2) If an NP-complete problem is PSPACE-complete, then NP = PSPACE.

We know that NP is a subset of NPSPACE and NPSPACE = PSPACE. We need to prove that PSPACE is a subset of NP.

Let L be a problem in PSPACE, let A be the NP-complete problem that is also PSPACE-complete. L <=P A (A is PSPACE-complete). A in in NP. L is in NP. We just proved that if L is in PSPACE then L is in NP, PSPACE is a subset of NP.

3) ALBA is PSPACE-complete. My solution: reduce all PSPACE languages to ALBA. Let L be a PSPACE language. We have TM M that decides L in space f(n) where f(n) is bounded by O(nk). Create M’ and w where M’ decides w and the head for M’ never leaves the cells for w. M’ = M. Let w = the input to M, but then padded with f(n) “special blank” symbols. M’ acts just as M but it uses the “special blank” as the blank symbol in the transition function. We created M’ and w using O(f(n)) space.

Show ALBA is in PSPACE. Given an M and w. Simulate M on w, reject if M loops. Count the total number of steps M takes an M loops of it takes more than d^(|w|) \* q \* |w|.

1) Show that if A is NP-complete then A is complete for co-NP.

Since A is NP-complete, given any L in NP, there is function f: Sigma\* → Sigma\*, x in L if and only if f(x) is in A.

Consider a language B in co-NP. There exists f such that for all x, x is in B if and only if f(x) is in A. That is equivalent to x is not in B if and only if f(x) is not in A. Equivalent to x is in B if and only if f(x) is in A. So we reduced B to A-complement. Since A is in NP, A-complement is in co-NP. A-complement is co-NP complete.

Examples of co-NP-complete problems:

1) complement of 3-SAT. The “string” is either not a 3-SAT instance, or it is a 3-SAT instance that has no solution.

We do not know how to have a short witness to a formula having no solution. So, we don’t have a polynomial time verifier for co-NP-complete problem.

Open question: Is NP = co-NP??

We will prove that NL = co-NL.

PATH is complete for NL. (Given a directed graph and 2 vertices, is there a path from the first to the second?)

PATH is complete for co-NL. (Given a directed graph and 2 vertices, there is no path from the first to the second vertex.)

We prove ~~PATH~~ is in NL.

(Given G, s and t)

Attempt 1: Nondeterministically guess a path from s to t. Verify the path. If we fail to reach t In n steps, accept (there is no path), If it reaches t within n steps reject.

What’s wrong? If it is possible to accept, a nondeterministic algorithm will always make guesses that lead to the accept state. It will guess something that is not a valid path from s to t. (Maybe it goes to a deadend, maybe it loops.)

Attempt 2: Guess the number of vertices that we can reach from s. Run through all vertices, and for each vertex either guess we can’t get to that vertex from s or guess we can and verify that by guessing the path to that vertex and checking. At the end, accept if (1) t was not one of the vertices we guessed we could reach from s and (2) the number of vertices we did reach from s is equal to the number we guessed at the start of the algorithm.

What’s wrong? The algorithm could “guess” fewer vertices than can actually be reached from s.

Algorithm idea:

For each k, we will make the algorithm determine how many vertices can be reached from s in k or fewer steps.

K= 0: There is only one vertex, s

For k = 1, the number of vertices we can reach from s can be checked easily, there must be an edge from s to that vertex.

Loop k = 0 to n-1:

let ck = the number of vertices we can reach in up to k steps from s.

loop through each vertex u, nondeterministically guess if we can reach that vertex from s within k steps.

If we guess we can, nondeterministically guess the path and verify the path (checking each edge exists, and keep a counter of how long the path is)

Add 1 to a counter (d)

For each vertex we can reach by an edge from u, add 1 to ck+1.

At end of the loop, verify the the counter d = ck.

We need one more trick to make sure we don’t double count vertices when incrementing ck+1. The trick is to embed this algorithm in one more loop.