Advanced Complexity

Jonathan Kelner
3/10/2011

Public vs. Private Coins

- Last time, showed how to convert any 2 round private coin protocol into a public coin one
  - People seemed a bit unsure about end of it, so I’d like to briefly review it and clarify things

- Basic idea:
  - Prover wants to convince the verifier (using public coins) that the private coin protocol would have accepted w.h.p.
  - If have a set \( S \) that verifier can check membership in, can use hashing and sampling to prove large
    - This is enough for GNI
  - For general case, not clear how to answer whether prover would have convinced to accept for given private coins
    - Prover could lie about what he would have done, since now knows the coins
  - Instead, make him say how he’d answer a specific question from verifier, and then make him show that this would correctly answer a lot of questions
    - Prover can do this for a lot of questions
      - See next slide for picture

\[
\begin{align*}
P: & \text{"There are } \geq 2^m \text{ questions where Accept has } \geq 2^m \text{ strings"} \\
V: & \text{Checks } 2^m \times m^2 \text{ enough. Sends:} \\
& h_i(\text{Questions}) \rightarrow \{0,1\}^{m^2}, \\
& t_i \in \{0,1\}^{m^1}
\end{align*}
\]

- P: Sends:
  - q s.t. \( h_i(q) = t_i \) and his answer to q
- V: Sends
  - \( h_j(\text{rand. bits}) \rightarrow \{0,1\}^{m^2} \)
  - \( t_j \in \{0,1\}^{m^2} \)
- P: Sends r s.t. \( h_j(r) = t_j \)
- V: checks r from same question q and would accept with given answer

- Prove #SAT \( \in \text{IP} \) in 18.404, but I’m going to review it
- Will try to relate it to some of the things we’ve done in this class

IP = PSPACE

- Note that IP not like alternation
  - Only \( \sum \), no \( \Pi \)s (i.e., no \( \forall \) quantifiers)
  - How would you prove formula unsatisfiable?
- Exact vs. approximate counting
- Surprisingly, can actually do exact counting in IP
  - \( \#P \subseteq \text{IP} \)
- In fact, can do anything in \( \text{PSPACE}! \)
  - This was very surprising when it came out
- Not hard to show \( \text{IP} \subseteq \text{PSPACE} \), so gives IP=PSPACE
- Proof rests on self reducibility
  - Basic point is can do anything self reducible in IP
- You showed #SAT \( \in \text{IP} \) in 18.404, but I’m going to review it
- Will try to relate it to some of the things we’ve done in this class
• Say have SAT formula \( \phi \), \( n \) vars, \( |\phi| = m \)
• \( P \) claims has \( N \) satisfying assignments
  - Good example to keep in mind: \( N = 0 \)
• \( V \) will ask questions like “how many satisfying assignments start with 01100”
  - Let \( f(a_{i-1}, a_0) \# \) SAT assign that start \( a_{i-1} \)
  - Self-reducibility: \( f(a_{i-1}, a_0) \# f(r_{i-1}, a_0) + f_1(a_{i-1}, a_0) \)
  - So if asked for \( f(a_{i-0}, a_0) \), \( f_1(a_{i-0}, a, 0), f_1(a_{i-0}, a, 1), \) could check if consistent
• If \( f_i \) is wrong, at least one of the \( f_{i+1}'s \) is wrong too
• So self-reducibility lets us verify recursively
• Of course, not efficient enough
  - **Usual explanation**: to verify, need to check whole tree, which is way too many verifications
• Alternative view:
  - **Randomized reduction**: To verify \( f(a_{i-1}, a_0) \), ask for \( f_{i-1}(a_{i-1}, a_0) \) and \( f_1(a_{i-1}, a_0, 0) \) randomly verify **one of them**
  - Now not too many calls, problem is just that error prob. is too high
  - Just need to find a way to improve probabilities
  - Do this with arithmetization
• This view is quite useful in other instances, will see others

