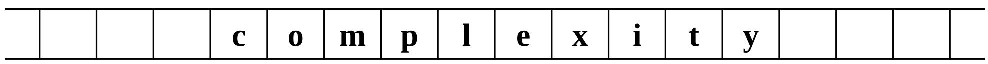
Computing Theory

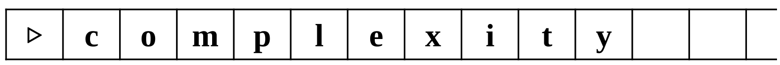
Midterm Examination on April 11, 2019

Spring Semester, 2019

Problem 1 (25 points) Define a single-string bidirectional Turing machine to be a single-string Turing machine which has infinite tapes in both directions (left and right). The computation is similar to an ordinary single-string Turing machine except that the cursor never encounters an end to the tape as it moves left. The tapes of a single-string bidirectional Turing machine is illustrated below.

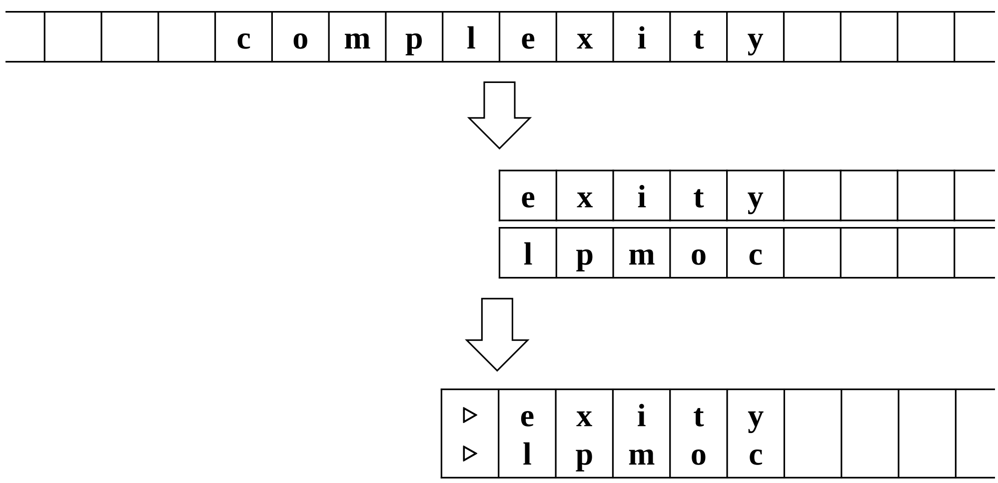


The tapes of an ordinary single-string Turing machine is illustrated below.



Sketch how given any single-string bidirectional Turing machine M operating within time $f(n)$, there exists an ordinary single-string Turing machine M' operating within time $O(f(n))$ such that $M(x) = M'(x)$ for any input x. (Remember to analyze the complexity.)

Proof: The construction of M' to simulate M is illustrated as below.



Construct M' by "folding" the tapes of M at an arbitrary location, say the input string's left border (and assume the cursor starts at the first symbol of the string, without loss of generality). If the symbol set of M is Σ, then the symbol set of M' contains Σ^2. We will work on the program of M to obtain the desired program for M'. Assume, without loss of generality, that the cursor of M' starts at the first symbol of the input string. To implement the two-way tape on a standard one-way tape, strings on the tape of M will be interpreted as a folded string: The string selected from the top symbols refers to the string of M to the right of the "fold", whereas the string selected from the bottom symbols refers to the string of M to the left of the "fold" but in a reverse order. See the above illustration. If M works to the right of the "fold", M' will work on the top symbols and follow the cursor instruction of M. If M moves to the left of the "fold", then M' will use the bottom symbols and change left movements into right movements.

