Problem Set 8

Due: **10:00pm, Monday, April 7**

This problem set focuses on the definition of Turing machines. In particular, we consider and discuss a couple of variants of Turing machines. We will also write a Turing to compute majority.

Write your answers in the ps8.tex LaTeX template. You will submit your solutions in GradeScope as a PDF file with your answers to the questions in this template. There are four "required" problems and one "bonus" practice, where the bonus gives extra points.

Collaboration Policy: You may discuss the problems with anyone you want. You are permitted to use any resources you find for this assignment **other than solutions from previous/concurrent CS3120 courses**. You should write up your own solutions and understand everything in them, and submit only your own work. You should note in the *Collaborators and Resources* box below the people you collaborated with and any external resources you used. You shall explicitly state the *content*, e.g., the main message in your collaborated discussion, the search keywords, the LLM/AI prompts, or the section in a book.

Collaborators and Resources: TODO: replace this with your collaborators and resources (if you did not have any, replace this with *None*)

To do this assignment:

1. Open this read-only Overleaf project located at https://www.overleaf.com/read/bqsbcnyzvbvb#2b6c04, and then copy this project.
2. Open your copy of the project and in the left side of the browser, you should see a file directory containing ps8.tex. Click on ps8.tex to see the LaTeX source for this file, and enter your solutions in the marked places.
3. The first thing you should do in ps8.tex is set up your name as the author of the submission by replacing the line, `\submitter{TODO: your name}`, with your name and UVA id, e.g., `\submitter{Haolin Liu (srs8rh)}`.
4. Write insightful and clear answers to all of the questions in the marked spaces provided.
5. Before submitting your PDF file, also remember to:
   - List your collaborators and resources, replacing the TODO in `\collaborators{TODO: replace ...}` with your collaborators and resources. (Remember to update this before submitting if you work with more people.)
   - Replace the second line in ps8.tex, `\usepackage{uvatoc}` with `\usepackage[response]{uvatoc}` so the directions do not appear in your final PDF. Starting from this Problem Set, we may deduct 3pt if you forgot this step.

Definition 1 (TCS textbook Turing machine) *A* Turing machine *is defined by a tuple* (Σ, k, δ) *such that*

– $k \in \mathbb{N}$ *is a finite number that describes the number of states.*

– $\Sigma = \{0, 1, \triangleright, \emptyset\}$ *is a finite set of symbols.*

– $\delta : [k] \times \Sigma \to [k] \times \Sigma \times \{L, R, S, H\}$ *is a function, called a transition function.*

Problem 1 (6pt) *Large alphabet Turing machines*

We sometimes assumed that the alphabet of Turing machine is binary, that is, $\Sigma = \{0, 1, \triangleright, \emptyset\}$. That is not necessary, and it is easier to write Turing machines given a larger alphabet.

For any $A \in \mathbb{N}$, define *A-alphabet TM* to be the variant of Turing machines where the alphabet is $\Sigma = [A] \cup \{\triangleright, \emptyset\}$. The definition of transition functions and the execution is the same as in Definition 1, and the input and output are defined to be a sequence in $[A]^*$. Notice that when $A = 10$, A-alphabet TM works on digits, and when $A = 10 + 26$, A-alphabet TM works on digits and a-z letters, and so on for more symbols.

Prove that for any constant $A \in \mathbb{N}$, for every function $F : \{0, 1\}^* \to \{0, 1\}^*$, F is computable by a Turing machine if and only if F is computable by a A-alphabet TM.

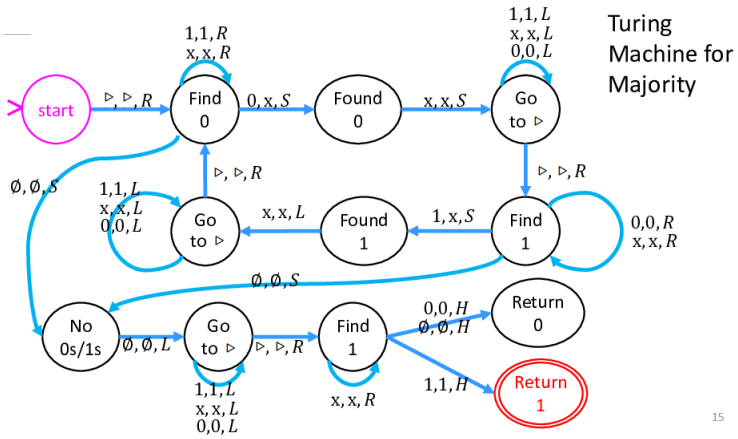
Answer:

Problem 2 (2pt each, 10pt total) *Turing Machine for Majority*

Consider a Turing machine for computing Majority (i.e., for a given input bit string, are there more 1s than 0s?). The behavior of this Turing Machine, informally, is as follows.

1. Start from the beginning of the tape where a "start of tape" symbol (the ▷) 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 ▷ (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 ▷ (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.

(e) A string of $\frac{n}{2}$ 0s followed by $\frac{n}{2}$ 1s (e.g. 00001111). You may assume n is even.

Answer:

(a)

(b)

(c)

(d)

(e)

Problem 3 (5pt) *Simplification*

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.

Answer:

Problem 4 (1,1,2pt) *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.

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 which other functions), and the program counter that identifies the next instruction to execute.

Answer the following sub-problems regarding machine configurations:

(a) List all of the things that should be included in the configuration of a nondeterministic 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.

Answer:

(a)

(b)

(c)

Practice 1 (Bonus, 2pt each, total 4pt) *How to Decide Differently*

In class, we saw that Turing machines write their outputs in their tape before halting. Consider an alternative way in which Turing machines could Accept or Reject (when the function to compute has Boolean output) as follows. Each machine will have a pair of special states q_A, q_R, but no halting state. The machine might enter either of q_A or q_R, after which it halts. If the machine enters state q_A that means the input is accepted; if it enters state q_R it means the input is rejected. We call this an "Accept/Reject" Turing Machine.

The next few questions will be a discussion of the similarities and differences between this variation of the Turing Machine and the "Output-on-Tape" Turing Machine description from class (and the book).

(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).

Answer:

(a)

(b)

Do not write anything on this page; leave this page empty.

This is the end of the problems for PS8. Remember to follow the last step in the directions on the first page to prepare your PDF for submission.