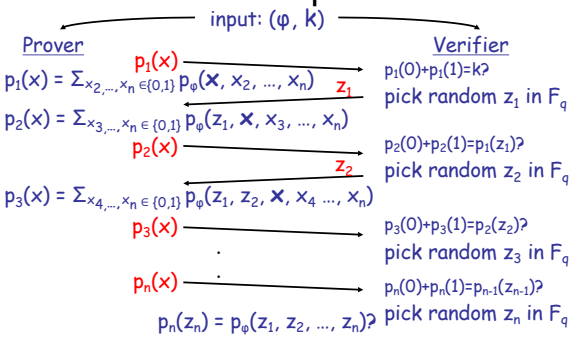
CS151
Complexity Theory

Lecture 14

May 13, 2021

The actual protocol



Analysis of protocol

- Completeness:
  - if (φ, k) ∈ L then honest prover on previous slide will always cause verifier to accept

Analysis of protocol

- Soundness:
  - let $p_i(x)$ be the correct polynomials
  - let $p_i^*(x)$ be the polynomials sent by (cheating) prover
  - $(\varphi, k) \notin L \Rightarrow p_1(0) + p_1(1) \neq k$
  - either $p_1^*(0) + p_1^*(1) \neq k$ (and V rejects)
  - or $p_1^* \neq p_1 \Rightarrow Pr_{z_1}[p_1^*(z_1) = p_1(z_1)] \leq d/q \leq |\varphi|/2^n$
  - assume $(p_{i+1}(0)+p_{i+1}(1)=)\ p_i(z_i) \neq p_i^*(z_i)$
  - either $p_{i+1}^*(0) + p_{i+1}^*(1) \neq p_i^*(z_i)$ (and V rejects)
  - or $p_{i+1}^* \neq p_{i+1} \Rightarrow Pr_{z_{i+1}}[p_{i+1}^*(z_{i+1}) = p_{i+1}(z_{i+1})] \leq |\varphi|/2^n$

Analysis of protocol

- Soundness (continued):
  - if verifier does not reject, there must be some i for which:

    $p_i^* \neq p_i$ and yet $p_i^*(z_i) = p_i(z_i)$
  - for each i, probability is $\leq |\varphi|/2^n$
  - union bound: probability that there exists an i for which the bad event occurs is

    $\leq n|\varphi|/2^n \leq poly(n)/2^n << 1/3$

Analysis of protocol

- Conclude: L = { (φ, k): CNF φ has exactly k satisfying assignments} is in **IP**
- L is **coNP**-hard, so **coNP**⊆ **IP**
- Question remains:
  - **NP, coNP** ⊆ **IP.** Potentially larger. How much larger?

IP = PSPACE

Theorem: (Shamir) **IP = PSPACE**

- Note: **IP ⊆ PSPACE**
  - enumerate all possible interactions, explicitly calculate acceptance probability
- interaction extremely powerful !
- An implication: you can interact with master player of Generalized Geography and determine if she can win from the current configuration even if you do not have the power to compute optimal moves!

IP = PSPACE

- need to prove **PSPACE ⊆ IP**
  - use same type of protocol as for **coNP**
  - some modifications needed

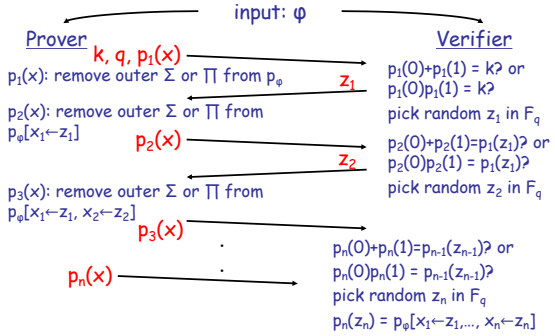
IP = PSPACE

- protocol for QSAT
  - arithmetization step produces **arithmetic expression** p_φ:
    - $(\exists x_i)\ \varphi \rightarrow \Sigma_{x_i = 0, 1}\ p_\varphi$
    - $(\forall x_i)\ \varphi \rightarrow \prod_{x_i = 0, 1}\ p_\varphi$
  - start with QSAT formula in special form ("simple")
    - no occurrence of x_i separated by more than one "∃" from point of quantification

