Computational Models, Spring 2014 Exrecise #3

Turing Machines

1. Give a verbal description of a Turing Machine that accepts the following languages. There is no need to give a formal definition but you should give a detailed explanation of each step.

   (a) $L = \{w \in \Sigma^* \mid \#_a(w) = \#_b(w) = \#_c(w)\}$ above $\Sigma = \{a, b, c\}$
   Main idea: Use multiple tapes: one for the input, one for counting the number of 'a's, one for counting the number of 'b's, and one for counting the number of 'c's. Move through input and for each symbol and a symbol on its tape. When input is read count that all three tapes are of same size.
   Pseudo-code:

   - **Part (1) - Initialize**
     - **Write \$ on tapes 2,3,4 and move Right on tapes 2,3,4**
     - **GoTo Part 2**
   - **Part (2) - Read**
     - **If input is** $(\sqcup, \sqcup, \sqcup, \sqcup)$ **GoTo Part 3.**
     - **If input is** $(a, \sqcup, \sqcup, \sqcup)$ **write** $(a, a, \sqcup, \sqcup)$ **and move right on tapes 1 and 2.**
     - **If input is** $(b, \sqcup, \sqcup, \sqcup)$ **write** $(b, \sqcup, b, \sqcup)$ **and move right on tapes 1 and 3.**
     - **If input is** $(c, \sqcup, \sqcup, \sqcup)$ **write** $(c, \sqcup, \sqcup, c)$ **and move right on tapes 1 and 4.**
   - **Part (3) - Check**
     - **If input is** $(\sqcup, \sqcup, \sqcup, \sqcup)$ **write** $(\sqcup, \sqcup, \sqcup, \sqcup)$ **and move left on tapes 2-4.**
     - **If input is** $(\sqcup, a, b, c)$ **write** $(\sqcup, \sqcup, \sqcup, \sqcup)$ **and move left on tapes 2-4.**
     - **If input is** $(\sqcup, \$, \$, \$)$ **accept.**
     - **If input contains one or two \$ signs, reject.**

   (b) $L = \{\{0\}^{n^2} \mid n \in N \text{ and } n > 0\}$ above $\Sigma = \{0, 1\}$
   Main idea: Use multiple tapes: one for the input, one to remember which n we are testing now, one to serve as a counter, and one to compute the current value of n^2 and to compare to original tape. Move through input and for each symbol and a symbol on its tape.
   Pseudo-code:

   - **Write 1 on the second and third tape**
   - **(†) Write on the fourth tape the number of zeros on the second tape.**
   - **Decrement counter on third tape.**
   - **If we have not reached zero, repeat step (†).**
   - **If we have reached zero, compare with first tape: return accept if equal, return reject if fourth tape is longer than first, else continue by**
     - **increment second tape and copy to third tape.**
     - **erase fourth tape.**
     - **GoTo step (†).**

   (c) $L = \{a^i b^j c^k \mid i \cdot j = k \text{ and } i, j, k \geq 0\}$ above $\Sigma = \{a, b, c\}$
   Main idea: The machine will read the tape and for each 'a' that it will read, will erase a 'c' for each 'b' there is.
   Pseudo-code:

   - **Check if the input is in the form $a^*b^*c^*$. If not, reject.**
   - **If the current letter is a blank - accept.**
   - **If the current letter is a 'c' - reject.**

- **(†) If the current letter is a 'b' - check that all c's were erases. If so, accept. If not, reject**
- **Erase first 'a' and move head right**
  - **As long as current letter is 'a' - move right**
  - **If the current letter is a 'c' - reject.**
  - **($\star$) If the current letter is a 'b' - replace with 'x' and move right.**
    - **As long as current letter is 'b' or 'y'- move right**
    - **If the current letter is a blank - reject.**
    - **If the current letter is a 'c' - replace by 'y' and move left until the first letter to the right of 'x'.**
    - **If the current letter is a 'y' - reconstruct all b's be replacing each 'x' with 'b'. If not GoTo($\star$).**
  - **Go left to the first non-blank and GoTo (†)**

2. In this question we are interested in *computing a function* using Turing Machines.

   (a) Given the function $f(x) = x + 1$, construct a TM that given a number x, represented in binary form on the tape where the least significant bit (lsb) is on the left-most position, returns the value $f(x)$. Give a *formal* description. There is no need to prove your answer.
   Define the following TM:

   $$M =< \{q_0\}, \{0,1\}, \{0,1,\sqcup\}, \delta, q_0, q_f >$$

   With transition function:
   $\delta(< q_0, 0 >) =< q_f, 1, R >$
   $\delta(< q_0, 1 >) =< q_0, 0, R >$
   $\delta(< q_0, \sqcup >) =< q_f, 1, R >$

