Computability & Complexity HW 7

Drew Blount 11/26/2014

1 A, the language of properly-nested parentheses, is in L.

Here's a description of a machine that decides A in L-space:

On input w from $\{(,)\}^*$, read w from left to right, keeping track of the difference between how many left parens you've seen, and how many right parens you've seen. Writing this down takes at most $lg(|w|)$ space, because you'll never have to store a number larger than w's length.

The machines rejects if its counter ever goes below zero, because in that case it has seen more closing parens than opening ones. It accepts if it reaches the end of w and its counter is zero, meaning that all opening parens were paired with a closing one; else it rejects.

2 A_{NFA} is NL-complete.

To see that $A_{NFA} \in$ L, note that an NFA requires no memory, only state, to operate. A TM simulating an NFA only needs to keep track of two things: The number of states it has visited, to make sure the machine does not get caught in loops and thus is a decider; and the current state that it is in. This first requirement takes $lg(|Q|)$, where Q is the set of the NFA's states, and is thus a subset of the TM's input string. The second requirement takes constant space.

To show that any language in L can be reduced to A_{NFA}, I will explain how $PATH$ can be log-space reduced to A_{NFA} with the log-space computable function $f(< G,s,t >)$, which constructs the following NFA:

- an empty alphabet
- a state for each vertex in G, with the state corresponding to s as the start state and the state corresponding to t as the only accept state
- An epsilon transition replicating each directed edge in the graph

This reduction function uses constant space, as it doesn't need to store anything, but simply copy and reformat information from the read-only input tape to the write-only output tape.

Now, if machine M' decides on input $(< M,w >)$ whether M is an NFA that accepts w, and the above reduction function is labelled f, $M(f(< G,s,t >),\varepsilon)$ solves $PATH$. This is true because when there is a path from s to t in G, exactly that path can be taken by ε transitions in the NFA constructed by f to get from the start state to the only accept state.

3 E_{DFA}, the language of empty DFAs, is NL-complete

To see that $E_{DFA} \in$ L, consider the following log-space computable function that reduces E_{DFA} to $PATH$:

- Convert the DFA into one an NFA with only one accept state. This step itself is log-space computable, as it requires only constant space on top of copying from an input tape to an output tape (which will be linear in the length of the input).

- Convert that NFA into a graph G, with every transition in the NFA (regardless of symbol) is a directed edge in the graph. Also, define the vertex s as the vertex corresponding with the NFA's start state, and the vertex t as the NFA's accept state. Again, this step is itself log-space computable as it needs only constant space on top of copying one tape to another.

- Deciding whether $< G,s,t > \in PATH$ says whether there was any possible string that would have been accepted by the original DFA, and so negating this output decides E_{DFA}.

To see that E_{DFA} is NL-hard, consider the fact that any NFA can be reduced to a DFA, and that the NFA constructed by function f to encode $PATH(< G,s,t >)$ in problem 2, describes a nonempty language only in the case that there is a path in G from s to t (in this case the language contains only the empty string). Thus, if we can reduce an NFA to an equivalent DFA in log-space with function g, and machine M decides E_{DFA}, then $\neg M(g(f(< G,s,t >)))$ decides $PATH$.

Unfortunately, the transformation from NFA to equivalent DFA presented in Theorem 1.39 in Sipser is not log space, as the DFA produced has an exponential number of states relative the NFA. I'm not sure how to get around this issue.