IP = PSPACE

- quantified Boolean expression φ is true if and only if $p_\varphi > 0$
- Problem: ∏'s may cause $p_\varphi > 2^{2^{|\varphi|}}$
- Solution: evaluate mod $2^n \le q \le 2^{3n}$
- prover sends "good" q in first round
  - "good" q is one for which $p_\varphi \bmod q > 0$
- Claim: good q exists
  - # primes in range is at least 2^n

The QSAT protocol



Analysis of the QSAT protocol

- Completeness:
  - if φ ∈ QSAT then honest prover on previous slide will always cause verifier to accept

Analysis of the QSAT protocol

- Soundness:
  - let $p_i(x)$ be the correct polynomials
  - let $p_i^*(x)$ be the polynomials sent by (cheating) prover
  - $\varphi \notin$ QSAT $\Rightarrow 0 = p_1(0) +/x\ p_1(1) \neq k$
  - either $p_1^*(0) +/x\ p_1^*(1) \neq k$ (and V rejects) — φ is "simple"
  - or $p_1^* \neq p_1 \Rightarrow Pr_{z_1}[p_1^*(z_1) = p_1(z_1)] \leq 2|\varphi|/2^n$
  - assume $(p_{i+1}(0) +/x\ p_{i+1}(1)=)\ p_i(z_i) \neq p_i^*(z_i)$
  - either $p_{i+1}^*(0) +/x\ p_{i+1}^*(1) \neq p_i^*(z_i)$ (and V rejects)
  - or $p_{i+1}^* \neq p_{i+1} \Rightarrow Pr_{z_{i+1}}[p_{i+1}^*(z_{i+1}) = p_{i+1}(z_{i+1})] \leq 2|\varphi|/2^n$

Analysis of protocol

- Soundness (continued):
  - if verifier does not reject, there must be some i for which:

    $p_i^* \neq p_i$ and yet $p_i^*(z_i) = p_i(z_i)$
  - for each i, probability is $\leq 2|\varphi|/2^n$
  - union bound: probability that there exists an i for which the bad event occurs is

    $\leq 2n|\varphi|/2^n \leq poly(n)/2^n << 1/3$
- Conclude: QSAT is in **IP**

Example

- Papadimitriou – pp. 475-480

$\varphi = \forall x \exists y (x \vee y) \wedge \forall z ((x \wedge z) \vee (y \wedge \neg z)) \vee \exists w (z \vee (y \wedge \neg w))$

$p_\varphi = \prod_{x=0,1} \Sigma_{y=0,1} [(x + y) * \prod_{z=0,1} [(xz + y(1-z)) + \Sigma_{w=0,1}(z + y(1-w))]]$

($p_\varphi = 96$ but V doesn't know that yet !)

Example

$p_\varphi = \prod_{x=0,1} \Sigma_{y=0,1} [(x + y) * \prod_{z=0,1} [(xz + y(1-z)) + \Sigma_{w=0,1}(z + y(1-w))]]$

Round 1: (prover claims $p_\varphi > 0$)

- prover sends q = 13; claims $p_\varphi = 96 \bmod 13 = 5$; sends k = 5
- prover removes outermost "$\prod$"; sends

  $p_1(x) = 2x^2 + 8x + 6$
- verifier checks:

  $p_1(0)p_1(1) = (6)(16) = 96 \equiv 5 \pmod{13}$
- verifier picks randomly: $z_1 = 9$

Example

$\varphi = \forall x \exists y (x \vee y) \wedge \forall z ((x \wedge z) \vee (y \wedge \neg z)) \vee \exists w (z \vee (y \wedge \neg w))$

$p_\varphi = \prod_{x=0,1} \Sigma_{y=0,1} [(x + y) * \prod_{z=0,1} [(xz + y(1-z)) + \Sigma_{w=0,1}(z + y(1-w))]]$

$p_\varphi[x \leftarrow 9] = \Sigma_{y=0,1} [(9 + y) * \prod_{z=0,1} [(9z + y(1-z)) + \Sigma_{w=0,1}(z + y(1-w))]]$

