Problem Set 5: Loopy Tunes

Due: **4:19pm Friday, 25 October 2019**

This assignment focuses on the material in Chapter 6 of the textbook, as well as finite state machines from Class 12 and Class 13. The textbook covers finite automata in Chapter 9, and there are many other sources for this material, but you should be able to understand enough from what we have done in class to answer the questions on this assignment without needing to consult other material.

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, but does include some problems using an on-line Turing Machine simulator. As with previous problem sets, the first thing you should do in p54.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{Alan Turing (amt01z)}`. 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 ps4.tex, `\usepackage{uvatoc}` with `\usepackage[response]{uvatoc}` so the directions do not appear in your final PDF.

Size Hierarchy

Problem 1 *Understanding the Size Hierarchy Proof*

The proof of the Size Hierarchy Theorem (Theorem 5.5 in the book, and Class 12) defined a sequence of functions, $f_0, f_1, \ldots$:

$$f_i(x) = \begin{cases} f^*(x) & lex(x) < i \\ 0 & \text{otherwise} \end{cases}$$

where f^* is some hard function, which we don't need to define but know must exist for sufficiently large n because of the number of functions in $SIZE(s)$.

(a) Prove that when $f^*(i) = 0$, $f_i = f_{i-1}$. That is, the two functions denoted by f_{i-1} and f_i are actually the same function.

(b) Explain why this is not a problem for the proof.

Problem 2 *Finer Hierarchy*

The Size Hierarchy Theorem says that for every sufficiently large n and $10n < s < 0.1 \cdot 2^n/n$, $SIZE_n(s) \subsetneq SIZE_n(s+10n)$ (where $SIZE_n(s)$ was the set of all n-input boolean functions that can be implemented with s or fewer $NAND$ gates). The need for the $10n$, means this does not tell us if, for a given s, $SIZE_n(s) \subsetneq SIZE_n(s+1)$. As computer scientists, we should be a bit bothered by this. (Make sure you understand what the proof method used to show the theorem resulted in this $10n$ term.)

(a) Prove that $SIZE_2(1) \subsetneq SIZE_2(2)$.

(b) Prove that for some $1 < s < 10$, $SIZE_2(s) = SIZE_2(s+1)$.

Programming Turing Machines

For this problem, you will use the Turing Machine simulator available at http://morphett.info/turing/turing.html. Before trying the problem, play around with the examples that are provided there (if you have a lot of time on your hands, try the 5-state Busy Beaver here: http://morphett.info/turing/turing.html?60b2d45a91bff93bead8e9fa13aadd9a) to get a feel for how the simulator works and what Turing Machine programs are like.

Problem 3 *Warm up: Implementing XOR*

Write a Turing Machine program that implements the infinite XOR machine from Class 12. Note that we implemented this in class using a Finite Automaton, so you can solve this problem by converting that to a Turing Machine, where the input is given on the initial tape. Your machine should end in the "halt-accept" state if the input has an odd number of `1` symbols, and end in the "halt" state if the input has an even number of `1` symbols. You can submit your answer by just cut-and-pasting the code from your machine into the verbatim block below.

```
; State 0:
0 0 0 r 0  ; you don't need to use this, but it is meant to be a useful hint
;; complete with your code

0 _ _ * halt ; reject if you end in state 0

;; complete with your code
```

Problem 4 *IsRepeat*

Define a Turing Machine that implements the function $IsRepeat$ defined for any input $x \in \{0,1\}^*$ by:

$$IsRepeat(x) = \begin{cases} 1 & \text{if there exists a } w \in \{0,1\}^* \text{ such that } x = ww \\ 0 & \text{otherwise} \end{cases}$$

For example, $IsRepeat(\texttt{0}) = 0$, $IsRepeat(\texttt{00}) = 1$, $IsRepeat(\texttt{011011}) = 1$, and $IsRepeat(\texttt{0111011}) = 0$. The machine your define should end in state "halt" if the output of $IsRepeat$ is 0, and in state "halt-accept" if the output of $IsRepeat$ is 1.

Problem 5 *Limits of Finite State Automata*

For each subproblem below, indicate if there is a Finite State Automaton that can compute the given function. You do not need to provide a formal proof, but should provide a short justification for your answer.

For all problems, you can assume there is a way to encode the input for the Turing Machine. For example, if there are multiple inputs, they would be separated by a special symbol # on the input tape to distinguish them.

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

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

(b) $IsRepeat$ (as defined in the previous problem)

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

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

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

$$WayCool(w,x) = \begin{cases} 1 & \text{if } x \text{ contains at least } val(w) \text{ symbols,} \\ 0 & \text{otherwise} \end{cases}$$

where $val(w)$ is the value of w interpreted as a binary number.

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

$$WahooWa(x) = \begin{cases} 1 & \text{if UVA will beat Virginia Tech in both cybersecurity and football in 2019,} \\ 0 & \text{otherwise} \end{cases}$$

Power of NAND-TM

The NAND-TM language is defined by adding both loops and arrays to NAND-CIRC. For the next two problems, we consider the impact of adding either just loops or just arrays to NAND-CIRC.

Problem 6 *Is NAND-LOOP more powerful than NAND-TM?*

Define the language NAND-LOOP as NAND-CIRC plus the MODANDJUMP instruction. Are there any functions that NAND-LOOP can compute that cannot be computed by NAND-TM?

(A good answer will either provide a proof that the languages are equivalent in power, or show that there exists a function which can be computed by NAND-LOOP but not by NAND-CIRC.)

Problem 7 *Is NAND-ARRAY more powerful than NAND-TM?*

Define the language NAND-ARRAY as NAND-CIRC plus the special integer variable `i` and array variables (that is, with everything in the first four bullets of the description of NAND-TM in Section 6.2.1, but without MODANDJUMP.

Are there any functions that NAND-ARRAY can compute that cannot be computed by NAND-TM?

Problem 8 *Define a language that extends NAND-CIRC, which is more powerful (in an interesting way) than NAND-CIRC, but less powerful (in an interesting way) than NAND-TM.*

One way to interpret "in an interesting way" is that your language should be able to implement an infinite set of functions that cannot be implemented by NAND-CIRC, but there should be an infinite set of functions that NAND-TM can implement which cannot be implemented in your language.

Problem 9 *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 10 *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 $\mathbb{N}$.

End of Problem Set 5