The more formal construction of M' is described as follows. Modify the original program of $M(K, \Sigma, \delta, s)$ to obtain the new machine $M'(K', \Sigma', \delta', s')$, where K' includes $\{(q, i) : q \in K, i \in \{t, b\}\}$, where t and b represent the modes of M' (the top and bottom modes), and Σ' includes $\{(\sigma_1, \sigma_2) : \sigma_1, \sigma_2 \in \Sigma\}$. For every instruction $\delta(q, \sigma) = (p, \rho, D)$ of M, M' will have the following instructions:

$$\delta'((q,t),(\sigma,x)) = ((p,t),(\rho,x),D) \text{ for all } x \in \Sigma.$$

$$\delta'((q,b),(x,\sigma)) = ((p,b),(x,\rho),D') \text{ for all } x \in \Sigma, \text{ where } D' = \begin{cases} -, & \text{if } D = -. \\ \leftarrow, & \text{if } D = \rightarrow. \\ \rightarrow, & \text{if } D = \leftarrow. \end{cases}$$

Also, M' has the following instructions to reverse directions:

$$\delta'((q,t),(\triangleright,\triangleright)) = ((q,b),(\triangleright,\triangleright),\rightarrow) \text{ for all } q \in K.$$
$$\delta'((q,b),(\triangleright,\triangleright)) = ((q,t),(\triangleright,\triangleright),\rightarrow) \text{ for all } q \in K.$$

The input $x' = (x'_1, x'_2, ..., x'_n)$ to M' is the same as M's input $x = (x_1, x_2, ..., x_n)$ except that $x'_i = (x_i, \sqcup)$ and $(\triangleright, \triangleright)$ is the first symbol.

M' takes at most 2 steps to simulate each step of M in the above (maybe less than complete) formulation. As there are $f(n)$ steps of M, M' operates within time $O(f(n))$.

■

Problem 2 (25 points) Define the language

$$H_\epsilon = \{M \mid M \text{ halts on the empty string } \epsilon.\}.$$

Prove that H_ϵ is undecidable by reducing the halting problem to it. (Do not use Rice's theorem.)

Proof: Given the question "$M; x \in H$?", we construct the following machine:

$$M_x(y) : M(x).$$

Clearly, M halts on x if and only if M_x halts on ϵ. In other words, $M; x \in H$ if and only if $M_x \in H_\epsilon$. So if H_ϵ were recursive, H would be recursive, a contradiction. ∎

Problem 3 (25 points) Prove that the language

$$k\text{-REACHABILITY} = \{(G, a, b, k) \mid G \text{ is a directed graph where there exists a path of length at most } k \text{ from node } a \text{ to } b.\}$$

is in NL = NSPACE($\log n$).

Proof: The nondeterministic algorithm of k-REACHABILITY works as follows. Start at node a and repeatly and nondeterministically select the next node from the current node for up to k steps. If node b is ever reached, accept the input. Otherwise, reject the input. The algorithm only needs to record the current node and the next node; hence it runs in nondeterministic logarithmic space. ∎

Problem 4 (25 points) A NAND gate is a logic gate which produces an output "false" only if all its inputs are true. The truth table of NAND gate is illustrated as bellow.

A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0

Define a NAND Boolean circuit to be a Boolean circuit which contains only NAND gates. The problem NAND CIRCUIT VALUE asks, given an NAND Boolean circuit and a truth assignment to the input, what is the value of the output? Prove that NAND CIRCUIT VALUE is P-complete.

Proof: It is clear that NAND CIRCUIT VALUE is in P. For any Boolean circuit, NOT, AND, and OR gates can be replaced by following rules.

$$\text{NOT } x = x \text{ NAND } x.$$
$$x \text{ AND } y = (x \text{ NAND } y) \text{ NAND } (x \text{ NAND } y).$$
$$x \text{ OR } y = (x \text{ NAND } x) \text{ NAND } (y \text{ NAND } y).$$

We can transform any Boolean circuit into a NAND Boolean circuit by the above local substitution. Thus we can reduce the problem CIRCUIT VALUE into NAND CIRCUIT VALUE. Since CIRCUIT VALUE is P-complete, NAND CIRCUIT VALUE is also P-complete. ∎