Assignment 8:
Prolog Programming

15-317: Constructive Logic

Anna Gommerstadt
Due: Wednesday 11/18/15 **1:30 PM EST**
Total points: 40

The purpose of this assignment is to familiarize you with the basics of programming in (nearly) pure Prolog, and to get you to experiment with the behavior of backtracking search on the deductive systems we studied in the first part of the semester.

Submit file, part1.pl, part2.pl, and part3.pl before the deadline by copying it to the directory

```
/afs/andrew/course/15/317/submit/<userid>/hw8
```

where `<userid>` is replaced with your Andrew ID.

1. Prolog programming

   For this part of the assignment, you will write and test some simple Prolog predicates, adhering to the following standards:

   - Don't use cut, conditional, or any other control operator.
   - Don't use negation-as-failure or disequality.
   - You may use any of the list processing predicates or arithmetic predicates provided by gprolog, except `maplist` or any of the sorts.
   - Backtracking must not result in nontermination or in wrong answers. That is, when the user types `;` in response to a query result, execution must terminate with a correct answer or `no`.
   - Backtracking may produce redundant correct answers.
   - When your code is loaded and compiled, no warnings should be printed.
   - Put your definitions in one file named `part1.pl`, *clearly delineating the parts with comments.*

[5] (a) Define the following operators for negation, conjunction, disjunction, and implication. In order of priority:

   1. Negation `neg` (prefix)
   2. Conjunction `and` (infix, left associative)
   3. Disjunction `or` (infix, left associative)
   4. Implication `implies` (infix, right associative)

Define predicate `eval/2`. The intended use is that the first argument is an input Boolean formula consisting of `true`, `false`, and logical operators, and the second is the result of evaluating that formula.

Example:

```
| ?- eval(true or a, V).

V = true ? ;

no
| ?- eval(true and a, V).

no
| ?- eval(true and false, V).

V = false

yes
```

5 (b) A propositional formula is in *conjunctive normal form* if it is a conjunction of clauses, where each clause is a disjunction of literals, and a literal is either an atom or the (single) negation of an atom.

Define the predicate `cnf/1`, which succeeds iff its argument is a formula in conjunctive normal form.

Example:

```
| ?- cnf(a and b or c).

no
| ?- cnf(a and (b or c)).

yes
| ?-
```

5 (c) Polynomials can be represented in Prolog as lists of pairs (c, n). Each pair represents one term with coefficient c and exponent n.

Define a predicate `poly_sum/3` to compute the sum of two polynomials represented in this way. The definition should not depend on any particular ordering of the lists, but its behavior in the presence of duplicate exponents in a list is undefined (you may do whatever you like in that case).

Example:

```
| ?- poly_sum([(1,5),(1,8),(1,0)],[(1,8),(2,0),(2,5),(2,1)],P).

P = [(3,5),(2,8),(3,0),(2,1)] ?
```

2. For this question you will write straightforward translations of natural deduction and the sequent calculus for minimal logic (there is no rule for falsehood) for pure propositions (no quantifiers). Your code should adhere to the standards above, except that it's not required to terminate, for obvious (we hope!) reasons.

Use the operators you defined above (`neg, and, or, implies`) to represent propositions. Put all the code for this part in a file named `part2.pl`.

5 (a) Define a predicate `prove/2`; `prove(Gamma,A)` should succeed iff there is a natural deduction proof of $\Gamma \vdash A$ that does not use falsehood elimination. E.g.

```
prove([], a implies a).

true
```

5 (b) Define a predicate `entails/2`; `entails(Gamma,A)` should succeed iff there is a sequent calculus proof of $\Gamma \Longrightarrow A$ that does not use the left rule for falsehood. E.g.

```
| ?- entails([a], a).

true
```

5 (c) In comments, give four different formulas for each predicate where the predicate succeeds. Also give a formula where `entails` succeeds but `prove` does not terminate, and explain why.

10 3. (a) In this question, you will need to implement graph coloring. As one example, it must work on the graph corresponding to the map below. Your job is to write a predicate `color_graph(graph, color_map)` that associates with `graph` all of the valid 4-colorings of the graph (no two vertices connected by an edge may have the same color). We will only test your program with planar graphs (the 4-color theorem says that all planar graphs are 4-colorable).

Put all the code for this part in a file named `part2.pl`.

1. You should define a `color/1` predicate with four colors.
2. Assume there are predicates `node/1` and `edge/2`. You will need to come up with some of these for testing purposes, but feel free to share your tests with the rest of the class on the discussion board. Your tests may even be used for grading purposes, so it would be convenient for you if you already knew you passed some of the test cases!
3. Implement `color_graph(graph, color_map)`, where `color_map` is a list of terms, where each term is of the form `paint(node, color)`.