedup Theorem)

For $t(n) \geq 2$:

$$\mathbf{AM}[2t(n)] = \mathbf{AM}[t(n)]$$

- The Arthur-Merlin Hierarchy collapses at its second level:

Theorem (Collapse Theorem)

For every $k \geq 2$:

$$\mathbf{AM} = \mathbf{AM}[k] = \mathbf{MA}[k+1]$$

Example

$$\mathbf{MAM} = (\exists\exists^{+}\exists/\forall\exists^{+}\forall) \overset{(1)}{\subseteq} (\exists\exists^{+}\forall\exists/\forall\forall\exists^{+}\forall) \subseteq (\exists\forall\exists/\forall\exists^{+}\forall) \overset{(2)}{\subseteq}$$
$$\subseteq (\forall\exists\exists/\exists^{+}\forall\forall) \subseteq (\forall\exists/\exists^{+}\forall) = \mathbf{AM}$$

Properties of Arthur-Merlin Games

Theorem (Speedup Theorem)

For $t(n) \geq 2$:

$$\mathbf{AM}[2t(n)] = \mathbf{AM}[t(n)]$$

- The Arthur-Merlin Hierarchy collapses at its second level:

Theorem (Collapse Theorem)

For every $k \geq 2$:

$$\mathbf{AM} = \mathbf{AM}[k] = \mathbf{MA}[k+1]$$

Example

$$\mathbf{MAM} = (\exists\exists^+\exists/\forall\exists^+\forall) \overset{(1)}{\subseteq} (\exists\exists^+\forall\exists/\forall\forall\exists^+\forall) \subseteq (\exists\forall\exists/\forall\exists^+\forall) \overset{(2)}{\subseteq}$$
$$\subseteq (\forall\exists\exists/\exists^+\forall\forall) \subseteq (\forall\exists/\exists^+\forall) = \mathbf{AM}$$

Properties of Arthur-Merlin Games

Theorem (Speedup Theorem)

For $t(n) \geq 2$:

$$\mathbf{AM}[2t(n)] = \mathbf{AM}[t(n)]$$

- The Arthur-Merlin Hierarchy collapses at its second level:

Theorem (Collapse Theorem)

For every $k \geq 2$:

$$\mathbf{AM} = \mathbf{AM}[k] = \mathbf{MA}[k+1]$$

Example

$$\mathbf{MAM} = (\exists\exists^+\exists/\forall\exists^+\forall) \overset{(1)}{\subseteq} (\exists\exists^+\forall\exists/\forall\forall\exists^+\forall) \subseteq (\exists\forall\exists/\forall\exists^+\forall) \overset{(2)}{\subseteq}$$
$$\subseteq (\forall\exists\exists/\exists^+\forall\forall) \subseteq (\forall\exists/\exists^+\forall) = \mathbf{AM}$$

Properties of Arthur-Merlin Games

Proof:

- The general case is implied by the generalization of BPP-Theorem (1) & (2):
- $(\mathbf{Q_1}\exists^+\mathbf{Q_2}/\mathbf{Q_3}\exists^+\mathbf{Q_4}) = (\mathbf{Q_1}\exists^+\forall\mathbf{Q_2}/\mathbf{Q_3}\forall\exists^+\mathbf{Q_4}) = (\mathbf{Q_1}\forall\exists^+\mathbf{Q_2}/\mathbf{Q_3}\exists^+\forall\mathbf{Q_4})$ $(\mathbf{1'})$
- $(\mathbf{Q_1}\exists\forall\mathbf{Q_2}/\mathbf{Q_3}\forall\exists^+\mathbf{Q_4}) \subseteq (\mathbf{Q_1}\forall\exists\mathbf{Q_2}/\mathbf{Q_3}\exists^+\forall\mathbf{Q_4})$ $(\mathbf{2'})$
- Using the above we can easily see that the Arthur-Merlin Hierarchy collapses at the second level. (*Try it!*) $\square$

Properties of Arthur-Merlin Games

Theorem (BHZ)

*If co***NP** $\subseteq$ **AM** *(that is, if* $\mathtt{GI}$ *is* **NP***-complete), then the Polynomial Hierarchy collapses at the second level, and* **PH** $= \Sigma_2^p =$ **AM***.*

Proof: Our hypothesis states: $(\forall/\exists) \subseteq (\forall\exists/\exists^+\forall)$

Then:

