CS 262: Computational Complexity *Instructor: Sandy Irani*

Homework 3

Due: April 25, 2018

1. A directed graph $G = (V, E)$ is *strongly connected* if for every pair of verties (x, y), there is a directed path from x to y and a directed path from y to x. Define STRONGLY-CONN to be the language consisting of all graphs that are strongly connected. Either show that this problem is in **L** or show a complexity consequence that results if this problem is in **L**.

   We will show that ST-CONN $\prec$ STRONGLY-CONN, by a log-space reduction. Since ST-CONN is **NL**-complete, this establishes that STRONGLY-CONN is also **NL**-complete, so if STRONGLY-CONN $\in$ Ł, then **NL** = Ł.

   The reduction will take as input $G = (V, E), s, t$ and will produce a new graph $G' = (V', E')$ such that there is a path from s to t in G if and only if G' is strongly connected. The vertex set of the two graphs is the same: $V = V'$. All the edges in E are also in E'. In addition, for every vertex $v \in V$, the edges (v, s) and (t, v) are added to E'. This reduction can be done easily on log-space. The reduction copies all the vertices and edges in G from the input tape to the output tape (requiring no extra workspace). Then on the work tape, enumerate each vertex v and write the two new edges (v, s) and (t, v) to the output tape.

   Now suppose that there is a path in G from s to t: $\langle s, v_1, v_2, \ldots, v_{l-1}, t\rangle$. Then for every two vertices x and y, the following path is a path from x to y in G': $\langle x, s, v_1, v_2, \ldots, v_{l-1}, t, y\rangle$. Therefore G' is strongly connected.

   Suppose that G' is strongly connected. Then there must be a path from s to t in G'. It is possible to remove cycles from the path to get a simple path (that doesn't repeat any vertices) from s to t in G': $\langle s, v_1, v_2, \ldots, v_{l-1}, t\rangle$. Since the path is simple, none of the verices v_1 through v_{l-1} can be s or t. Therefore, the path does not contain any of the new edges which have the form (v, s) or (t, v). Since the path from s to t does not contain any of the new edges, there must also be a path from s to t in G.

2. A *strong* nondeterministic Turing Machine has, in addition to q_{acc} and q_{rej} states, a special state $q_?$. The state $q_?$, like q_{acc} and q_{rej}, is a terminating state in that there are no transitions out of state $q_?$. A strong NTM *accepts* its input if all computation paths lead to q_{acc} or $q_?$ states. A strong NTM *rejects* its input if all computation paths lead to q_{rej} or $q_?$ states. Also, on each input, there is at least one computation path that leads to q_{acc} or q_{rej}. In order for a language L to be decided by a strong NTM, the NTM must accept or reject every input x according to whether $x \in L$. Note that the fact that a strong NTM decides a language implies that there can be no input that has compuation paths that lead to both q_{acc} and q_{rej}.

Show that the class of languages decided by a strong nondeterministic Turing Machine in polynomial time is exactly **NP** $\cap$ **co-NP**.

First assume that language L is decided by a strong NTM M. We will show a polynomial time NTM that decides L and a polynomial time NTM that decides co-L, which will establish that $L \in$ **NP** $\cap$ **co-NP**. By definition, on every input x, every computation path of M is at most polynomial in length. Furthermore, if $x \in L$, then every computation path leads to $q_?$ or q_{acc}, and at least one computation path leads to q_{acc}. Similarly, if $x \notin L$, then every computation path leads to $q_?$ or q_{rej}, and at least one computation path leads to q_{rej}. Consider a new NTM M' that performs exactly as M, except that whenever M is about to transition to $_?$, M' will transition to q_{rej} instead. M' is a normal (not strong) NTM that decides L. Similarly, consider a new NTM M" that performs exactly as M, except that whenever M is about to transition to $_?$, M" will transition to q_{rej} instead. In addition, when M" swaps the role of q_{acc} and q_{rej}, so when M is about to transition to q_{acc} then M" transitions to q_{rej} instead, and when M is about to transition to q_{rej} then M" transitions to q_{acc} instead. M' is a normal (not strong) NTM that decides co-L.

Now assume that there is an NTM M that decides L and an NTM M' that decides co-L. We will show a strong NTM (M") that decides L. On input x, first run M. If a computation path of M leads to an accepting state, then accept. If the computation path leads to a rejecting state, instead of transitioning to q_rej, run M' on input x. If a computation path of M' leads to a rejecting state, transition to $q_?$ instead. If a computatino path of M' leads to an accepting state, then transition to q_{rej} instead. Suppose $x \in L$, then there is at least one computation path in M that leads to q_{acc}. This computation path also leads to q_{acc} in M". Any computation path in M that leads to q_{rej} is followed by a computation of M' on input x. Since $x \in L$ and M' decides co-L, all computation paths of M' on input x will be rejecting paths. These paths lead to $q_?$ in M". Suppose $x \notin L$. Then there will be no accepting computation paths of M on input x. All computation paths will be followed up by a computation of M' on input x. Since $x \notin L$ and M' decides co-L, there will be at least one accepting computation path of M' on input x. This will lead to q_{rej} in M". Any rejecting computation paths of M' on input x will end in $q_?$ in M".

3. An alternative definition of the class **NL** makes use of a Turing Machine with a special read-once tape. The head on a read-once tape starts at the left-most end of the non-blank symbols written on the tape and can only move to the right or stay in the same place (i.e. it can never move left). The alternative definition says that a language L is in **NL** if there is a deterministic Turing Machine R (called a *verifier*) with a special read-once tape and a polynomial p such that for every $x \in \Sigma^*$,

$$x \in L \Leftrightarrow \exists u \in \Sigma^{p(|x|)} \text{ such that } R(x,u) = 1,$$

where $R(x,u)$ is the output of R when x is placed on the input tape and u is placed on its special read-once tape and M uses $O(\log n)$ space on its work tape for every input x.

Prove that this definition is equivalent to the definition using non-deterministic Turing Machines discussed in class.

Suppose that there is a log-space NTM M that decides L. We will show a witness-version of L. The witness y is the same as in the proof for NP. The string consists of the set of all nondeterministic choices made by that decides L. The string y can be formatted as a sequence of triples: $(q, a, L), (q', b, R), \ldots$. Note that since M is a log-space NTM, M is also has a time bound of polynomial in the length of the input, which means that the length of y is also at most a polynomial in the length of the input. The head reading the string y never needs to move to the left. Each triple (q, a, L) can be stored in the state of R which simulates M on a separate work tape. R can then check if a give triple is a valid next step based on the state of M.

Now suppose that L is a language such that there exists an NTM R with a special read-once tape such that for every $x \in \Sigma^*$,

$$x \in L \Leftrightarrow \exists u \in \Sigma^{p(|x|)} \text{ such that } M(x, u) = 1.$$

Consider a new NTM M that does not have the work tape that R has. R will have a special tape cell which stores a guess for the current symbol of R's write-once tape. Whenever the head on R's read-once tape moves to the right, M will guess a new symbol for to put in the cell. Since the head on the read-once tape never moves left, M never has to remember what was written in the cell earlier in the computation. If there is a sequence of guesses for the tape cell that makes M accept, then the sequence of symbols is a string y that causes R to accept x. If there is a string y that causes R to accept, then that sequence of guesses will be an accepting computation of M on input x.