
Problem 1 (grade completely)

To show that the adversary's task is in NP $\cap$ coNP, we will show that the language $L = \{y : g(f^{-1}(y)) = 1\}$ is in NP and coNP.

L is in NP
For any y in L, the witness w is the pre-image x such that $f(x) = y$. Since f is a one-way permutation, f^{-1} exists for every y. The verifier V would do the following: Take the input (y, w). Check that $f(w) = y$ (this is possible in polynomial time since f is length-preserving and we can compute f in polynomial time). Compute $g(w)$ (which is also possible in polynomial time). Accept if $g(w) = 1$. Since all steps of V can be done in polynomial time, L is in NP.

L is in coNP
For any y not in L, the witness w would again be the pre-image x such that $f(x) = y$. The verifier V' for $\overline{L}$ would do the following: Take the input (y, w). Check that $f(w) = y$ (verifiable in polynomial time). Compute $g(w)$ (verifiable in polynomial time). Accept if $g(w) = 0$. This verifier V' shows that the complement of L, $\overline{L}$, is in NP, which implies L is in coNP.

Therefore, L is in NP $\cap$ coNP because we have constructed polynomial-time verifiers for both L and its complement $\overline{L}$.

Problem 2

a.
To show that DEADLOCK is NP-complete, we must show that it is in NP and is NP-hard. We can see that DEADLOCK is in NP because given any state, we can simply check that there are no such pairs, $((v_i, v_i'), (v_j, v_j'))$, in the given list P for all i, j. This process occurs in polynomial time.

To show that DEADLOCK is NP-hard, we can reduce it from 3-SAT. Let us consider some instance ϕ of 3-SAT with m total clauses. We can map ϕ to H where $H = (G_1, G_2, ..., G_m, P)$. Each G_i is a directed graph and P is the list of pairs of edges. For each clause in ϕ, the associated graph, G_i, is a directed graph relying on 3 vertices where each vertex represents a literal in the clause and the directed edges representing the transition from one state to another. Then, for each literal x_i, if the negation, $\neg x_i$, also exists in some separate clause, then we can add the pair of vertices, $((x_i, l_1), (\neg x_i, l_2))$, to P where l_1 is a literal in the same clause as x_i and l_2 is some literal in the same clause as $\neg x_i$. Each clause is generated in polynomial time and the pairs of transitions are added to P also occur in polynomial time.

To show that yes maps to yes, let us consider that ϕ has satisfying assignment, H. We want to show that H has a deadlock state. Since ϕ is satisfiable, it follows that there is at least one true literal in every clause. Thus, the vertices that corresponds to each true literal in their respective graph make up the state. Now, there are several cases to consider. In the case that ϕ contains no $\neg x_i$ for every x_i, P will be an empty set. It follows that there are no available transitions and H will have a deadlock state. In the case that ϕ contains both x_i's and $\neg x_i$'s but each in their own clause, P is again an empty set and H will have a deadlock state. In the final case that ϕ has both x_i's and $\neg x_i$'s in different clauses, we know that either x_i or $\neg x_i$ can be true but not both. Thus, let's suppose that x_i is true. By our reduction from ϕ to H, we can consider two processes where their respective transitions can be defined as (x_i, l_1) and $\neg x_i, l_2)$ with the pair of transitions, $((x_i, l_1), (\neg x_i, l_2)) \in P$. However, there are no available transitions because the state contains either x_i or $\neg x_i$ but not both and as a result, there we have a deadlock state. Thus, we have shown that yes maps to yes.

To show that no maps to no, let us consider some instance H that has a deadlock state. We can then choose the vertices/literals that correspond to the vertices specified by the deadlock state. Since the current state consists of one vertex from each graph, it follows that we have at least one literal in each clause that is true. It also follows that we have an x_i and $\neg x_i$ that both cannot be in the state because otherwise, H would not have a deadlock state. Thus, we have shown that no maps to no.

Since we have shown a valid reduction from 3-SAT to DEADLOCK, it follows that DEADLOCK is NP-complete.

b.

To show that REACHABLE DEADLOCK is PSPACE-hard, we will reduce it from some language L in PSPACE. We can let M be the Turing Machine that decides this language L. We can represent M as a tableau where the rows represent the current state. Our initial state will be the first row of M. The language will consist of the tape language as well as the pairs, (a, q), where a is some tape symbol and q is a state in M. The transition function allows the head to move either left, right, or stay at its current position. Suppose our initial configuration is $x, (\alpha, \beta), y$.

- Left transition: $x, (\alpha, \beta), y \to (x, \beta'), \alpha', y$
- Right transition: $x, (\alpha, \beta), y \to x, \alpha', (y, \beta')$
- Stay transition: $x, (\alpha, \beta), y \to x, (\alpha', \beta'), y$

where α' is some symbol in the tape alphabet and β' is some state. The accept state does not have a transition and the reject state transitions to some other state before transitioning directly back to the reject state (creating an infinite loop). We can represent a graph, G_i, by a column of the tableau, and we can let P be the transition functions. Additionally, each cell of the tableau represents a state of the graph of the column it is in. Thus, each row would represent the overall state of all graphs. For each row, the reduction maps at most n^m cells to at most n^m states, where n is the input length to M. There are both a constant number of rows and vertices. Thus, this reduction is polynomial.

To show that yes maps to yes, let us consider some string $s \in L$ that is accepted by M. From our reduction, it follows that there are no further transitions from an accept state. Thus, a deadlock is reached.

To show that no maps to no, let us consider some string $s \notin L$ that is rejected by M. From our reduction, it follows that it will enter an infinite loop from the reject state to some other state back to the reject state. Thus, a deadlock is never reached.

Since we have shown a valid reduction from some language L in PSPACE to REACHABLE DEADLOCK, it follows that REACHABLE DEADLOCK is PSPACE-hard.

Problem 3

To show that a function h, which is the composition of two functions f and g both computable in $O(\log n)$ space, is also computable in $O(\log n)$ space, we can argue as follows:

Since g is computable in $O(\log n)$ space, there exists a Turing machine M_g that computes $g(w)$ using $O(\log n)$ space. Similarly, since f is computable in $O(\log n)$ space, there exists a Turing machine M_f that computes $f(u)$ for some string u using $O(\log n)$ space. For the composition $h(w) = f(g(w))$, we can construct a Turing machine M_h by chaining the computation of M_g and M_f as follows:

- Use M_g to compute $g(w)$ with the input w on tape 1. The result $g(w)$ is written to a portion of tape 2, using $O(\log n)$ space.
- Once $g(w)$ is computed, we can use M_f to compute $f(g(w))$. Machine M_f will treat the segment of tape 2 containing $g(w)$ as its input tape.
- The computation of $f(g(w))$ will use another segment of tape 2, also bounded by $O(\log n)$ space.

Because $\log n + \log n = O(\log n)$, the total space used on tape 2 is still within $O(\log n)$ space. It is crucial that after M_g finishes computing $g(w)$, the space it used can be reused by M_f for computing $f(g(w))$, thus ensuring that the total space used does not exceed $O(\log n)$. Note that this argument relies on the fact that the logarithm function is closed under addition, which is a key property allowing us to state that the space complexity of the composition is the same as that of the individual functions. The output of $h(w)$ will be written to tape 3. This shows that h is computable in $O(\log n)$ space.