Correction Challenge Activity 2 – CFGs and PDAs

1. The language L1 = {w#w | w is in {0, 1}*} is not a context-free language (CFL). (0 points)

2. Justify, using what CFGs are capable of, why L1 is not a CFL; (1 point)

   Solution: Context free grammars are not capable of storing, and in this case we would need to keep in memory the first string w, so that the grammar could latter, after consuming #, recognize it. Also, due to the structure of productions in a context-free grammar, there isn't a way to specify the repetition of a string in the same order (in the reverse order is possible, e.g. recognizing a palindrome). Given these limitations, there is no CFG that can recognize L1, and therefore L1 is not a context free language.

3. Justify, using what PDAs are capable of, why L1 is not a CFL; (1 point)

   Solution: Pushdown automata are capable of storing a variable amount of symbols in a stack, and therefore could store the first string w so that it could be recognized after. However, a stack stores memory in a LIFO (last in first out) manner, and a pushdown automata has only access to the element in the top of the stack. Therefore, when storing the first string w, the top element of the stack will be the last symbol of the string. This makes it impossible to compare both strings w, since they would be stored reversed relative to one another. When encountering the first symbol of the second string w, the stack would have to discard all the elements of the stack to reach the bottom, where the first symbol of the first string is stored. But then, the rest of the string would be lost. Given these limitations, there is no PDA that can recognize L1, and therefore L1 is not a context free language.

4. As you know, the complement of a CFL may not be a CFL. Show that the complement of L1 is a CFL by presenting a possible CFG for the complement of L1; (7 points)

   Solution: The following CFG recognizes the complement of L1:

   S → Z1T | I0T | TKE | EKT | C
   Z → KZK | 0T#
   I → KIK | 1T#
   E → KEK | #
   C → KC | #F | ε

$F \rightarrow KF \mid \#B$

$K \rightarrow 0 \mid 1$

$T \rightarrow KT \mid \varepsilon$

$A \rightarrow K \mid \#$

$B \rightarrow AB \mid \varepsilon$

The two first productions of S, Z1T and I0T, ensure a string of the form w#x will have at least one symbol in w which is different from the analogous symbol (same position counting from the left) in x. This leaves out strings of the form w#w.

The third and fourth productions of S, TKE and EKT, ensure that in a string of the form w#x, the size of w will be different from the size of x ($|w| \neq |x|$). Another alternative to the grammar would be removing the productions TKE and EKT from S and adding L (left side of # is longer) and R (right side of # is longer):

$S \rightarrow \ldots \mid L \mid R \mid \ldots$

$L \rightarrow TK\# \mid KLK$

$R \rightarrow \#KT \mid KRK$

The last production of S, C, ensures all strings with 0, 2 or more # are accepted. Another alternative to the grammar would be to replace the productions of C, and deleting the productions of F, with:

$C \rightarrow B\#B\#B \mid KT$

5. Instead of a stack such as in the PDAs, one can think about other FA extensions. Select the kind of structures that could be used to extend an FA and make it able to implement L1: (1 point)
   1. queue
   2. 2 stacks
   3. none of the other options;

Solution: The correct options are **1 and 2**. Adding a queue to a PDA (option 1) would allow it to store the string in the correct order. Adding two stacks to a PDA (option 2) would allow one string to be stored in one stack, and the other string in the other stack, allowing then to compare both. Also, a PDA with two stacks is equivalent to a Turing machine in processing power, so it would be able to recognize L1.