Week 8: Turing Time

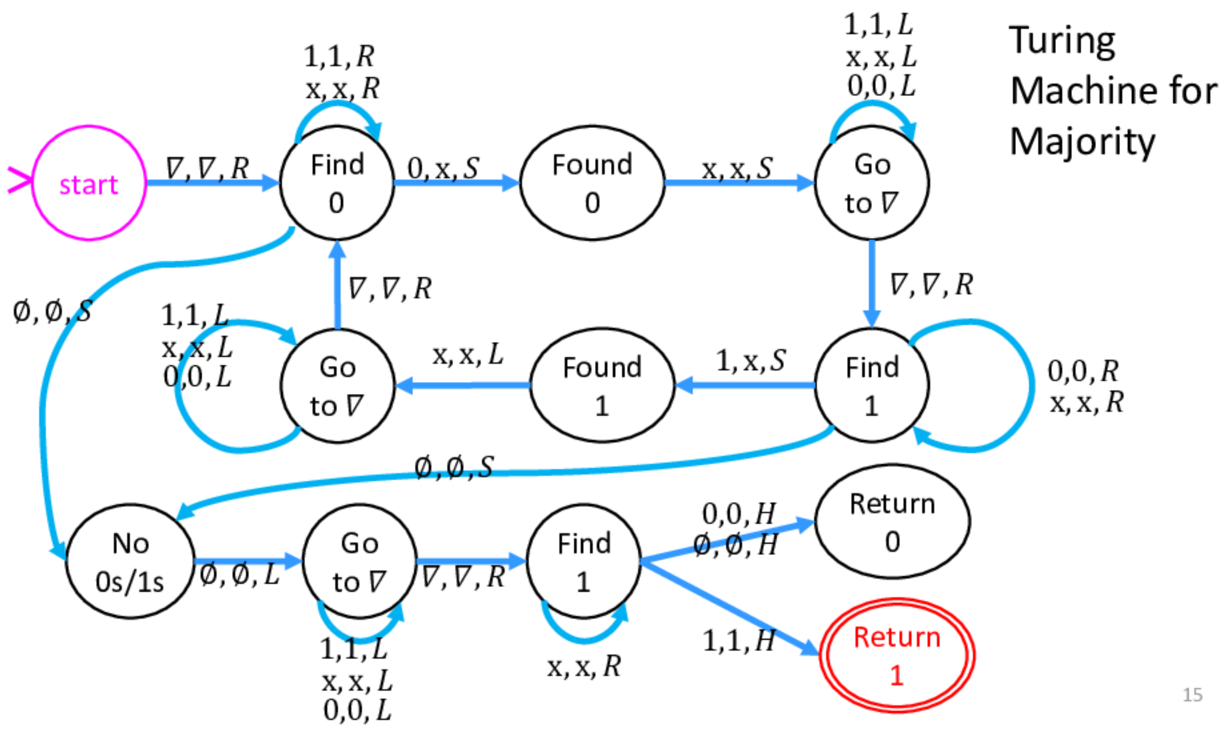
Collaboration Policy: You should work on the problems yourself, before discussing with others, and with your cohorts are your cohort meeting. By the Assessed Cohort Meeting, you and all of your cohortmates, should be prepared to present and discuss solutions to all of the assigned problems (including the programming problems). In addition to discussing with your cohortmates, you may discuss the problems with anyone you want, and use any resources you want except for any materials from previous offerings of this course, which are not permitted. You should document any resources you use (beyond the provided course materials) in your problem write-up.

Problem 1 *Turing Machine for Majority*

In the *Turing Machine Examples* video we begin but do not complete an "Accept/Reject" Turing machine for computing Majority (i.e., for a given input bit string, are there more 1s than 0s?). The behavior of the Turing Machine is as follows (note that this description roughly matches the expected level of specificity of Turing Machine descriptions):

1. Start from the beginning of the tape where a "start of tape" symbol (the $\triangle$) precedes the input
2. Move right through the tape, looking for a 0.
3. If a 0 is found, write over it with an x, then go back to the $\triangle$ (start of tape)
4. Move right through tape, looking for a 1.
5. If a 1 is found, write over it with an x, then go back to the $\triangle$ (start of tape)
6. Repeat the "look for 0, mark if found, look for 1, mark if found" steps until a blank cell (the symbol) is reached (meaning there are either no more 0s or no more 1s)
7. Go back to the beginning of the tape, looking for 1s
8. if a 1 is found along the way, return 1. Otherwise, return 0.

