CS331 Practice Problem Set #4 Solutions

Your solutions to problems on undecidability are expected to be of the following rigor. Don't be unnecessarily lengthy by proving results covered in class. At the same time, it is important to include all relevant details, if you want to get full credit. In particular, pay attention to the direction of reductions.

1. Given two CFLs L_1 and L_2, we know that the problem of determining if $L_1 = L_2$ is undecidable (covered in class). In the following, we show that if we had an algorithm for determining if CFL $L = L^R$, we would also have an algorithm for the $L_1 = L_2$ problem – an impossible proposition.

   From our study of context-free languages, we know that given a CFG/PDA for a CFL L, there exists an algorithm (equivalently, Turing Machine M_1) to derive a CFG/PDA for L^R (problem 7 in Practice Homework 2). So, given CFLs L_1 and L_2 (i.e., corresponding CFGs or PDAs), we first transform them to CFLs L_1 and L_2^R. We next derive the CFG/PDA for $L_3 = L_1\#L_2^R$ from those for L_1 and L_2^R. The symbol # is assumed to be $\notin \Sigma_1 \cup \Sigma_2$. Once again, from our study of CFGs, we know there exists an algorithm (equivalently, Turing Machine M_2) to derive the CFG/PDA for L_3 from those for L_1 and L_2^R. Note that L_3 is a CFL if L_1 and L_2 are.

   We now claim that $\mathbf{L_1 = L_2}$ **if and only if** $\mathbf{L_3 = L_3^R}$.
   [**Only if:**] Trivial since $L_3^R = L_2\#L_1^R$.
   [**If:**] Suppose $L_1 \neq L_2$. There are two cases:

   (a) $L_1 \cap L_2^c \neq \phi$, so let $x \in L_1 \cap L_2^c$. For every string $y \in L_2$, $x\#y^R \in L_3$ but $\notin L_3^R$ since $L_3^R = L_2\#L_1^R$ and $x \notin L_2$.

   (b) $L_2 \cap L_1^c \neq \phi$. A similar argument shows that there are strings in L_3^R that are not in L_3.

   Thus, $L_1 \neq L_2$ implies $L_3 \neq L_3^R$, and our claim is proved.

   We have thus reduced the "$L_1 = L_2$" problem for CFLs to the "$L = L^R$" problem. Since the former problem is undecidable, the latter must be undecidable too.

2. This problem involves a reduction from the halting problem and is actually quite simple.

   Given an instance of the halting problem, i.e., (M, x), we construct a TM M' that contains all state transitions of M. Suppose M has n states labeled $q_1, \ldots q_n$. We now add n more states (labeled $s_1, \ldots s_n$) and corresponding transitions to M' as follows.

   - For every halting state q_h of M, we add transitions $\delta(q_h, X) = (s_1, \#, R)$, where $\#$ is assumed to be not in the tape alphabet of M, and X denotes any tape symbol other than $\#$.
   - For $1 \leq i \leq n$, we add $\delta(s_i, X) = (q_i, \#, L)$, where X denotes any tape symbol (including $\#$).
   - For $1 \leq i < n$, we add $\delta(q_i, \#) = (s_{i+1}, \#, R)$.

   Thus, if M halts in state q_h on input x, M' on x first simulates M and reaches state q_h. Subsequently it visits all its states in the following sequence: $q_h \rightarrow s_1 \rightarrow q_1 \rightarrow s_2 \rightarrow q_2 \cdots \rightarrow s_n \rightarrow q_n$.

   On the other hand, if M doesn't halt on input x, then M' never enters state s_1 (or for that matter any s_i).

   Thus, M halts on x if and only if M' on input x enters every state at least once. Effectively, we have reduced the halting problem to the problem of determining if a TM on a given input enters every state at least once. Since the former problem is undecidable, the latter must necessarily be undecidable too.

   It is not hard to see that the code for M' can be mechanically constructed given the code for M. Try this out taking the cue from Example 8.2 of Hopcroft and Ullman.