$\Sigma_2^p = (\exists\forall/\forall\exists) \overset{Hyp.}{\subseteq} (\exists\forall\exists/\forall\exists^+\forall) \overset{(2)}{\subseteq} (\forall\exists\exists/\exists^+\forall\forall) = (\forall\exists/\exists^+\forall) =$ **AM** $\subseteq (\forall\exists/\exists\forall) = \Pi_2^p$. $\square$

Properties of Arthur-Merlin Games

Theorem (BHZ)

*If co***NP** $\subseteq$ **AM** *(that is, if $\mathtt{GI}$ is* **NP***-complete), then the Polynomial Hierarchy collapses at the second level, and* **PH** $= \Sigma_2^p =$ **AM**.

Proof: Our hypothesis states: $(\forall/\exists) \subseteq (\forall\exists/\exists^+\forall)$

Then:

$\Sigma_2^p = (\exists\forall/\forall\exists) \overset{Hyp.}{\subseteq} (\exists\forall\exists/\forall\exists^+\forall) \overset{(2)}{\subseteq} (\forall\exists\exists/\exists^+\forall\forall) = (\forall\exists/\exists^+\forall) =$ **AM** $\subseteq (\forall\exists/\exists\forall) = \Pi_2^p$. $\square$

Properties of Arthur-Merlin Games

Theorem (BHZ)

*If co***NP** $\subseteq$ **AM** *(that is, if* $\mathtt{GI}$ *is* **NP***-complete), then the Polynomial Hierarchy collapses at the second level, and* **PH** $= \Sigma_2^p =$ **AM**.

Proof: Our hypothesis states: $(\forall/\exists) \subseteq (\forall\exists/\exists^+\forall)$

Then:

$\Sigma_2^p = (\exists\forall/\forall\exists) \overset{Hyp.}{\subseteq} (\exists\forall\exists/\forall\exists^+\forall) \overset{(2)}{\subseteq} (\forall\exists\exists/\exists^+\forall\forall) = (\forall\exists/\exists^+\forall) =$ **AM** $\subseteq (\forall\exists/\exists\forall) = \Pi_2^p$. $\square$

Properties of Arthur-Merlin Games

Theorem (BHZ)

*If co***NP** $\subseteq$ **AM** *(that is, if* $\mathtt{GI}$ *is* **NP***-complete), then the Polynomial Hierarchy collapses at the second level, and* **PH** $= \Sigma_2^p =$ **AM**.

Proof: Our hypothesis states: $(\forall/\exists) \subseteq (\forall\exists/\exists^+\forall)$
Then:

$$\Sigma_2^p = (\exists\forall/\forall\exists) \overset{Hyp.}{\subseteq} (\exists\forall\exists/\forall\exists^+\forall) \overset{(2)}{\subseteq} (\forall\exists\exists/\exists^+\forall\forall) = (\forall\exists/\exists^+\forall) = \mathbf{AM} \subseteq (\forall\exists/\exists\forall) = \Pi_2^p. \ \square$$

Properties of Arthur-Merlin Games

Theorem (BHZ)

*If co***NP** $\subseteq$ **AM** *(that is, if* $\mathtt{GI}$ *is* **NP***-complete), then the Polynomial Hierarchy collapses at the second level, and* **PH** $= \Sigma_2^p =$ **AM**.

Proof: Our hypothesis states: $(\forall/\exists) \subseteq (\forall\exists/\exists^+\forall)$

Then:

$$\Sigma_2^p = (\exists\forall/\forall\exists) \overset{Hyp.}{\subseteq} (\exists\forall\exists/\forall\exists^+\forall) \overset{(2)}{\subseteq} (\forall\exists\exists/\exists^+\forall\forall) = (\forall\exists/\exists^+\forall) = \mathbf{AM} \subseteq (\forall\exists/\exists\forall) = \Pi_2^p. \ \square$$

Measure One Results

- $\mathbf{P}^A \neq \mathbf{NP}^A$, for almost all oracles A.
- $\mathbf{P}^A = \mathbf{BPP}^A$, for almost all oracles A.
- $\mathbf{NP}^A = \mathbf{AM}^A$, for almost all oracles A.

