CS 332: *Theory of Computation* — Prof. Mark Bun
Boston University — October 15, 2022

Homework 5 – Due Thursday, October 20, 2022 at 11:59 PM

Reminder Collaboration is permitted, but you must write the solutions *by yourself without assistance*, and be ready to explain them orally to the course staff if asked. You must also identify your collaborators and write "Collaborators: none" if you worked by yourself. Getting solutions from outside sources such as the Web or students not enrolled in the class is strictly forbidden.

Note You may use various generalizations of the Turing machine model we have seen in class, such as TMs with two-way infinite tapes, stay-put, or multiple tapes. If you choose to use such a generalization, state clearly and precisely what model you are using.

Problems There are 5 required problems and 2 bonus problems.

1. (**Recognizability vs. Decidability**) Recall the high-level description of a TM recognizer for the language $\{\langle p\rangle \mid p$ is a k-variate integer polynomial and there exists $x_1, \ldots, x_k$ such that $p(x_1, \ldots, x_k) = 0\}$ that we described in class.

   Input : Encoding of k-variate polynomial p
   1. For every possible setting of $x_1, \ldots, x_k$ to integer values:
   2. Evaluate $p(x_1, \ldots, x_k)$. If it equals 0, *accept*.

   Explain in a few sentences what is **wrong** about the following attempt to construct a TM decider for the same language:

   Input : Encoding of k-variate polynomial p
   1. For every possible setting of $x_1, \ldots, x_k$ to integer values:
   2. Evaluate $p(x_1, \ldots, x_k)$.
   3. If any evaluation equals 0, *accept*. Otherwise, *reject*.

2. (**Closure properties**)

   (a) Show that the class of **decidable** languages is closed under complement.

   (b) Explain why your construction from part (a) **fails** to show that the **Turing-recognizable** languages are closed under complement. (That is, if L is Turing-recognizable, explain why your construction from part (a) does not necessarily produce a TM recognizing $\overline{L}$.)

3. (**String sorting**) We've introduced Turing machines as recognizers / deciders for languages, i.e., decision problems with yes-no answers. But it also often useful to consider Turing machines that compute functions with more complex outputs. Namely, a TM computes a function $f : \Sigma^* \to \Sigma^*$ if given $w \in \Sigma^*$ on its input tape, it halts with $f(w)$ on its output tape.

(a) Given a string w, let $\text{sort}(w)$ denote the string obtained by sorting the characters of w so that all 0's appear before all 1's. For example, $\text{sort}(011011) = 001111$.

Write a Turing machine M_{sort} that, given a string w on its tape, halts with $\text{sort}(w)$ on its tape. Implement your TMs in the following environment: http://morphett.info/turing/turing.html. Your solution should contain:

(i) An implementation-level description of your code.

(ii) Code that we can copy from your submission and run directly on that website. (Please add comments and make it as readable as possible.) There is a separate dropbox on Gradescope that will accept your code submissions.

(b) Define the shuffle operation on languages by $\text{shuffle}(L) = \{w \mid \text{sort}(w) \in L\}$. Show that the class of decidable languages is closed under shuffle. You can use an implementation-level description on a multi-tape TM to solve this part of the problem.

Hint: You may want to quote the TM M_{sort} you constructed in part (a) as a subroutine. If you do so, there's no need to rewrite out the description of this TM. You can just include a statement like "Run TM M_{sort} on ...."

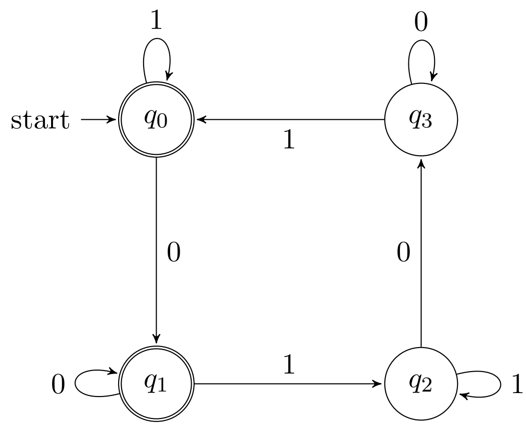
4. (**Universal DFA**) This short programming exercise aims to give you intuition about Turing machines that take more complicated objects as inputs – namely, DFAs. This week, we'll study the language $A_{\text{DFA}} = \{\langle D, w\rangle \mid D \text{ is a DFA accepting input } w\}$. Deciding membership in this language corresponds to the following computational problem: Given a DFA D and a string w, does D accept on input w?.

(a) The file `universal_dfa.py` contains starter code that will help you implement a ~~Turing machine~~ Python program solving this problem. Implement a program that prompts the user for an (appropriately encoded) DFA D and a binary string w, outputting i) the sequence of states D enters when run on w, and ii) whether D accepts or rejects input w. The starter code file describes the expected syntax for the input and output of your solution. There is a separate dropbox on Gradescope that will accept your code submissions.

This is not a software engineering class, so your program is allowed to fail arbitrarily (including failing silently) if its inputs do not correctly encode a DFA and a binary string.

If you don't like Python, you can implement your program in another high-level programming language (Java, C++, Haskell, ...) that the grading staff can read. (No Malbolge, please.) The downside is that you won't have the starter code to parse the input for you.

(b) Let w be the result of converting the numeric part of your BU UID to binary. (It doesn't matter exactly how you do this conversion. I just want you to feel some personal attachment to the string w you generate here.) Record the input and output of your program when you use it to determine whether the DFA represented by the following state diagram accepts input w.



5. (**Eco-friendly TM**) An *eco-friendly* Turing machine (ETM) is the same as an ordinary (deterministic) one-tape Turing machine, but it can read and write on both sides of each tape square: front and back.

   At the end of each computation step, the head of the eco-friendly TM can move left (L), move right (R), or flip to the other side of the tape (F).

   (a) Give a formal definition of the syntax of the transition function of an eco-friendly TM. (Modify Part 4 of Definition 3.3 on page 168 of the textbook.)

   (b) Show that eco-friendly TMs recognize the class of Turing-recognizable languages. That is, use a simulation argument to show that they have exactly the same power as ordinary TMs.

   You may use implementation-level descriptions of multi-tape TMs to solve this problem.

6. (**Bonus problem**) Let A be a Turing-recognizable language which is not decidable. (We will prove later in the course that such languages exist.) Consider a TM M that recognizes A. Prove that there are infinitely many input strings on which M loops forever.