Exercise 1: Alternating Turing machines with negations

Let us define an alternating Turing machine with negations as a Turing machine where the set of non-halting states is partitioned into the set of existential states, the set of universal states and the set of negation states. Moreover there is the restriction that each configuration on a negation state has exactly one successor configuration. Remark that we do not require that the machine always halts.

For such a machine $M$ we define the set of eventually accepting configurations, and the set of eventually rejecting configurations as the minimal sets of configurations satisfying the following conditions:

- if $C$ is an accepting configuration, then $C$ is eventually accepting;
- if $C$ is an existential configuration and there exists a successor configuration $C'$ of $C$ (i.e., $C \rightarrow_{M} C'$) which is eventually accepting, then $C$ is eventually accepting;
- if $C$ is a universal configuration, and all successor configurations $C'$ of $C$ are eventually accepting, then $C$ is eventually accepting;
- if $C$ is a negation configuration and the (unique) successor configuration $C'$ of $C$ is eventually rejecting, then $C$ is eventually accepting;
- if $C$ is a rejecting configuration, then $C$ is eventually rejecting;
- if $C$ is an existential configuration and all successor configuration $C'$ of $C$ are eventually rejecting, then $C$ is eventually rejecting;
- if $C$ is universal configuration, and there exists a successor configuration $C'$ of $C$ which is eventually rejecting, then $C$ is eventually rejecting;
- if $C$ is a negation configuration and the (unique) successor configuration $C'$ of $C$ is eventually accepting, then $C$ is eventually rejecting.

The machine accepts an input $x$ iff the initial configuration on input $x$ is eventually accepting. The language accepted by an alternating Turing machine with negations $M$ is the set of all $x$ accepted by $M$.

Prove that any alternating Turing machine $M$ with negations can be simulated by an alternating Turing machine $M^*$ without negations, with no extra cost in time or space. More precisely prove that there exists a configuration reachable in $n$ steps and using $m$ working tape cells in $M$ iff there exists a configuration reachable in $n$ steps and using $m$ working tape units in $M^*$. Do not assume any space or time bound on $M$. 
Exercise 2: Alternating logarithmic time vs logarithmic space
Show that \( \text{ATIME}(\log n) \neq \text{L} \).
\( \text{Hint: show that the language of palindromes is in one class but not the other} \)

Exercise 3: Yet another padding argument
Show that \( \text{EXPSPACE} = \text{AEXPTIME} \). (skip this exercise if you have seen it in class)

Exercise 4: Linearly and logarithmically bounded alternations
Let \( \text{AP}(O(n)) \) (resp. \( \text{AP}(O(\log n)) \)) be the class of problems which can be decided by an alternating polynomial time Turing machine whose computations have a linear (resp. logarithmic) number of alternations (in the size of the input).

- Is \( \text{QBF} \) in \( \text{AP}(O(n)) \) ? In \( \text{AP}(O(\log n)) \) ?
- Can we conclude \( \text{PSPACE} = \text{AP}(O(n)) \)? \( \text{PSPACE} = \text{AP}(O(\log n)) \) ?

Exercise 5: \( \text{P} \)-complete problems
Show the following problems to be \( \text{P} \)-complete:

- INPUT: \( G \) a context-free grammar
  - QUESTION: \( \mathcal{L}(G) = \emptyset \) ?

- INPUT: \( G \) a context-free grammar, and \( w \) a word
  - QUESTION: \( w \in \mathcal{L}(G) \) ?
  \( \text{Hint: use the Chomsky Normal Form: given a grammar, one can compute in logarithmic space an equivalent grammar with production rule of the form} \ X \rightarrow YZ; X \rightarrow a \text{ or } S \rightarrow \varepsilon. \)

- INPUT: A planar circuit \( C \)
  - QUESTION: Does \( C \) evaluate to true ?

Exercise 6: Closure under morphisms
Given a finite alphabet \( \Sigma \), a function \( f : \Sigma^* \rightarrow \Sigma^* \) is a morphism if \( f(\Sigma) \subseteq \Sigma \) and for all \( a = a_1 \cdots a_n \in \Sigma^* \), \( f(a) = f(a_1) \cdots f(a_n) \) (\( f \) is uniquely determined by the value it takes on \( \Sigma \)).

1. Show that \( \text{NP} \) is closed under morphisms, that is: for any language \( L \in \text{NP} \), and any morphism \( f \) on the alphabet of \( L \), \( f(L) \in \text{NP} \).
2. Show that if \( \text{P} \) is closed under morphisms, then \( \text{P} = \text{NP} \).

Exercise 6: Unary Languages

1. Prove that if a unary language is \( \text{NP} \)-complete, then \( \text{P} = \text{NP} \).
   \( \text{Hint: consider a reduction from SAT to this unary language and exhibit a polynomial time recursive algorithm for SAT} \)

2. Prove that if every unary language in \( \text{NP} \) is actually in \( \text{P} \), then \( \text{EXP} = \text{NEXP} \).
   \( \text{Hint: remember we can always restrict our attention to Turing machines on alphabet \{0,1\}.} \)

3. Show the converse.