Mod Comp CSCI-2400
Solutions to Homework #6

(The solution is provided by your classmate Matt Leotta)

1. No, the requirement that a Turing machine must halt only in a final state does not restrict the power of the machine. Let's call the modified kind of machines, which halt only on final states, as the "final-halt" machines. We need to prove that any language accepted by a standard Turing Machine is also accepted by a no-halt machine, and vice-versa.

(a) Any language accepted by a standard Turing machine is also accepted by a final-halt machine.

Proof by construction
We can convert any standard Turing machine into a final-halt machine using the following method:

1. Create a new "trap state" with transitions to itself for all inputs
2. For each undefined transition from a non-final state, define a transition to the trap state

For all instances where the standard machine previously halted in a non-final state, the final-halt machine now enters an infinite loop in a trap state. Therefore, all previous inputs which where not accepted are still not accepted because the machine loops forever. All input strings previously accepted are still accepted since transitions towards final states are not modified.

(b) Any language accepted by a final-halt machine is also accepted by a standard Turing Machine.

Trivially true, since any final-halt machine is also a standard Turing machine.

2. For any string $w \in \{a, b, c\}^+$, let w_i denote the i-th symbol of w, namely, $w = w_{|w|}\, w_{|w|-1} \ldots w_2\, w_1$. Assign to letters a, b, and c the respective values 1, 2, and 3. The function $f(w)$ that returns the index for all $w \in \{a,b,c\}^+$ in the proper ordering is:

$$f(w) = w_{|w|}3^{|w|-1} + w_{|w|-1}3^{|w|-2} + \ldots + w_2\,3 + w_1$$

3. Let L_1 be recursive and L_2 recursively enumerable. Show that $L_2 - L_1$ is necessarily recursively enumerable.

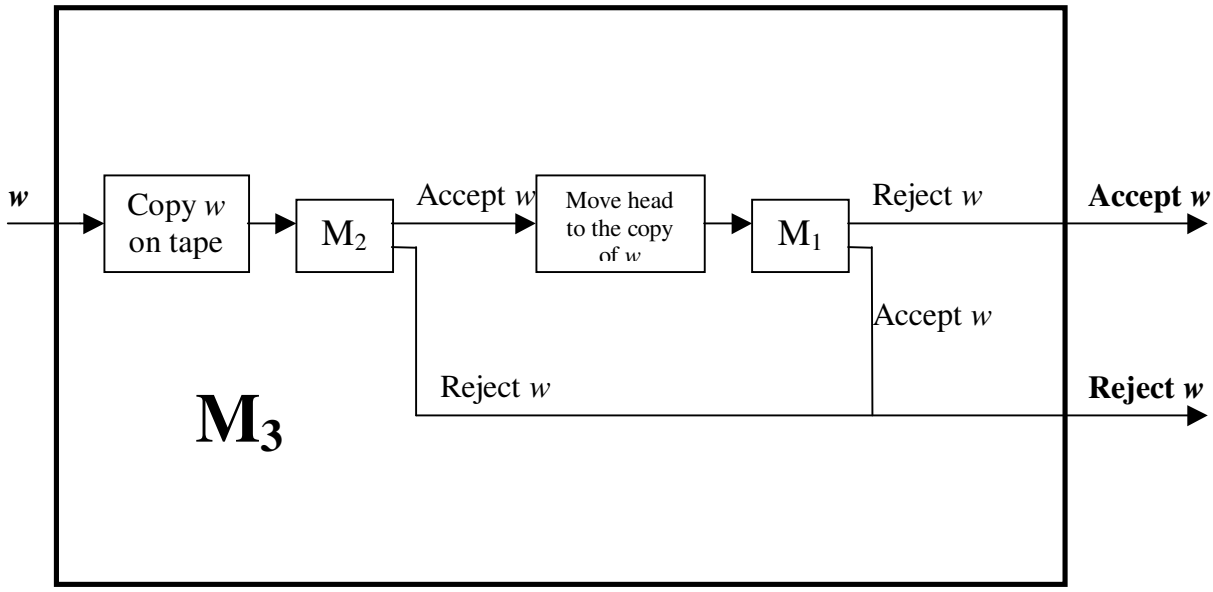
Proof by construction:
There exists a Turing machine M_1 that accepts L_1 and halts on all inputs.
There exists a Turing machine M_2 that accepts L_2 and may not halt for all inputs.
We can construct a new Turing machine M_3 from M_1 and M_2 that accepts $L_2 - L_1$.
M_3 should accept all strings accepted by M_2 but not accepted by M_1.

Machine M_3 first needs a new Turing machine to duplicate the input string on the tape. Next M_3 will run M_2 and then M_1 on the copy of the input with the following changes:

1. Add transitions from all final states of M_2 (and make them non-final states) to the starting state of M_1 (moving the read-write head to the second copy of the input string in the process). Therefore, if a string is accepted by M_2 it will be tested on M_1.

2. Convert all final states of M_1 into non-final states. Furthermore, convert any other hatling states of M_1 to final states. Therefore, if a string is accepted by M_1 it is rejected in M_3, and furthermore if a string is rejected by M_1 it is accepted by M_3.

M_3 will run M_2 and then M_1.



Since M_3 contains M_2 there are must be instances of w for which M_3 does not halt (M_2 doesn't halt for these instances). Therefore, $L_2 - L_1$ is necessarily recursively enumerable. However, if L_2 was recursive then M_3 would always halt and $L_2 - L_1$ would be recursive as well.

4. If there exists a Turing machine that enumerates the elements of L in proper order then it must be a recursive language.

Proof by construction:
By definition, L is recursive if and only if there exists a membership algorithm for it. We will construct a Turing machine M that determines membership in language L, namely, for any input string w, machine M determines if w is in L or not.

Let M_1 be the Turing machine that enumerates the elements of L in proper order.

Machine M compares the input string w with each output from M_1.
If string w equals an output from machine M_1 then machine M returns "yes".
If the length of an output string from M_1 exceeds the length of w then M returns "no".

This works because strings in proper order are in order of increasing length. Thus, if M_1 reaches a point where it is producing strings with a length greater than that of w, we know that it will never produce w. This membership algorithm always produces an answer in finite time (always halts) so L must be recursive.