Problem Set 8: Introducing Time Complexity
CS254, Fall 2019, Anna Rafferty

Solution: Please do not post these solutions or share them with anyone outside this class. Thanks!

You are welcome to talk to others about the problems on this problem set, but you should **read and attempt each problem prior do discussing it with others.** The write up must be your own work. That means that **you should write your answers by yourself, without looking at notes from sessions with collaborators, and you must be able to explain any answer you write down.** You should list anyone you collaborated with in your collaboration form.

Due dates: All five problems are due Wednesday, November 6, as there is an exam on Monday, November 4. I recommend doing problem 1 prior to the exam. As for all homeworks, if a problem is due on a specific day it must be turned in by the 9PM on that day. See homework handout (a copy is posted on Moodle) for more specific homework policies. Failing to name your files properly may result in a loss of points and potentially receiving no credit for that problem; please name your files properly!

1. Consider the language $XOR = \{\langle M, x, y\rangle \mid M$ is a TM that halts on precisely one of the strings x or $y\}$. Use mapping reductions to show that XOR is neither r.e. nor co-r.e.

Solution: We show that XOR is neither r.e. nor co-r.e. by showing that $\overline{HALT} \leq_m \overline{XOR}$ and $\overline{HALT} \leq_m XOR$.

First, we show $\overline{HALT} \leq_m \overline{XOR}$. We construct our function f as follows:

f(z):

- Decode z into M, w.
- Checks if $w = \epsilon$. If so, the string below that is denoted y is equal to '1'. If not, the string below that is denoted y is equal to ϵ.
- Constructs M': On input s:
    1. Checks if $s = y$. If so, it loops forever (e.g., continually moves right an stays in the same state).
    2. Otherwise, checks if $s = w$. If so, runs M on input w and outputs what it outputs.
    3. If neither of the above is true, accepts s.
- Outputs $(\langle M'\rangle, y, w)$

This is a computable function: there is no possibility of looping, as M' is constructed, not run, and checking whether strings are equal to one another is computable. We see that if $\langle M, w\rangle \notin \overline{HALT}$, then M' halts on w, since when its input is w, it mimics the behavior of M'. Additionally, we see that y is (a) guaranteed to be a not equal to w, and (b) not a string that M' halts on. Thus, $(\langle M'\rangle, y, w) \notin \overline{XOR}$ since M' halts on w and not on y. If $\langle M, w\rangle \in \overline{HALT}$, then M' halts on neither w or y, so $(\langle M'\rangle, y, w) \in \overline{XOR}$. Thus, $\langle M, w\rangle \in \overline{HALT}$ iff $(\langle M'\rangle, y, w) \in \overline{XOR}$. This shows that $\overline{XOR}$ is not r.e. (since $\overline{HALT}$ is not r.e.), and thus XOR is not co-r.e.

Next, we show that that $\overline{HALT} \leq_m XOR$. We construct our function f as follows:

$f(z)$:

- Decode z into M, w.
- Checks if $w = \epsilon$. If so, the string below that is denoted y is equal to '1'. If not, the string below that is denoted y is equal to ϵ.
- Constructs M': On input s:
    1. Checks if $s = y$. If so, it immediately accepts.
    2. Otherwise, checks if $s = w$. If so, runs M on input w and outputs what it outputs.
    3. If neither of the above is true, accepts s.

- Outputs $(\langle M' \rangle, y, w)$

This is a computable function: there is no possibility of looping, as M' is constructed, not run, and checking whether strings are equal to one another is computable. We see that if $\langle M, w \rangle \in \overline{HALT}$, then M' does not halt on w, since when its input is w, it mimics the behavior of M'. Additionally, we see that y is (a) guaranteed to be a not equal to w, and (b) a string that M' halts on. Thus, $(\langle M' \rangle, y, w) \in XOR$ since M' halts on y and not on w. If $\langle M, w \rangle \notin \overline{HALT}$, then M' halts on both w or y, so $(\langle M' \rangle, y, w) \notin XOR$. Thus, $\langle M, w \rangle \in \overline{HALT}$ iff $(\langle M' \rangle, y, w) \in XOR$. This shows that XOR is not r.e. (since $\overline{HALT}$ is not r.e.), and thus XOR is neither r.e. nor co-r.e.

