1. Write a CFG for the regular expression \((01 + 10)^*\). Provide justification why it is correct. (6)

\[
S \rightarrow SS \mid A \mid \epsilon \\
A \rightarrow 01 \mid 10
\]

The first rule will replicate \(S\) as many times as necessary and the second rule will substitute using either 01 or 10.

2. Let \(L = \{w \in (a + b)^* \mid w\ \text{contains even number of a's}\}\). Here even does not include the case 0. Given the grammar

\[
S \rightarrow AA \\
A \rightarrow a \mid bA \mid Ab
\]

Does it generate \(L\)? If not, add one more production so that it generates \(L\). (6)

No it doesn’t. We cannot generate any string with 4 a’s. To fix it, add the rule \(A \rightarrow AAA\). This will increase the number of \(A\)’s by two each time we use it. The first rule produces 2 \(A\)’s and therefore the number of \(A\)’s is even. Any string with even number of a’s and any number of b’s can be written as concatenation of \(b^*a\) and \(ab^*\). Both of these can be generated using \(A \rightarrow bA\mid Ab\).
3. A shuffle of two strings $x, y \in \Sigma^*$ denoted by $x||y$ is the set of strings that can be obtained by inter-leaving the strings $x$ and $y$ in any manner. For example $ab||cd = \{abcd, acbd, acabd, cadb, cdbd\}$. (The strings need not be of the same length.) For two sets of strings $A, B$, the shuffle is defined as $A||B = \bigcup_{x \in A, y \in B} x||y$.

If $A, B$ are CFL, is $A||B$ a CFL? (10)

No CFL is not closed under $||$. Consider the two CFLs $L_1 = a^i b^j$ and $L_2 = c^j d^i$. Consider a string $a^i c^j b^i d^i \in L_1||L_2$. We can use pumping lemma for sufficiently large $i, j$ to produce strings with unequal number of $a'$ and $b'$s or $c'$s and $d'$s (done in class).

Any other argument based on non-specific use of pumping lemma (not corresponding to any specific language and strings) is not acceptable.

4. Give an algorithm to decide if a given CFG generates exactly 100 strings. You can make use of any procedure that was discussed in class but for anything new you must argue correctness and termination. (8)

Assume that the CFL is described using a CNF. So the pumping lemma constant is $N = 2^k$ where $k$ is the number of variables. We can easily argue that if the number of strings in the CFL is finite then there is no string $s \in L$ such that $N \leq |s| < 2N - 1$. We use the CYK algorithm for membership testing of all possible strings having lengths in this range. This procedure will stop in finite time.

If there is such a string then the answer is NO (infinite strings) otherwise

If there is no such string then we will try all possible strings of lengths $< N$ using the membership testing algorithm. If exactly 100 such strings are accepted the answer is YES else NO.

Note Some of the solutions were based on counting directly the number of strings generated by the grammar $G$. This approach suffers from the following problems
(i) Often the grammar was not converted to a normal form like CNF. As discussed in the class, an arbitrary grammar may have epsilon productions and there is no easy way of generating a parse tree for any specific string.
(ii) The solutions were based on enumerating parse trees which was not discussed in class and it has to be algorithmically described.
(iii) Finding cycles between variables based on common variables in production does not imply infinite strings. All the variables may not yield any terminal string. It must be understood that in the PL proof, there is a long enough string in the language that forces a repetition of variable in the derivation tree and not the other way round.
(iv) In the recursive counting strategy, you have to also check for duplicates since it is not a disjoint union over all productions.
5. How would you compare the computational power of a PDA with two stacks and a deterministic Turing machine. The transition function of such a PDA is

\[ \delta : Q \times \Sigma \times \Gamma \times \Gamma \Rightarrow Q \times \Gamma^* \times \Gamma^*. \]

Here \( Q \) is the set of states, \( \Sigma \) is input alphabet and \( \Gamma \) is stack alphabet.

Consider the IDs of deterministic TM \( \Gamma^* \cdot q \cdot \Gamma^* \cdot B^+ \). Use two stacks \( S_l, S_r \) to store the the tape symbols on either side of the tape head. If the tape head moves left, then pop a symbol from \( S_l \) and push it to \( S_r \). The transition function only looks at the top of \( S_r \) (and ignores the input tape). Initially \( S_l \) is empty and \( S_r \) contains the input with the leftmost symbol on top of the stack. For accepting a string the PDA can do it by final state. So a two stack PDA can simulate a TM.

In the other direction, we can use a multitape TM with one tape containing the input, and two tapes for each of the stacks. When the tape head moves left, a blank is written. Wlog, we can assume that a PDA pushes exactly one symbol on top. (If not we can modify the transition function that pushes one at a time). So a multitape TM can easily simulate a 2 stack PDA.

*Note* Many solutions only presented one of the simulations and got only half the marks.