Homework 6 — Prolog — assigned Wednesday April 20 — due Friday 6 May

6.1 Multiplication [A.1; K.2.6] (20pts)

Define numerals as follows:

```
%% num(X) means: X is a numeral.
num(0).
num(s(X)) :- num(X).

%% sum(X, Y, Z) means: X, Y, Z are numerals
%%         such that Z is the sum of X and Y.
sum(X, 0, X) :- num(X).
sum(X, s(Y), s(Z)) :- sum(X, Y, Z).
```

Write the rules for multiplication `mult(X, Y, Z)`, meaning that `Z` is the product of `X` and `Y`. Multiplication is defined by the following axioms:

- $x \cdot 0 = 0$
- $x \cdot s(y) = (x \cdot y) + x$

6.2 Arithmetic evaluator [A.2; K.1.2; K.2.6] (25pts)

In this exercise you should use the built-in arithmetic of Prolog. The relation `ev` is between an expression in a very restricted arithmetic language described below, and a number which that expression evaluates to, in this little language. You can assume that the first argument is instantiated.

```
?- ev(3,N).
N=3

?- ev([+,3,4], N).
N=7

?- ev([+,3,[*,8,0.25]], N).
N=5
```

Expressions in the "little language" are defined as

```
E ::= number
    | [+, E, E]
    | [*, E, E]
    | [exp, E]
    | [sin, E]
    | [cos, E]
```

These have the obvious numeric meanings when evaluated. Write the rules for `ev`.

6.3 Tic-tac-toe [K.1.2] (25pts)

Write a Prolog program that allows a human user to play tic-tac-toe against the computer. Either the computer or the human can move first. The computer must use a policy that guarantees it will not lose if it plays first. The first player to move is the X player, the second is the O player.

Suggested representations of a tic-tac-toe board include these possibilities: `[x,o,o,b,x,b,b,b,x]` or `[[x,o,o],[b,x,b],[b,b,x]]`. (`b` stands for an unoccupied square.) Both of these examples are lists intended to represent a winning board for X in which X has conquered the bend (the diagonal from top left to bottom right).

You do not have to worry about interactive input-output. It suffices to write a predicate `computersmove(L1,L2)`, which, when given an instantiated board L1, produces a unique board L2 as the outcome of the computer's move. Assume that the human will repeatedly query this predicate.

Hints: Consider writing a predicate `xwins(L)`, meaning that the board L is a winning board for X, and a similar predicate `owins(L)`. Consider writing a predicate `okmove(L1,L2)`, meaning that from the board L1 it is legal to get to the board L2 in the very next move.

6.4 Grammars [A.7; K.1.2; K.2.6] (30pts)

The following language over the alphabet $\{a,b,c\}$:

$$L = \{a^n b^n c^n \mid n \in \mathbb{N}\}$$

can be generated by the following grammar:

$$G = (\{S,X,Y\},\{a,b,c\},S,F),$$

where the productions are:

$$F = \{S \to abc, S \to aXbc, Xb \to bX, Xc \to Ybcc, bY \to Yb, aY \to aaX, aY \to aa\}$$

This grammar is not context-free, because on the left-hand sides of some productions there are strings longer than single non-terminals. (In fact, there is no context-free grammar for this language.) However, the productions in this grammar satisfy a *length-increasing* property: the number of symbols on the right is never less than the number on the left. Such grammars are called context-sensitive. (Note that there are grammars, and their languages, that are not even context-sensitive.)

Write a Prolog program that determines whether a string belongs to the language L. Strings over the alphabet $\{a,b,c\}$ will be represented as Prolog lists of Prolog atoms `a`, `b`, and `c`. You should split the program into a set of predicates that are general-purpose (for any context-sensitive grammar), and a set of predicates that describe the grammar G.

The program should result in queries like these:

```
?- accepts([]).
```

```
No
?- accepts([a]).
No
?- accepts([a,b]).
No
?- accepts([a,b,b]).
No
?- accepts([a,b,c]).
Yes
?- accepts([a,a,b,b,c,c]).
Yes
?- accepts([a,a,a,a,a,a,b,b,b,b,b,b,c,c,c,c,c,c]).
Yes
?- accepts([a,a,a,a,a,a,b,b,b,b,b,b,c,c,c,c,c,c,a]).
No
```

You may find useful the predefined predicate `length/2`, which works as follows: given a list `X`, the goal `length(X,N)` succeeds and sets N to the integer equal to the length of the list `X`.

You should test your program on the queries above and others you find useful to cover the various cases of strings in the language and outside the language. Make sure that your program does not go into an infinite loop on any inputs. Do not submit your program unless it has passed all these tests.

6.5 Diplomacy [C; K.1.2; K.2.6] (60pts extra credit)

The rules of the game *Diplomacy* are available at the URL
`http://www.diplomacy-archive.com/resources/rulebooks/2000AH4th.pdf`.

A wealth of other information about the game can be found at:
`http://www.diplomacy-archive.com/`.

Obtain the rules and familiarize yourself with the game.

In each move of the game, each of the seven players writes a list of *orders* for the units controlled by the player. All the orders are revealed at once, and at that point any conflicts between orders are resolved according to the rules of the game. Some moves succeed, while others fail.

Devise a representation for a configuration of the game in Prolog: represent the map (provinces and adjacency; supply centers), the units (armies and fleets), and the orders.

Write a Prolog predicate which, given a complete list of orders for one move, determines which of the orders succeed and which fail.

6.6 Regular languages [A.1; A.7] (60pts extra credit)

Consider deterministic finite automata with N states over an alphabet of K symbols $\{\sigma_1, \ldots, \sigma_K\}$. An automaton is described by giving its start state, its set of accept states, and its transition function. How many distinct automata are there for some given N and K?

Call the number above $F(N,K)$. Each of the $F(N,K)$ automata defines a language, but not all these languages are different. Let $G(N,K)$ be the number of distinct languages among the languages defined by deterministic finite automata with N states over an alphabet of K symbols $\{\sigma_1,\ldots,\sigma_K\}$. Find a closed form for $G(N,K)$.

6.7 Scheme [K.1.2; K.1.4; K.2.6; K.2.7] (50pts extra credit)

Design a representation for Scheme values and expressions in Prolog, and write a predicate `evaluatesto/2` which relates a Scheme expression to its value, if any. Carefully choose the Scheme subset you wish to handle—make it as small as possible to keep the exercise manageable. Remember that `lambda` is sufficient, and that without `lambda` you don't have Scheme.

How to turn in

Turn in your code by running

~jackmp/cs451TA/handin your-file

on a regular UNM CS machine. You should use whatever filename is appropriate in place of your-file.

Include the following statement with your submission, signed and dated:
I pledge my honor that in the preparation of this assignment I have complied with the University of New Mexico Board of Regents' Policy Manual.