Definition

$$almost\mathcal{C} = \left\{ L | \mathbf{Pr}_{A \in \{0,1\}^*} \left[L \in \mathcal{C}^A \right] = 1 \right\}$$

Theorem

1. $almost\mathbf{P} = \mathbf{BPP}$ *[BG81]*
2. $almost\mathbf{NP} = \mathbf{AM}$ *[NW94]*
3. $almost\mathbf{PH} = \mathbf{PH}$

Measure One Results

Theorem (Kurtz)

For almost every pair of oracles B, C:

1. $\mathbf{BPP} = \mathbf{P}^B \cap \mathbf{P}^C$
2. $almost\mathbf{NP} = \mathbf{NP}^B \cap \mathbf{NP}^C$

Indicative Open Questions

- Does exist an oracle separating **AM** from *almost***NP**?
- Is *almost***NP** contained in some finite level of Polynomial-Time Hierarchy?
- *Motivated by [BHZ]*: If $co\mathbf{NP} \subseteq almost\mathbf{NP}$, does it follow that **PH** collapses?

The power of Interactive Proofs

- As we saw, **Interaction** alone does not gives us computational capabilities beyond **NP**.
- Also, **Randomization** alone does not give us significant power (we know that **BPP** $\subseteq \Sigma_2^p$, and many researchers believe that **P** $=$ **BPP**, which holds under some plausible assumptions).
- How much power could we get by their *combination*?
- We know that for fixed $k \in \mathbb{N}$, **IP**$[k]$ collapses to

$$\mathbf{IP}[k] = \mathbf{AM} = \mathcal{BP} \cdot \mathbf{NP}$$

  a class that is "close" to **NP** (under similar assumptions, the non-deterministic analogue of **P** vs. **BPP** is **NP** vs. **AM**.)
- If we let k be a polynomial in the size of the input, how much more power could we get?

The power of Interactive Proofs

- Surprisingly:

Theorem (L.F.K.N. & Shamir)

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

The power of Interactive Proofs

Lemma 1

$$\mathbf{IP} \subseteq \mathbf{PSPACE}$$

Warmup: Interactive Proof for UNSAT

Lemma 2

$$\mathbf{PSPACE} \subseteq \mathbf{IP}$$

- For simplicity, we will construct an Interactive Proof for UNSAT (a *co*$\mathbf{NP}$-complete problem), showing that:

Theorem

$$co\mathbf{NP} \subseteq \mathbf{IP}$$

- Let N be a prime.
- We will translate a **formula** ϕ with m clauses and n variables $x_1, \ldots, x_n$ to a **polynomial** p over the field $(modN)$ (where $N > 2^n \cdot 3^m$), in the following way:

Arithmetization

- Arithmetic generalization of a CNF Boolean Formula.

$$\begin{array}{rcl} \mathtt{T} & \longrightarrow & 1 \\ \mathtt{F} & \longrightarrow & 0 \\ \neg x & \longrightarrow & 1 - x \\ \wedge & \longrightarrow & \times \\ \vee & \longrightarrow & + \end{array}$$

Example

$$(x_3 \vee \neg x_5 \vee x_{17}) \wedge (x_5 \vee x_9) \wedge (\neg x_3 \vee x_4)$$
$$\downarrow$$
$$(x_3 + (1 - x_5) + x_{17}) \cdot (x_5 + x_9) \cdot ((1 - x_3) + (1 - x_4))$$

- Each literal is of degree 1, so the polynomial p is of degree at most m.
- Also, $0 < p < 3^m$.

Warmup: Interactive Proof for UNSAT

Prover		**Verifier**
Sends primality proof for N	$\longrightarrow$	checks proof

Warmup: Interactive Proof for UNSAT

Prover		**Verifier**
Sends primality proof for N	$\longrightarrow$	checks proof
$q_1(x) = \sum p(x, x_2, \ldots x_n)$	$\longrightarrow$	checks if $q_1(0) + q_1(1) = 0$

Warmup: Interactive Proof for UNSAT

Prover		**Verifier**
Sends primality proof for N	$\longrightarrow$	checks proof
$q_1(x) = \sum p(x, x_2, \ldots x_n)$	$\longrightarrow$	checks if $q_1(0) + q_1(1) = 0$
	$\longleftarrow$	sends $r_1 \in \{0, \ldots, N-1\}$

