CS 332: *Theory of Computation* — Prof. Mark Bun
Boston University — March 5, 2021

Homework 5 – Due Thursday, March 11, 2021 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 and 2 bonus problems.

0. (**Midterm feedback survey**) Please fill out the mid-semester feedback form here `https://forms.gle/dffQdxXnJSrMuiTJA` to let us know what's working, what isn't working, and what we can do to improve. (You'll earn a bit of participation credit for acknowledging that you completed it in response to this question, but the survey is anonymous, so it's on the honor system.)

1. (**More closure properties**)

   (a) Given Turing machines M_1, M_2, give a high-level description of a **nondeterministic** (multi-tape) TM recognizing $L(M_1) \circ L(M_2)$. 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) Explain why part (a) implies that the Turing-recognizable languages are closed under concatenation.

   (c) 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.

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

   (e) Explain (briefly) how you would modify your previous constructions and their analyses to show that the **decidable** languages are closed under concatenation and 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.

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.

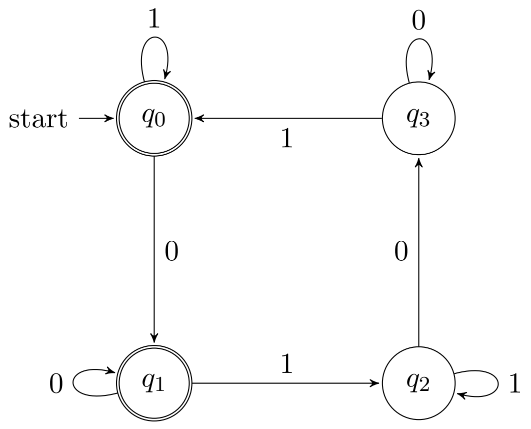
3. (**Universal DFA**) This short programming exercise aims to make the idea of designing a Turing machine that takes a DFA as input less disturbing. Recall the language $A_{\text{DFA}} = \{\langle D, w\rangle \mid D \text{ is a DFA accepting input } w\}$. Deciding membership in this language corresponds to the computational problem of determining whether DFA D accepts 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.

   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 x be your special binary input from HW4 Problem 3a. 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 x.



4. ($\text{EQ}_{\text{DFA,REX}}$) Consider the following computational problem: Given a DFA D and a regular expression R, is the language recognized by D equal to the language generated by R?

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

   (b) Show that $\text{EQ}_{\text{DFA,REX}}$ is decidable.

   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.

5. (**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.

6. (**Bonus problem**) Extend your code from Problem 3 to solve the problem corresponding to the language

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

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