CS 21 Decidability and Tractability	**Winter 2018**

Solution Set 7

Posted: March 7

If you have not yet turned in PS7, you should not consult these solutions.

1. To see that the language is in NP, we simply note that a witness for an n-symbol input y is the unique n-symbol string x for which $f(x) = y$. To verify that $y \in L$ given this witness, we check that $f(x) = y$ (poly-time computable), and that $g(x) = 1$ (also poly-time computable).

   To see that the language is in coNP, we give a witness non-membership in L. For an n-symbol input y, the witness is the unique n-symbol string x for which $f(x) = y$. To verify that $y \notin L$ given this witness, we check that $f(x) = y$ (poly-time computable), and that $g(x) = 0$ (also poly-time computable).

2. (a) DEADLOCK is in NP because given a state $v = (v_1, \ldots, v_n)$ that is a purported deadlock state, we can check in polynomial time that for all i, j pairs, there is no pair $((v_i, v_i'), (v_j, v_j'))$ in the list P (there are n^2 pairs of i, j to check, and for each we need to traverse the list P).

   To prove that DEADLOCK is NP-hard, we reduce from 3-SAT. Given an 3-CNF formula ϕ with n variables and m clauses, we produce m directed graphs $G_1, \ldots, G_m$. Each $G_i = (V_i, E_i)$ is a directed cycle on 3 vertices; the vertices are labelled with the literals appearing in clause i. Our list P consists of all pairs of edges $(e_i = (v_i, v_i'), e_j = (v_j, v_j'))$ with $e_i \in E_i$ and $e_j \in E_j$ for which the literals labelling v_i and v_j are negations of each other. Now, suppose we started with a satisfiable 3-CNF formula, and fix a satisfying assignment. Then the state $v = (v_1, v_2, \ldots, v_m)$ in which each v_i is labelled with a true literal in that satisfying assignment is a deadlock state. In the other direction, if $v = (v_1, v_2, \ldots, v_m)$ is a deadlock state, then the literals labelling the vertices $v_1, \ldots, v_m$ must be consistent (meaning that no two of these literals are negations of each other), because otherwise, there is an available transition, and so v could not have been a deadlock state. But then the assignment that sets the literals labelling $v_1, v_2, \ldots, v_m$ to TRUE is a satisfying assignment to the original 3-CNF.

   (b) We give a generic reduction from a language L in PSPACE. Fix such a language, and let M be the Turing machine that decides L using at most n^k space. We first modify M to prevent it from attempting to move off the beginning tape. At the beginning of its computation, we have M place a special marker at the beginning of its n-bit input (shifting the input to the right by 1). Any move by M onto this special marker is now followed by a move back to the right (onto the first "real" tape square); this prevents moves off the left of the tape. Second, we modify M so that if it is about to reject, it instead enters an infinite loop. So, the modified machine simulates M, never moves outside the first $n^k + 1$ tape squares (counting the special marker), it halts if its input x is in L (the simulation eventually accepts), and it enters an infinite loop if x is not in L

Our reduction is given an instance of L, the n-bit string x. Define $\Gamma = \Sigma \cup (\Sigma \times Q)$. These are the symbols that appear in a tableau of M's computation on x (as we used in the reduction showing that CIRCUIT-SAT was NP-complete). Let $m = n^k + 1$, and define the graphs $G_1, G_2, \ldots, G_m$ to each be the complete directed graph in $|\Gamma|$ vertices. We identify the vertices of each G_i with Γ. Now we produce the list P consisting of all pairs $((v_i, v_i'), (v_{i+1}, v_{i+1}'))$ for $v_i, v_i' \in V_i$ and $v_{i+1}, v_{i+1}' \in V_{i+1}$ for which

- $v_i, v_{i+1}' \in (\Sigma \times Q)$ and $v_{i+1}, v_i' \in \Sigma$, or
- $v_i', v_{i+1} \in (\Sigma \times Q)$ and $v_{i+1}', v_i \in \Sigma$,

and adjacent cells v_i, v_{i+1} in one row of the tableau become v_i', v_{i+1}' in the next row of the tableau, according to M's transition function.

If we take $v = (v_1, \ldots, v_m)$ to be the sequence in Γ^m corresponding to the first row of the tableau (of M with input x, padded with blanks), then there is exactly one reachable state (since M is deterministic), which corresponds to the second row of the tableau. From that state, there is exactly one reachable state corresponding to the third row of the tableau, and so on. Thus if M eventually accepts, there will be a deadlock state (since M halts upon accepting, and there are no further rows of the tableau from that point). The other alternative (recall our initial modification to M) is for M to loop forever; consequently we will never reach a deadlock state.

This completes our reduction from a generic language L in PSPACE to REACHABLE DEADLOCK.

3. We want to compute $f(g(x))$ in logspace. We cannot compute $g(x)$ and then evaluate f on it, because we don't have enough storage space to write-down the intermediate result $g(x)$, which may be polynomially long in the input length $|x|$. Note that $g(x)$ can *only* be polynomially long in $|x|$ (not more), because it is produced by a logspace machine, which therefore has only polynomially many configurations. It does not loop, so it can only run for polynomially many steps before halting.

   Let M_f and M_g be Turing Machines that compute f and g in logspace, respectively. We build a new Turing Machine that simulates M_f, and whenever M_f would have read the i-th bit of its input, we pause (remembering M_f's state in $O(\log |x|)$ bits), and simulate M_g on input x (ignoring what it would have written on its output tape, and remembering only the position of the head on the output tape) *but keeping track of what it writes in the i-th bit* of $g(x)$. After this simulation of M_g completes, we have the necessary bit to continue simulating M_f. At every point in this simulation we need only remember the state of one of the two machines (requiring space $O(\log |x|)$), while running the other, so the total space is $O(\log |x|)$ as desired.