CSE 130 - Programming Assignment #7

Prolog

100 points

(see submission instructions below)

(click your browser's refresh button to ensure that you have the most recent version)

(Programming Assignment #7 FAQ)

Note: To download and install SWI-Prolog on your home machines see this page. Remember that this is only to enable you to play with the assignment at home: The final version turned in must work on the ACS Linux machines. While you can use Windows to begin working with Prolog, the code you turn in must be that required for the ACS Linux environment.

Code Documentation and General Requirements

Code for all programming assignments should be **well documented**. A working program with no comments will **receive only partial credit**. Documentation entails writing a description of each predicate as well as comments throughout the code to explain the program logic. Comments in Prolog begin with a percent sign (%) and are terminated by a newline/carriage return. It is understood that some of the exercises in this programming assignment require extremely little code and will not require extensive comments. Nevertheless, comments describing recursions and helper predicates are required.

While few programming assignments pretend to mimic the "real" world, they may, nevertheless, contain some of the ambiguity that exists outside the classroom. If, for example, an assignment is amenable to differing interpretations, such that more than one algorithm may implement a **correct** solution to the assignment, it is incumbent upon the programmer to document not only the functionality of the algorithm (and more broadly his/her interpretation of the program requirements), but to articulate **clearly** the reasoning behind a particular choice of solution.

Assignment Overview

The overall objective of this assignment is for you to gain some hands-on experience with problem solving using Prolog, using simple facts and rules, recursion, and database handling capabilities of the language.

It is a good idea to start this assignment early; Prolog programming, while not inherently difficult, often seem somewhat foreign at first, particularly when it comes to recursion and list manipulation.

So as not to make the code overly long, it is not required that you deal with user errors: you can assume that the user always types valid commands (e.g., if a predicate is supposed to take an atom as argument, you do not have to check whether the argument is instead a list and throw an error). Note that the Prolog interpreter catches a good number of user errors anyway.

Submission Guidelines/Instructions

1. Testing

Your functions/programs **must** compile and/or run on a **UNIX** ACS machine, as this is where the verification of your solutions will occur. While you may develop your code on any system, ensure that your code runs as expected on an ACS machine prior to submission. Code which does not compile and/or produce the desired results on the testing machine will receive little or no credit. You should test your code in the directories from which the tar file (see below) will be created, as this will approximate the environment used for grading the assignment.

2. Submitting the program via the `turnin_pa7` program

You will turn in your solution, by filling in the template file: misc.pl.

This file is submitted using turnin as follows:

```
turnin_pa7 misc.pl
```

The `turnin_pa7` program will provide you with a confirmation of the submission process; make sure that the size of the file indicated by `turnin_pa7` matches the size of your file. Note that you may submit multiple times, but your latest submission overwrites previous submissions, and will be the ONLY one we grade. If you submit before the assignment deadline, and again afterwards, we will count it as if you only submitted after the deadline.

General Hint:

- you may use built-in functions `append, reverse, bagof` judiciously

Problem #0 - Lists

The skeleton file provides several useful list functions (`isin, zip, assoc, remove_duplicates, union, intersection`) that you may use in the next problem if you wish. Although these functions have been defined for you, make sure you understand how they work.

Problem #1 - Taqueria Database

Prolog can be used as a sophisticated database system in which data is stored in the form of structured predicates. In this problem we consider a database of taquerias that sell various items, for a variety of budgets and palates. First, we have a set of facts that encode the price of different ingredients. For example, the first fact stipulates that a serving of carne asada costs 3 dollars (this is high-quality, organic, grass-fed beef).

```
cost(carne_asada,3).
cost(lengua,2).
cost(birria,2).
cost(carnitas,2).
cost(adobado,2).
cost(al_pastor,2).
cost(guacamole,1).
cost(rice,1).
cost(beans,1).
cost(salsa,1).
cost(cheese,1).
cost(sour_cream,1).
cost(taco,1).
cost(tortilla,1).
cost(sopa,1).
```

Next, we have a list of menu items, and for each item, we have a fact that tells us which ingredients go into the item. For example, the carnitas taco, is a tasty snack comprising carnitas, salsa and guacamole, generously stuffed into a fresh taco.

```
ingredients(carnitas_taco, [taco,carnitas, salsa, guacamole]).
ingredients(birria_taco, [taco,birria, salsa, guacamole]).
ingredients(al_pastor_taco, [taco,al_pastor, salsa, guacamole, cheese]).
ingredients(guacamole_taco, [taco,guacamole, salsa,sour_cream]).
ingredients(al_pastor_burrito, [tortilla,al_pastor, salsa]).
ingredients(carne_asada_burrito, [tortilla,carne_asada, guacamole, rice, beans]).
ingredients(adobado_burrito, [tortilla,adobado, guacamole, rice, beans]).
ingredients(carnitas_sopa, [sopa,carnitas, guacamole, salsa,sour_cream]).
ingredients(lengua_sopa, [sopa,lengua,beans,sour_cream]).
ingredients(combo_plate, [al_pastor, carne_asada,rice, tortilla, beans, salsa, guacamole, cheese]).
ingredients(adobado_plate, [adobado, guacamole, rice, tortilla, beans, cheese]).
```

