Exercise Set 6: Heavy Machinery

The first thing you should do in exercise6.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 email id, e.g., `\submitter{Grace Hopper (gmh1a)}`.

Before submitting , 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 exercise6.tex, `\usepackage{uvatoc}` with `\usepackage[response1]{uvatoc}`, `\usepackage[response2]{uvatoc}`, etc. for the appropriate problem submission.

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

Computable Languages

Definition 1 (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 2 (Computable Language) *We say that a language $\mathcal{L}_c$ is computable is there is some always-halting Turing Machine M such that $\mathcal{L}_c = \mathcal{L}(M)$.*

Exercise 6-1: Computability is closed under Union

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

Exercise 6-2: Halts in k steps

Consider the Language:

$H_k = \{M | M$ is the description of a Turing Machine which halts in k or fewer steps when it receives no input$\}$

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

Definition 3 (H_*) *Define the language* $H_* = \bigcup_{k \in \mathbb{N}} H_k$.

Exercise 6-3: A flawed proof

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:

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

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 $H_0 \cup H_1 \cup ... \cup H_n$ is computable, so is $H_0 \cup H_1 \cup ... \cup H_n \cup H_{n+1}$.

Proof: By the inductive hypothesis, $H_0 \cup H_1 \cup ... \cup H_n$ 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 $H_0 \cup H_1 \cup ... \cup H_n \cup H_{n+1}$ is computable as well. So, but induction, we have shown that $P(n)$ holds for all $\mathbb{N}$, and H_* is computable.

Exercise 6-4: Maximum Recursion Depth

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 when running on a given input is not computable.

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 defined 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"
```

Limits of FSA

Exercise 6-5: What can a FSA do?

For each subproblem below, indicate if there is a Finite State Automaton (NFA or DFA) 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 $x \in \{\texttt{0},\texttt{1}\}^*$,

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

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

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

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

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

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

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

$$Wahoo\,Wa(x) = \begin{cases} 1 & \text{if UVA will beat Virginia Tech in both cybersecurity and football in 2020,} \\ 0 & \text{otherwise} \end{cases}$$

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

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

For example, $IsRepeat(\texttt{0}) = 0$, $IsRepeat(\texttt{00}) = 1$, $IsRepeat(\texttt{011011}) = 1$, and $IsRepeat(\texttt{0111011}) = 0$.