HW 4 - Computational Models - Spring 2014 - Solutions sketch

1. For each of the following languages, find a minimal (relative to inclusion) class it belongs to (if any) between R/RE/coRE. Prove your answer.

   (a) $\text{Halt}^* = \{\langle M\rangle \mid M \text{ is a Turing machine that halts for each possible input}\}$
   Solution: in $\overline{RE \cup coRE}$: $\overline{H_{TM}} \leq_M \text{Halt}^*$ ($f(< M, w >) =< M_0 >$ where M_0 on input y simulates M on input w for $|y|$ steps and enters a loop if M stopped, halts otherwise), $H_{TM} \leq_M \text{Halt}^*$ ($f(< M, w >) =< M_0 >$ where M_0 on input y simulates M on input w).

   (b) $\text{ALL}_{\text{TM}} = \{\langle M\rangle \mid M \text{ is a Turing machine and } L(M) = \Sigma^*\}$
   Solution: in $\overline{RE \cup coRE}$: $\text{Halt}^* \leq_M \text{ALL}_{\text{TM}}$ ($f(< M >) =< M_0 >$ where M_0 on input y simulates M on input y moving to state q_r whenever M moves to q_a).

   (c) $L = \{\langle M\rangle \mid M \text{ is a Turing machine that halts on all inputs or } |\langle M\rangle| \geq 2014\}$
   Solution: $\overline{L}$ is finite and therefore in R.

   (d) $L_1 = \{\langle M\rangle \mid M \text{ is a Turing machine and } |L(M)| \leq 2014\}$
   Solution: in $coRE - R$: Not in R by Rice. $\overline{L_1}$ in RE: Given $< M >$, first check if it is a valid encoding of a TM (accept if not), and then simulate the execution of M on all words in Σ^* (interleaved, in lexicographic order) until the 2015th word is accepted (and then accept).

   (e) $L_2 = \{\langle M\rangle \mid M \text{ is a Turing machine and } |L(M)| \geq 2014\}$
   Solution: in $RE - R$: Not in R by Rice. L_2 in RE: Given $< M >$, first check if it is a valid encoding of a TM (reject if not), and then simulate the execution of M on all words in Σ^* (interleaved, in lexicographic order) until the 2014th word is accepted (and then accept).

   (f) $L_3 = L_1 \cap L_2$
   Solution: in $\overline{RE \cup coRE}$: See question 2 Moed B 2013 for reductions $\overline{H_{TM,\epsilon}} \leq_M L_3$, and $H_{TM,\epsilon} \leq_M L_3$.

2. (a) Describe a reduction $H_{TM} \leq \overline{H_{TM}}$ (use a TM that decides $\overline{H_{TM}}$ to construct a TM that decides H_{TM}).

   Solution: A TM that decides H_{TM} given input $< M, w >$ feeds the input to a TM that decides $\overline{H_{TM}}$ and flips the decission.

   (b) Can there be a mapping reduction $H_{TM} \leq_M \overline{H_{TM}}$? Prove your answer.

   Solution: No. Otherwise, since $\overline{H_{TM}} \in coRE$ we would conclude (from the mapping reduction) that $H_{TM} \in coRE$ but this is impossible (why !?).

3. We define the following total function over the natural numbers: for a given n, it is the smallest TM (in terms of number of states) that makes at least n steps on input ϵ and eventually halts. Using the definition of S_n as defined in class, we formally have (denoting by $|T(\epsilon)|$ the number of steps taken by the TM T on input ϵ): $FF(n) = \min\{m \mid \exists T \in S_m \text{ such that } |T(\epsilon)| \geq n\}$. Prove that FF is not computable. (Hint: use a reduction from BB).