Example

$p_1(9) = \Sigma_{y=0,1} [(9 + y) * \prod_{z=0,1} [(9z + y(1-z)) + \Sigma_{w=0,1}(z + y(1-w))]]$

Round 2: (prover claims this = 6)

- prover removes outermost "Σ"; sends

  $p_2(y) = 2y^3 + y^2 + 3y$
- verifier checks:

  $p_2(0) + p_2(1) = 0 + 6 = 6 \equiv 6 \pmod{13}$
- verifier picks randomly: $z_2 = 3$

Example

$\varphi = \forall x \exists y (x \vee y) \wedge \forall z((x \wedge z) \vee (y \wedge \neg z)) \vee \exists w(z \vee (y \wedge \neg w))$

$p_\varphi = \prod_{x=0,1} \Sigma_{y=0,1}[(x + y) * \prod_{z=0,1}[(xz + y(1-z)) + \Sigma_{w=0,1}(z + y(1-w))]]$

$p_\varphi[x \leftarrow 9, y \leftarrow 3] = [(9 + 3) * \prod_{z=0,1}[(9z + 3(1-z)) + \Sigma_{w=0,1}(z + 3(1-w))]]$

Example

$p_2(3) = [(9 + 3) * \prod_{z=0,1}[(9z + 3(1-z)) + \Sigma_{w=0,1}(z + 3(1-w))]]$

Round 3: (prover claims this = 7)

- everyone agrees expression = 12*(…)
- prover removes outermost "∏"; sends

$p_3(z) = 8z + 6$

- verifier checks:

$p_3(0) * p_3(1) = (6)(14) = 84;\ 12*84 \equiv 7 \pmod{13}$

- verifier picks randomly: $z_3 = 7$

Example

$\varphi = \forall x \exists y (x \vee y) \wedge \forall z((x \wedge z) \vee (y \wedge \neg z)) \vee \exists w(z \vee (y \wedge \neg w))$

$p_\varphi = \prod_{x=0,1} \Sigma_{y=0,1}[(x + y) * \prod_{z=0,1}[(xz + y(1-z)) + \Sigma_{w=0,1}(z + y(1-w))]]$

$p_\varphi[x \leftarrow 9, y \leftarrow 3, z \leftarrow 7] =$

$12 * [(9*7 + 3(1-7)) + \Sigma_{w=0,1}(7 + 3(1-w))]$

Example

$12*p_3(7) = 12 * [(9*7 + 3(1-7)) + \Sigma_{w=0,1}(7 + 3(1-w))]$

Round 4: (prover claims = 12*10)

- everyone agrees expression = 12*[6+(…)]
- prover removes outermost "Σ"; sends

$p_4(w) = 10w + 10$

- verifier checks:

$p_4(0)+p_4(1) = 10 + 20 = 30;\ 12*[6+30] \equiv 12*10 \pmod{13}$

- verifier picks randomly: $z_4 = 2$
- Final check:

$12*[(9*7+3(1-7))+(7+3(1-2))] = 12*[6+p_4(2)] = 12*[6+30]$

Arthur-Merlin Games

- **IP** permits verifier to keep coin-flips private
  - necessary feature?
  - GNI protocol breaks without it
- Arthur-Merlin game: interactive protocol in which coin-flips are public
  - Arthur (verifier) may as well just send results of coin-flips and ask Merlin (prover) to perform any computation Arthur would have done

Arthur-Merlin Games

- Clearly **Arthur-Merlin ⊆ IP**
  - "private coins are at least as powerful as public coins"
- Proof that **IP** = **PSPACE** actually shows

  PSPACE ⊆ Arthur-Merlin ⊆ IP = PSPACE
  - "public coins are at least as powerful as private coins" !

