Homework 5

Due: Thursday, 3/23/17 at 9:00pm
Reading: Chapter 3

Before you start: 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 a table or a state transition diagram). The only exception is that omitted transitions can be assumed to go to $q_{rej}$.

- 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”).

- 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.

Recall from class that an input-output TM is defined similarly to a standard TM, except it has only one halting state $q_{halt}$ (replacing $q_{acc}$ and $q_{rej}$), and everything else is defined just like a standard TM, with a computation halting as soon as the machine goes to the $q_{halt}$ 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.

1. (a) Provide a **formal level** description of a input-output TM that adds 1 to a binary number.

   Specifically, the input alphabet should consist of $\{0, 1, \#\}$, and we denote the binary representation of a number $x$ by $\langle x \rangle$. Your TM should compute the function

   $$f(\#\langle x \rangle) = \#\langle x + 1 \rangle$$

   For example, on input $\#1010$ your machine should terminate with $\#1011$.

   (b) Provide the complete sequence of configurations of your TM when started on the input $\#101$.

2. Provide an **implementation level** description of a input-output TM that computes the function $f(\#\langle x \rangle) = \#(3x)$ where $\langle x \rangle$ is the binary representation of the number $x$.

   (For example, on input $\#1010$ your machine should terminate with $\#11110$).
3. 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 ϵ, 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 \{0^n1^n2^n : n ≥ 0\} (you do not need to submit this, but you should do it for your own understanding).

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

   (a) Show that for any k ≥ 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.

4. (a) Prove that the class of Turing-decidable languages is closed under the set difference operation.

   That is, given TM deciders M_1 and M_2 for languages L_1, L_2 respectively, construct a TM decider M for the language L_1 \ L_2 = \{w : w ∈ L_1, w /∈ L_2\}. (You may assume both languages are over the same alphabet Σ).

   Note that we have not yet proved in class any closure of the class of decidable languages under any operation, so your solution should consist of a direct construction as above.

   Don’t forget to argue that your constructed TM M is indeed a decider, and that it accepts a string w if and only if w ∈ L_1 \ L_2.

   (b) Does the same proof you provided in part (a) work to prove that the class of TM-recognizable languages is closed under set difference? That is, given M_1, M_2 recognizing languages L_1, L_2 respectively, would the TM M that you constructed above recognize L_1 \ L_2? Justify your answer.

5. **Extra Credit** We define a counter c to be a data structure that stores a value in \{0, 1, 2, 3, …\}, and which supports the following three operations:

   - increment: c ← c + 1
   - decrement: c ← c − 1 (assuming the current value c > 0; decrementing when the counter is at c = 0 is not defined)
   - zero-test: check whether current value c = 0?

   We define a k-counter NFA A to be an NFA additionally equipped with k counters, which are initialized to 0 in the beginning of the computation. As usual for NFAs, A will read its input left to right, and on each input symbol (or ϵ), A will either increment, decrement, or do nothing to each of its counters, possibly conditioned on which are
equal to 0 (there could be more than one option as this is non-deterministic). We assume that attempting to decrement a counter which is 0 will go to a dead (rejecting) state. A string is accepted if there exists a computation that ends up in an accepting state after the input has been read.

Your (extra credit) assignment: Prove that any TM has an equivalent 3-counter NFA. Hint: Using problem 3 above, it is sufficient to show that any 2-stack PDA has an equivalent 3-counter NFA.

Further remarks on the Power of counter NFAs: It turns out that a 1-counter NFA is strictly more powerful than an NFA, and strictly weaker than a PDA (but equivalent to a PDA with one stack symbol to signify empty stack and only one other stack symbol). Any $k$-counter NFA can easily be simulated by a $k$-tape TM (and thus by a standard TM), as the three supported operations are easy to implement on a TM (you did the first one in problem 1). On the other hand, as indicated by your assignment, any TM can be simulated by a 3-counter NFA. In fact, this can be done even with a 2-counter NFA, as one can show that a 3-counter NFA can be simulated by a 2-counter NFA. Thus, 2-counter NFAs are equivalent in power to TMs.