Homework 6 – Due Tuesday, March 18, 2025 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 4 required problems.

1. (**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 use 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 ...."

2. (**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.

3. (**Nondeterministic Turing machines**)

   (a) Give a high-level description of a **nondeterministic** (multi-tape) TM deciding the following language over $\{0, 1, \#\}$: $\{s\#a_1\#a_2\#\ldots\#a_n \mid$ where n is a positive integer, $a_1, \ldots, a_n$ are binary integers such that some subset of $a_1, \ldots, a_n$ sums to exactly $s\}$. Explain why your NTM is correct.

   Note: It is possible to do this with a deterministic TM, but we want to give you practice with the concept of nondeterminism. So your solution must use an NTM's ability to nondeterministically guess in a meaningful way.

   (b) Given a Turing machine M, give a high-level description of a **nondeterministic** (multi-tape) TM recognizing $(L(M))^*$. Again, your solution must use nondeterminism in a meaningful way. Explain why your NTM is correct.

   (c) Explain why part (b) implies that the Turing-recognizable languages are closed under star.

   (d) Explain (briefly) how you would modify your previous construction and its analyses to show that the **decidable** languages are closed under star.

   Hint: Recall that a nondeterministic TM is a decider if it halts on every input, on every computation branch. The class of languages decided by NTMs is exactly the class of decidable languages.

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?.

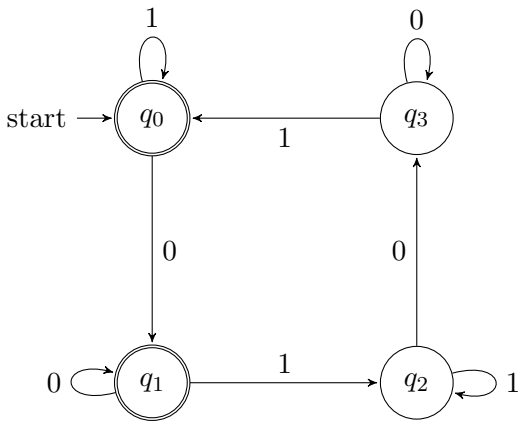
   (a) The file `universal_dfa.py` contains starter code that will help you implement a ~~Turing machine~~ Python program solving this problem. Implement a program that prompts the user for an (appropriately encoded) DFA D and a binary string w, outputting i) the sequence of states D enters when run on 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.

   IMPORTANT: DO NOT CHANGE THE NAME OF THE FILE `universal_dfa.py`. If you do, it will not be correctly auto-graded.

   (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.



The following two problems are postponed until Homework 7, but I'm leaving them here in case you want to start thinking about them now.

5. (**$\mathrm{SUB}_{\mathrm{DFA,REX}}$**) Consider the following computational problem: Given a DFA D and a regular expression R, is the language recognized by D a subset of the language generated by R?

   (a) Formulate this problem as a language $\mathrm{SUB}_{\mathrm{DFA,REX}}$.

   (b) Show that $\mathrm{SUB}_{\mathrm{DFA,REX}}$ is decidable by giving a high-level description of a Turing machine that decides it, together with an explanation of correctness.

   Hint: Following the examples in Sipser Chapter 4.1, you may assume that the procedures we've seen in class for converting back and forth between automata and regular expressions can be implemented on Turing machines.

6. (**Bonus problem**) Extend your code from Problem 4 to solve the problem corresponding to the language

$$E_{\mathrm{DFA}} = \{\langle D \rangle \mid D \text{ is a DFA recognizing the empty language}\}.$$

   That is, your program should prompt the user for an encoded DFA D and output "accept" if $L(D) = \emptyset$ and "reject" otherwise.