Problem Set 6: Dodge Duck Dip Dive Decide

Due: **4:19pm Friday, 1 November 2019**

This assignment focuses on the material in Chapter 6 and Chapter 8 of the textbook. In particular, we will use the definitions of Turing Machines covered in Chapter 6. From Chapter 8 we will be using the sections discussing uncomputability (Sections 8.2-8.5). This is the last problem set before Exam 2, which will be in class on Wednesday, 6 November.

Collaboration Policy: You should do this assignment by yourself and submit your own answers. You may discuss the problems with anyone you want and it is also fine to get help from anyone on problems with LaTeX or debugging your Turing machine code. You are permitted to use any resources you find for this assignment. You should note in the *Collaborators and Resources* box below the people you collaborated with and any external resources you used.

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

This problem set does not include any Jupyter notebook components. As with previous problem sets, the first thing you should do in ps6.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{Alonzo Church (a0c03z)}`. Before submitting your PDF, also remember to (1) list your collaborators and resources, replacing the TODO in `\collaborators{TODO: replace ...}`, and (2) replace the second line in ps6.tex, `\usepackage{uvatoc}` with `\usepackage[response]{uvatoc}` so the directions do not appear in your final PDF.

Getting Busy

These are the two problems that were deferred from Problem Set 5.

Problem 1 *Prove that the Busy Beaver Problem (defined in Class 14 and below) is uncomputable.*

Define the the Busy Beaver Problem as:

Definition 1 (Busy Beaver Problem) *For any $n \in \mathbb{N}$, define $BB_2(n)$ as the maximum number of steps for which a Turing Machine with n states and 2 symbols can execute and halt, starting from a blank tape.*

Your proof could show that assuming a Turing Machine that can solve the Busy Beaver problem (that is, output the correct value of $BB_2(n)$ for any input $n \in \mathbb{N}$, leads to a contradiction (that it, it would allow for a TM that can compute some problem we know is undecidable).

Problem 2 *Prove that the Busy Boa Problem (defined below) is uncomputable.*

Define the the Busy Boa Problem as:

Definition 2 (Busy Boa Problem) *For any $n \in \mathbb{N}$, define $BOA(n)$ as the largest integer that a idealized Python function implemented using at most n characters, and which takes no input, can return (and halt).*

The *idealized Python* language is the Python language you are familiar with, but without any arbitrary limits. So, for example, it provides a + operation that works on all natural numbers, unlike any actual Python implementation which can only implement + for a subset of $\mathbb{N}$.

Computable Languages

These definitions formalize notions we used in Class 15:

Definition 3 (Language of a Turing Machine) *We say that the language of a Turing Machine M, denoted $\mathcal{L}(M)$, is the set of all input strings which M accepts (i.e. halts and returns 1).*

Definition 4 (Computable Language) *We say that a language $\mathcal{L}_c$ is computable is there is some always-halting Turing Machine M such that $\mathcal{L}_c = \mathcal{L}(M)$.*

Problem 3 *Computability is closed under Union*

Show that if the language L_1 is computable (i.e. there is some Turing Machine M_1 such that $L_1 = \mathcal{L}(M_1)$), and language L_2 is computable (i.e. there is some Turing Machine M_2 such that $L_2 = \mathcal{L}(M_2)$), then the language $L_1 \cup L_2$ is also computable.

Computability Step-by-Step

Problem 4 *Halts in k steps*

Consider the Language:

$$H_k = \{M | M \text{ is the description of a Turing Machine which halts in } k \text{ or fewer steps}\}$$

Show that H_k is computable for every choice of $k \in \mathbb{N}$.

Problem 5 *Prove that H^* (defined below) is not computable.*

Define the language $H^* = \bigcup_{k \in \mathbb{N}} H_k$.

Problem 6 *The "proof" below seems to prove that H^* is computable. Identify and explain a flaw in the proof.*

Consider the following bogus "proof" by induction that H^* is computable:

Our inductive hypothesis is:

$$P(n) := H_0 \cup H_1 \cup \ldots \cup H_n \text{ is computable.}$$

Base case: H_0 is computable.

Proof: We showed (in your correct answer to Problem 4, which we will assume exists) that $\forall k \in \mathbb{N}$, H_k is computable.

Inductive case: We show $P(n) \implies P(n+1)$. That is, if $H_0 \cup H_1 \cup \ldots \cup H_n$ is computable, so is $H_0 \cup H_1 \cup \ldots \cup H_n \cup H_{n+1}$.

Proof: By the inductive hypothesis, $H_0 \cup H_1 \cup \ldots \cup H_n$ is computable. By Problem 4, H_{n+1} is computable. By Problem 3, the union of two computable languages is computable. Thus, it must be that $H_0 \cup H_1 \cup \ldots \cup H_n \cup H_{n+1}$ is computable as well. So, but induction, we have shown that $P(n)$ holds for all $\mathbb{N}$, and H^* is computable.

Decidability

Problem 7 *Rice's Theorem*

For each subproblem, indicate whether Rice's Theorem applies. If it applies, answer if the problem is computable or uncomputable. If it does not apply, just indicated that it doesn't apply (it is not necessary to determine whether or not it is computable if Rice's theorem does not apply).

(a) Given the description of a Turing Machine, does that machine always return 0.

(b) Given the description of a Turing Machine, does that machine always return 1 when it receives no input.

(c) Given the description of a Turing Machine, does that machine ever use more than 10,000 cells on its tape.

(d) Given the description of a Turing Machine, is the language of that machine recognizable.

(e) Given the description of a Turing Machine, does the Turing machine have exactly 50 states.

Problem 8 *Maximum Recursion Depth Exceeded is Uncomputable*

Python has a default recursion depth of 1000 calls. This means that if ever we have a function which calls itself over 1000 times (nested), Python will throw an error. Show that, in general, the problem of determining whether a given Python program will exceed the maximum recursion depth a given input is not computable. (Note for this problem, you can assume an "idealized" version of Python with no other implementation limits, other than the setting for recursion depth.) In other words, show that the language of Python, input pairs such that the Python program has no more than 1000 nested recursive calls is not computable.

For example, calling the `rec(x)` function dedinfed below with input `x = 500` would terminate happily, but on input `x = 1001` would result in a recursion depth exceeded error.

```
def rec(x):
    if x > 0:
        return rec(x-1)
    else:
        return "Done"
```

Problem 9 ($\star\star$) *Compressibility*

Show that the problem of determining whether a $s \in \{0,1\}^*$ is *compressible* (as defined below) is not computable.

Definition 5 (Compressibility) We can represent a string $s \in \{0,1\}^*$ as the pair (w, x) where w is a Turing machine description and x is a string, such that running $\mathcal{M}(w)$ on input x halts and leaves the string s on its tape. ($\mathcal{M}(w)$ is the machine represented by w, as used in Class 15.)

A string s is *compressible* if we can represent s as (w, x) where $|w|+|x| < |s|$ (i.e., the Turing Machine-input representation of s is shorter that s itself).

End of Problem Set 6