2. Sipser 7.6.

Solution:

Throughout this problem, we ignore the fact that simulating some machine may be most easily done using multiple tapes, since if a machine runs in polynomial time with multiple tapes, it also runs in polynomial time with a single tape.

We first show P is closed under union. We can construct M' that decides $L_1 \cup L_2$ in polynomial time, assuming that L_1 and L_2 are languages for which there exist TMs M_1 and M_2 that decide the in polynomial time.

M' works as follows: On input w, it runs M_1 on w. If M_1 accepts, it accepts. If not, it runs M_2 on w. If M_2 accepts, it accepts. Otherwise, it rejects.

This machine runs in polynomial time since each stage runs in polynomial time; i.e., if M_1 takes time $O(f(n))$ and M_2 takes time $O(g(n))$, this machine takes time $O(f(n) + g(n))$; since $f(n)$ and $g(n)$ are polynomial in n, $f(n) + g(n)$ is also polynomial in n.

Next we show that P is closed under concatenation. Again, we can construct M' that decides $L_1 \circ L_2$ in polynomial time, assuming that L_1 and L_2 are languages for which there exist TMs M_1 and M_2 that decide the in polynomial time.

M' works as follows: On input w, it iterates through each possible division of $w = w_1 w_2$. It runs M_1 on w_1 and M_2 on w_2. If both accept, it accepts. If w is not accepted after all iterations, it rejects.

M' considers $O(|w|)$ ways to split w, and for each of these, it runs M_1 and M_2. The running time for each try on each of M_1 and M_2 is upper bounded by $O(f(|w|))$ and $O(g(|w|))$, where M_1 runs in $O(f(n))$ and M_2 runs in $O(g(n))$ (since actually they may be running on smaller strings). Altogether, this is $O(|w|(f(|w|) + g(|w|)))$. Since f, g are polynomial in the length of the input, this will also be polynomial in the length of the input, where the resulting polynomial is increased by a factor of n.

Finally, we show that P is closed under complement. Again, we can construct M' that decides $\overline{L}$ in polynomial time. Assume M is the machine that decides L in time $O(f(n))$, where $f(n)$ is a polynomial in n.

M' works as follows: On input w, it runs M on w. It then accepts if M rejects and rejects if M rejects.

This machine clearly takes only constant time more than M (ignoring the fact that we might use multiple tapes to simulate - but we've shown that if something can be decided in polynomial time on a multi-tape TM, it can also be decided in polynomial time on a single-tape TM), and thus still runs in time polynomial in the length of the input.

3. Sipser 7.8. Your analysis here can be similar in rigor to the analyses of TMs given in sections 7.1 and 7.2.

Solution: The machine starts off by selecting and marking a node - reaching that node takes at most $O(n)$ time. It then has a loop that repeats at most $|V|$ times, where V is the set of vertices. For each iteration of the loop, it must scan over all of the nodes in G and check whether it is connected by an edge to a node that is already marked. Considering all pairs of nodes is at most $O(|V|^2)$ and we also look at up to all edges for each pairs of nodes, meaning the loop takes at most $O(|V|^2|E|)$ per iteration. All together, steps 2-3 take up to $O(|V|^3|E|)$ time. Finally, step 4 scans through all the nodes in G; this can take no more than $O(n)$ time. $|E|$ and $|V|$ are upper bounded by n, so we can replace all occurrences with n. Then we have the full TM takes at most $O(n) + O(n^4) + O(n)$ steps, which is $O(n^4)$. This is in P.

