CS3200: Programming Languages
Homework 7: Prolog

For the following problems, you are required to submit a sample prolog tests that demonstrates each predicate working correctly.

Required Problems

1. In this problem, you're going to do a few list problems to get warmed up:

   (a) Write a prolog predicate `zip(L1,L2,L3)` that is true of the list L3 is obtained by zipping (or shuffling or interleaving) the elements of L1 and L2 Note that L1 and L2 can have different lengths; in this case, just append the remaining amount of which is remaining at the end.

   Here are a few example runs (although I will test more!):

```
?- zip([1,2],[a,b],[1,a,2,b]).
true.

?- zip([1,2],[a,b],X).
X = [1, 2, a, b] ;
X = [1, 2, a, b] ;
X = [1, a, 2, b] ;
X = [1, a, b, 2] ;
X = [a, 1, 2, b] ;
X = [a, 1, b, 2] ;
X = [a, b, 1, 2] ;
X = [a, b, 1, 2] ;
false.

?- zip([1,2],[a,b],[1,2,a,b]).
true.

?- zip(X,[a,b],[1,a,2,b]).
X = [1,2]
true.

?- zip([1,2],X,[1,a,2,b]).
X = [a,b]
true.
```

   (b) Now, write a prolog predicate assoc that implements associative lists (or dictionaries) in prolog. Specifically, write a predicate `assoc(L,X,Y)` so that `assoc([[k1,v1],[k2,v2],...,[kn,vn]],X,Y)` is true if X equals some ki and Y is the corresponding vi in the list.

   Some test runs:

```
?- assoc([[a,1],[b,2],[c,3],[d,4],[b,5]],c,3).
```

```
true.

?- assoc([[a,1],[b,2],[c,3],[d,4],[b,5]],f,Y).
false.

?- assoc([[a,1],[b,2],[c,3],[d,4],[b,5]],X,99).
false.

?- assoc([[a,1],[b,2],[c,3],[d,4],[b,5]],b,Y).
Y = 2 ;
Y = 5
false.

?- assoc([[a,1],[b,2],[c,3],[d,1],[b,5]],X,1).
X = a ;
X = d
false.
```

(c) Now, write a prolog predicate for unions of sets (so without repetition). So `union(L1,L2,L3)` is true if L3 is equal to the list which is the set theoretic union of elements in L1 and L2 - so all elements in L1 or in L2 (or both).

Some test runs:

```
?- union([1,2,3,4],[1,3,5,6],[1,2,3,4,5,6]).
true.

?- union([1,2,3,4],[1,3,5,6],X).
X = [1,2,3,4,5,6].

?- union([1,2,3,4],[1,3,5,6],X).
X = [1,2,3,4,5,6].

?- union([1,2,3],[4,3],[1,2,3]).
false.
```

2. For this problem, we'll implement a database type system in prolog, where data is stored in our predicates. We'll be running a few restaurants that sell various items for a variety of budget and palate combinations. (Note that all of these can be grabbed from the course git repo.)

First, we'll have some initial facts:

```
cost(carne_asada,6).
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).
```

Next, we'll have some menu items, as well as a list of ingredients for each:

```
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, salsa, 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 have some restaurants, each of which has a slightly different menu and list of employees:

```
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]).
```

Now, for this problem, you'll implement the following predicates:

(a) `availableAt(X,Y)`, which is true when menu item X is available at restaurant Y. A sample run:

```
?- 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.

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

(b) `totalCost(X,K)` that is true if the sum of the costs of the ingredients of item X is equal to K. Sample run:

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

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

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

(c) `hasIngredients(X,L)` that is true if the item X has all the ingredients listed in L. Sample run:

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

?- has_ingredients(X,[salsa,guacamole,cheese]).</font><br>
X = al_pastor_taco ;
X = combo_plate.
```

(d) `disgruntled(X)` that is true if person X works at more than one taqueria. Sample run:

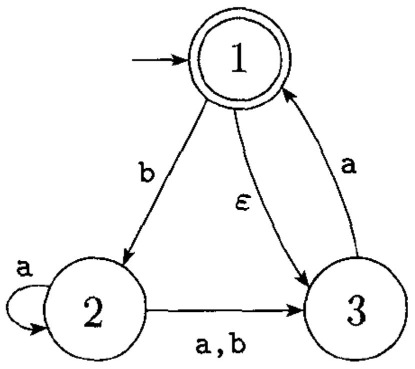
```
?- disgruntled(maria).
true.

?- disgruntled(carlos).
false.

?- disgruntled(X).
X = maria ;
X = miguel .
```

(e) `avoidsIngredients(X,L)` that is true if the item X does not have any of the ingredients listed in L. Sample run:

```
?-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.

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

3. As mentioned in class, Prolog is ideal for simulating NFA's as well as DFA's, since the language will backtrack and explore alternate possible valid parsings if some parsing fails. (See the schedule page to look at some finite state machine examples.)

   Code the specification for the following NFA in prolog as NFA.pl, and test your code by showing the derivations for several strings both in the language and not in the language. Show at least one string which has several possible parsings (although only one may end up in an accept state).



4. Extra Credit: Use prolog to write a set of rules to determine whether a number is prime. A user should be able to enter the goal: prime(10) (for example) and get a yes or no.

   Hints: The base cases are that 1 and 2 are prime. (Technically, 1 isnt prime, but it might help you to assume it is depending on how you decide to write this.) Youll want to have a helper rule that tells you whether a number evenly divides another and handles the recursion - you'll probably use the "is" predicate to check if numbers are equal, and might also want to use the "mod" command to check remainders.