   (b) How would your answer to (a) change if the number was represented when the most significant bit (msb) is on the left-most position? No need to formally build the TM, explain in pseudo-code your algorithm.
   - **Shift input right and mark leftmost cell with $**
   - **Move head from right to left while exchanging every 1 with 0 until we reach a 0 or the $**
   - **Exchange the 0 (or $) with 1**
   - **If we haven't written on the $ then shift result left**

   (c) For the following question, assume that the Turing machine has a an *input* tape where the input is written (it is read only) and an *output tape* where the output is written. Describe a Turing Machine such that given an input x (in Unary representation) returns the value $\lfloor \log_2(x) \rfloor$. No need to formally build the TM, explain in pseudo-code your algorithm and explain its correctness.
   - **While there is a '1' on input tape**
     - **Write '1' on output tape**
     - **Replace every second '1' of the input with 'x'.**
   - **Remove one '1' from the output tape**

3. Let us define a generalization of Turing Machines to include a *finite memory* of size n. We denote such a Turing Machine formally as:

   $$M_{\text{mem}} = (Q, \Sigma, \gamma, \delta_{\text{mem}}, n, q_0, qa, q_r),$$

   where all the definitions are identical to the Turing Machine defined in class except that there is the *finite* memory size n and the transition function δ_{mem}. At each step, the transition depends on the

current state, the input on the tape and all the memory. The transition to the next step can update the entire memory. Formally: $\delta_{\text{mem}} : Q \times \Gamma \times \Gamma^n \to Q \times \Gamma \times \Gamma^n \times \{L, R\}$.

Given a finite memory Turing Machine M_{mem}, define *formally* a (standard) Turing Machine M such that $L(M) = L(M_{\text{mem}})$. Namely, both Turing Machines accept the same language. Explain your answer.

Bonus: Show more than one formal construction.

Given a *finite memory* TM

$$M_{\mathbf{mem}} = (Q, \Sigma, \gamma, \delta_{\mathbf{mem}}, n, q_0, q_a, q_r).$$

We define a (standard) Turing Machine M as follows:
(a) Extend γ to account for the memory:

$$M = (Q, \Sigma, \gamma', \delta', , q_0, q_a, q_r)$$

Where: $\gamma' = \Gamma^{n+1}$ and
$\delta'(q, (\sigma_0 \dots \sigma_n)) = (r, (\tau_0 \dots \tau_n), D)$ **for every:**
$\delta(q, \sigma_0, \dots, \sigma_n)) = (r, \tau_0, \dots, \tau_n, D)$
(b) Similarly, one can extend the states to account for the memory.

4. Let $L_1, L_2 \in RE \setminus R$ can the following occur? If your answer is "yes" give a concrete example. If your answer is "no" give a sketch of a proof.

   (a) $L_1 \cap L_2 \in R$
   Yes, take

   $$L_1 = \{< M, x >, \textbf{M is a TM that halts on x and has an even number of states}\}$$

   $$L_2 = \{< M, x >, \textbf{M is a TM that halts on x and has an odd number of states}\}$$

   It is easy to see that $L_1, L_2 \in RE \setminus R$ but their intersection is empty hence in R.

   (b) $L_1 \cup L_2 \in R$
   Yes, take

   $$L_1 = \{< M, x >, \textbf{M is a TM that halts on x or has an even number of states}\}$$

   $$L_2 = \{< M, x >, \textbf{M is a TM that halts on x or has an odd number of states}\}$$

   It is easy to see that $L_1, L_2 \in RE \setminus R$ but their union is all pairs $< M, x >$ hence in R.

   (c) $L_1 \cap L_2 \in R$ and $L_1 \cup L_2 \in R$
   No, then we could construct a TM that decides L_1: Let M_1, M_2 be the TM that accepts L_1 and L_2, respectively. Let M_u, M_i be the TM that decides $L_1 \cap L_2$ and $L_1 \cup L_2$, respectively.
   The TM that decides L_1: If $M_i(x) = T$ return T.
   If $M_u(x) = F$ return F.
   If not, the run M_1, M_2 simultaneously on x until one accepts.
   If $M_1(x)$ accepts return T.
   If $M_2(x)$ accepts return F.

5. Prove or contradict: R is closed under infinite (countable) union. Namely, given $L_1, L_2 \dots$ such that $L_i \in R$ then $\bigcup_{i=1}^{\infty} L_i \in R$.
   The claim is not true. Let L be a language not in R (hence it has an infinite number of words), and enumerate the words in some order. For every i, let L_i be the language that contains only the i'th word of L. Clearly, every L_i is decidable as it is finite but $L = \bigcup_{i=1}^{\infty} L_i \notin R$.