Arthur-Merlin Games

- Delimiting # of rounds:
  - **AM[k]** = Arthur-Merlin game with k rounds, Arthur (verifier) goes first
  - **MA[k]** = Arthur-Merlin game with k rounds, Merlin (prover) goes first

Theorem: **AM[k]** (**MA[k]**) equals **AM[k]** (**MA[k]**) with perfect completeness.
  - i.e., $x \in L$ implies accept with probability 1
  - proof on problem set

Arthur-Merlin Games

Theorem: for all constant $k \geq 2$

$$\mathbf{AM[k] = AM[2]}.$$

- Proof:
  - we show **MA[2]** $\subseteq$ **AM[2]**
  - implies can move all of Arthur's messages to beginning of interaction:

AMAMAM…AM = AAMMAM…AM

… = AAA…AMMM…M

Arthur-Merlin Games

- Proof (continued):
  - given $L \in$ **MA[2]**

$$x \in L \Rightarrow \exists m \; Pr_r[(x, m, r) \in R] = 1$$

order reversed

$$\Rightarrow Pr_r[\exists m \; (x, m, r) \in R] = 1$$

$$x \notin L \Rightarrow \forall m \; Pr_r[(x, m, r) \in R] \leq \varepsilon$$

$$\Rightarrow Pr_r[\; \forall m \; (x, m, r) \in R] \leq 2^{|m|}\varepsilon$$

  - by repeating t times with independent random strings r, can make error $\varepsilon < 2^{-t}$
  - set $t = m+1$ to get $2^{|m|}\varepsilon < ½$.

MA and **AM**

- Two important classes:
  - **MA** = **MA[2]**
  - **AM** = **AM[2]**
- definitions without reference to interaction:
  - $L \in$ **MA** iff $\exists$ poly-time language R

$$x \in L \Rightarrow \exists \; m \; Pr_r[(x, m, r) \in R] = 1$$

$$x \notin L \Rightarrow \forall \; m \; Pr_r[(x, m, r) \in R] \leq ½$$

  - $L \in$ **AM** iff $\exists$ poly-time language R

$$x \in L \Rightarrow Pr_r[\; \exists \; m \; (x, m, r) \in R] = 1$$

$$x \notin L \Rightarrow Pr_r[\; \exists \; m \; (x, m, r) \in R] \leq ½$$

MA and **AM**

  - $L \in$ **AM** iff $\exists$ poly-time language R

$$x \in L \Rightarrow Pr_r[\; \exists \; m \; (x, m, r) \in R] = 1$$

$$x \notin L \Rightarrow Pr_r[\; \exists \; m \; (x, m, r) \in R] \leq ½$$

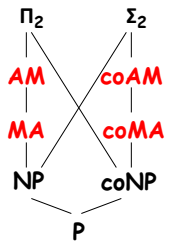
- Relation to other complexity classes:
  - both contain **NP** (can elect to not use randomness)
  - both contained in $\prod_2$. $L \in \prod_2$ iff $\exists \; R \in$ **P** for which:

$$x \in L \Rightarrow Pr_r[\; \exists \; m \; (x, m, r) \in R] = 1$$

$$x \notin L \Rightarrow Pr_r[\; \exists \; m \; (x, m, r) \in R] < 1$$

  - so clear that **AM** $\subseteq \prod_2$
  - know that **MA** $\subseteq$ **AM**

MA and **AM**



MA and AM

Theorem: **coNP ⊆ AM ⇒ PH = AM.**

- Proof:
  - suffices to show $\Sigma_2 \subseteq$ **AM** (and use **AM** $\subseteq \Pi_2$)
  - L $\in \Sigma_2$ iff ∃ poly-time language R

    $x \in L \Rightarrow \exists\, y\, \forall\, z\, (x, y, z) \in R$

    $x \notin L \Rightarrow \forall\, y\, \exists\, z\, (x, y, z) \notin R$
  - Merlin sends y
  - 1 AM exchange decides **coNP** query: $\forall\, z\, (x, y, z) \in R$?
  - 3 rounds; in **AM**

MA and AM

- We know **Arthur-Merlin = IP**.
  - "public coins = private coins"

Theorem (GS): **IP[k] ⊆ AM[O(k)]**

- stronger result
- implies for all constant k ≥ 2,

  IP[k] = AM[O(k)] = AM[2]

- So, GNI ∈ **IP[2] = AM**

Back to Graph Isomorphism

- The payoff:
  - not known if GI is **NP**-complete.
  - previous Theorems:

    if GI is **NP**-complete then **PH = AM**
  - unlikely!

  - Proof: GI **NP**-complete ⇒ GNI **coNP**-complete ⇒ **coNP ⊆ AM ⇒ PH = AM**