24.118: Paradox and Infinity, Spring 2019
Problem Set 9: Computability

How these problems will be graded:

- In Part I there is no need to justify your answers. Assessment will be based on whether your answers are correct.
- In Part II you must justify your answers. Assessment will be based both on whether you give the correct answer and on how your answers are justified. (In some problem sets I will ask you to answer questions that don't have clear answers. In those cases, assessment will be based entirely on you justification. Even if it is unclear whether your answer is correct, it should be clear whether or not the reasons you have given in support of your answer are good ones.)
- *No answer may consist of more than 150 words.* Longer answers will not be given credit. (Showing your work in a calculation, a proof, or a computer program does not count towards the word limit.)
- You may consult published literature and the web. You must, however, credit all sources. Failure to do so constitutes plagiarism and can have serious consequences. For advice about when and how to credit sources see: https://integrity.mit.edu (You do not need to credit course materials.)

Important: Answers that call for a Turing Machine program will only be given credit if they are submitted as part of a PDF whose code can be copied and pasted onto the following simulator: `morphett.info/turing/turing.html`. Please test your PDF using Adobe's Acrobat Reader before submitting it, by making sure your code works as intended after being copied and pasted into the simulator. (LaTeX users might consider using the "verbatim" environment, or an environment intended for computer code.)

Part I

1. The aim of this problem is to get you to think about how to code Turing Machines as natural numbers. Please use the coding system described in Section 9.1.5 of the textbook.

   (a) i. Which natural number codes the following Turing Machine? (5 points)

   ```
   0 _ _ 1 1
   ```

   ii. Give an informal description of the behavior of that Turing Machine, when run on an empty input. (2 points)

(b) i. Which Turing Machine is coded by the number 11,550? (5 points)

ii. Give an informal description of the behavior of that Turing Machine, when run on an empty input. (2 points)

(c) Could there be a function f such that, for distinct numbers n and m, n and m both code Turing Machines that compute f? (6 points)

2. Ternary notation is to 3 what decimal notation is to 10 and what binary notation is to 2. In other words: one works with three digits ("0", "1", and "2") and lets the string "$d_k \ldots d_1 d_0$" refer to the number $d_0 \cdot 3^0 + d_1 \cdot 3^1 \cdot \cdots \cdot d_k \cdot 3^k$.

   Design a Turing Machine that does the following: when given as input a natural number $n \geq 1$ in ternary notation, followed by a blank, followed by natural number $m \geq 1$ in ternary notation, it halts with the number $n + m$ in ternary notation on an otherwise blank tape.

   Here is an example, to illustrate how your Turing Machine ought to work. Suppose $n = 47$ and $m = 64$. In ternary notation, "1202" refers to the number 47 and "2101" refers to the number 64. So your machine should start out with the following string of symbols on an otherwise blank tape:

   1202 2101

   Since $47 + 64 = 111$ (and since, in ternary notation, "11010" refers to the number 111), your Turing Machine should halt with the following string of symbols on an otherwise blank tape:

   11010

   And, of course, you want this to work for arbitrary n and m. You may use auxiliary symbols, if you need them. (5 points)

3. Suppose you're interested in minimizing the number of states required by your Turing Machine. One strategy is to come up with a clever algorithm. Another strategy is to start with an algorithm that requires many states and bring down the number of states by increasing the number of auxiliary symbols that your Turing Machine is allowed to print on the tape. Use either of these strategies (or a combination of the two) to solve the following problems. Your Turing Machines are allowed to use as many auxiliary symbols as they need. (10 points each)

   (a) Design a Turing Machine that has no more than two states and behaves as follows: whenever it is given a sequence of n ones as input ($n > 0$), it halts with a sequence of n ones, followed by a blank, followed by a one, with the reader positioned at the left-most one of the initial sequence.

   (b) Design a Turing Machine that has no more than two states and behaves as follows: for some $n \geq 20$, when run on an empty input, it halts with at least n cells of the tape containing non-blanks. (No need to worry about where the reader ends up.)

