CS 280 Homework: The Polynomial Hierarchy
Due in class Tuesday May 1st. (No late assignments accepted.)
This problem set will be graded as if it were a quiz.

1. Problem 10.12.

2. Problem 10.13.

3. Problem 10.14. In class we showed that by the "oracle" definition of the polynomial hierarchy, the class Σ_2^P is defined as NP^{NP}, or equivalently, NP^{SAT}. We then tried to give an intuition as to why NP^{NP} should be the same as the alternating machine definition of $\Sigma_2 P$. This problem asks you to more formally prove that the why NP^{NP} is the same as the alternating machine definition of $\Sigma_2 P$.

4. A Boolean formula can be thought of as a blueprint for an electronic circuit, where the propositions are the inputs and the value of the formula, 1 or 0 for true or false respectively, is the one-bit output of the circuit. Consider the following problems that arise in circuit design, and determine where they fall in the polynomial hierarchy, using just the $\Sigma_i P$ and $\Pi_i P$ classes. For each, describe an i-alternating Turing Machine that solves the problem, where the machine accepts if the answer is "yes" to the problem. Your choice of complexity class should always be the lowest (least powerful) class that you can prove contains the problem. Once you find an initial solution, think carefully about whether there might not be another way to solve the problem using a less power alternating TM. Keep in mind the following definitions and facts about Boolean logic:

- A formula F is satisfiable if some assignment of values to its propositions makes it true, and is unsatisfiable otherwise.
- A formula F is valid if it is true under every assignment of values to its propositions.
- A formula F is valid if and only if the formula $\neg F$ is unsatisfiable.

a. Given two formulas F_1 and F_2, determine if they compute the same one-bit function.

b. Given a formula F, determine if F is minimal - that is, determine if there is no other smaller formula that computes the same function. (The size of a formula is the total number of characters needed to write down the formula. You can assume that any proposition can be written down as a single character.)

c. Given a formula F, determine if the output of F is ever 0.

d. There are many situations in which one wants a device to continue to operate properly even if some of its sub-components fail. Let us say we can "corrupt" a given formula by changing one or more of its logical connectives - that is, changing an "and" to "or" or vice-versa, or changing a "not" to a new connective, "identity", which has the same value as its argument. We will say that

a formula is *k-fail-safe* it computes the same function even if up to k corruptions are introduced. Given a formula F and an integer k, determine if F is k-fail-safe.

e. Determine if k corruptions can cause a given formula F to always output 1.

f. Suppose a formula is corrupted. It is always possible to "repair" the formula, that is, make it compute the original function, by corrupting the corrupted formula: new each corruption would simply undo one of the original corruptions. However, there may a way of repairing the formula by making *fewer* new corruptions than originally occurred. The new formula would not be identical to the original formula, but would still compute the same function.

Let us say that a formula is *k-j-repairable* if the effect of up to *k* corruptions can always be repaired by making no more than *j* further changes, for $k \geq j$.

i. Redefine the notion of k-fail-safe in terms of k-j-repairable.
ii. Describe the complexity class and the corresponding i-alternating Turing Machine for the problem of given a formula F and integers *k* and *j*, determines if F is k-j-repairable.