In-Class Problems Week 3, Fri.

Problem 1.
Let's refer to a programming procedure (written in your favorite programming language —C++, or Java, or Python, ...) as a *string procedure* when it is applicable to data of type `string` and only returns values of type `boolean`. When a string procedure, P, applied to a `string`, s, returns **True**, we'll say that P *recognizes* s. If $\mathcal{R}$ is the set of strings that P recognizes, we'll call P a *recognizer* for $\mathcal{R}$.

(a) Describe how a recognizer would work for the set of strings containing only lower case Roman letter —`a,b,...,z`—such that each letter occurs twice in a row. For example, `aaccaabbzz`, is such a string, but `abb`, `00bb`, `AAbb`, and `a` are not. (Even better, actually write a recognizer procedure in your favorite programming language).

A set of `string`s is called *recognizable* if there is a recognizer procedure for it.

When you actually program a procedure, you have to type the program text into a computer system. This means that every procedure is described by some `string` of typed characters. If a `string`, s, is actually the typed description of some string procedure, let's refer to that procedure as P_s. You can think of P_s as the result of compiling s.[1]

In fact, it will be helpful to associate every string, s, with a procedure, P_s; we can do this by defining P_s to be some fixed string procedure —it doesn't matter which one —whenever s is not the typed description of an actual procedure that can be applied to `string` s. The result of this is that we have now defined a total function, f, mapping every `string`, s, to the set, $f(s)$, of `string`s recognized by P_s. That is we have a total function,

$$f : \texttt{string} \to \mathcal{P}(\texttt{string}). \tag{1}$$

(b) Explain why the actual range of f is the set of all recognizable sets of strings.

This is exactly the set up we need to apply the reasoning behind Russell's Paradox to define a set that is not in the range of f, that is, a set of strings, $\mathcal{N}$, that is *not* recognizable.

(c) Let

$$\mathcal{N} ::= \{s \in \texttt{string} \mid s \notin f(s)\}.$$

Prove that $\mathcal{N}$ is not recognizable.

Hint: Similar to Russell's paradox or the proof of Theorem **??**.

[1]The string, s, and the procedure, P_s, have to be distinguished to avoid a type error: you can't apply a string to string. For example, let s be the string that you wrote as your program to answer part (a). Applying s to a string argument, say `oorrmm`, should throw a type exception; what you need to do is apply the procedure P_s to `oorrmm`. This should result in a returned value **True**, since `oorrmm` consists of three pairs of lowercase roman letters

(d) Discuss what the conclusion of part (c) implies about the possibility of writing "program analyzers" that take programs as inputs and analyze their behavior.

Problem 2.
The Axiom of Choice can say that if s is a set whose members are nonempty sets that are *pairwise disjoint* —that is no two sets in s have an element in common —then there is a set, c, consisting of exactly one element from each set in s.

In formal logic, we could describe s with the formula,

$$\text{pairwise-disjoint}(s) ::= \quad \forall x \in s.\, x \neq \emptyset QAND \forall x, y \in s.(x \neq y) \text{ IMPLIES } (x \cap y = \emptyset).$$

Similarly we could describe c with the formula

$$\text{choice-set}(c, s) ::= \quad \forall x \in s.\, \exists! z.\ z \in c \cap x.$$

Here "$\exists! z.$" is fairly standard notation for "there exists a *unique* z.

Now we can give the formal definition:

Definition (Axiom of Choice)**.**

$$\forall s.\, \text{pairwise-disjoint}(s) \text{ IMPLIES } \exists c.\, \text{choice-set}(c, s).$$

The only issue here is that Set Theory is technically supposed to be expressed in terms of *pure* formulas in the language of sets, which means formula that uses only the membership relation, $\in$, propositional connectives, and the two quantifies $\forall$ and $\exists$. Verify that the Axiom of Choice can be expressed as a pure formula, by explaining how to replace all impure subformulas above with equivalent pure formulas.

For example, the formula $x = y$ could be replaced with the pure formula $\forall z.\ z \in x$ IFF $z \in y$.

Problem 3.
There are lots of different sizes of infinite sets. For example, starting with the infinite set, $\mathbb{N}$, of nonnegative integers, we can build the infinite sequence of sets

$$\mathbb{N},\ \mathcal{P}(\mathbb{N}),\ \mathcal{P}(\mathcal{P}(\mathbb{N})),\ \mathcal{P}(\mathcal{P}(\mathcal{P}(\mathbb{N}))),\ \ldots.$$

By Theorem **??** from the Notes, each of these sets is *strictly bigger*[2] than all the preceding ones. But that's not all: if we let U be the union of the sequence of sets above, then U is strictly bigger than every set in the sequence! Prove this:

Lemma. *Let $\mathcal{P}^n(\mathbb{N})$ be the nth set in the sequence, and*

$$U ::= \bigcup_{n=0}^{\infty} \mathcal{P}^n(\mathbb{N}).$$

Then

[2] Reminder: set A is *strictly bigger* than set B just means that A surj B, but NOT(B surj A).

1. U surj $\mathcal{P}^n(\mathbb{N})$ *for every* $n \in \mathbb{N}$, *but*
2. *there is no* $n \in \mathbb{N}$ *for which* $\mathcal{P}^n(\mathbb{N})$ surj U.

Now of course, we could take $U, \mathcal{P}(U), \mathcal{P}(\mathcal{P}(U)), \ldots$ and can keep on indefinitely building still bigger infinities.