CSCI 3434, CU Boulder **Prof. Rafael Frongillo**
Fall 2020 **Problem Set 7**
Collaborators: Frank Stinar **Student: Alex Book**

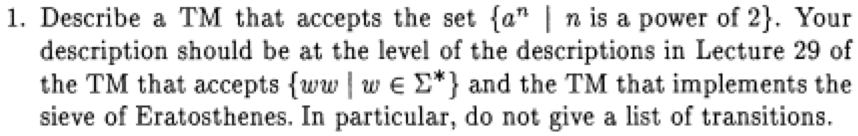
- **Do not use external sources** beyond the materials linked to on the course website to solve these problems.
- You are encouraged to work with ($\leq$ 2) other students, but you must **write your solutions independently**.
- Be sure to **list your collaborators** by name clearly at top of your submission, or "no collaborators" if none.
- Recall: letters correspond to the section of the book: [H]omeworks starting page 301, [M]iscellaneous [E]xercises starting page 315.

1. Basics - Computability (S18)
Coming soon – look out for "PS7 - S18" in Canvas starting around Monday Oct 19.

Completed on Canvas.

2. Designing Turing Machines (S19)
Your descriptions should be at the same level as the examples in the book (see H8.1).

a. H8.1


1. Describe a TM that accepts the set $\{a^n \mid n \text{ is a power of } 2\}$. Your description should be at the level of the descriptions in Lecture 29 of the TM that accepts $\{ww \mid w \in \Sigma^*\}$ and the TM that implements the sieve of Eratosthenes. In particular, do not give a list of transitions.

We will use a two-tape Turing machine. The first tape holds the input string. The second tape starts with one 'a'.

i. The head on the first tape starts at the left end symbol, and the head on the second second tape starts at the last non-empty symbol (initially it starts on the single 'a', but the need for the specification of starting on the last non-empty symbol will become apparent below).

ii. The head on the first tape moves to the right each step unless otherwise specified. The head on the second tape moves left each step until it reaches its left end symbol, where it switches to moving right each step (unless otherwise specified).

iii. If the head on the second tape reaches its last non-empty symbol before the head on the first tape reaches its last non-empty symbol, the following happens:

   A. The second tape doubles its amount of a's, which is done trivially (we can mark every original 'a', then read through the string and write each "companion" 'a' to the end of the string (writing over ␣'s), each time removing the mark from one of the original a's, resulting in double the amount of a's as the string started with).

   B. The process resets to step (i) while maintaining the tapes the entire time (so the second tape's doubling in its amount of a's is preserved).

iv. If the head on the first tape reaches its last non-empty symbol before the head on the second tape reaches its last non-empty symbol, the Turing Machine halts and rejects.

v. If the head on each tape reaches its last non-empty symbol at the same time (on the same step), the Turing Machine halts and accepts.

b. Let $A = L(1(1+0)^* + 0)$ be the set of binary strings with no leading zeros. Given $x \in A$, let $\#x$ denote the number x represents in binary. Your task is to implement a binary counter using a Turing machine: Design a TM M such that, given input $x \in A$, halts with just y on the tape, where $\#y = \#x + 1$. More precisely, the final tape should be $\vdash y \sqcup^\omega$. For example, on input 10011, the machine halts with only 10100 on the tape.
Hint: Sometimes the output will need to be longer than the input.

Note: In saying "overflow," I mean that adding 1 to the input string will result in a string that is longer than the input.

First, we will scan left to right and observe the entirety of the input string x. There are three possible findings:

i. If we see a 0 first and non-empty characters after, we halt and reject, as the input isn't of the proper form.

ii. If we see a 0 first and empty characters after, we write a 1 in its place, then halt and accept. The resultant string has a value of one more than the input string.

iii. If the string has some mixture of 0's and 1's (at least one of each), we know that adding 1 to the input won't cause overflow, as the addition process will be "caught" by a 0 at some point.In this case, when scanning back to the right, we will perform the addition of 1 to the input.
   - If we see a zero first, we write a 1 in its place, then halt and accept. The resultant string has a value of one more than the input string.
   - If we see a 1 before a 0, we write a 0 on that tape spot. As long as we encounter another 1, we write a 0 on that tape spot. As soon as we encounter a 0, we write a 1 on that tape spot, then halt and accept. The resultant string has a value of one more than the input string.

iv. If the input string has only 1's, adding 1 to it will cause overflow, as the necessary resultant string would have a length one greater than the input string (it would be one 1, followed by a number of 0's equal to the length of the input string; i.e. for input string 1111, the desired resultant string would be 10000). In order to create this resultant string, we will scan back to the beginning while doing nothing (writing exactly what we read), then begin scanning back to the left. We then do as follows:
   - At the first character, we write a 1 and move to the right.

- Then, as long as we encounter a 1, we write a 0 on that tape spot and move to the right.
- When we encounter a $\sqcup$, we write a 0, then halt and accept. The resultant string has a value of one more than the string.