The following image depicts one potential implementation of the machine described above:



In this machine, each transition is a triple, where the first value is the symbol read from the tape when taking that transition, the second is the symbol to write onto the tape in its place, and the third symbol is a control instruction (L for move left in the tape, R for move right in the tape, S for stay put in the tape, and H for halt).

Consider that this machine is given each of the inputs describe below, all of which are n bits long. Give a Θ bound on the total number of transitions this machine would take before halting as a function of the input length n.

a. A string of n 1s

b. A string of n 0s

c. A string of $\frac{n}{2}$ copies of 01 pairs (e.g. 010101...). You may assume n is even.

d. A 1 followed by $\frac{n-1}{2}$ copies of 01 pairs (e.g. 1010101...). You may assume n is odd.

Problem 2 *Simplify the Machine*

Modify or re-design the Majority Turing machine provided above so that it uses at least one less state but still computes the same function.

Problem 3 *Configurations*

A machine *configuration* contains all of the information needed to completely represent the "status" (we're avoiding using "state" here because of the confusion with the internal machine state, which does not fully capture the execution state of a Turing machine) of an execution. Essentially, a configuration should have all of the necessary details to be able to pause an execution and save its status so that it can be resumed later.

As an example of this idea, operating systems must be able to occasionally pause computer programs temporarily in order to use the CPU for another process. When doing this, the operating system will store a configuration of that program (often called an *image*) so it can perfectly pick up from where it left off. For a Python program, this configuration would most likely include the values of all variables, the contents of all data structures, the current call stack (which functions have invoked with other functions), and the program counter that identifies the next instruction to execute.

Answer the following regarding machine configurations.

a. List all of the things that should be included in the configuration of a Deterministic Finite State Automaton.

b. List all of the things that should be included in the configuration of a Turing Machine (select any definition of a TM you would like).

c. One necessary reason that Turing Machines are more powerful than finite state automata is that a particular machine could have an infinite number of possible configurations. Identify what component(s) of your Turing Machine configuration would allow for an infinite number of configurations.

d. ($\star$) Show that any Turing machine with only finitely many possible configurations computes a regular language.

Problem 4 *A Tale of Two Turing Machines*

In the *first Turing Machines video* we present a model where the machine has special "Accept" and "Reject" states, and output is determined by entering one of those states. In the *Turing Machine Execution video* we present a slightly different model (similar to the one in the TCS book) in which the Turing Machine writes its output on a special tape. The next few questions will be a discussion of the similarities and differences between these models.

a. First, show that any "Accept/Reject" Turing Machine can be converted into an equivalent "Output on Tape" machine

b. Discuss whether, in the general case, an "Output on Tape" machine can be converted into an equivalent "Accept/Reject" machine (or multiple such machines).

c. By this point in the semester, we've seen lots of subtly different models of computing all trying to accomplish similar goals. Discuss why there's so much disagreement about how a Turing Machine should be defined. (There's no proof or formal answer we're expecting for this part, but a thoughtful discussion of why there are different ways of defining the TM and when it might matter which one we choose and when it doesn't. Especially thoughtful answers will consider whether the situation is the same or different for other models we've explored such as Boolean circuits, DFAs, and regular expressions.)

Problem 5 *Computable Numbers*

Review the *discussion of computable numbers*. Be prepared to answer questions about that discussion such as (but not limited to): *are all integers computable?*, *what does it mean for an irrational number to be computable?*, *how do we know uncomputable numbers exist?*, *does the way we represent numbers impact what is computable?*, *would aliens have the same notion of uncomputable numbers as we do?* and *does it matter that there are uncomputable numbers?*.

Problem 6 *NAND-TM Closure*

Using the NAND-TM model of computing from *TCS Chapter 7* (we don't discuss this in the lectures, but you should be able to follow the book's presentation linked here), demonstrate that computable languages are closed under `AND`. In other words, show that if $f_1 : \{0,1\}^* \to \{0,1\}$ and $f_2 : \{0,1\}^* \to \{0,1\}$ can each be computed by a NAND-TM program, then $f_{\text{AND}}(x) = \texttt{AND}(f_1(x), f_2(x))$ can also be computed by a NAND-TM program. To do this, you may use implementations of f_1 and f_2 as "sugar" (like function calls) when implementing f_{AND}.