1) If P = NP then every language except the empty language and all strings is NP-complete.

L, x in L, y not in L.

Consider a NP-complete problem A. We need to reduce A to L. A can be solved in polynomial time. The reduction f is as follows. On input w, solve w to see if it is in A. If w is in A, then f(w) = x. If w is not in A, then f(w) = y. Now w is in A if and only if f(w) is in L.

2) Prove that LPATH is NP-complete. We are given a graph G and an integer k, there is a simple path of at least k vertices in G. To show LPATH is in NP, guess a sequence of k vertices and verify that this is a path.

Ham. Cycle <= LPATH. Take G, this input to Ham Cycle problem and create G’ that is G plus 3 new vertices. If G’ has a path of n+3 vertices then G has a Ham. Cycle.

3) Set Splitting. We have a collection of sets of elements, we need to color the elements with 2 colors (R, B) such that every set is not colored with just one color.

Set Splitting in in NP: Guess a coloring to each element and in linear time check that every set has 2 colors.

3-SAT <= Set Splitting. Given a set of m clauses over n variables, 3 literals per clause. Create a set of elements. I will have m + n + 5 sets over 2n + 6 elements.

For each literal x, (NOT x). create an element. Make a set { x, NOT x}. For each clause (a, b, c) in the 3-SAT formula, make a set {a, b, c, F}. The goal is that F will be colored with the “false” color. Then at least one of a, b, or c must be colored with the “true” color. Create 2 elements, T, F, and a set {T, F}. Create four new elements. (X, NOT X, Y, NOT Y), and create the sets: {X, Y, F}, {X, NOT Y, F}, {NOT X, Y, F}, {X, NOT X, F}, {Y, NOT Y, F}. This allows all combinations of setting true/false to X and Y except both false. X and Y must be set the same color.

X NOT X

Y NOT Y

(a,b,c,F), (d,e,f,F), (a,b,f,F) ….

PSPACE: The set of language that can be decided my a deterministic single tape turing machine using at most a polynomial (in terms of the input size) number of cells.

PSPACE = NPSPACE

A language L is PSPACE-complete (NPSPACE-complete) if

a) L is in PSPACE

b) For every language A in PSPACE then A <=P L.

(This is a polynomial time reduction.)

TQBF (totally quantified boolean formula). We take a boolean formula (ex: SATISFIABILITY or 3-SAT) and place a quantifier on all n of the variables.

(There exists x, for all y, for all z, there exists w … (x OR (NOT Y) OR …))

a) Show TQBF is in PSPACE.

For each variable in order, try setting the variable to T, recurse, when it returns, set the variable to F and recurse on the subformula. The height of the recursion is n.

b) For any PSPACE language L we can reduce L to TQBF. If L is in PSPACE, there is a deterministic Turing machine M that decides L. M uses at most PSPACE, takes at most 2^(O(nk)) time. We consider the states (configurations) of M. Figure out if M can go from the initial configuration to an accepting configuration (assume just one configuration) in t steps (but t can be exponential in size!)

Create a variable for each cell in the configuration, the head location, the state (just like Cook’s theorem). Create a formula that says if it is legal to go from one configuration to the next.

Create a formula that says: Can I go from configuration c1 to configuation c2 in t steps? This will be created recursively. (Pc1, c2, t  = There exists an x Pc1, x, t/2 AND Px, c2, t/2).

There exists an x such that for all (c1,x), (x, c2) , we have P(c1,x), (x, c2), t/2)