Warmup: Interactive Proof for UNSAT

Prover		**Verifier**
Sends primality proof for N	$\longrightarrow$	checks proof
$q_1(x) = \sum p(x, x_2, \ldots x_n)$	$\longrightarrow$	checks if $q_1(0) + q_1(1) = 0$
	$\longleftarrow$	sends $r_1 \in \{0, \ldots, N-1\}$
$q_2(x) = \sum p(r_1, x, x_3, \ldots x_n)$	$\longrightarrow$	checks if $q_2(0) + q_2(1) = q_1(r_1)$

Warmup: Interactive Proof for UNSAT

Prover		**Verifier**
Sends primality proof for N	$\longrightarrow$	checks proof
$q_1(x) = \sum p(x, x_2, \ldots x_n)$	$\longrightarrow$	checks if $q_1(0) + q_1(1) = 0$
	$\longleftarrow$	sends $r_1 \in \{0, \ldots, N-1\}$
$q_2(x) = \sum p(r_1, x, x_3, \ldots x_n)$	$\longrightarrow$	checks if $q_2(0) + q_2(1) = q_1(r_1)$
	$\longleftarrow$	sends $r_2 \in \{0, \ldots, N-1\}$

Warmup: Interactive Proof for UNSAT

Prover		**Verifier**
Sends primality proof for N	$\longrightarrow$	checks proof
$q_1(x) = \sum p(x, x_2, \ldots x_n)$	$\longrightarrow$	checks if $q_1(0) + q_1(1) = 0$
	$\longleftarrow$	sends $r_1 \in \{0, \ldots, N-1\}$
$q_2(x) = \sum p(r_1, x, x_3, \ldots x_n)$	$\longrightarrow$	checks if $q_2(0) + q_2(1) = q_1(r_1)$
	$\longleftarrow$	sends $r_2 \in \{0, \ldots, N-1\}$
	$\vdots$	
$q_n(x) = p(r_1, \ldots, r_{n-1}, x)$	$\longrightarrow$	checks if $q_n(0) + q_n(1) = q_{n-1}(r_{n-1})$

Warmup: Interactive Proof for UNSAT

Prover		**Verifier**
Sends primality proof for N	$\longrightarrow$	checks proof
$q_1(x) = \sum p(x, x_2, \dots x_n)$	$\longrightarrow$	checks if $q_1(0) + q_1(1) = 0$
	$\longleftarrow$	sends $r_1 \in \{0, \dots, N-1\}$
$q_2(x) = \sum p(r_1, x, x_3, \dots x_n)$	$\longrightarrow$	checks if $q_2(0) + q_2(1) = q_1(r_1)$
	$\longleftarrow$	sends $r_2 \in \{0, \dots, N-1\}$
	$\vdots$	
$q_n(x) = p(r_1, \dots, r_{n-1}, x)$	$\longrightarrow$	checks if $q_n(0) + q_n(1) = q_{n-1}(r_{n-1})$ picks $r_n \in \{0, \dots, N-1\}$

Warmup: Interactive Proof for UNSAT

Prover		**Verifier**
Sends primality proof for N	$\longrightarrow$	checks proof
$q_1(x) = \sum p(x, x_2, \ldots x_n)$	$\longrightarrow$	checks if $q_1(0) + q_1(1) = 0$
	$\longleftarrow$	sends $r_1 \in \{0, \ldots, N-1\}$
$q_2(x) = \sum p(r_1, x, x_3, \ldots x_n)$	$\longrightarrow$	checks if $q_2(0) + q_2(1) = q_1(r_1)$
	$\longleftarrow$	sends $r_2 \in \{0, \ldots, N-1\}$
	$\vdots$	
$q_n(x) = p(r_1, \ldots, r_{n-1}, x)$	$\longrightarrow$	checks if $q_n(0) + q_n(1) = q_{n-1}(r_{n-1})$ picks $r_n \in \{0, \ldots, N-1\}$ checks if $q_n(r_n) = p(r_1, \ldots, r_n)$

Warmup: Interactive Proof for UNSAT

- If ϕ is **unsatisfiable**,then

$$\sum_{x_1\in\{0,1\}} \sum_{x_2\in\{0,1\}} \cdots \sum_{x_n\in\{0,1\}} p(x_1,\ldots,x_n) \equiv 0 \ (modN)$$

  and the protocol will succeed.
- Also, the arithmetization can be done in polynomial time, and if we take $N = 2^{\mathcal{O}(n+m)}$, then the elements in the field can be represented by $\mathcal{O}(n+m)$ bits, and thus an evaluation of p in any point of $\{0,\ldots,N-1\}$ can be computed in polynomial time.
- We have to show that if ϕ is satisfiable, then the verifier will **reject** with high probability.
- If ϕ is satisfiable, then $\sum_{x_1\in\{0,1\}} \sum_{x_2\in\{0,1\}} \cdots \sum_{x_n\in\{0,1\}} p(x_1,\ldots,x_n) \neq 0 \ (modN)$

