CS 332: *Theory of Computation* Prof. Mark Bun
Boston University October 21, 2021

# Homework 6 – Due Tuesday, October 26, 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 5 required problems.

1. **(**$\text{ALL}_{\text{REX}}$**)** Consider the following computational problem: Given a regular expression $R$ (over some alphabet $\Sigma$), is the language generated by $R$ equal to $\Sigma^*$?

   (a) Formulate this problem as a language $\text{ALL}_{\text{REX}}$, in the style of the languages described in Sisper Chapter 4.1.

   (b) Show that $\text{ALL}_{\text{REX}}$ is decidable. You may give a high-level description of the Turing machine your construct.

   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, and just cite these constructions.

2. (**Universal TM**) In this short programming exercise, you will (sort of) build the universal Turing machine. Hopefully this makes the idea of designing a Turing machine that takes another Turing machine as input less disturbing. Recall the language $A_{\text{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM accepting input } w\}$. Determining membership in this language corresponds to the computational problem of determining whether TM $M$ accepts input $w$.

   The file `universal_tm.py` contains starter code that will help you implement a ~~Turing machine~~ Python program recognizing this language. Implement a program that prompts the user for an (appropriately encoded) TM-with-stay-put $M$ and a binary string $w$, outputting i) the sequence of configurations $M$ enters when run on $w$, and ii) whether $M$ accepts or rejects input $w$, if it terminates. Your program is (necessarily) allowed to run forever if $M$ runs forever on $w$. 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 TM and a binary string.

   If you really 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.

3. (**Code as data**) Consider the following description of a three-tape TM $H$.

```
Algorithm: $H(\langle M, N \rangle)$
  Input : Binary encoding of two basic TMs $M$ and $N$, i.e., $\langle M, N \rangle \in \{0,1\}^*$
  1. Copy the string $\langle M \rangle$ to tape 2.
  2. Copy the string $\langle N \rangle$ to tape 3.
  3. Repeat the following three steps forever:
  4.    Simulate $N$ for one step on tape 2.
  5.    Simulate $M$ for one step on tape 3.
  6.    If either machine accepts, accept. If both machines have rejected, reject. Else, continue.
```

Also, define the following three TMs, whose descriptions will be supplied as input to $H$.

```
Algorithm: $M_1(x)$
  Input : String $x \in \{0,1\}^*$
  1. Ignore the input $x$ and reject.
```

```
Algorithm: $M_2(x)$
  Input : String $x \in \{0,1\}^*$
  1. If $x = \langle M_1 \rangle$, accept. Otherwise, reject.
```

```
Algorithm: $M_3(x)$
  Input : String $x \in \{0,1\}^*$
  1. Let $M$ be the TM such that $\langle M \rangle = x$.
  2. For $i = 1, 2, 3, \ldots$:
  3.    For each string $y$ where $|y| \leq i$:
  4.       Run $M$ on input $y$ for $i$ steps. If it accepts, accept.
```

(a) What is $L(M_1)$?

(b) What is $L(M_2)$?

(c) What is $L(M_3)$?

(d) Is $\langle M_1, M_2 \rangle \in L(H)$? Explain why or why not.

(e) Is $\langle M_1, M_3 \rangle \in L(H)$? Explain why or why not.

(f) What is the language $L(H)$ recognized by $H$?

(g) Is $H$ a decider for the language $L(H)$? Explain why or why not.

4. (EVEN$_{\text{TM}}$) Consider the following computational problem: Given a Turing machine $M$ (over some alphabet $\Sigma$), does there exist a string $w$ of even length such that $M$ accepts input $w$?

   (a) Formulate this problem as a language EVEN$_{\text{TM}}$.

   (b) Show that EVEN$_{\text{TM}}$ is Turing-recognizable. You may give a high-level description of the Turing machine your construct.

   Hint: Think about the TM $M_3$ from Problem 3 above. Why was it constructed that way? A similar idea will be helpful for this problem.

5. (**Countable sets**)

   (a) Aliens from the planet Foobar have finite single-strand DNA sequences consisting of the nucleobases A, C, G, and T. For example, ACGTTAG and CGATCGACTGCA are both possible DNA sequences. Let $\mathcal{F}$ be the set of all possible DNA sequences for residents of Foobar. Show that $\mathcal{F}$ is countable.

   (b) Aliens from the planet Foobarbaz have finite single-strand DNA sequences from a countably infinite set of nucleobases $\{A_1, A_2, A_3, \dots\}$. For example, $A_2A_4A_8A_1A_9$ and $A_3A_{747}A_{9999999}A_4$ are both possible DNA sequences. Let $\mathcal{B}$ be the set of all possible DNA sequences for residents of Foobarbaz. Show that $\mathcal{B}$ is countable.