Homework 2 Solutions
Csc 4330/6330, Spring 2018

1. Prove that the following grammars are ambiguous (**Note:**In the following grammars, we have two terminal symbols, 'a' and 'b' and all non-terminal symbols are in upper-case letters, such as 'A' and 'S'. 'S' is the start symbol and ε is the empty string.):

   Grammar G1

```
S -> SS
S -> ab
S -> ba
S -> ε
```

   w = ab
   S => ab
   S => SS => abS => ab

   Grammar G2

```
S -> AabaA
A -> aA
A -> bA
A -> ε
```

   w = ababa
   S => AabaA => abaA => ababA => ababaA => ababa
   S => AabaA => Aaba => aAaba => abAaba => ababa

   Grammar G3

```
S -> aSb
S -> bSa
S -> SS
S -> ε
```

   w = abab
   S => aSb => abSab => abab
   S => SS => aSbS => aSbaSb => abaSb => abab

2. Consider the following grammar:

```
S -> aScB | A | b
A -> cA | c
B -> A | d
```

Here again, we assume lower-case symbols such as `a,b,c,d` are terminals and upper-case symbols `S`, `A`, and `B` are non-terminals and `S` is the start symbol. Which of the following sentences are in the language generated by the grammar:

1. `abcd`

   Yes
   S => aScB => abcB => abcd

2. `acccbd`

   No
   S => aScB must be the first rule applied to generate the leading 'a' and to generate the trailing 'd', B => d must be applied to get aScd which means that the last 'd' cannot be preceded by a 'b'

3. `accbcc`

   No
   S => aScB must be the first rule applied to generate the leading 'a'. Now if S is used to generate the second 'c', the 'b' cannot be generated and if S is used to generate the b then no 'c's can be generated between 'a' and 'b'.

4. `acd`

   No;
   S => aScB must be the first rule applied to generate the leading 'a'. Now S does not derive the empty string and therefore some non-empty string must appear between 'a' and 'c'; hence "acd" cannot be derived.

5. `accc`

   Yes;
   S => aScB => aAcB => accB => accA => accc

3. Write ANTLR4 grammar and lexer rules for "Datalog Rules". A Datalog rule is of the form:

```
p :- q1, ..., qn.
```

where `n>0` and `p`, `q1`, ..., and `qn` are atomic formulas of the form:

```
r(x1, ..., xm)
```

where `m>0` and `r` is a relation name and `x1` , ..., `xm` are either numbers or variables. Relation names and variables are made up of alphabetic letters or digits and start with a lower-case alphabetic letter. Numbers are positive integers or zero very much like numbers in the WAE example. Some examples of Datalog rules are:

```
ancestor(x1,y1) :- parent(x1,y1).
teaches(tno,cno) :- faculty(tno), course(cno), assigned(tno,cno).
p(x,y) :- q(2,x), r(y,45,z), s(a,b,22).
```

Put the grammar in a file, DLG.g4 and test it out.

```
grammar DLG;

dlog : relation IF body PERIOD;
relation : ID LPAREN args RPAREN;
args : arg | arg COMMA args;
arg : NUMBER | ID;
body : relation | relation COMMA body;

fragment VALID_ID_START : ('a'..'z');
fragment VALID_ID_CHAR : ('a'..'z') | ('A'..'Z') | ('0'..'9');
NUMBER : ('0'..'9')+;
LPAREN : '(';
RPAREN : ')';
PERIOD : '.';
COMMA : ',';
IF : ':-';
ID : VALID_ID_START VALID_ID_CHAR*;
WS : [ \r\n\t]+ -> skip;
```

4. Write a grammar for the language consisting of strings that have `n` copies of the letter `a` followed by the same number of copies of the letter `b`. For example, the strings `ab`, `aaabbb`, and `aaaaabbbbb` are in the language but `aaabb`, `bbaa`, and `aba` are not. Draw the parse tree for `aaabbb`.

S -> aSb | epsilon

S => aSb => aaSbb => aaaSbbb => aaabbb