- So, $p_1(01) + p_1(1) \neq 0$, so if the prover send p_1 we 're done.
- If the prover send $q_1 \neq p_1$, then the polynomials will agree on at most m places. So, $\mathbf{Pr}\left[p_1(r_1) \neq q_1(r_1)\right] \geq 1 - \frac{m}{N}$.
- If indeed $p_1(r_1) \neq q_1(r_1)$ and the prover sends $p_2 = q_2$, then the verifier will reject since $q_2(0) + q_2(1) = p_1(r_1) \neq q_1(r_1)$.
- Thus, the prover must send $q_2 \neq p_2$.
- We continue in a similar way: If $q_i \neq p_i$, then with probability at least $1 - \frac{m}{N}$, r_i is such that $q_i(r_i) \neq p_i(r_i)$.
- Then, the prover must send $q_{i+1} \neq p_{i+1}$ in order for the verifier not to reject.
- At the end, if the verifier has not rejected before the last check, $\mathbf{Pr}\left[p_n \neq q_n\right] \geq 1 - (n-1)\frac{m}{N}$.
- If so, with probability at least $1 - \frac{m}{N}$ the verifier will reject since, $q_n(x)$ and $p(r_1, \ldots, r_{n-1}, x)$ differ on at least that fraction of points.
- The total probability that the verifier will accept if at most $\frac{nm}{N}$.

Arithmetization of QBF

$$\exists \longrightarrow \sum$$
$$\forall \longrightarrow \prod$$

Example

$$\forall x_1 \exists x_2 [(x_1 \wedge x_2) \vee \exists x_3 (\bar{x}_2 \wedge x_3)]$$

$$\downarrow$$

$$\prod_{x_1 \in \{0,1\}} \sum_{x_2 \in \{0,1\}} \left[(x_1 \cdot x_2) + \sum_{x_3 \in \{0,1\}} (1 - x_2) \cdot x_3 \right]$$

Theorem

A closed QBF is true if and only if tha value of its arithmetic form is non-zero.

Arithmetization of QBF

- If a QBF is `true`, its value could be quite large:

Theorem

Let A be a closed QBF of size n. Then, the value of its arithmetic form cannot exceed $\mathcal{O}\left(2^{2^n}\right)$.

- Since such numbers cannot be handled by the protocol, we reduce them modulo some -smaller- prime p:

Theorem

Let A be a closed QBF of size n. Then, there exists a prime p of length polynomial in n, such that its arithmetization

$$A' \neq 0 (mod p) \Leftrightarrow A \text{ is true.}$$

Arithmetization of QBF

- A QBF with all the variables quantified is called **closed**, and can be evaluated to either `True` or `False`.
- An **open** QBF with $k > 0$ free variables can be interpreted as a boolean function $\{0,1\}^k \to \{0,1\}$.
- Now, consider the language of all true quantified boolean formulas:

$$\texttt{TQBF} = \{\Phi | \Phi \text{ is a true quantified Boolean formula}\}$$

- It is known that `TQBF` is a **PSPACE**-complete language!
- So, if we have a interactive proof protocol recognizing `TQBF`, then we have a protocol for every **PSPACE** language.

Protocol for TQBF

- Given a quantified formula

$$\Psi = \forall x_1 \exists x_2 \forall x_3 \cdots \exists x_n \; \phi(x_1, \ldots, x_n)$$

we use *arithmetization* to construct the polynomial P_ϕ. Then, $\Psi \in \texttt{TQBF}$ if and only if

$$\prod_{b_1 \in \{0,1\}^*} \sum_{b_2 \in \{0,1\}^*} \prod_{b_3 \in \{0,1\}^*} \cdots \sum_{b_n \in \{0,1\}^*} P_\phi(b_1, \ldots, b_n) \neq 0$$

PRABs

Definition (PRABs)

A Positive Retarded Arithmetic Program with Binary Substitutions (PRAB) is a *sequence* $\{p_1, \ldots, p_t\}$ of "instructions" such that, for every k, one of the following holds:

1. p_k is constant (0 or 1).
2. $p_k = x_i$, for some $i \leq k$.
3. $p_k = 1 - x_i$, for some $i \leq k$.
4. $p_k = p_i + p_j$, for some $i, j \leq k$.
5. $p_k = p_i p_j$, for some i, j, such that $i + j \leq k$.
6. $p_k = p_j(x_i = 0)$ or $p_j(x_i = 1)$, for some $i, j \leq k$.

