1 Turing Machines

1.1 Designing TMs that compute simple functions

The following practice question asks to fully and formally flesh out a TM computing a simple function. The point is to ensure you understand the formal definitions perfectly.

Question 1. *Give a full and formal description of a Turing machine that shifts its input to the right, i.e. maps any input $x \in \{0,1\}^*$ to the output $0x$. (Your description should include the alphabet, the states, and the transition function.) Prove that your TM indeed computes this function and that it runs in linear time.*

Hopefully your solution also convinces you that we can compute $x \mapsto 1x$ in linear time, shift parts of an input that appear after a delimiter in linear time, and so on.

In class we saw a TM that computes $f(x) = x + 1$ when x is parsed as an integer with the LSB bit appearing first (i.e., left-most) and the MSB bit last (i.e., right-most).

Question 2. *Describe a TM that computes $f(x) = x + 1$ when integers are represented in the reverse order to the above, i.e. the MSB is left-most and the LSB is right-most.*

In the exercise above, you don't have to fully and formally describe the TM (alphabet, states, transition function): use words, at an abstraction level similar to what you see in the tutorials, and just make sure to be detailed and explicit enough. A reasonable reader should be convinced beyond any doubt that this is implementable as a TM (without needing to fill in non-obvious gaps).

Finally, we consider the language of palindromes, which decides whether a given string reads left-to-right the same way it reads right-to-left.

Question 3. *Let $L = \{w : w = w^{reverse}\}$ where $w^{reverse} = w_{|w|}w_{|w|-1}w_{|w|-2}, ..., w_1$ is the string w in reverse order. Describe a TM that decides L in quadratic time.*[1]

1.2 Variants of TMs

In many texts, TMs are described for the special case of deciding languages. Instead of having a state s_{end} that causes the machine to halt and output the worktape contents, such texts require two states s_{accept} and s_{reject}, and whenever the machine reaches any of these states, it halts and the output is defined accordingly.

Question 4. *Prove that any TM of the form above (i.e., with two final states for accept and reject) running in time $T(n)$ can be simulated by the standard TM in time $O(T(n))$.*

In class we explained how to simulate any two-tape machine by a standard (one-tape) machine. The explanation was rushed, giving the high-level idea but not fleshing it out in sufficient detail. The next question asks you to fully flesh it out.

[1] You may use the fact, which was stated in class and that you're asked to prove below, that two-tape TMs can be simulated by standard TMs with quadratic time overhead.

Question 5. *Prove that any two-tape TM running in time $T(n)$ can be simulated by the standard TM in time* $\mathrm{poly}(T(n))$.

Question 6. *Prove that for any constant $k \in \mathbb{N}$, any k-tape TM running in time $T(n)$ can be simulated by the standard TM in time* $\mathrm{poly}(T(n))$.

We mentioned in class that we can replace the one-way-infinite tape of a TM by a two-way-infinite tape, but we did not prove this. You will do so now.

Question 7. *Consider a TM model in which the head starts at location 1, with the input x written in locations $1, ..., |x|$, and the head can move not only indefinitely right as usual (wherein there are blank β symbols) but also indefinitely left (wherein there are also blank β symbols). Simulate any such TM that runs in time $T(n)$ by a standard TM in time* $\mathrm{poly}(T(n))$.

1.3 A sanity check

Recall that we formally only allow inputs and outputs that are Boolean strings. However, we sometimes informally describe problems as mapping non-Boolean strings to non-Boolean strings, as long as the alphabet is constant (e.g. allowing a delimiter symbol "," when we want to encode pairs of objects).

We learned in class to formalize this by encoding each non-Boolean symbol by a short Boolean string. The next question asks to verify that this trick does not cause a significant increase in the running time of any computation.

Question 8. *Let Σ be a constant-sized alphabet, and let M be a TM that computes a function $g\colon \Sigma^* \to \Sigma^*$ in time $T(n)$. Let $f\colon \{0,1\}^* \to \{0,1\}^*$ be a Boolean representation of g (i.e., each Σ-symbol is encoded by $\lceil \log |\Sigma| \rceil$ bits).*[2] *Prove that f is computable by a TM in time $O(T(n))$.*

1.4 Lower bounds for TMs

The following two questions ask to prove lower bounds on TMs. These lower bounds will rely on the fact that the input (respectively, the output) is long.

Question 9. *Prove that there is a language that cannot be decided in time $o(n)$.*

Question 10. *Prove that there is a function $f\colon \{0,1\}^* \to \{0,1\}^*$ that cannot be computed in time $o(2^{2^{2^n}})$ (or any other very big function that you like).*

These lower bounds are unsatisfying (try to articulate why!), and we will be interested in lower bounds that are super-linear (in fact, super-polynomial) and that hold for more interesting functions than the one you used to solve the second question.

[2] There are different ways to identify Σ-symbols with strings of length $\lceil \log(|\Sigma|) \rceil$, and the statement is true given any injective mapping $\Sigma \to \{0,1\}^{\lceil \log |\Sigma| \rceil}$ that defines f.