PS8: Turing Machines

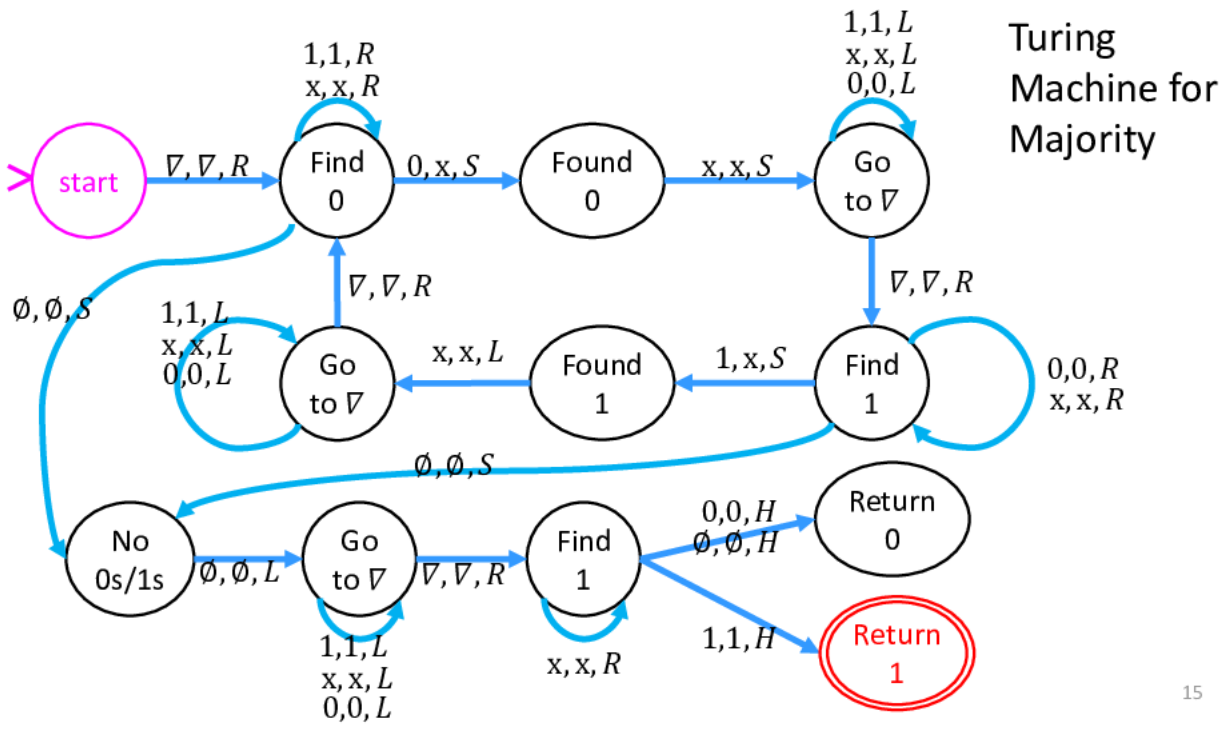
Collaboration: You should work on the problems yourself, before discussing with others, including your cohorts at your cohort meeting. In addition to discussing with your cohortmates, you may discuss the problems with any other current CS3102 students you want, and use any resources you want except for any materials from previous offerings of this course or complete solutions that might be available on the web, which are not permitted. Sharing and subsequently submitting any text, code, images, figures, etc. constitutes plagiarism, so make sure all submitted materials are created exclusively by members of your cohort.

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 1 s than 0 s?). The behavior of the Turing Machine is as follows (note that this description roughly matches the expected level of specificity of Turing Machine “psuedo-code”):

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 0 s or no more 1 s)
7. Go back to the beginning of the tape, looking for 1 s
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. You should informally justify how you came to that conclusion (e.g. "after each 0 we overwrite we need to take one more transition"), but a formal proof is not necessary.

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. n 0's followed by n 1s (e.g. 00001111).

e. 10 0's followed by $n-10$ 1s (e.g. 00000000001111 or 00000000001111111111111 which is 10 zeros then 13 1s).

f. $n-10$ 1's followed by 10 0s (e.g. 11110000000000 or 11111111111110000000000 which is 13 1s then 10 zeros).

Problem 2: Simplify The Machine

Modify or re-design the Majority Turing machine provided above so that it uses at least one fewer state but still computes the same function. You may either re-draw the machine (or a portion of the machine) or else clearly describe what change(s) you would make. Either way, be sure to mention which state(s) were removed.

Problem 3: Configurations

A machine *configuration* contains all of the information needed to completely represent a snapshot (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 and save an execution so that it can be resumed later. By convention, the description of the machine itself and the input is not included in the configuration (unless necessary according to the model of computing).

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/Java 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. Essentially, all the information you would see if using a debugging tool in an IDE.

As another example, a deterministic finite state automaton would only need to record the current state it's in. Consider what should be included in the configuration of a NAND circuit and non-deterministic finite state automaton. You don't need to submit an answer for this, but please discuss with cohort-mates, TAs, or discord to verify you understand what we mean by configuration. After you've done this, answer the questions below.

Answer the following regarding Turing machine configurations.

(a) 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).

(b) 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.

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

Problem 4: Describe the TM

Each subproblem below gives a function that can be computed by a Turing machine. Describe, using high level Turing machine "pseudo-code", how a Turing machine could compute the given function. For example, your pseudo-code might have statements like "scan left until you find a 1, then cross it off", or "halt and return 1 if there you get to the start of the tape without seeing a 1". Refer to Problem 1 to see an example.

For all problems, you can assume there is a way to encode the input for the Turing Machine as a string (that's not necessarily binary). For example, if there are multiple inputs, they would be separated by a special symbol $\#$ on the input tape to distinguish them.

(a) for $x \in \{\texttt{0}, \texttt{1}\}^*$,

$$OR(x) = \begin{cases} 1 & \text{if there exists a } \texttt{1} \text{ in } x \\ 0 & \text{otherwise} \end{cases}$$

(b) for $x \in \{\texttt{0}, \texttt{1}\}^*$,

$$CoolEnough(x) = \begin{cases} 1 & \text{if } x \text{ contains at least 3102 } \texttt{1s} \\ 0 & \text{otherwise} \end{cases}$$

(c) for $w \in \{\texttt{0}, \texttt{1}\}^*, x \in \{\texttt{0}, \texttt{1}\}^*$,

$$MoreOnes(w, x) = \begin{cases} 1 & \text{if } x \text{ has more } \texttt{1s} \text{ than } w \text{ does.} \\ 0 & \text{otherwise} \end{cases}$$

Problem 5: Treat yourself

I wanted to take this time to make sure you're taking care of yourself. At this point we're at week 11 of 15 weeks for the semester, and half way between spring break and final exams. This is the time in the semester when many of you will be having your second midterms in your courses, your projects really getting underway, your final papers assigned, and just in general the content is getting *real*. Instead of another theory problem, I hope you can take this time to do something for yourself. For example:

– Plant something

– Change your bed sheets

– Vacuum

– Sun bathe

– Make a loved one smile

– Change a lightbulb

– Do a craft

– Bake NAND-Gate Cookies

– Alphabetize something

– De-alphabetize something

– Deliver encouragements to your neighbors/hallmates/suitemates

If you're in a cohort, share what you did/plan to do with your cohort. I also encourage you to share in the PS8 channel of discord. Pics are encouraged!

You do not need to submit anything for this problem, we'll just trust that you did something.