- Such a program defines a sequence $\tilde{p}_k$ of polynomials in an obvious way!
- We say that P computes $\tilde{p}_t$, the last member of the sequence.

PRABs

- A family $P_1, P_2, \ldots$ of PRABs is **uniform**, if, upon input 1^n, a polynomial-time deterministic TM computes P_n, and the polynomial $\tilde{P}_n$ computed only depends on $x_1, \ldots, x_n$.

Theorem 1 (Characterization of $\#P$)

For a function $f : \{0,1\}^* \to \mathbb{Z}^+$, the following are equivalent:

1. $f \in \#\mathbf{P}$
2. There exists a uniform family of PRABs P_n, such that for every $x \in \{0,1\}^*$,
$$f(x) = \tilde{P}_{|x|}(x)$$

- By $P(x)$ we mean $P(x_1, \ldots, x_n)$, where $x = x_1 x_2 \cdots x_n \in \{0,1\}^n$

Reminder: Operators on Complexity Classes

Let **C** be an arbitrary complexity class.

- $L \in \mathcal{P} \cdot \mathbf{C}$ if there exists $L' \in \mathbf{C}$ and $p \in poly$ such that $\forall x \in \{0,1\}^*$:
  - $x \in L \Rightarrow \exists_{1/2} y \; L'(<x,y>)$
  - $x \notin L \Rightarrow \exists_{1/2} y \; \neg L'(<x,y>)$
- $L \in \mathcal{BP} \cdot \mathbf{C}$ if there exists $L' \in \mathbf{C}$ and $p \in poly$ such that $\forall x \in \{0,1\}^*$:
  - $x \in L \Rightarrow \exists^+ y \; L'(<x,y>)$
  - $x \notin L \Rightarrow \exists^+ y \; \neg L'(<x,y>)$
- $L \in \bigoplus \cdot \mathbf{C}$ if there exists $L' \in \mathbf{C}$ and $p \in poly$ such that $\forall x \in \{0,1\}^*$:
  - $x \in L \Rightarrow \bigoplus y \; L'(<x,y>)$
  - $x \notin L \Rightarrow \bigoplus y \; \neg L'(<x,y>)$

where for every certificate y: $|y| = p(|x|)$, and by $\bigoplus y$ we mean that the number of y's satisfying the condition is **odd**.

Theorem 2

For a fuction $f : \{0,1\}^* \to \{0,1\}$. the following are equivalent:

1. $f \in \mathcal{BP} \cdot \oplus \cdot \mathbf{P}$.
2. There exists a uniform family of PRABs P_n, such that the polynomial $\tilde{P}_n$ computed by P_n has $n + m(n)$ variables for $m \in poly(n)$, and $\forall x \in \{0,1\}^*$:

$$f(x) = \tilde{P}_{|x|}(x, r) \mod 2$$

for at least $2/3$ of the strings $r \in \{0,1\}^{m(|x|)}$.

(The same result holds for $\mathcal{P} \cdot \oplus \cdot \mathbf{P}$.)

Proof: By definition, $f \in \mathcal{BP} \cdot \oplus \cdot \mathbf{P}$ iff

$$(\exists g \in \#\mathbf{P})(\exists^+ r \in \{0,1\}^{m(|x|)})(\forall x \in \{0,1\}^*)\ f(x) = g(x, r) \mod 2$$

The claim is immediate from Theorem 1. Analogously for $\mathcal{P} \cdot \oplus \cdot \mathbf{P}$.

□

- Based on the previous results, we can also show that:

Theorem 3

$$\mathcal{P} \cdot \oplus \cdot \mathbf{P} \subseteq \mathbf{P}^{\#\mathbf{P}}$$

Proof (*Toda*):

...

