Autumn 1997–98, Prof. Boneh
Solution Set
CS 154, Problem Set 5

5.1 (3 points)

PROBLEM: **Prove that the diagonal language $L_D = \{w_k : w_k \notin L(M_k)\}$ is not R.E.**

This is just like in class. If L_D were R.E., there would be some machine accepting it (by the definition of R.E.). This machine is M_i for some i (where i is the row in the table where the TM for L_D is found). Now what does M_i do for w_i? If it accepts, then w_i does not belong to $L_D = L(M_i)$, and M_i accepts some word not in $L(M_i)$. Obviously impossible. If M_i does not accept w_i, then w_i does not belong to $L(M_i)$. But, by definition, such a word should belong to L_D, indicating once again L_D and $L(M_i)$ are different. We must conclude no TM has L_D as the language it accepts; that is, L_D is not R.E.

5.2 (4 points)

PROBLEM: **(a) Prove that $L_{\text{NOT-EMPTY}} = \{\langle M\rangle : M$ is a TM s.t. $L(M) \neq \emptyset\}$ is R.E. (b) Prove that $L_{\text{NOT-EMPTY}}$ is undecidable. (c) Deduce that $L_\emptyset = \{\langle M\rangle : M$ is a TM s.t. $L(M) = \emptyset\}$ is not R.E.**

(a) Remember the diagonalization in class to enumerate two infinities at once (an infinite number of TMs, each computing for an infinite number of steps)? Well, it's easy to do in 3 dimensions as well. Order all the TMs and all the words. Now, we let c range from 0 to ∞, and for a given value of c we check the first c TMs on the first c words, running each TM for at most c steps. If machine i accepts word j after k steps, then we will figure out M_i is in L after c is set to $\max\{i, j, k\}$. If M_i accepts no word, of course, we will never discover this. Since we can, in this manner, enumerate all machines that accept some input, we have shown L is R.E.

(b) This is a pretty easy reduction from L_{HALT}. We show how, if $L_{\text{NOT-EMPTY}}$ were decidable, we could use our decider for $L_{\text{NOT-EMPTY}}$ to decide L_{HALT} as well. Let $\langle M\rangle, w$ be our input to the halting problem: we want to know if w halts when run on M.

We must modify M so an oracle for the not-empty problem will give us useful information for the halting problem. In this case, we modify M by adding a preamble that erases Ms original input and writes w instead. Thus, this machine always does its thing on w, no matter what the input. (This is like writing a computer program that says, "enter a number," and then outputs "3 is prime" no matter what number you put in). If M accepts w out modified machine M' accepts w all inputs; if M doesn't accept w, M' accepts no input. This is because M' ignores its input. Thus, M accepts w iff our not-empty decider says $L(M')$ is not empty.

Since an oracle for $L_{\text{NOT-EMPTY}}$ decides the halting problem, which is known to be undecidable, no oracle for $L_{\text{NOT-EMPTY}}$ can exist. Thus $L_{\text{NOT-EMPTY}}$ is undecidable.

(c) $\overline{L_{\text{NOT-EMPTY}}} = L_\emptyset \cup \{w : w$ does not encode a TM$\}$. That is, the two languages are not exact complements, since a string not encoding a turing machine is not in either language. Now, $\{w : w$ does not encode a TM$\}$ is clearly R.E. — just enumerate all strings, and check each string to see whether it encodes a TM. If $L_\emptyset$ were also R.E., then $L_\emptyset \cup \{w : w$ does not encode a TM$\}$ would be R.E., since the union of two R.E. languages is R.E. (Here's why: take the two enumerators and alternate between them, enumerating both languages.) In other words, $\overline{L_{\text{NOT-EMPTY}}}$ would be R.E.

$L_{\text{NOT-EMPTY}}$ we know to be R.E. from part **(a)**. If both $L_{\text{NOT-EMPTY}}$ and its complement were R.E., then $L_{\text{NOT-EMPTY}}$ would be recursive (we know this from the previous problem set). But from part **(b)** we know $L_{\text{NOT-EMPTY}}$ is *not* recursive. The only possible flaw in our argument was the claim $L_{\emptyset}$ is R.E., so it must not be.

5.3 (3 points)

PROBLEM: **Show that the following language is undecidable:**

$$L = \left\{ < M > \;\middle|\; \begin{array}{l} M \textbf{ is a TM that on all inputs doesn't move} \\ \textbf{its head left of the left end of the tape} \end{array} \right\}$$

Again, the technique is to show how an oracle for this problem can be used to decide a known-to-be-undecidable problem. In this case, I choose $L_{\text{NOT-EMPTY}}$ as the known-to-be-undeciable problem.

So suppose $\langle M \rangle$ is the input that we want to know if it's in $L_{\text{NOT-EMPTY}}$. First thing we do is modify M to get a new machine M'. The hope is that by deciding if M' is in L, we can decide if M is in $L_{\text{NOT-EMPTY}}$.

For the oracle to be useful, we need to carefully control when M' moves off the left end of the tape. First, we make sure it never does so during the normal course of action. We do this by adding a preamble to M, so that, before doing its normal computation, it replaces its input w by $\$w$, where $\$$ is a new letter. It then adds transitions so when the head is looking at $\$$, it goes to q_r. With this modification, the machine mimics the behavior of falling off the left edge of the tape without actually doing so. Obviously, $L(M') = L(M)$.

Now we add special cases where we make M' fall off the left edge. In particular, when M would go into q_a, we make it go left forever. To do this we add a new state that goes left on all letters (even $\$$). Now we're done. The new machine M' goes off the left edge exactly for those words M accepts. In particular, if M accepts no words, M' never goes off the left edge.

The reduction now works as follows: given an input $\langle M \rangle$ to the not-empty problem, we convert it into M' and call on the decider for L. If L says M' goes off the left edge of the tape for some input, we say M is in $L_{\text{NOT-EMPTY}}$. If it says M' never goes of the left edge of the tape, we say M is not in $L_{\text{NOT-EMPTY}}$.