Partners: Michael Sanchez and Simrun Heir
I pledge my honor that I have abided by the Stevens Honor System.

Problem 1 If G is a CFG in Chomsky Normal Form, show that a string of terminal symbols of length n≥ 1 is generated by the application of exactly 2n − 1 rules of G.

Answer:
A CFG in CNF has every rule in the form

- A → BC where B, C are variables in V-{S}, or
- A → a where a is a terminal in Σ, or
- S → ε where S is the start variable in V

Let n be the length of the string, note n≥1.
Start constructing the string by using the start symbol, which has a length of n=1.
Next, using n-1 rules of form A → BC, you can construct a string of length n.
For each non-terminal symbol in the constructed string, apply the rules of form A → a, meaning the rules of form A → were applied n times.
In the process of constructing the string of length n, n-1+n rules of the CFG were applied.
n-1+n = 2n-1.
When constructing a string of length n from a CFG in CNF, the string is generated by applying exactly 2n-1 rules of G

Problem 2 Let G = (V, Σ, R, S) be a CFG where V = {S, T, U}, Σ = {0, #}, and R is the set of rules:
S → TT | U
T → 0T | T0 | #
U → 0U00 | #
Describe the language L(G) in English and prove that it is not regular.

Notes:
Possible strings:
S → U → (0)^i # 0^2i
S → TT → ## | string with two #'s and an even number of 0s, where #'s can be anywhere within the string, but if one # is at the start of the string then the other # must be at the end of the string or there must be at least 1 0 between them, and the same goes of the string terminates in a #. If the string starts and terminates in a zero then the #'s can either be next to each other somewhere in the string or be separated by a number of zeroes.

Note any string made by U can't be in TT and vice-versa

Answer:
Let G be the CFG for CFL L(G).

L(G) is able to produce strings of two different forms

- Form 1:
    - A string with a certain number of 0s followed by a # and twice as many 0s that preceded the # symbol
    - The number of 0s before and after the # can be any number greater than or equal to 0
    - This string is derived from S → U
- Form 2:
    - A string with an even number of 0s, which includes no zeroes, and two # symbols
    - The #'s can be anywhere within the string
        - If one # is at the start of the string then the other # must be at the end of the string or there must be at least 1 0 between them
            - The same goes of the string terminates in a #
        - If the string has no zeroes then it is just the two #'s next to each other
        - If the string starts and terminates in a zero then the #'s can either be next to each other somewhere in the string or be separated by a number of zeroes
    - This string is derived from S → TT.
- Note that L(G) can produce a string that is derived from U or that is derived from TT, and both strings have different forms, but both strings are in the language.

Proof:
Let p be the pumping length for G that is guaranteed to exist by the pumping lemma.
Select the string s = $0^{p}\#0^{2p}$
s is a member of G and of length at least p.
By the pumping lemma s can be divided into a string s=uv^(i)xy^(i)z.
Note that # cannot be pumped, therefore # cannot be contained within v or y.
The string will always have 0s in either the v or the y, resulting in three cases.
Case 1:

- vxy contains p 0s and z = the #0^(2p).
    - When pumped down to the string uxz, there are no zeroes proceeding the # yet there are 2p zeroes after the #, thus creating a string out of the language, creating a contradiction
    - When pumped up, for example uy^(2)xv^(2)z, creates a string where the number of 0s after the # does not have twice the amount of 0s that proceed the #, creating a contradiction

Case 2:

- vxy contains p 0s, u=0^(p)#, z=0^(p)
    - When pumped down, the string uxz, the number of 0s before and after the # are equal, creating a string out of the language, creating a contradiction
    - When pumped up, for example uv^(2)xy^(2)z, creates a string where there are more than 2p 0s after the #. Since the number of 0s before the # is still p, as u is not the portion of the string that can be pumped, this string is out of the language, creating a contradiction

Case 3:

- x=#, |vy| ≤ p-1, v and y contain only 0s

- Note that the number of 0s in v cannot be equal to the number of 0s in y when p is an equal number
  - This is because if p is even then p-1 is odd, so the amount of 0s in v and y cannot be split evenly
- When pumped up to where i is an even number, for example uv^(2)xy^(2)z, the amount of 0s before the # do not increase in a way that allows for there to be twice as many 0s after the # as there are 0s before the #. This creates a string that is out of the language, creating a contradiction.
  - The same occurs when pumped down, string uxz

Note that because the strings derived from U are in L(G), but not in TT and vice-versa. This also creates a contradiction.

