Homework 5

Due: Friday, 3/23/18 at 12 noon

Instructions on expected level of detail: Read “Terminology for describing Turing Machines”, starting on page 184 in the text (p. 156 in the second edition). In particular:

- For formal level description, you must use the standard definition of TM that we gave (single tape, deterministic), and provide all transitions in detail (using either a table/list or a state transition diagram). The only exception is that omitted transitions can be assumed to go to $q_{\text{rej}}$.

- For implementation level description, you are also allowed to use one of the TM variants we proved equivalent (e.g., a multi-tape TM or a non-deterministic TM), and you can use low-level subroutines like those we saw in class (for example, you can say “insert a blank and shift the rest of the tape content one space to the right” or “copy the content of the first tape onto the second tape”).

- For high level description you do not need to discuss tapes and head, just make sure each step in your algorithm is clear and well defined (you should be convinced that with enough time and patience that step could be translated to implementation-level description, and from there to formal level description).

For all three, you should add whatever explanations are necessary to make sure it is clear how your machine works (please be concise). In general, from now on, unless otherwise specified, a high-level description of a TM is sufficient.

Reminder: Recall from class that an input-output TM is defined similarly to a standard TM, except it has only one halting state $q_{\text{halt}}$ (replacing $q_{\text{acc}}$ and $q_{\text{rej}}$), and everything else is defined just like a standard TM, with a computation halting as soon as the machine goes to the $q_{\text{halt}}$ state. A given input-output TM computes the function (or partial function) $f$ if for any input $x$ on which $f$ is defined, when the machine starts from the start configuration on input $x$, the machine halts with $f(x)$ written on its tape, followed by blanks.

For simplicity, when you are asked to construct an input-output TM for a function that expects its input in a certain format (e.g., $\#\langle x \rangle$ for some positive integer $x$, like in problem 3), you may ignore the case that the input is of the wrong format. In examples like this it is easy to check whether the input is of the correct format and output an error message or a special symbol if it’s not, but you are not required to do so.
1. Consider the following TM:

\[ M = (\{q_0, q_1, q_2, q_{\text{acc}}, q_{\text{rej}}\}, \{0, 1\}, \{0, 1, \omega\}, \delta, q_0, q_{\text{acc}}, q_{\text{rej}}) \]

Describe the resulting language \( L(M) \) if we define \( \delta \) according to each of the following sets of rules (with all missing transitions going to \( q_{\text{rej}} \)):

(a) \( \delta(q_0, 1) = (q_1, 0, R) \); \( \delta(q_1, 0) = (q_0, 1, R) \); \( \delta(q_1, \omega) = (q_{\text{acc}}, \omega, R) \)

(b) \( \delta(q_0, 1) = (q_1, 0, R); \delta(q_1, 0) = (q_2, 1, L); \delta(q_2, 0) = (q_0, 0, R); \delta(q_1, \omega) = (q_{\text{acc}}, \omega, R) \)

2. (a) Let \( \Sigma = \{1, \#, \hat{\_}\} \). Provide a formal level description of a input-output TM computing the following function.

On input \( w \in \Sigma^* \), if \( w \) contains at least two \# symbols, insert a \( \hat{\_} \) before the second \# (shifting the rest of the input to the right). If the input contains fewer than two \# symbols, don’t change anything.

For example, on input \( \#11\#\hat{\_}1\# \), the output should be \( \#11\hat{\_}\#\hat{\_}1\# \)

(b) Provide the complete sequence of configurations of your TM when started on the above example input, \( \#11\hat{\_}1\# \).

3. Let \( \Sigma = \{\#, 0, 1\} \). Provide an implementation level description of a input-output TM that computes the function

\[ f(\#\langle x \rangle) = \begin{cases} \#\langle x/2 \rangle & \text{if } x \text{ is even} \\ \#\langle 3x + 1 \rangle & \text{otherwise} \end{cases} \]

where \( \langle x \rangle \) stands for the binary representation of the number \( x \).

(For example, if the TM starts with \( \#100 \) on the tape it should halt with \( \#10 \) on the tape; if it starts with \( \#11 \), it should halt with \( \#1010 \).)

You may use a TM with more than one tape – in this case the output should be written on the first tape.

4. Let a “k-PDA” be a pushdown automaton with \( k \) stacks. So, on each step, the PDA reads an input symbol and pops \( k \) symbols (one from the top of each stack). Then, according to these \( k + 1 \) symbols, the machine pushes \( k \) symbols (one onto each stack), changes state, and moves its read head forward on the input. Note that any of the symbols (read or stack) might be \( \epsilon \), as in a normal PDA.

Thus, 0-PDAs are just NFAs (recognizing regular languages), and 1-PDAs are just PDAs (recognizing context free languages, and thus more powerful).

You can convince yourself that 2-PDAs are more powerful than 1-PDAs, by constructing a 2-PDA that can recognize the non-context free language \( \{ww \mid w \in \{0, 1\}^*\} \) (you do not need to submit this).

Here you will prove that \( k \)-PDAs are no more powerful than 2-PDAs, for any \( k \geq 2 \). In fact, you will prove that a 2-PDA is equivalent to a TM.

(a) Show that for any \( k \geq 2 \), a \( k \)-PDA can be simulated by a multi-tape non-deterministic Turing machine. This simulation should be described at an implementation level.

(b) Show that a TM can be simulated using a 2-PDA. Again, use an implementation level description.