Problem Set #1

1 Theory Questions

Subjects: R, RE.

1. Infinity country was founded 100 years ago, and will exist forever. A simple lottery game takes place in this country once a week. Each participant chooses a natural number between 1 and 10^{10}, and writes it on a card. In the lottery day, a natural number between 1 and 10^{10} is randomly chosen - and this is the winning number for this week! The big prize is divided between all the participants that chose this exact number.

   Define the language L to be the set of all numbers that have won the big prize in the Infinity country lottery game, or will win at some time in the future.
   Is L decidable? Explain your answer.

2. Recall that we showed some closure properties of R in the recitation. In this question we will deal with a few others.

   (a) Prove that if $L_1, L_2 \in R$ then $L_1 \cap L_2 \in R$, and conclude that R is closed under finite intersection.

   (b) Is R closed under infinite unions? Under infinite intersections? Prove your answer.

   (c) Given languages L_1, L_2, the *concatenation of L_1 and L_2* is noted by $L_1 \circ L_2$ and defined by:

   $$L_1 \circ L_2 = \{x \circ y \mid x \in L_1 \wedge y \in L_2\}$$

   where $x \circ y$ denotes string concatenation.
   Prove that if $L_1, L_2 \in R$ then $L_1 \circ L_2 \in R$.

3. Recall that up until now we saw one example of a recursively enumerable language that is not recursive: the language L_{Acc} (proven in class). In this question we will introduce another example of such language: the language L_{Halt} (defined below).

   We say that a program *halts* if its execution terminates after a finite number of steps.
   Let

   $$L_{\text{Halt}} = \{(P, w) \mid P \textit{ is a program, } w \textit{ is a string, and } P \textit{ halts when executed with input } w\}$$

   We will prove that $L_{Halt} \in RE \setminus R$. Follow the steps below:

   (a) Prove that $L_{Halt} \in RE$ by providing a recognizer for it.

   (b) Before turning to prove that $L_{Halt} \notin R$, we will first introduce a simple program that would be useful during the proof. Look at the program **loop_if_true** defined by:

   $$loop\_if\_true := [\lambda x.\, if\ (x == True)\ then\, loop\_if\_true(x)\, else\, 17]$$

   Assume that the possible inputs for **loop_if_true** are the truth values **True** and **False**. Describe what happens when it executed on each of those values.

   (c) Now prove that $L_{Halt} \notin R$: Assume the contrary (for contradiction). Given a decider d for L_{Halt}, define a program A by:

   $$A := [\lambda w.\, loop\_if\_true(d(w, w))]$$

   Ask whether or not A halts when executed with input A, and show that in both cases we get a contradiction. Conclude that the assumption $L_{Halt} \in R$ must be wrong.

4. We say that a list of strings is sorted in a *shortlex order* if the strings are ordered by length (from the shortest one to the longest one), and strings with the same length are ordered by a lexical order.

   For example, the following list of colors names is sorted in a shortlex order: **red**, **blue**, **pink**, **black**, **green**, **white**, **orange**, **yellow**.

   Prove that if a language L has an enumerator that prints the words of L in a shortlex order, then L is decidable.
   (Hint: deal separately with the case that L is finite and with the case that L is infinite.)

2 Python programming

Subjects: One liners.

We got an anonymous complaint that the exercises in your courses tend to be too long. So, to start off, and **for this exercise only**, you must write only one-line functions! That means:

- Lines length may be as long as you want.
- The body of the function (namely all the code that appears after the **def** line) must contain exactly one line.
- You may use as many external modules (imports) as you like.
- You may thoroughly search the Python documentation.
- You may use the **dir()** and **help()** functions.
- For example, the following code is a one-liner function that prints the string: "Hello Algorithms for Computational Linguistics!":

```
def welcome():
    print('Hello Algorithms for Computational Linguistics!')

>>> welcome()
'Hello Algorithms for Computational Linguistics!'
```

triple

Write the one-line function **triple** that receives a string, and returns another string where each character in the original string is tripled.

For example:

```
>>> triple('hello')
'hhheeellllllooo'
```

cleaning right

Oh no! Someone spilled coffee over our strings, and now we have strings that end with randomly ordered 'c', 'o', 'f', 'e' letters on the right. That is, "hello world" suddenly became "hello worldfofoeccc".

Write the one-line function **cleaning_right** that receives a list of strings, and cleans this mess up! That is, it returns a new list with the original strings – after cleaning their right edges.

Note: in case the original string had ended with one of the 'c', 'o', 'f', 'e' characters before the coffee was spilled – you can treat it as mess and clean it up.

For example:

```
>>> cleaning_right(['hell', 'hello', 'hello worldcofofcfee'])
['hell', 'hell', 'hello world']
```