
Complete this checkpoint **on your own, and email your solution to ndobson@andrew.cmu.edu by 11:59 PM on March 29th** with "`[FLAC checkpoint 7]`" in the subject.

Your work should be legible and preferably typed, but you don't have to use TeX.

1 Quiniforous Quines

Say we had two different encodings of computer programs which are equivalent (e.g. turing machines and pushup automata). The first encoding is the standard definition as given in lecture. Assume the recursion theorem for each encoding on its own. Assume that there is a computable function from each encoding to the other that maintains behavior (i.e. we can compile each encoding into the other). Prove that there exists programs M in one encoding and N in the other such that M prints $\langle N \rangle$ on the empty input and N prints $\langle M \rangle$ on the empty input.

Explain how this solves the problem:
Let $\langle M \rangle_1$ be the first encoding of machine M. Let $\langle M \rangle_2$ be the second encoding of machine M.

Consider the following machine:
def $M(w)$:
    let $s = \langle M \rangle_1$
    def $N(w')$:
        print s
    print $\langle N \rangle_2$

2 Don't overthink it

1. Prove that some language is not in NP.
2. Is the empty language (not $E_{\texttt{TM}}$, but the empty set) in NP?

3 This is the best name for a problem.

What would be the problem with defining Kolmogorov complexity in terms of English? In other words, if the Kolmogorov complexity of a string were the length of the shortest English phrase that describes that word. Hint: the problem is more fundamental than spelling or grammatical ambiguity.

4 König's lament

Recall the following definitions with regard to nondeterministic turing machines (NTMs):

1. A nondeterministic decider is an NTM such that on every input, the computation history contains no infinite paths.
2. For deciders, the number of steps needed is the length of the longest path in the computation history.
3. For non-deciders, the number of steps needed is not defined.

Determine if the following nondeterministic programs would be deciders, and if so, state their asymptotic complexity (e.g. constant, polynomial, exponential, etc.).

```
def fuel(s):
    in one branch:
        accept
    in the other branch:
        loop

def melt(s):
    in one branch:
        accept
    in the other branch:
        wait |s| steps
        accept

def beams(s):
    let w = 0^|s|
    for i in range(|s|):
        nondeterministically do one of the following:
            1. w[i] = 0
            2. w[i] = 1
    if w == s :
        accept
    else :
        reject
```