COMP 356 Homework Assignment 9

Acknowledgment. This assignment was written by Prof. Tim Wahls, with minor changes by John MacCormick.

1. Consider the following Prolog program:

```
parent(X, Y) :- mother(X, Y).
parent(X, Y) :- father(X, Y).
grandparent(X, Y) :- parent(X, Z), parent(Z, Y).
ancestor(X, Y) :- parent(X, Y).
ancestor(X, Y) :- parent(Z, Y), ancestor(X, Z).
```

   Extend the above program with rules defining the following relations:

   (a) (5 pts) A relation `fullSibling(X, Y)` that determines whether two people are full siblings (have the same mother and father). Note: According to this definition, any given person is sibling to him or herself. Your relation definition should permit this possibility.

   (b) (5 pts) A relation `firstCousins(X, Y)` that determines whether two people are first cousins. The first cousins of a person P are the children of P's parents' brothers and sisters. For this question, "brothers and sisters" means "brothers, sisters, half-brothers and half-sisters." In other words, the definition is not restricted just to full siblings as in the previous part of this question. Important hint: you'll need to use the built-in not-provable operator `\+` here. For technical reasons, the `\+` clause should be the *last* subgoal in any rule where you use it. Otherwise, you will see unexpected results.

2. (5 pts) The following Prolog relation reverses a list:

```
reverse([], []).
reverse([A | X], Z) :- reverse(X, Y), append(Y, [A], Z).
```

   Use this relation to define a relation that determines whether a list is a palindrome (is the same when read from right to left as when read from left to right). For example, if your relation is called `palindrome`, then the query:

```
palindrome([1,2,3,2,1]).
```

   should succeed, while the query:

```
palindrome([1,2,3,1]).
```

   should return `Failure`.

   Note that `reverse` is built-in to XGP, but not SWI-Prolog. So, if you are using SWI-Prolog, you will need to include the two rules above in your program.

3. (5 pts) Define a Prolog relation that finds the intersection of two sorted lists of numbers (in ascending order) – i.e. all elements that occur in both of the argument lists. That is, if your relation is called `intersectS`, then the query:

```
intersectS([1, 2], [], Z).
```

should return: `Z=[]`, and the query:

```
intersectS([1, 2, 3, 4], [2, 4, 6], Z).
```

should return: `Z=[2, 4]`.

You are NOT allowed to use the relation `intersect` that is built-in to XGP, and you must call your relation `intersectS`. For full credit, your relation must take advantage of the fact that the lists are sorted.

4. Give the substitution needed to unify each of the following pairs of terms, or explain why they cannot be unified. Your answers for this question should be placed as comments in your Prolog source file. Prolog uses `/* ... */` style comments, and you should use the symbol `->` to express a binding in a substitution, e.g. `{X -> 2, Y -> 3}`.

   (a) (2 points) `[X, Y | Z]` and `[1, 2, 3, 4]`

   (b) (3 points) `node(node(L, D, R), D, empty)` and
   `node(node(node(empty, 2, empty), 4, empty), 5, empty)`

To turn in this assignment, submit your solutions as a single Prolog file to Moodle. Your relations will be graded on correctness, compliance with the above guidelines, and coding style.