Exercises for Computability and Complexity, Spring 2017, Sheet 2 – Solutions

Please return your solutions in class, in the Thursday lecture on Feb 16.

Note: You may work in teams up to size 2.

Exercise 1. If one would admit TMs with countably many states, would this extend the set of TM-computable functions on the integers? In other words, is there a function $f: \mathbb{N} \to \mathbb{N}$ which can be computed by some TM with countably infinitely many states, but not by any ordinary TM? Sketch a proof for your answer.

Solution. With infinitely many states one can indeed "compute" more functions than with finitely many states. (In fact, with such a machine one could "compute" *every* function on the integers.) To see why, let $f: \mathbb{N} \to \{0, 1\}$ be *any* function with binary values on the integers (that is, f picks a subset of the integers – and any subset can be thus picked by some such f – that is, there must be uncountably many such f, which in turn means that almost all of these f are not Turing-computable). Arrange an infinite-state TM M with state set $K \supseteq \{s_1, s_2, \ldots\}$ such that on input n, M first goes to s_n (how can this be done? needs a subroutine) and then outputs $f(n)$ due to a hardwired answer-table-lookup rule of the form $\delta(s_n, a) = (h, f(n), -)$.

Exercise 3 (a) Are the functions $f(n) = \exp(n)$ and $g(n) = \exp(2n)$ polynomially related? **(b)** What about $f(n) = \exp(n)$ and $g(n) = \exp(n^2)$? Prove your answers.

Solution. (a) Yes, by the quadratic polynomial $p(n) = n^2$. We clearly have $f(n) \leq p(g(n))$, and conversely, $g(n) = (\exp(n))^2 = p(f(n))$.
(b) No. Assume there were a polynomial $p(n) = n^a$ such that $g(n) = \exp(n^2) = \leq p(f(n)) = \exp(na)$. Then for $m > a$, we would have $g(m) = \exp(mm) \geq \exp(ma)$, contradiction.

Exercise 4 Show that $L = \{w \in \{1\}^* \mid |w| \text{ is a power of } 2\} \in$ **TIME**$(O(n \log n))$, by describing in words (and maybe sketches of interesting configurations) a TM (with possibly several tapes) that does this job.

Solution. Set up a 2-tape TM, as follows. The first tape contains the input word, is read-only, and the cursor here never moves left. While the first cursor moves right, on the second tape a binary-coded count of the number of 1's visited is constructed. Whenever the first cursor moves to the right, the count on tape 2 is updated (which may take some operations where the first cursor does not move). The update is a combination of the add-1 and shift-right, single-tape TMs from the lecture notes, which per add-1 operation may require 2 full back-and-forth traversals of the word b written on tape 2 up to that point, that is, 4 |b| TM cycles. When the last 1 on tape 1 has been processed, our TM enters a final round of checking whether the 2nd tape word b is of the form 10....0. If yes, the input is accepted, if no, not. This final check can be clearly effected in another $|b|$ steps. Since $|b| \leq \log_2(|w|)$, we find that our TM uses at most $\log_2(|w|)(4\, n) + \log_2(|w|) = O(n \log n)$ steps.