   Solution: Observe first that $FF(n)$ is monotone ($n_1 \geq n_2$ implies $FF(n_1) \geq FF(n_2)$) and unbounded (for an arbitrary n let M be the maximal number of steps taken by TMs in the finite set $\cup_{m \leq n} S_m$ and we have $FF(M+1) > n$). Now, note that if for some k we have $FF(k) \leq n$ and $FF(k+1) > n$ then it must be that $BB(n) = k$. Therefore, we can compute $BB(n)$ by computing $FF(1), FF(2), \ldots$ until for some k we have $FF(k) \leq n$ and $FF(k+1) > n$. Since we know $BB(\cdot)$ is not computable we conclude that $FF(\cdot)$ is not computable.

4. Assume you have a python "stepper", an object that is initialized with a function and its parameters, and has a step() method that makes one subsequent computation step of the function each time called (e.g. as in a debugger).

   Describe ("high level", no code required) how would you implement a python function that receives a list of functions $[f_1, \ldots, f_k]$ and a list of inputs $[x_1, \ldots, x_k]$, and prints all returned values $f_i(x_j)$ (and each such returned value only once) for each pair (f_i, x_j) such that f_i returns when its input is x_j (note that a python function might loop forever or crash). Your function does not have to halt.

   Solution: We will simulate in parallel the executions of $f_i(x_j)$ for all pairs i, j step by step (using the stepper) and print the reutrn value whenever any execution returns.

5. Two python boolean functions f and g are said to be equivalent (denoted $f \equiv g$) if when given the same input they return the same value (or both don't return/crash).

Let C be a condition on python boolean functions (meaning, C is a predicate that may occur or not. For example, "the function returns true only on odd-value inputs"). If $f \equiv g \Rightarrow C(f) \iff C(g)$ we say that such a condition is "semantic" (meaning, C is a condition on the behavior of the function and not on its implementation).

Let the python function c be a decider for the set of boolean python functions for which the predicate C holds, that is, when given a function f, $c(f)$ returns true if the condition C holds for f and returns false otherwise.

Finally, let *no* be a python function that never returns.

(a) Assume for a semantic condition C that $C(no) = false$ and there exists a python function *yes* such that $C(yes) = true$. Consider the following python function

```
def halt(p,w):
    return c(lambda y: yes(y) if p(w)==p(w) else False)
```

Prove that *halt* decides the halting problem for python functions.

Solution: If p halts on w then the input of c is a function that is equivalent to *yes* and therefore c will return true. If p doeas not halt on w then the input of c is a function that never returns (equivalent to *no*) and therefore c will return false.

(b) What is the reduction implemented by *halt* above ? Is it a mapping reduction? If yes, what is the mapping?

Solution: It is a mapping reduction. Tha mapping function maps a pair p, w to the anonymous python function

```
lambda y: yes(y) if p(w)==p(w) else False
```

(c) Assume now that the semantic condition C is such that $C(no) = true$ and there exists a python function *yes* such that $C(yes) = false$. Define a python function (that uses c, i.e. a reduction as above) that decides the halting problem for python functions in this case.

Solution:

```
def halt(p,w):
    return not c(lambda y: yes(y) if p(w)==p(w) else False)
```

(Note: This is no longer a mapping reduction!)

(d) State the Rice theorem for python functions just proved by the above reductions.

Solution: A nontrivial language of python functions induced by a semantic condition is undecidable.

6. Assume $A \leq_M B$, $B \leq_M C$, $E \leq_M B$, $B \leq_M D$. For each of the following statements chose and prove one of the following: (1) The statement is true for any choice of A, B, C, D, E, (2) The statement is false for any choice of A, B, C, D, E. (3) Not (1) and not (2).

   (a) $C \notin R$ and $E \in R$.

   (b) $D \in RE$ and $D \notin R$ and $A \notin RE$.

   (c) $D \in$ Co-$RE \Rightarrow \overline{E} \in RE$.

   (d) $C \in RE$ and $D \in$ Co-$RE \Rightarrow B \in R$.

Solution: See question 2 (Part A) Moed B 2013.