Complexity Theory - Tutorial

Ivan Gavran

February 7th, 2017

Part 1

1. Decide whether the following statements are true or false and justify your decision.

   (a) For every class C and $A \in C$; if A is C-complete under Karp reductions, then $C \subset P^A$
   Solution *True.* Any $B \in C$ can be transformed into instance of A. But this - by the definition - means that B can be decided by polynomial computation (do the transformation into an instance of A) and then asking oracle A.

   (b) If L is NP-complete and L' is coNP complete, then $L \cap L'$ is $NP \cap coNP$ complete.
   Solution *False.* SAT is NP-complete. NOSAT is coNP-complete. Their intersection is empty (so that language would trivially always return false, but we know that there are non-trivial languages in $NP \cap coNP$, primality, for example.).

   (c) If PH=PSPACE, then PH collapses to some level Σ_k
   Solution *True.* There is a complete problem for PSPACE (TQBF, for example). Since PH = PSPACE, this problem is in PH and therefore it is in Σ_k, for some k. But now all the problems in PH can be in polynomial time reduced to that one. Therefore, PH collapses to Σ_k.

   (d) There is an undecidable language in $\mathrm{P}/_{poly}$
   Solution *True* All unary languages are in P/poly, unary variant of halting problem is in P/poly.

   (e) If P=NP, then NP=coNP.
   Solution *True* Since NP = P and $P \subset coNP$ it is clear that $NP \subset coNP$. Take $L \in coNP$. This means that $\bar{L} \in NP = P$. Then there exists a poly-time Turing machine M that decides $\bar{L}$. Create $M' = 1 - M$, that machine decides L in polynomial time. Therefore, $coNP \subset P = NP$

   (f) If #L is #P-complete, then L is NP-complete (you may assume $P \neq$ NP).
   Solution *False.* #2-SAT is #P-complete, while 2-SAT is in P.

(g) Show that every circuit with only $\wedge$ and $\vee$ gates can be replaced by a circuit containing only *majority* gates.

2. DOUBLE-SAT $= \{\phi : \phi$is a Boolean formula with 2 satisfying assignments$\}$. Show that DOUBLE-SAT is NP-complete.
Solution DOUBLE-SAT is clearly in NP (the two mentioned assignments could be given as a witness. Now we show how to reduce SAT to DOUBLE-SAT. Let ϕ be a 3-CNF formula. We define $\phi'(y, x) = \phi(y) \wedge (x \vee \bar{x})$. If $\phi \notin 3SAT$ then ϕ' is obviously not in SAT. On the other hand, if $\phi \in 3SAT$, it means there is a satisfying assignment for ϕ. But from that one we can derive two satisfying assignments for ϕ'.

3. Show that $UPATH \in RL$. You may use the following fact: if graph G has a path from s to t, then a random walk of length $8 \cdot |V(G)| \cdot |E(G)|$ visits t with probability $\geq \frac{1}{2}$

4. Show that $SPACE(f(n)) \subset RSPACE(f(n)) \subset NSPACE(f(n))$

Part 2

5. We showed that there is an undecidable language in P/poly. Show that there are decidable languages in P/poly that are not in P. Use the following waypoints:

   (a) show that there is a decidable language L that is not in EXP

   (b) define $L' := \{1^m : m \in L\}$ and show that L' is decidable and member of P/poly

   (c) show that L' is not in P

   Solution We know that there is a decidable language L that is not in EXP (by the time-hierarchy theorem). With the definition of L', we know that L' is decidable (because we can translate 1^m into m. It is also in P/poly because every unary language is in P/poly. Finally, in order to prove that L' is not contained in P, we assume contrary, $L' \in P$. Then starting from $x \in L$ we can turn it into 1^x in exponential time and then in polynomial time (in the size of unfolded string) we can decide whether the original one was in L. But this all together makes an exponential algorithm for L which is a contradiction with L not being in EXP.