(c) Design a Turing Machine that has no more than two states and behaves as follows: when given as input a natural number $n \geq 1$ in binary notation, it halts with the number $4n + 3$ in binary notation on an otherwise blank tape. (No need to worry about where the reader ends up.)

Part II

4. The following questions are meant to get you to think about the Halting Function.

   For each of the descriptions below, determine whether there could be a Turing Machine satisfying that description. (5 points each; don't forget to justify your answers)

   (a) A Turing Machine M that behaves as follows when given the code of a Turing Machine M' as input:

   - If M' halts when run on an empty input, M halts.
   - If M' doesn't halt when run on an empty input, M doesn't halt

   (b) A Turing Machine M that behaves as follows when given the code of a Turing Machine M' as input:

   - If M' halts when run on an empty input, M outputs a 1
   - If M' doesn't halt when run on an empty input, M outputs a 0

   (c) For a given Turing Machine M', a Turing Machine M that behaves as follows on an empty input:

   - If M' halts when run on an empty input, M outputs a 1
   - If M' doesn't halt when run on an empty input, M outputs a 0

5. Section 9.2.2 of the textbook contains a proof that the Busy Beaver function is not Turing-computable. The following problems are aimed at getting you to think about that proof. (5 points each; don't forget to justify your answers.)

   (a) For k an arbitrary positive integer, design a (one-symbol) Turing Machine that does the following, given an empty input: it produces a sequence of k ones, brings the reader to the beginning of the sequence, and halts. Please make sure your machine has $k + C$ states for some constant C.[1]

   (b) Design a (one-symbol) Turing machine that does the following, given a sequence of n ones as input: it produces a sequence of $2n$ ones, brings the reader to the beginning of the sequence, and halts.

   (c) Design a (one-symbol) Turing machine that does the following, given a sequence of n ones as input: it produces a sequence of $n + 1$ ones, brings the reader to the beginning of the sequence, and halts.

[1] This condition is not required to prove that the Busy Beaver function is not Turing-computable, but it'll simplify our discussion below.

(d) The proof in Section 9.2.2 of textbook works with a hypothetical Turing Machine M^I. The characterization of M^I presupposes a Turing Machine M^{BB}, which computes the Busy Beaver Function. Since the Busy Beaver Function is not Turing-Computable, M^{BB} doesn't actually exist (and so neither does M^I). For the purposes of this exercise, however, I'd like you to pretend that M^{BB} does exist, and has the following program:

```
; fake version of BB machine
0 _ _ l 1
0 1 1 r 0
1 _ _ r halt
1 1 1 l 1
```

Use this pretense, together with your answers to problems (5*a*)–(5*c*), to explicitly write out a program for M^I for the special case in which $k = 3$.

(*Request:* as a courtesy to your TA, please annotate your program so as to make it easy to understand which parts of your code correspond to the fake version of M^{BB} above and which correspond to your answers to problems (5*a*)–(5*c*).)

(e) Your answer to (5d) is a Turing Machine program. Count the states in that program. Now assume that instead of working on the assumption that $k = 3$, you had been asked to work with an arbitrarily given k. How many states would your code for M^I have had in that case, as a function of k?

(*Hint:* your answer should be of the form $k + C$ for some constant C.)

(f) If the two-state machine I supplied in problem (5d) had really computed the Busy Beaver Function, then your code for M^I would have computed the function $BB(2k)+1$. So your code would have been more productive—by one—than the most productive Turing Machine with $2k$ states or fewer. How big must k be in order for your code for M^I to have $2k$ states or fewer (and therefore be more productive—by one—than it could possibly be)?

(*Hint:* Your answer to problem (5e) should be of the form $k + C$ for some constant C. Your answer to the present question should be of the form $k \geq X$, where X is specified using the value you used for C in (5e).)

MIT OpenCourseWare
https://ocw.mit.edu/

24.118 Paradox and Infinity

Spring 2019

For information about citing these materials or our Terms of Use, visit: https://ocw.mit.edu/terms.