CSE 431
Introduction to the Theory of Computation
Sample Midterm - Solutions

1. (18 points) For each of the following questions answer True, False, or Unknown and briefly JUSTIFY your answer.

   (a) It is undecidable to tell, given two Turing machines M_1 and M_2, whether or not $L(M_1)$ is the complement of $L(M_2)$.

   Solution:

   True: We use this to solve E_{TM} by converting the input $\langle M \rangle$ to the pair $\langle M, M_{\Sigma^*} \rangle$, where M_{Σ^*} is a trivial machine that always accepts, and using the resulting answer (this is a reduction from E_{TM}). This is correct since $\varnothing$ is the complement of Σ^*.

   (b) There is a regular language whose complement is undecidable.

   Solution:

   False: The complement of any regular language is also regular and all regular languages are decidable.

   (c) There is no Turing machine that decides whether or not two context-free grammars generate the same language.

   Solution:

   True: One could use this to decide ALL_{CFG} by setting one of the grammars to a trivial grammar that generates of Σ^*. However ALL_{CFG} is undecidable.

2. (20 points) Show that the language

   $$BH = \{\langle M \rangle \mid \text{Turing Machine } M \text{ halts when started at the left end of a blank tape}\}$$

   is undecidable.

   Solution:

   We prove that $HALT_{TM} \leq_m BH$. Since $HALT_{TM}$ is undecidable, this is enough.

   Given a pair $\langle M, w \rangle$ encoding a Turing machine M and input w to M, the function f will create a new TM (a familiar TM): M_w which ignores its input x and runs M on the hardcoded value w and does exactly what M does.

   f is certainly computable. It remains to show that f is a correct reduction.

   Now if $\langle M, w \rangle$ is in $HALT_{TM}$ then M halts on input w and so M_w will halt on all inputs, including empty string input which corresponds to a blank tape. Therefore $\langle M_w \rangle$ is in BH.

On the other hand if $\langle M, w\rangle$ is not in $HALT_{TM}$ then M runs forever on input w and so M_w will run forever on all inputs, including empty string input which corresponds to a blank tape. Therefore $\langle M_w\rangle$ is not in BH.

Therefore this is a correct reduction.

3. (20 points)

   (a) Give the full formal definition of what it means for a set A to be mapping reducible to a set B.

   (b) Prove that if a set C is co-Turing-recognizable but not decidable then it cannot satisfy $C \leq_m A_{TM}$.

   Solution (b):

   If $C \leq_m A_{TM}$ then C must be Turing-recognizable because A_{TM} is Turing-recognizable. Therefore C is both Turing-recognizable and co-Turing-recognizable and hence decidable, which is a contradiction.

4. (22 points)

   (a) State the Church-Turing Thesis.

   (b) Why can't we ever prove the Church-Turing Thesis?

   (c) What is the evidence for it?

   (d) A queue automaton is a bit like a Turing machine except that it has a single queue of symbols instead of a tape. The input followed by a single blank symbol is initially in the queue with the first character at the head of the queue. In each step the queue automaton pops (reads and removes the symbol) at the head of the queue,o and then, based on that symbol and the current state, changes state and pushes one or two symbols onto the tail of the queue.

   Give the rough ideas for why a queue automaton is equivalent to a TM. (The tricky part is seeing how you can simulate TM moves in each direction on the queue automaton.)

   Solution (d):

   The general idea is to use the queue together with the state of the queue automaton to keep track of the TM configuration. In general, configuration $uqav$ will be represented as the queue string $av\#u$ where the head is on the a. The state will hold q.

   To do a right move $\delta(q, a) = (p, b, R)$ of the TM the queue machine will remove a from the head, change to state p, and add b to the tail, yielding queue contents $v\#ub$.

   To do a left move $\delta(q, a) = (p, b, L)$ of the TM, suppose that the queue contents are $av\#uc$. The queue machine will remove a from the head, add $\$b$ to the tail to reach $v\#uc\$b$ and record in its state that it will want to move to state p. Before that can happen it will move one character at a time from the head to the tail, provided that the following character is not \$. (The putting on the tail will be one step after reading from

the head.) At this point, it can end up with $\$bv\#u$ and if we allow editing the head of the queue, we would replace the \$ by the c it has remembered yielding $cbv\#u$. (If editing of the head is not allowed, the \$ can be read immediately and the remembered c can be used to trigger the next state change.)

5. (20 points) If language L is defined over alphabet Σ then the set of prefixes of strings in L,

$$Prefix(L) = \{x \mid \text{there exists a } y \in \Sigma* \text{ such that } xy \in L\}.$$

(a) Show that if L is Turing-recognizable then $Prefix(L)$ is Turing-recognizable.

Solution:

There are three different natural constructions:

(1) Since L is Turing-recognizable there is an enumerator E for L. We create a recognizer M' for $Prefix(L)$ as follows:
M': "On input x run E but whenever E outputs a string w, check to see whether x is a prefix of w; if so, accept."

(2) Since L is Turing-recognizable there is an enumerator E for L. We create an enumerator E' for $Prefix(L)$ as follows.
E': "Run enumerator E but whenever E prints a string w, E' also prints all prefixes of w."

(3) (Dovetail) Given a TM M that recognizes L, create a recognizer M' for $Prefix(L)$ as follows:
M': "On input x, for $t = 1$ to ∞: for all $y \in \Sigma^*$ with length at most t, run M on input xy for at most t steps; if M accepts, then accept"

(b) Intuitively, why doesn't the method you used for part (a) also show that if L is decidable then $Prefix(L)$ is decidable?

Solution:

This depends on the answer from part (a); If (1) then if there may be no such w, in which case the algorithm will run forever; if (2) then the list of strings is definitely not in lexicographic order even if the w are, because the prefixes are very short; if (3) then again, the algorithm will run forever if no y exists.¡/i¿

(c) (BONUS - Do not attempt unless you are finished early and have already checked over your paper) Actually come up with an example where you can prove that L is decidable but $Prefix(L)$ is not decidable.