Finally, we the database has a set of facts specifying the different taqueries, and for each taqueria, the list of employees, and the list of comestibles available for purchase at the taqueria.

```
taqueria(el_cuervo, [ana,juan,maria],
        [carnitas_taco, combo_plate, al_pastor_taco, carne_asada_burrito]).

taqueria(la_posta,
        [victor,maria,carla], [birria_taco, adobado_burrito, carnitas_sopa, combo_plate, adobado_plate]).

taqueria(robertos, [hector,carlos,miguel],
        [adobado_plate, guacamole_taco, al_pastor_burrito, carnitas_taco, carne_asada_burrito]).

taqueria(la_milpas_quatros, [jiminez, martin, antonio, miguel],
        [lengua_sopa, adobado_plate, combo_plate]).
```

The first store has three employees and sells four different items, the second store has three employees and sells five different items, and so on. You can assume that there are no duplicates (`carnitas` is not listed twice in any ingredient list, `maria` is not listed twice in any employee list, the same menu item is is not listed twice in any store menu, etc.). Given a database of facts, the questions below have you write predicates that implement queries to the database.

(a) available_at (5 Points)

Write a Prolog predicate `available_at(X,Y)` that is true if the item `X` is available at the taqueria `Y`. When you are done, you should get the following behavior:

```
?- available_at(lengua_sopa,el_cuervo).
false

?- available_at(X,Y).
X = carnitas_taco,
Y = el_cuervo ;
X = combo_plate,
Y = el_cuervo ;
X = al_pastor_taco,
Y = el_cuervo ;
X = carne_asada_burrito,
Y = el_cuervo ;
X = birria_taco,
Y = la_posta ;
X = adobado_burrito,
Y = la_posta ;
X = carnitas_sopa,
Y = la_posta ;
X = combo_plate,
Y = la_posta ;
```

```
X = adobado_plate,
Y = la_posta ;
X = adobado_plate,
Y = robertos ;
X = guacamole_taco,
Y = robertos ;
X = al_pastor_burrito,
Y = robertos ;
X = carnitas_taco,
Y = robertos ;
X = carne_asada_burrito,
Y = robertos ;
X = lengua_sopa,
Y = la_milpas_quatros ;
X = adobado_plate,
Y = la_milpas_quatros ;
X = combo_plate,
Y = la_milpas_quatros ;
false.
```

Similarly, if one has a hankering for a carnitas taco, one can query the database thus:

```
?- available_at(carnitas_taco,Y).
Y = el_cuervo ;
Y = robertos ;
false
```

(b) multi_available (10 Points)

Write a Prolog predicate `multi_available(X)` that is true if the item `X` is available in more than one place. For example:

```
?- multi_available(carnitas_taco).
true

?- multi_available(lengua_sopa).
false

?- multi_available(X).
X = carnitas_taco ;
X = carne_asada_burrito ;
X = adobado_plate ;
X = combo_plate ;
false
```

(c) overworked (10 Points)

Write a Prolog predicate `overworked(X)` that is true the person `X` works at more than one taqueria. For instance:

```
?- overworked(maria).
true

?- overworked(carlos).
false

?- overworked(X).
X = maria ;
X = miguel ;
false
```

(d) total_cost (15 Points)

Write a Prolog predicate `total_cost(X,K)` that is true if the sum of the costs of the ingredients of item `X` is equal to `K`. When you are done, you should get the following:

```
?- total_cost(carnitas_taco,3).
false

?- total_cost(carnitas_taco,X).
X = 5

?- total_cost(X,5).
X = carnitas_taco ;
X = birria_taco ;
X = lengua_sopa ;
false
```

(e) has_ingredients (20 Points)

Write a Prolog predicate `has_ingredients(X,L)` that is true if the item `X` has all the ingredients listed in `L`. When you are done, you should get the following:

```
?- has_ingredients(lengua_sopa,[cheese,lengua]).
false

?- has_ingredients(X,[salsa,guacamole,cheese]).
X = al_pastor_taco ;
X = combo_plate ;
false
```

(e) avoids_ingredients (20 Points)

Write a Prolog predicate `avoids_ingredients(X,L)` that is true if the item `X` does not have any of the ingredients listed in `L`. When you are done, you should get the following:

```
?- avoids_ingredients(lengua_sopa,[cheese,lengua]).
false

?- avoids_ingredients(lengua_sopa,[cheese,tortilla]).
true

?- avoids_ingredients(X,[guacamole]).
X = al_pastor_burrito ;
X = lengua_sopa ;
false

?- avoids_ingredients(X,[salsa,guacamole]).
X = lengua_sopa ;
false
```

(f) find_items (10 + 10 Points)

In the given file, we have filled in an implementation for a predicate `find_items(L,X,Y)` that is true if `L` is the list of *all items* that contain all the ingredients in `X` and do not contain any of the ingredients in `Y`. This predicate is specified using two helper predicates `p1,p2` that *you must implement*, to obtain a complete specification for `find_items`. When you are done, you should obtain the following results:

```
?- find_items(X,[taco,guacamole],[cheese]).
X = [carnitas_taco, birria_taco, guacamole_taco]

?- find_items(X,[tortilla],[rice]).
X = [al_pastor_burrito]

?- find_items(X,[],[rice,guacamole]).
```

```
X = [al_pastor_burrito, lengua_sopa]

?- find_items(X,[rice,guacamole],[salsa]).
X = [carne_asada_burrito, adobado_burrito, adobado_plate]

?- find_items(X,[guacamole,cheese,salsa],[]).
X = [al_pastor_taco, combo_plate]
```