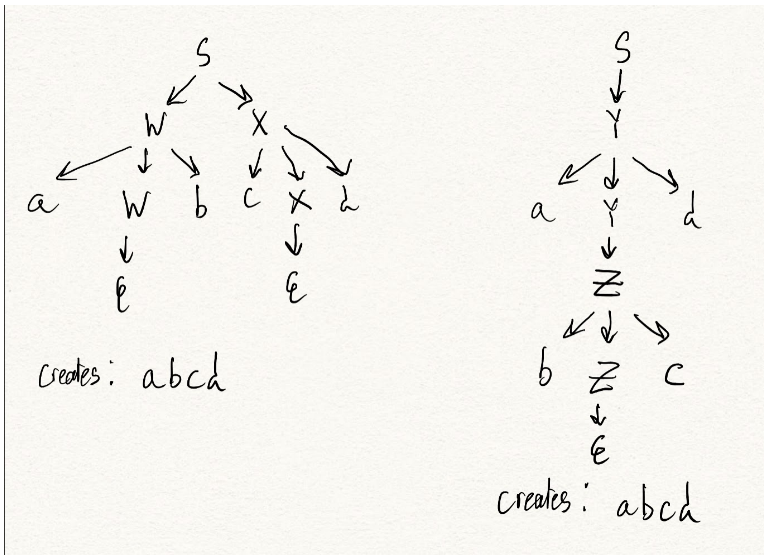
In all cases, ∃i: uv^ixy^iz that violates the pumping lemma by creating a string that does not belong to L(G), therefore L(G) is not context free.

Problem 3 Give a CFG for $\{a^ib^ic^kd^k : i,k \geq 0\} \cup \{a^ib^kc^kd^i : i,k \geq 0\}$. Is your grammar ambiguous?

Answer:
Let G = (V,Σ,R,S) be the CFG for the language given in the problem, where V = {S, W, X, Y, Z} and R is the set of rules:

S → WX | Y
W → aWb | ε
X → cXd | ε
Y → aYd | Z | ε
Z → bZc | ε

Check to see if ambiguous:
Parse trees for string abcd in grammar G



Grammar G is ambiguous as it has two different leftmost derivations for strings where a=b=c=d, as in the number of each is the same. This is done by building the parts that are equal in length, and then moving on to build the next equal length segment of the string, since there are two ways to do this when a=b=c=d there are two different leftmost derivations.

Problem 4 Let $L_{add} = \{a^i b^{i+j} c^j : i,j \geq 0\}$ and $L_{mult} = \{a^i b^{ij} c^j : i,j \geq 0\}$. For each language, either give a CFG for it, or prove that is not a CFL

Answer:
$L_{add} = \{a^i b^{i+j} c^j : i,j \geq 0\}$
Let ADD = (V, Σ, R, S) be a CFG for L_{add} where V = {S, F, H}, Σ = {a, b, c}, and R is the set of rules::
S → aFbH | FbHc | ε
F → aFb | ε
H → bHc | ε

$L_{mult} = \{a^i b^{ij} c^j : i,j \geq 0\}$:
Not a CFL
Proof:
Assume L_{mult} is a CFL
Let p be the pumping length, guaranteed to exist by the pumping lemma
Let s = *$\{a^p b^{p\wedge 2} c^p\}$*:
By the pumping lemma, we can break s into five parts, uv^ixy^iz, where:

