

CPSC 468/568: Answer Key for Exam 1

October 15, 2009

Question 1

- (a) Let $V = \{x, y\}$ and E consist of a single edge with endpoints x and y . Then the game ends after Player 1 makes his first move; thus this is a yes instance.
- (b) Let $V = \{x, y, u, v\}$ and $E = \{\{x, y\}, \{y, u\}, \{u, v\}\}$. If Player 1 chooses x first, then Player 2 will choose either u or v ; in either case, Player 1 will be left with no vertex to choose. If Player 1 chooses y first, then Player 2 will choose v , and again Player 1 will be left with no vertex to choose. By symmetry, Player 1 will lose if he chooses either v or u first. Because Player 1 does not have a winning strategy, this is a no instance.
- (c) We give a recursive algorithm and then compute its space complexity. The input to this algorithm is a pair (i, G) , where $i \in \{1, 2\}$ is the player whose turn it is to move. If $G = (V, E)$ is a graph and $V' \subseteq V$, then $G - V'$ is the graph with vertex set $V - V'$ and edge set $\{\{x, y\} \in E \text{ such that } x \notin V' \text{ and } y \notin V'\}$. For any vertex x , the set $N(x)$ consists of all neighbors of x , *i.e.*, all vertices y such that $\{x, y\} \in E$. If the input pair contains the empty graph (*i.e.*, the graph with no vertices and no edges), the algorithm returns “no” if $i = 1$, because Player 1 has no first move, and it returns “yes” if $i = 2$. If the input graph has one vertex (with either no edges or one edge that is a self-loop), the algorithm returns “yes” if $i = 1$ and “no” if $i = 2$, because, after Player i chooses the one vertex, Player $3 - i$ has no move. If the input graph $G = (V, E)$ has at least two vertices, then, for each $x \in V$, the algorithm makes a recursive call on input $(3 - i, G - (x \cup N(x)))$; if $(3 - i) = 1$, then the algorithm returns “yes” if and only if at least one of these calls returns “yes” and otherwise returns “no”; if $(3 - i) = 2$, then it returns “yes” if and only if all of these calls return “yes” and otherwise returns “no.” To compute the space complexity of this algorithm, we use the crucial fact that, after a recursive call is made, the space used for it can be reused for the next recursive call. Let S_n be the space needed to run this algorithm on a graph on n vertices; note that such a graph can be represented by a string of $O(n^2)$ bits. Just one bit is required to keep track of the AND or the OR of the results of all recursive calls that have been done so far. The input to a recursive call can be specified with $O(n^2)$ bits. Thus $S_n = S_{n-1} + O(n^2)$ and $S_1 = S_0 = O(1)$. This implies that $S_n = O(n^3)$, as desired.

Question 2

- (a) Suppose that $T(n)$ were time-constructible. Then the two classes that would be equal by the Gap Theorem are $\text{DTIME}(T(n))$ and $\text{DTIME}(W(n))$, where $W(n) = (T(n))^2$, and W would be time-constructible as well. Furthermore, $T(n) \log T(n) = o(W(n))$. The Time-Hierarchy Theorem would then give us that $\text{DTIME}(T(n))$ is properly contained in $\text{DTIME}(W(n))$ – a contradiction.
- (b) Alternating Turing Machines are defined on page 99 of Arora-Barak. See Definition 5.7 on pages 99-100 for $\text{ATIME}(T(n))$. If we alter Definition 5.7 to require that M use space $S(n)$, *i.e.*, that, on every $x \in \{0, 1\}^*$ and for every sequence of transition-function choices, M uses at most $S(|x|)$ space, we get the definition of $\text{ASPACE}(S(n))$.

Question 3

- (a) Different Turing Machines have different (and different-sized) transition tables, just as different real-world computers have different (and different-sized) instruction sets. Similarly, different Turing Machines have different (and different-sized) alphabets. Thus, the notion of “one unit of time (resp. space)” cannot be pinned down precisely enough to distinguish $c_1T(n)$ from $c_2T(n)$ (resp. $c_1S(n)$ from $c_2S(n)$).
- (b) Relativization applies to machines¹, not to classes. For example, “ P^B ” (resp., “ NP^B ”) refers to all languages recognizable by deterministic (resp., nondeterministic), polynomial-time Turing Machines that have access to B . If $P = NP$, then, if M is a nondeterministic, polynomial-time Turing Machine *that does not have access to any oracle*, the language $L(M)$ can also be recognized by a deterministic, polynomial-time Turing Machine that does not have access to any oracle. This still leaves open the possibility that there is such an M that could make use of some oracle B in a way that enables it to recognize a language that cannot be recognized by any deterministic, polynomial-time Turing Machine with access to B .

Question 4

- (a) False. The set UHALT defined on page 110 of Arora-Barak is an undecidable, unary language.
- (b) Unknown. If TQBF were PH-complete, then we would have that PSPACE = PH and that the PH collapses, but we don’t know whether either of those is true.
- (c) False. There are undecidable languages in P/poly but none in NP.
- (d) True. PATH is the canonical NL-complete language, and, by the Immerman-Szelepcsényi Theorem, NL = coNL.

Question 5

- (a) By the Immerman-Szelepcsényi Theorem, NL is closed under complement. We will thus prove that the complement of DISCONN is in NL, *i.e.*, that the set of *strongly connected, directed graphs is in NL*. Assume that directed graphs on n vertices are represented as n -by- n adjacency matrices, *i.e.*, by bit strings of length n^2 . A read-once, logspace-verifiable certificate of strong connectedness consists of a sequence

$$v_{1,2,1} \dots v_{1,2,l_{1,2}}, v_{1,3,1} \dots v_{1,3,l_{1,3}}, \dots, v_{n,n-1,1} \dots v_{n,n-1,l_{n,n-1}}.$$

Here, the subsequence $v_{i,j,1} \dots v_{i,j,l_{i,j}}$ is a directed path from node i to node j . The verifier must check that paths for all pairs (i, j) such that $1 \leq i, j \leq n$ and $i \neq j$ are present, that the length of each path is at most $n - 1$, and that all adjacent pairs $(v_{i,j,k}, v_{i,j,k+1})$ correspond to directed edges that are actually in the graph. Clearly, all of this can be checked in space $O(\log n)$

Technically speaking, input bitstrings that are not of length n^2 for some positive integer n are also in the complement of DISCONN, but inputs of that form are also clearly verifiable in space $O(\log n)$.

¹Actually, it applies to computational devices generally – for example, one can give a circuit access to an oracle.

- (b) First, note that, if (1) is false, then (3) is also false, regardless of whether $\text{NP} \subseteq \text{P/poly}$. Next, recall that there is a polynomial-time (and, *a fortiori*, P/poly) reduction from search to decision for SAT. If $\text{NP} \subseteq \text{P/poly}$, this reduction can be applied to the polynomial-sized circuit family in (2), which tests whether the formula determined by ϕ and u is satisfiable, to produce the polynomial-sized circuit family in (3), which finds a satisfying assignment for the formula determined by ϕ and u if one exists. Thus, if $\text{NP} \subseteq \text{P/poly}$ and (1) is true, (2) and (3) are true as well.

There is a notional liberty taken in this problem: There are polynomials $q(n)$ and $q'(n)$ such that the inputs to circuits C_n and C'_n are of length at most $q(n)$ and $q'(n)$, respectively, but neither of these polynomials is precisely n .