# CS 383
# HW 6 Solutions

1. Convert the following grammar into Chomsky Normal Form:
   S => ASB | $\varepsilon$
   A => aAS | a
   B => SbS | A | bb

   If you remove the $\varepsilon$-rule and the one unit rule (B => A), the rest is just splitting the rules into 2 nonterminals on the right hand side:
   - S => $S_1$B | AB
   - $S_1$ => AS
   - A => $A_1$S | $A_2$A | a
   - $A_1$ => $A_2$A
   - $A_2$ => a
   - B => $B_1$S | $B_2$S | S$B_2$ | $A_1$S | $A_2$S | a | $B_2B_2$
   - $B_1$ => S$B_2$
   - $B_2$ => b

2. Chomsky Normal Form forces parse trees to be binary trees. Some people like trinary trees. Say that a grammar is in "Bobsky Normal Form" if all rules have the form A => BCD or A => a. Can all context free grammars be converted to Bobsky Normal Form? Either find a grammar that can't be converted or prove that all can.

   No, Bobsky Normal Form is stupid. A => aa can't be converted to Bobsky Normal Form. For that matter, any string derived from a Bobsky Normal Form grammar must have odd length.

3. Show that $\{0^i1^j2^k \mid i < j < k\}$ is not context-free
   Suppose it is context-free; let p be its pumping constant. Let z = $0^p1^{p+1}2^{p+2}$ . Let z=uvwxy be any pumping decomposition of z with |vwx| <= p. vwx can't contain both 0s and 2s because they are separated by p+1 symbols. If vwx contains any 0s then $uv^3wx^3y$ has more 0s than 2s. If vwx has 1s but no 0s then $uv^0wx^0y$ does not have more 1s than 0s. So vwx can't have any 0s or any 1s. If it is all 2s then $uv^0wx^0y$ does not have more 2's than 1's. Any way you try, the string z is not pumpable.

4. For each of the following languages either prove that the language is context-free or prove that it isn't:
   a. $\{0^n1^m \mid n, m > 0\}$
   b. $\{0^n1^m \mid n > 0, m=n\}$
   c. $\{0^n1^m \mid n > 0, 0 < m < 2n\}$
   d. $\{0^n1^m2^n \mid n, m > 0\}$
   e. $\{0^n1^m2^n$ n, m > 0, 0 < m < n}

   (a) is regular (and so context-free): $0^+1^+$
   (b) is context free: S => 0S1 | 01
   (c) is context-free: make a PDA that on seeing a 0 nondeterministically pushes 2 0s onto the stack. When it sees a 1 transitions to a state that pops a 0 every time it sees a 1. If there are still 0s on the stack at the end of the input it accepts.
   (d) is context-free: make a PDA that reads 0s and pushes them onto the stack. When it sees a 1 it transitions to a state that reads and ignores 1, not changing the stack. When it sees a 2 it pops the stack and transitions to a state that reads 2s and pops the stack. The stack should be empty at the end of the input.
   (e) is not context-free. If it was let p be its pumping constant, let z=$0^{p+1}1^p2^{p+1}$. A pumping decomposition z=uvwxzy can't have vwx containing both 0s and 2s. If it contains 0s and not 2s or 2s and not 0s then after pumping we get different numbers of 0s and 2s. If it contains only 1s then after pumping we get more 1s than 0s or 2s. Any way we slice it, z can't be pumped.

5. Give an algorithm for determining if the language derived from a given context-free grammar is infinite.

   Here is how I would go about this. You can give a different algorithm based on grammars (by checking if it is possible to have a repeated symbol in a grammar derivatiuon). Let p be the language's pumping constant ($2^{\text{\# nonterminals in a CNF grammar}}$) If the language contains a string longer than p that string is pumpable and the language is infinite. If the language has a long string and we pump it 0 times we remove at most p symbols from it, so the language is infinite if and only if it contains a string w of length between p and 2p. So generate all of the strings with length between p and 2p and check each for membership in the language. This might take a while, but it will eventually terminate.

6. Give an algorithm for determining if the language derived from a context-free grammar G is empty (i.e., the grammar derives no strings).

   In class we gave an algorithm for marking the nonterminal symbols that generate terminal strings (If A => w is a rule where w consists only of terminal symbols, mark A. If A=>$\alpha$ is a rule where every symbol in a is either terminal or marked, mark A. Repeat until nothing else can be marked) The language is empty if and only if the start symbol for the grammar is not marked.