1. For all i ≥ 0, uv^ixy^iz, ϵ L_{mult}
2. |vy| > 0
3. |vxy| ≤ p

For 5 different ways to break up the string:

1. vxy = a^t for some t ≤ p, z=$b^{p\wedge 2}c^p$. When only pumping down, creating the string uxz, |b| ≠ |a|*|c|, which exits the language.
2. vxy = a^tb^k for some t and k where t+k ≤ p, z=c^p. When pumping down creating the string uxz, |b| ≠ |a|*|c|, which exits the language.
   When pumped up, ∃i: uv^ixy^iz does not belong to L_{mult}, as |b|/|a| ≠ |c|
3. vxy = b^t for some t ≤ p. When pumping down, creating the string uxz, |b| ≠ |a|*|c|, which exits the language.
4. vxy = b^tc^k for some t and k where t+k ≤ p, u=a^p. When pumping down, creating the string uxz, |b| ≠ |a|*|c|, which exits the language.
   When pumped up, ∃i: uv^ixy^iz does not belong to L_{mult}, as |b|/|c| ≠ |a|
5. vxy = c^t for some t ≤ p, u=$a^pb^{p\wedge 2}$.When pumping down, creating the string uxz, |b| ≠ |a|*|c|, which exits the language.

In all cases, ∃i: uv^ixy^iz does not belong to L_{mult}, therefore, L_{mult} is not a CFL.

Optional Problem 5 Let Σ={a,b}. Give a CFG to generate all and only strings which contain twice as many a's as b's. Give a proof that your grammar is correct.

Answer:
Let G = (V,Σ,R,S) be the CFG for CFL A, where V = {S}, and R is the set of rules:

S → bSaSa | aSbSa | aSaSb | SS | ε

Proof:
Let G represent the CFG for CFL A. Then L(G) = L.

For G to be correct L(G) ⊆ L and L ⊆ L(G).
L(G) ⊆ L

- The application of the rules in G always produces a string that has twice as many a's as b's
- In order for this to be true each string will have to produce 2 a's every time it produces a b, which it does (S → bSaSa | aSbSa | aSaSb), or produces nothing (S → SS | ε)
- Therefore any string generated by G must have twice as many a's as b's.
- Then L(G) ⊆ L

L ⊆ L(G)

- The smallest possible strings are x0∈{ε, aab, aba, baa} all of which have Na(x0)=2*Nb(x0), where Na(x) represents the number of a's in string x and Nb(x) represents the number of b's in the string x.
- Assume Na(xn) = 2*Nb(xn) holds for all values of n.
- Basis:
    - x=x0
    - Na(x0)=2*Nb(x0) is true because x0∈{ε, aab, aba, baa}.
- Inductive Hypothesis:
    - Na(xn) = 2*Nb(xn) holds for all values of n
- Inductive Step:
    - Let s represent the string that is inserted to the string every time a rule is carried out
        - s∈{ε, aab, aba, baa}
    - x(n+1) = xn + s
    - Every insertion of s into the strings either adds 0 a's and b's or 2 a's and 1 b.
    - Case of adding 0 a's and b's
        - Na(x(n+1)) = Na(x(n))
        - Nb(x(n+1)) = Nb(x(n))
        - Na(x(n)) = 2*Nb(x(n))
        - Na(x(n+1)) = 2*Nb(x(n+1))
    - Case when 2 a's and 1 b are inserted
        - Na(x(n+1)) = Na(x(n)) +2
        - Nb(x(n+1)) = Nb(x(n)) + 1
        - Na(x(n)) + 2 = 2*(Nb(x(n)) +1)
        - Na(x(n+1)) = 2*Nb(x(n+1))
- Therefore, by induction, all strings generated using the grammar contain twice as many a's as b's.

Since $L(G) \subseteq L$ and $L \subseteq L(G)$, the grammar G correctly generates the strings in language L, meaning the grammar G is the correct CFG for CFL A.