PRABs and Polynomial Hierarchy

- Can we describe the Polynomial Hierarchy by such programs?
- We encode quantified Boolean Formulas with a bounded number of quantifier alternations:

$$\psi_i(x_{i+1}, \ldots, x_d) = \mathbf{Q}_i x_i \psi_{i-1}(x_i, \ldots, x_d)$$

, where $\mathbf{Q}_i \in \{\exists, \forall\}$, and ψ_0 is a 3CNF formula.

PRABs and Polynomial Hierarchy

- Can we describe the Polynomial Hierarchy by such programs?
- We encode quantified Boolean Formulas with a bounded number of quantifier alternations:

$$\psi_i(x_{i+1}, \ldots, x_d) = \mathbf{Q}_i x_i \psi_{i-1}(x_i, \ldots, x_d)$$

, where $\mathbf{Q}_i \in \{\exists, \forall\}$, and ψ_0 is a 3CNF formula.

Theorem 4

Partially quantified Boolean formulas with a bounded number of quantifier alternations can be represented probabilistically by PRABs *mod*2 in the sense that for any ψ_i, there exists a PRAB P^i such that:

$$\tilde{P}^i(x_{i+1}, \ldots, x_d, r_1, \ldots, r_i) = \psi_i(x_{i+1}, \ldots, x_d)$$

for all but an arbitrarily exponential small fraction of r_j's, $|r_j| \leq p(n)$ for some $p \in poly$.

PRABs and Polynomial Hierarchy

- So, finally, we have:
- *Theorem 2 & 4* $\Rightarrow \mathbf{PH} \subseteq \mathcal{BP} \cdot \bigoplus \cdot \mathbf{P}$
- And by using *Theorem 3*: $\mathcal{P} \cdot \bigoplus \cdot \mathbf{P} \subseteq \mathbf{P}^{\#\mathbf{P}}$
  we obtain an alternative proof of a famous result:

PRABs and Polynomial Hierarchy

- So, finally, we have:
- *Theorem 2 & 4* $\Rightarrow \mathbf{PH} \subseteq \mathcal{BP} \cdot \oplus \cdot \mathbf{P}$
- And by using *Theorem 3*: $\mathcal{P} \cdot \oplus \cdot \mathbf{P} \subseteq \mathbf{P}^{\#\mathbf{P}}$
  we obtain an alternative proof of a famous result:

Toda's Theorem

$$\mathbf{PH} \subseteq \mathbf{P}^{\#\mathbf{P}}$$

- The "connecting" inclusion $\mathcal{BP} \cdot \oplus \cdot \mathbf{P} \subseteq \mathcal{P} \cdot \oplus \cdot \mathbf{P}$ follows trivially.

Epilogue: Probabilstically Checkable Proofs

- But if we put a **proof** instead of a Prover?

Epilogue: Probabilstically Checkable Proofs

- But if we put a **proof** instead of a Prover?
- The alleged proof is a string, and the (probabilistic) verification procedure is given direct (oracle) access to the proof.
- The verification procedure can access only *few* locations in the proof!
- We parameterize these Interactive Proof Systems by two complexity measures:
  - **Query** Complexity
  - **Randomness** Complexity
- The effective proof length of a PCP system is upper-bounded by $q(n) \cdot 2^{r(n)}$ (in the non-adaptive case). (How long can be in the adaptive case?)

PCP Definitions

Definition

PCP Verifiers Let L be a language and $q, r : \mathbb{N} \to \mathbb{N}$. We say that L has an $(r(n), q(n))$-**PCP** verifier if there is a probabilistic polynomial-time algorithm V (the verifier) satisfying:

- *Efficiency*: On input $x \in \{0,1\}^*$ and given random oracle access to a string $\pi \in \{0,1\}^*$ of length at most $q(n) \cdot 2^{r(n)}$ (which we call the proof), V uses at most $r(n)$ random coins and makes at most $q(n)$ non-adaptive queries to locations of π. Then, it accepts or rejects. Let $V^\pi(x)$ denote the random variable representing V's output on input x and with random access to π.
- *Completeness*: If $x \in L$, then $\exists \pi \in \{0,1\}^* : \mathbf{Pr}\left[V^\pi(x) = 1\right] = 1$
- *Soundness*: If $x \notin L$, then $\forall \pi \in \{0,1\}^* : \mathbf{Pr}\left[V^\pi(x) = 1\right] \leq \frac{1}{2}$

We say that a language L is in $\mathbf{PCP}(r(n), q(n))$ if L has a $(\mathcal{O}(r(n)), \mathcal{O}(q(n)))$-**PCP** verifier.

