CSC 520 Fall 2023
Final Exam
175 points/10 extra credit points

Submit a .zip archive that contains these files:

- A plain text .txt file that affirms the honor code: **`On my honor as an SFSU student, I, `*`name`*`, have neither given or received inappropriate help with this final exam.`**
- A .jff file containing your solution for problem 1. (You can also email this file to mpico@sfsu.edu)
- A plain text .txt file containing your RL pumping lemma proof for problem 2.
- A .py Python source code file containing your solution for either problem 3.a or 3.b, but not both.

1. (60 points/10 extra credit points for the minimal correct solution. Submit your solution as a .jff file) The input to TM M is $n_1\$n_2$, where n_1 and n_2 are strings of *1's* and *0's*. If $n_1 = n_2$, the output is $n_1{=}n_2$; when n_1 and n_2 are interpreted as the binary numbers n_1 and n_2 respectively, with leading zeroes ignored, the output is $n_1{*}n_2$ if $n_2 = 2*n_1 + 1$, and $n_1\#n_2$ if n_1 and n_2 differ in any other way. A correct implementation of M will have the transducer results below for the inputs in test_cases.txt, included with the test materials.

Input	Output	Result
101$0101	101=0101	Accept
0101$101	0101=101	Accept
101$01011	101*01011	Accept
0101$01011	101*01011	Accept
101$11	101#11	Accept
01$10	01#10	Accept
01$011	01*011	Accept

The file final.jff, also included with the test materials, encodes a partial implementation of M. It handles $n_1 = n_2$ correctly, but tries to compute the < and > relations instead of the * and # relations defined above. Enhance final.jff to implement M. To run the test cases in transducer mode, select Multiple Run (Transducer) from the Input menu, and then select Load Inputs from the buttons at the bottom of the right panel.

2. (55 points. Submit your solution as a .txt plain text file). $\Sigma = \{C, A, G, T\}$, $L = \{ w : w = C^cA^{a1}GA^{a2}GA^{a3}T^t,$ $c \% 2 = 0$; $a_1, a_2, a_3 > 0$; $(a_1 + a_2) > a_3$; and $t = (a_1 + a_2) - a_3 \}$. Note that to be in L, a string must have 3 A regions in the middle, separated by single *G's*; start with an even number of *C's*, where 0 counts as an even number; and end with at least one T. For example, $C^4A^5GA^3GA^7T \in L$; $A^5GA^3GA^7T$ is also in L (it has 0 C's at the start of the string, and 0 is an even number); $CCA^6GA^3GA^7T^3 \notin L$ because there should be 2 *T's* instead of 3; and $C^5A^5GA^3GA^7T \notin L$ because it does not start with an even number of *C's*.

Prove that L $\notin$ RLs using the regular language pumping lemma. Start by defining a string *S* such that <u>*S* ∈ L and |*S*| ≥ N, N ≥ 1</u>. Please <u>do not use any of the example strings</u> above in your proof, <u>or define *x, y,* and *z* substrings</u> in the style found in the 520 archives. Note that for *S* to be at least N symbols long, it <u>must</u> use N, and/or an expression derived from N, as a repetition operator. For example, $A^5GA^3GA^7T$ could not work as an *S* although it is in L, because it is not defined in terms of N — thus it is meaningless to ask if it is at least as long as N. And $C^cA^NGA^{N-1}GA^{N-2}T^t$ could not work as an *S* because c and t are undefined outside the characteristic function for L.

Other points to bear in mind:

- The only valid assumption about N is that is at least 1. For example, consider the language $L = \{ w : w = G^jT^k, j \% 2 = 1, j < k\}$. $S = G^NT^{N+1}$ would not work, because you cannot assume that N is odd.
- After defining *S*, the next step is to show that for <u>all</u> non-null substrings *R* within the first symbols of *S*, there is <u>some</u> way to pump (remove or replicate) *R* to generate *S'* $\notin$ L. Just saying that this could be done is insufficient.
- Unless you divide all possible R's into a finite number of categories, it is impossible to prove that all R's can be pumped. Within each category, removing or replicating R must yield S' $\notin$ L. For example, suppose your R was contained in a region containing N C's. Then your pumping must yield S $\notin$ L regardless of whether |R| is odd or even.

3. (**60 points. Submit your answer to either 3.a or 3.b (next page), but not both, as a .py Python source code file.**)

3.a A positive instance of the decision procedure Mod2InMod3Out(*P*) $\overset{\text{def}}{=}$ $\forall_I(P(I) = I'$, such that $|I'| \% 3 = (|I| \% 2) + 1$. That is, the length of P's output mod 3 equals the length of its input mod 2, plus 1. Consider these examples:

```
def f1(inString): return inString
def f2(inString):
    if len(inString) % 2 = 0 return 'x'
```

f_1 is a negative instance of Mod2InMod3Out, because if `instring == 'xx', len('xx') % 2 + 1 == 1` **!=** `len('xx') % 3 == 2`. f_2 is also a negative instance of Mod2InMod3Out although it handles even length input correctly, because if `instring == 'xxx', len('xx') % 2 + 1 == 2` **!=** `len('xxx') % 3 == 0`.

Show that Mod2InMod3Out(P) is undecidable by modifying yesOnStringViaMod-template.py (included with the test materials) to implement YesOnString(*P*,*I*) $\le_T$ Mod2InMod3Out(P). In the multi-line comment at the end of the file, argue that you have shown that Mod2InMod3Out(*P*) is undecidable.

3.b Prove that HalfIndependentSet is NP-hard by polyreduction from HasNodeCover, where HalfIndependentSet is the problem defined for Midterm #2 (in test materials). Complete the template file ConvertNodeCoverToHalfIndependentSet-template.py (in test materials), adding a multi-line comment at the end arguing that your code is a valid proof. You are welcome to call ConvertCliqueToHalfIndependentSet (in test materials), and any of the function defined above the main conversion routine, but you are not required to do so. It is possible to make the needed code changes in as few as a couple of lines. The test harness works in the usual way.

The WCBC lecture slides on Karp's problems are included as a reference. The slides most relevant to this polyreduction are #9, #10, and #11.