Exercises for Computability and Complexity, Spring 2019, Sheet 5 – Solutions

Please return on Tuesday, March 12, in class. As usual you are invited but not requested to work in teams of size at most 2.

Exercise 1 (rather easy) Prove that $H_2 = \{<M>;x \mid Code(<M>) \text{ and } Standard(x) \text{ and there exists some } y \text{ with } Standard(y) \text{ such that } M(x) = y\}$ from Proposition 6.3 is undecidable.

Solution. Take any word $<N>$ with $Code(<N>)$. We can effectively construct a TM $K_{<N>}$ with tape alphabet $\{0, 1, \#\}$ which, for all inputs $x \in \{0, 1, \#\}^*$, yields the following result:

$$K_{<N>}(x) \quad = \quad \text{if } N(x) \text{ halts then } K_{<N>}(x) = 1, \text{ else } K_{<N>}(x) = \nearrow$$

($K_{<N>}$ simply simulates $N(x)$, and if this halts, $K_{<N>}$ erases its tape and writes a 1, then halts). It clearly holds that $N(x)$ halts iff there exists some y such that $K_{<N>}(x) = y$. (namely, $y = 1$), which in turn is equivalent with $< K_{<N>} >;x \in H_2$. If H_2 were decidable, so would H, mission impossible.

Exercise 2 (medium difficult) Show that the language

$$L = \{<M> \in \{0, 1, \#\}^* \mid M \text{ halts on no input}\}$$

is not recursively enumerable. *Hint: in addition to a reduction argument, you might wish to also work in Proposition 3.1 from the lecture notes.*

Solution. First consider the complement language

$$L^c = \{w \in \{0, 1, \#\}^* \mid w \text{ is not a codeword } w = <M> \text{ for any TM } M, \text{ or } w \text{ is a codeword } w = <M> \text{ for some TM } M, \text{ and } M \text{ halts on some input}\}$$

L^c is recursively enumerable: it can be accepted by a TM N which first checks whether w is a valid TM codeword. If no, N immediately accepts. If yes, that is, if $w = <M>$, N simulates M on all input words $<x_1>, <x_2>, \ldots$ in a "dovetailing" fashion, that is, N first simulates M on input x_1 for k steps, then on inputs x_1 and x_2 for $2k$ steps each, then on inputs x_1, x_2 and x_3 for $3k$ steps, etc. If in one of these stages M is found to halt, N accepts.

Now if L would be recursively enumerable too, then L would be decidable. This can be seen, e.g., by reducing the language $H_0 = \{<M> \mid Code(<M>) \text{ and } M \text{ halts on the empty input}\}$ from the lecture notes to L: assume L is decidable. Modify M, obtaining M' such that M' behaves like M on the empty input and runs into infinity on any nonempty input. Then, $<M'> \in L$ iff $<M> \in H_0$, thus we could decide H_0, contradiction.

Challenge problem (optional, not easy) Prove the following claim: If L is recursively enumerable but not recursive, then there exists another language L' which is likewise r.e. but not recursive, such that $L \cup L'$ is recursive.

Solution (the one that I found; if you find a simpler one I'd be happy to learn about it). Let $L \subset \Sigma^*$ be recursively enumerable but not recursive, and M a Turing machine that accepts it. From M we construct another TM M' which accepts a language L' such that L' is r.e. but not recursive, and furthermore $L \cup L' = \Sigma^*$, i.e. this is recursive.

Let $(w_n)_{n = 1, 2, \ldots}$ be the alphabetical enumeration of Σ^*, and for $w \in \Sigma^*$, let $I(w)$ be the index of w in this enumeration.

We first show that there is a totally defined, recursive function $f: \mathbb{N} \to \mathbb{N}$, such that there exist infinitely many $v \in L$ where M needs at most $f(I(v))$ steps to accept v. One way to obtain such f goes like this:

Initialize $p = 0$.

By a dovetailing scheme, simulate M first for 1 step on w_1, then for 2 steps on w_1 and w_2, ... etc, – in the k-dovetail run, for k steps on w_1 to w_k. Whenever this simulation finds that M accepts w_l in m steps, and l is greater than p, set $f(n) = m$ for all $p \leq n \leq l$. Update p to l.

It is straightforward to show that f is total recursive and there exist infinitely many $v \in L$ where M needs at most $f(I(v))$ steps to accept v.

Using f we construct M' as follows. On input w, M' simulates M for at most $f(I(w))$ steps. If M does not accept w within this time, then M' accepts w (from this it follows that $L \cup L' = \Sigma^*$). If M accepts w within this time, M' first computes the number $k(w) = | \{i \leq I(w) \mid$ runtime of M on input w_i is at most $f(i)\}$ (in order to compute k, M' has to simulate M on all words v that come before w in the alphabetical enumeration, but only up to $f(I(v))$ steps). Then M' simulates M on input w_k. It is easy to see that in this way, M' simulates M on all words $u \in \Sigma^*$, ultimately running the simulation of M on u_i when M' is started on that w that has $k(w) = i$. When M accepts input w_k, M' accepts too (namely its original input w); otherwise M', simulating M, runs forever. The language L' thus accepted by M' is not recursive, because if it would be, then L could be decided with the use of M' (how? an extra little sub-exercise).