Main Results

- Obviously:

$$\mathbf{PCP}(0,0) = \text{?}$$
$$\mathbf{PCP}(0, poly) = \text{?}$$
$$\mathbf{PCP}(poly, 0) = \text{?}$$

Main Results

- Obviously:

$$\mathbf{PCP}(0,0) = \mathbf{P}$$
$$\mathbf{PCP}(0, poly) = \mathbf{?}$$
$$\mathbf{PCP}(poly, 0) = \mathbf{?}$$

Main Results

- Obviously:

$$\mathbf{PCP}(0,0) = \mathbf{P}$$
$$\mathbf{PCP}(0, poly) = \mathbf{NP}$$
$$\mathbf{PCP}(poly, 0) = \textbf{?}$$

Main Results

- Obviously:

$$\mathbf{PCP}(0,0) = \mathbf{P}$$
$$\mathbf{PCP}(0, poly) = \mathbf{NP}$$
$$\mathbf{PCP}(poly, 0) = co\mathbf{RP}$$

Main Results

- Obviously:

$$\mathbf{PCP}(0,0) = \mathbf{P}$$
$$\mathbf{PCP}(0, poly) = \mathbf{NP}$$
$$\mathbf{PCP}(poly, 0) = co\mathbf{RP}$$

- A suprising result from Arora, Lund, Motwani, Safra, Sudan, Szegedy states that:

The PCP Theorem

$$\mathbf{NP} = \mathbf{PCP}(\log n, 1)$$

Main Results

- The proof is **constructive**: Transform any **NP**-witness into an oracle that makes the PCP verifier accept with probability 1.

Proof Overview

- $\mathbf{NP} \subseteq \mathbf{PCP}(\log n, \text{poly}\log n)$
- $\mathbf{NP} \subseteq \mathbf{PCP}(\text{poly} n, 1)$
- Compose the above two: The "inner verifier" is used for probabilistically verifying the acceptance criteria of the "outer" verifier.

Main Results

- The proof is **constructive**: Transform any **NP**-witness into an oracle that makes the PCP verifier accept with probability 1.

Proof Overview

- $\mathbf{NP} \subseteq \mathbf{PCP}(\log n, \text{poly} \log n)$
- $\mathbf{NP} \subseteq \mathbf{PCP}(\text{poly} n, 1)$
- Compose the above two: The "inner verifier" is used for probabilistically verifying the acceptance criteria of the "outer" verifier.

- The composition of the two yields a PCP with: $r(n) = r'(n) + r''(q'(n))$ and $q(n) = q''(q'(n))$

Further Reading

- S. Arora, B. Barak: *Computational Complexity: A Modern Approach*, Cambridge University Press, 2009
- Oded Goldreich: *Computational Complexity: A Conceptual Perspective*, Cambridge University Press, 2008
- L. A. Hemaspaandra & M. Ogihara, *The Complexity Theory Companion*, Springer, 2002
- L. Trevisan, Lecture Notes in Computational Complexity, 2004, UC Berkeley.
- S. Goldwasser and M. Sipser, *Private coins versus public coins in interactive proof systems*, In Proceedings of the eighteenth annual ACM symposium on Theory of computing, STOC '86
- L. Babai & S. Moran, *Arthur-Merlin Games: A randomized proof system, and a hierarchy of complexity classes*, J. Comput. Syst. Sci., 36(2):254-276, 1988

- Stathis Zachos and Martin Fürer, *Probabilistic quantifiers vs. distrustful adversaries*, In Foundations of Software Technology and Theoretical Computer Science, volume 287 of Lecture Notes in Computer Science, pages 443-455, Springer Berlin / Heidelberg, 1987.
- S. Zachos, *Probabilistic quantifiers, adversaries, and complexity classes: an overview*, In Proc. of the conference on Structure in complexity theory, pages 383-400, New York, NY, USA, 1986. Springer-Verlag New York, Inc.
- A. Shamir, $IP = PSPACE$, J. ACM, 39:869-877, October 1992
- L. Babai and L. Fortnow, *Arithmetization: A new method in structural complexity theory*, Computational Complexity, 1:41-66, 1991
- S. Arora, *How NP got a new definition: a survey of probabilistically checkable proofs*, CoRR cs.CC/0304038, 2003

Thank You!