Exam 2 practice. Each question is graded as 0, 1/3, 2/3, 3/3.

1. Give a DFA, NFA, RE for a regular language.

 L = {strings of a's and b's that contain at least one a, at least one b, and does not contain either aaa or bbb}

 Strings not in L: aaa, bbb, ababbba, abbbaabaaab

See exam2\_practice\_1.jff (open with JFLAP, see video solutions from <https://cs.indstate.edu/wiki/index.php/Theory_of_Computation> for information about JFLAP)

2. Give a CFG for a CFL language.

 L = {strings of (, ), [, ], {, } such that it is properly parenthesised}

 Strings not in L: ([)], (, {], (()))

S -> S S | (S) | [S] | {S} | epsilon

3. Give a PDA for a CFL language.

 L = {w w^R | w is a string of a's and b's that starts with a and ends with b}

 Strings in L: aabbaa

 Strings not in L: abaaba, bbabbabb, abb, baa, abbabb

See exam2\_practice\_3.jff (open with JFLAP, see video solutions from <https://cs.indstate.edu/wiki/index.php/Theory_of_Computation> for information about JFLAP)

4. Prove that a non-CFL language is not CF.

 L = {w wR w | w is a string of a's and b's, and w^R is w in reverse}

Suppose L is CF, and let p be the pumping length from the pumping lemma for CFL’s. Let s = apbpbpapapbp. Note that s is in L and |s| >= p. Then the pumping lemma for CFL’s guarantees that s can be written as s = uvxyz where |vxy| <= p, |vy| >= 1, and uvixyiz is in L for all i >= 1. We consider the cases for what part of the string vxy is within.

Case 1: vxy is completely contained within the a’s in the first third of s. In this case, uvvxyyz will have more a’s in the first portion than in the second two parts, and it is not of the form wwRw.

Case 2: vxy is completely contained within the b’s in the first third of x. Similar to case 1, there are more b’s in the first part than the second two parts, and it is not in L.

Case 3: vxy is between the a’s and b’s in the first part with v in the a’s and y in the b’s. In this case there are more a’s and b’s in the first part than in the second two parts, and uvvxyyz is not in L.

Case 4: vxy is in between the a’s and b’s in the first part with either v or y crossing the boundary. In this case there will be an extra alternation between a’s and b’s in uvvxyyz, and this string is not in L.

Case 5: vxy is completely contained within the second third or last third of s. These are similar to the cases already considered.

Case 6: vxy crosses in between the first-second thirds of s or in between the second-last thirds of s. These are similar to cases 3 and 4.

In all cases, uvvxyyz is not of the form wwRw and is not in L. Therefore it could not have been that L is CF.

5. Given the following CFG -

 PAGE -> <html>CONTENT</html>

 CONTENT -> <p>CONTENT</p> |

 <b>CONTENT</b> |

 <h1>CONTENT</h1> |

 <h2>CONTENT</h1> |

 LIST | dummy-string

 LIST -> <ul>ITEMS</ul>

 ITEMS -> ITEM ITEMS | epsilon

 ITEM -> <li>CONTENT</li>

Note - variables in the grammar are PAGE, CONTENT, LIST, ITEMS, ITEM. epsilon is the empty string. Everything else is a terminal symbol.

Note – I added the rule CONTENT -> dummy-string after posting the practice exam, because without a rule like that there would be infinite recursion in generating strings (the grammar wouldn’t really be a valid grammar).

 Give a string that is a member of the language:

 <html><p>dummy-string</p></html>

 Give a string that is not a member of the language:

 dummy-string

 Is this language itself closed under star? That is, if a string w is in the language, are ww, www, wwww, etc. also in the language?

 This is not closed under \*. If the language were closed under star, then the string

<html>dummy-string</html><html>dummy-string</html>

would be in the language. But there is no way to generate that string.

6. Write out the transition table for the following PDA:

 https://www.tutorialspoint.com/automata\_theory/images/pda\_for\_l1.jpg

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| state | Input symbol | Stack symbol | Go to state | Put on stack |
| q1 | epsilon | epsilon | q2 | $ |
| q2 | a | epsilon | q2 | a |
| q2 | b | epsilon | q2 | b |
| q2 | epsilon | epsilon | q3 | epsilon |
| q3 | a | a | q3 | epsilon |
| q3 | b | b | q3 | epsilon |
| q3 | epsilon | $ | q4 | epsilon |

 Give a string that is accepted by the PDA:

 aabb

 Give a string that is not accepted by the PDA:

 ab

 What is a shortest string accepted by the PDA?

 Empty string. Next after that would be aa and bb

 If the set of accept states was swapped (so q1, q2, q3 are accept states) - what would the language be?

 In this case, the language would contain all strings – because the PDA could just stay in state q2 through the end of the input. Note that this is at least a little interesting – it is NOT the case that you get the complement of the language just be swapping what the accept states are (why? Because of non-determinism).

7. Give the proof idea for the following statement that is true.

 Statement: if a language L is regular, then it is CF as well.

One way to see this is to see how to convert a RE to a CFG. In particular, I describe how to handle each of the regular operations – star, union, concatenation. Suppose we have regular expressions X and Y and already have CFG’s for X and Y with start symbols SX and SY

Union – we can simulate the regular expression X | Y by using a new start symbol S and adding the rules S -> SX and S -> SY.

Concatenation – we can simulate the regular expression X Y by using a new start symbol S and adding the rule S -> X Y

Star – we can simulate the regular expression X\* by using a new start symbole and adding the rules S -> X, S-> S S, S -> epsilon

Since we can simulate the regular operations we can build a CFG to simulate any RE. To make this a formal proof we would use a proof by induction based on the number of operations in the RE. The base case would be a RE that is just a single terminal symbol. If there is a regular expression something like “a”, then we would use a CFG S -> a for that.

8. The following statement is not true in general for CF's. Give an example language where the statement is indeed true, and give another example where the statement is not true.

 Statement: any language that is CF has a PDA that only uses one sybmol for it's stack (so stack alphabet Gamma is just a single letter, say 'a').

 Language that has the property – L = {anbk | n >= 0 and k <= n}. Note that the PDA for this language only needs to keep track of how many a’s there are, and then pop these a’s off the stack as it reads through the b’s. This only requires one stack symbol.

 Language that does not have the property – L = {anbn | n >= 0}. The usual PDA for this language uses some symbol to keep track of how many a’s (push the symbol when reading a’s, pop the symbol when reading b’s). The PDA also needs some special symbol (e.g., $, or B) to use to mark the bottom of the stack so that we can make sure there are as many b’s as a’s (so we don’t leave the stack with a’s still there). I don’t think we can give a correct PDA without using the special symbol to mark the bottom of the stack.

9. Convert the following CFG into Chomsky normal form.

 S -> a U b T c | epsilon

 U -> c U | epsilon

 T -> T a | V

 V -> b V | b

# rules based on S’s rules

S -> A S1
S1 -> U S2
S2 -> B S3

S3 -> T C
S -> epsilon

B -> b
A -> a

C -> c

# rules based on rules for U, T, V

U -> C U
T -> T A
V -> B V
V -> b

# T-> V rule – add in “V” to any rule where there was T
S3 -> V C
T -> V A

# U -> epsilon rule – add in rule leaving U out anywhere there was a U
#S1 -> S2
#U -> C
# But those are not of the required form, so put in rule with S2 in place of S1 and C in place of U anywhere those rules showed up
S -> A S2
S1 -> C S2
U -> C C

I think that’s it. I could have missed something…