---

**Arithmetization**

• Make multivariate polynomial \( p \) over finite field \( F \) s.t. \( p(a_1, ..., a_n) \)
  - agrees with \( \phi \) when all \( a_i \in \{0, 1\} \)
  - Don’t restrict \( p \) when \( a_i \) not all in \( (0,1) \)
  - Field \( F \) chosen to be large: \( |F| > 2^m \)
  - Can construct & do arithmetic over \( F \) in poly time
  - **Key point**: polys of reasonable degree that are different are different at a lot of points (1-degree/2^n fraction)
• Turn logic into arithmetic:
  - \( f \land g \to fg \)
  - \( f \lor g \to 1-(1-f)(1-g) \)
  - **Note**: not \( f \lor g \)
  - \( \neg f \to 1-f \)
• SAT formula \( \phi \) becomes poly \( p \) of degree \( \leq m \)
  - **Redefine** \( f \) using \( p \) to be poly that agrees with previous definition on \( \{0,1\}^n \): \( f(a_1, ..., a_n) = \sum_{a_{n-1}, ..., a_0 \in \{0,1\}} p(a_1, ..., a_n) \)
• Still have \( f(a_{i-1}, a_0) = f_1(a_{i-1}, a_0, 0) + f_1(a_{i-1}, a, 1) \)
• But now can check at many points, and, if wrong, polys diff at most of them

---

**Generalizing to PSPACE**

• What did we use?
  - each poly to have bounded degree
  - each poly to have bound \( \# \) vars
  - to be able to compute \( f_i \) in poly time
  - to be able to compute \( f \) in terms of a small \# of values of \( f_i \)
• Can almost get these for TQBF
  - \( \exists \) is just an \( \land \), \( \forall \) is just an \( \lor \)
• **Problem**: degree blows up
• See board.
  - Add “linearization” quantifiers \( L_i \) as on board, get formula like
    - \( \forall x_1, l_1 \forall x_2, l_2 \exists x_3, l_3 \exists x_4, l_4 \exists l_5 \ldots p(x_1, x_2, x_3, x_4, x_5, \ldots) \)
  - where \( p \) is arithmetization of \( \phi \)
• Total \# quantifiers (\( \forall, \exists \), or \( L_i \)) is \( O(m^2) \)
IP Protocol for PSPACE

- **Phase 0:** \([f_0(j)]\)
  - P sends \(f_0(j)\)
  - V checks if \(f_0(j)=1\) (so that P claims TQBF is true)
  
  ... 

- **Phase \(k\):** (reduce verifying \(f_{k-1}(r_1,\ldots)\) to verifying \(f_k(r_1,\ldots)\))
  - P sends coefficients of \(f_k(r_1,\ldots)\) as poly in \(z\)
  - V checks that deg right & vals consistent:
    - If P quantifier is \(\forall\): check \(f_k(r_1,\ldots) = f_{k-1}(r_1,\ldots,0)*f_{k-1}(r_1,\ldots,1)\)
    - If P quantifier is \(\exists\): check \(f_k(r_1,\ldots) = 1-((1-f_{k-1}(r_1,\ldots,0))*(1-f_{k-1}(r_1,\ldots,1)))\)
    - If P quantifier is \(L_i\): check \(f_k(r_1,\ldots,1) = (1-r)*f_{k-1}(r_1,\ldots,0) + r*f_{k-1}(r_1,\ldots,1)\)

  Note: V and \(\exists\) change #vars, but \(L_i\) doesn't, so # vars = \(i\), # rounds: some \(N > n\)

Picks random \(r_2 \in F\) and sends to P. (If at an \(L_i\), replace old \(r\) with new one)

... 

- **Final phase:** [check \(f_n(r_1,\ldots,r_n)\)]
  - V checks directly that \(f_n(r_1,\ldots,r_n) = p(r_1,\ldots,r_n)\).

  Note: final round, just had to verify \(p\), which was arithmetization of original formula, so we can do this without help from prover

- Earlier rounds: only sent coefficients of 1-variable polynomials, so didn't ever have to send too much information