CISC 462 — Computability and Complexity
Problem Set 4
Solutions

Queen's University, Winter 2017

1. First, we observe that an NFA with n states has an equivalent DFA with at most 2^n states. Obviously, we can't compute the DFA, since it would exceed the polynomial space bound. Instead, we observe that we simulate the run of NFAs in polynomial space (in fact, we showed that this problem is in **NL**). Thus, we can simulate a run of both NFAs on the same word in polynomial space and check if the two differ on any input string.

   However, in order for this algorithm to work, we need to ensure that we only test a finite number of strings. Indeed, we use the observation that for an n-state NFA, there are at most 2^n possible subsets of states that can be reached. Therefore, if A has n_A states and B has n_B states, we only need to test strings of length up to $2^{n_A}2^{n_B}$, since otherwise, we can find a shorter string where they would have disagreed.

   We can use a nondeterministic algorithm to guess such a string and each branch of computation will then clearly require at most polynomial space to simulate the run of both A and B. Thus, we have shown that EQ_{NFA} is in **NPSPACE**. However, by Savitch's theorem, **NPSPACE** = **PSPACE**, so EQ_{NFA} is in **PSPACE**.

2. Every regular language L can be recognized by a deterministic finite automaton A. We can simply observe that a DFA is a 2-tape Turing machine with a read-only input tape and a constant-size work tape, and this is clearly deterministic and log-space bounded.

   Alternatively, we can show that we can construct a log-space bounded deterministic machine that simulates a DFA A, since all it needs to keep track of is the current input symbol and the current state. This can clearly be stored in logarithmic space and since the DFA is deterministic, the simulation will be deterministic as well.

3. To see that $A_{LBA} \in$ **PSPACE**, we observe that simulating A on w will require at most $O(|w|)$ space, since A is an LBA. Linear space is clearly polynomial, so the problem is in **PSPACE**.

   Now we will show that A_{LBA} is **PSPACE**-hard. Let $L \in$ **PSPACE**. Suppose L is recognized by a Turing machine M that runs in n^k space. Given an input word w, we want to construct an LBA M' and input word w' such that w' is recognized by M' if and only if $w \in L$. We construct M' to simulate M while treating the symbol # as a blank symbol $\square$. Then we construct our input word $w' = w\#^{|w|^k}$. This construction is clearly possible to perform in polynomial time.

   Since M' just simulates M, it is not difficult to see that M' accepts w' iff M accepts w. Now, we simply need to verify that M' is an LBA. We observe that since M uses

n^k space, M' also recognizes w' in n^k space. However, $w' = w\#^{|w|^k}$, which means that n^k is $O(|w'|)$. Thus, M' recognizes w' in $O(|w'|)$ space, as required.

4. First, we show $NE_{NFA} \in \mathbf{NL}$. Recall that for an NFA A, $L(A) \neq \emptyset$ if it accepts some word. In other words, there needs to be some path from the initial state of A to a final state of A. We can test this by treating A like a directed graph and checking for each final state of A whether it is reachable from the initial state. This is clearly in **NL**.

   Now, to show hardness, we reduce PATH to NE_{NFA}. Given a directed graph $G = (V, E)$ and vertices s and t, we construct an NFA $A_G = (Q, \Sigma, \delta, q_0, F)$ that is non-empty if and only if G contains a path from s to t. We will construct A_G to be an NFA over the unary alphabet $\Sigma = \{a\}$. We let our state set be $Q = V$. For each edge $(u, v) \in E$, we add the transition $v \in \delta(u, a)$. Then we set our initial state $q_0 = s$ and our final states to be $F = \{t\}$. Since we are basically reproducing the graph as an NFA, it is not difficult to see that the construction can be performed in logarithmic space.

   To see that our construction is correct, we will first suppose that G contains a path from s to t. Then by our construction, there is a series of transitions in A_G from $q_0 = s$ to the state $t \in F$ that correspond to the edges in the path from s to t. Therefore, there must be some word that A_G accepts by following the path and therefore A_G is non-empty.

   Now suppose that A_G is nonempty. Then this means that there is some path from the initial state $q_0 = s$ to the sole final state $t \in F$. From our construction, this is just a path that already exists in G.

5. Let L be a language that is **PSPACE**-hard. Then every language $L' \in \mathbf{PSPACE}$ is polynomial time reducible to L and we have $L \leq_P L'$. Recall that $\mathbf{NP} \subseteq \mathbf{PSPACE}$. In other words, if $L'' \in \mathbf{NP}$ then $L'' \in \mathbf{PSPACE}$. Then since L is **PSPACE**-hard, we have $L \leq_P L''$ for every $L'' \in \mathbf{NP}$. But this means that every language in **NP** is polynomial time reducible to L and therefore L is **NP**-hard.

6. We will show that L is in **NPSPACE**. First, we can immediately *reject* if $|u| \neq |v|$.

   Now, let $|u| = |v| = n$ and consider the set of words of length exactly n, denoted Σ^n. Beginning from u, we will nondeterministically search for v. At each step on a word $w_1 \in \Sigma^n$, we will guess a word w_2 which differs from w_1 by exactly one symbol and we will check that M accepts w_2. Then we repeat this process with w_2 only if it is accepted by M. This guess and check clearly requires polynomial space, since the space required is only as much to store the current word w_1 and the space required to check w_2 on the DFA A.

   If we begin this process on u and some computation branch reaches v, we *accept*. Otherwise, if this process exceeds Σ^n steps, then v is unreachable from u and we *reject*.