6. Suppose that we have a poly-time algorithm A such that for $\phi \in$ USAT, $A(\phi) = 1$ and for $\phi \notin SAT$, $A(\phi) = 0$. Show that then NP = RP. Afterwards, show that AM[2] = BPP.
Solution First we show that NP = RP. $RP \subset NP$: consider the language $L \in$ RP and a computation of the RP machine. If $\alpha \in L$, at least half of the branches are accepting (therefore, there is one that accepts as well). If $\alpha \notin L$ not a single branch accepts, which is exactly what we need.

$NP \subset RP$: take ϕ, an instance of SAT. By Valiant-Vazirani theorem (17.18 from Arora-Barak book), there is a probabilistic polynomial time algorithm f such that for every n-variable Boolean formula ϕ if $\phi \in SAT$, $f(\phi) \in USAT$ with probability $\geq \frac{1}{8n}$ and for $\phi \notin SAT$ the probability that $f(\phi)$ would be in SAT equals to zero. But the we have - in case that ϕ is not satisfiable - probability zero that $A(f(\phi))$ would answer zero and for the cases when ϕ is satisfiable, probability that $A(f(\phi))$ equals 1 could be boosted to a higher value.
Now we show that AM[2]=BPP. The first inclusion - BPP $\subset$ AM is easy: a verifier just ignores what the prover sent and does all the computation on its own. For the other inclusion, we know that NP = RP $\subset BPP$. Recall the (alternative) definition of AM: $L \in$ AM if there exists a (poly-time, deterministic) machine M such that for every input x of length n

- if $x \in L$, $\mathop{\mathbb{P}}_{y}[\exists z : M(x, y, z) = 1] \geq \frac{2}{3}$
- if $x \notin L$, $\mathop{\mathbb{P}}_{y}[\forall z : M(x, y, z) = 0] \geq \frac{2}{3}$

Since we are working under the assumption that $NP \subset BPP$, we know that there is a branching machine M' such that $M'(x, y) = \exists z : M(x, y, z) = 1$ for $x \in L$ and $M'(x, y) = \forall z : M(x, y, z) = 0$, namely, a BPP computation that can distinguish between these two situations (deterministically). But this exactly tells us that $L \in BPP$.

7. Show that regular languages are in NC^1. (Hint: note that the final state can be reached from the initial one in n steps if there is some intermediate state q that can be reached from the initial one in $\frac{n}{2}$ steps and the final state can be reached from q also in $\frac{n}{2}$ steps. Repeat the same idea to the leaves.)
Solution Assume that there is a DFA M that decides a regular language and assume that M has t states and (for simplicity of explanation) that it has a single accepting state. We are going to create a circuit of logarithmic depth that would do the same. Let's observe circuit of (roughly speaking) $\log(n)$ levels of computation. At the first level, compute whether it is possible to go from q_i to q_j in 1 step. At the second level, compute whether the same is possible in 2 steps, then in 4 steps etc. So, at the i-th level, compute whether it is possible to go from q_i to q_j in 2^i steps. Finally, we will get the answer whether it is possible to go from the initial state to the final state in n steps having a circuit of depth log(n). To put it more formally, let M_a be a square boolean matrix of dimension t that has 1 on position (i,j) if and only if on reading symbol a the machine M transitions from q_i to q_j. If the input is given as $a_1a_2a_3 \dots a_n$, then consider the iterated Boolean matrix product $P := M_{a_1} \cdot M_{a_2} \cdot \ldots \cdot M_{a_n}$. A matrix multiplication here takes a circuit of constant depth. This series of multiplications can be done in parallel so that $\log(n)$ of multiplications is needed. One can prove that the position (i,j) of matrix P tells us if the state q_j can be reached from the state q_i reading the input $a_1a_2a_3 \dots a_n$.

NOTE: Some of the problems and solutions are taken from

- http://soltys.cs.csuci.edu/blog/wp-content/oldpage/cu-f07/chp5.pdf
- http://cse.iitkgp.ac.in/~abhij/course/theory/CC/Spring04/soln5.pdf