4. Let A be any regular language. Show that $A \in \mathsf{TIME}(n)$.

Solution: Build a Turing machine for A that acts identically to a DFA for the language A. Specifically, for a DFA $\langle Q, \Sigma, \delta, q_0, F\rangle$, well build a Turing machine $M = \langle Q \cup \{q_{acc}, q_{rej}\}, \Sigma, \Gamma, \delta', q_0, q_{acc}, q_{rej}\rangle$ where $\Gamma = \Sigma \cup \{\textvisiblespace, \vDash\}$, and we define δ' as follows for all states q:

$$\delta'(q,a) = (p,a,R) \qquad \text{for states, symbols where } \delta(q,a) = p$$

$$\delta'(q,\textvisiblespace) = \begin{cases} (q_{acc}, \textvisiblespace, R) & \text{if } q \in F \\ (q_{rej}, \textvisiblespace, R) & \text{if } q \notin F \end{cases}$$

The Turing machine M mimics the operation of the DFA and accepts exactly the same strings, only moves right, and halts when it hits the first blank tape cell. So the number of steps on input w is precisely $|w| + 1$, which means that M runs in $O(n)$ time.

5. Let $f : \mathcal{N} \to \mathcal{N}$ be a function that grows without bound; i.e., $\lim_{n\to\infty} f(n) = \infty$. First, give definitions of $O(2^{f(n)})$ and $2^{O(f(n))}$. Then, show that $O(2^{f(n)})$ is a proper subset of $2^{O(f(n))}$. To do this, first prove that if $g(n) \in O(2^{f(n)})$, then $g \in 2^{O(f(n))}$. Then additionally prove that there is a $g(n) \in 2^{O(f(n))}$ that is not in $O(2^{f(n)})$.

Solution: By the definition of big-O, we know that $g(n) \in O(2^{f(n)})$ means there exists some constant C and positive integer N such that for all $n \geq N$, $g(n) \leq C \cdot 2^{f(n)}$. $g(n) \in 2^{O(f(n))}$) means there exists some constant C and positive integer N such that for all $n \geq N$, $g(n) \leq 2^{C \cdot f(n)}$.

We first prove that if $g(n) \in O(2^{f(n)})$, then $g(n) \in 2^{O(f(n))}$. Since $g(n) \in O(2^{f(n)})$, we know there exist C, N where for all $n \geq N$, $g(n) \leq C \cdot 2^{f(n)}$.

$$g(n) \leq C \cdot 2^{f(n)} \iff g(n) \leq 2^{\log_2 C} \cdot 2^{f(n)} \tag{1}$$

$$\iff g(n) \leq 2^{f(n)+\log_2 C} \tag{2}$$

Since $f(n)$ grows without bound, we choose N' to be the maximum of N and the smallest integer such that $f(n) \geq 1$. Then $2^{f(n)+log_2 C} \leq 2^{f(n)+f(n)\log_2 C}$. Continuing from above:

$$g(n) \leq 2^{f(n)+f(n)\log_2 C} \iff (n) \leq 2^{f(n)(1+\log_2 C)} \tag{3}$$

Letting $C' = (1 + \log_2 C)$, we see that for all $n \geq N'$, $g(n) \leq 2^{C' f(n)}$, so $g(n) \in 2^{O(f(n))}$.

Next, we show that for $f(n) = n$, $g(n) = 2^{3f(n)}$ is not in $O(2^{f(n)})$, although it is clearly in $2^{O(f(n))}$. Consider the following limit:

$$\lim_{n\to\infty} \frac{g(n)}{2^{f(n)}} = \lim_{n\to\infty} \frac{2^{3n}}{2^n} = \lim_{n\to\infty} 2^{3n-n} = \lim_{n\to\infty} 2^{2n} = \infty \tag{4}$$

This limit is not finite, meaning $g(n) \notin O(2^{f(n)})$.

6. Optional problems: You do not need to turn this question in and if you do, it will not be graded. However, to solidify your understanding of big-O and little-o, I recommend trying Sipser 7.1 and 7.2.