CS 538

Project #3

Programming in Prolog

Due: Friday, May 3, 2002

(Not accepted after Friday, May 10, 2002)

1. Let `L` be a list containing distinct (non-repeating) integers. Moreover, assume `L` contains an *odd* number of values. Assume we want to define a Prolog predicate `median(L,M)` that is true when `M` is the median of `L`. One way to implement this predicate is to sort `L` and then extract the "middle" element of the sorted list. This however is a bit of overkill.

   A simpler implementation can be extracted from the definition of what it means for `M` to be `L`'s median: `M` must occur in `L` and if `M` is used to partition `L` into two sublists, one containing values greater than `M` and the other containing values smaller than `M`, then the resulting sublists must be equal in length.

   Use this definition to implement `median(L,M)`.

   (Remember, you don't have to tell Prolog how to find `M`. Just define the relationship between `M` and `L` and let Prolog do the searching!)

2. Recall that in languages like Scheme, ML and Prolog, sets can be represented as lists. However, unlike lists, the order of values in a set is not significant. Thus both `[1,2,3]` and `[3,2,1]` represent the same set.

   (a). Write facts and rules that define a Prolog relation `setEq(S1,S2)` that tests whether sets `S1` and `S2` (represented as lists) are equal. Two sets are equal if they contain exactly the same members, ignoring ordering. In this part you may assume that sets contain only atomic values (numbers, and symbols). You may also assume that `S1` and `S2` are ground (that is, they are constants or are bound to fixed values). For example

```
setEq([1,2,3], [3,2,1]). ⇒  yes
setEq([1,2], [3,2,1]).   ⇒  no
setEq([curly,larry,moe], [moe,larry,curly]). ⇒  yes
```

(b). In general sets can contain other sets. Extend your solution to part (a) to allow sets to contain other sets. For example,

```
setEq([1,[2,3]], [[3,2],1]). ⇒ yes
setEq([1,2,3], [[3,2],1]).   ⇒ no
setEq([1,2,3], [[1,2,3]]).   ⇒ no
```

3. "Send more money" is a well-known puzzle. Each of the letters, D, E, M, N, O, R, S and Y represents a **different** digit. Moreover, when each letter is mapped to its corresponding digit the equation SEND + MORE = MONEY holds.

   Since there are 8 letters to be solved, we can simply explore the 10*9*...*3 mappings of letters to digits. This could well be too slow. A little insight can simplify things. Clearly, SEND < 9999 and MORE < 9999. Thus MONEY < 19998 and hence M = 1. Now we have SEND + 1ORE = 1ONEY. Again SEND < 9999 and now 1ORE < 1999 so 1ONEY < 11998. Since M is already bound to 1, O must be bound to 0. A little more thought shows that S must be bound to 8 or 9, and that N = E + 1.

   Using these insights to reduce the number of solutions that must be explored, write a Prolog predicate soln([D,E,M,N,O,R,S,Y]) that solves this puzzle by binding the correct digits to each of the variables in the list.

4. Let S be a list representing a set of atomic values (integers or symbols). Let PS be a list of lists. Let the Prolog relation subsets(S,PS) be true when PS represents the power set of S (that is, the set of all subsets of S).

   (a). The most common way to use the subsets relation is to fix S to a known value, and let PS be free (a variable). Then Prolog will set PS to be the power set of S. Give facts and rules for the subsets relation that will allow PS to be correctly computed given S. For example,

   ```
   subsets([],PS).      ⇒ PS = [[]]
   subsets([1],PS).     ⇒ PS = [[],[1]]
   subsets([a,b],PS).   ⇒ PS = [[],[b],[a],[a,b]]
   ```

   (b). Another way to the subsets relation might be used is to fix both S and PS to known values. Then the relation would test whether PS really is a power set of S. For example,

   ```
   subsets([],[[]]).       ⇒ yes
   subsets([1],[[],[1]]).  ⇒ yes
   subsets([1],[[1],[]]).  ⇒ yes
   subsets([1],[[1]]).     ⇒ no
   ```

   Does your solution to part (a) correctly handle the case in which both S and PS are fixed to known values (that is, **ground**)? If so, you are done with this part. If not, extend your solution to part (a) to handle this case. Be sure to note that the set values that are bound to PS need not be in the same order as the values that would be computed by subsets given S (though they must represent the same set).

(c). It may also happen that the `subsets` relation is used when both S and PS are free (that is variables, not bound to any values). In this case the relation would generate pairs of possible values for S and PS, starting with the simplest (the empty list), and then the list with one element, then two elements, etc. Since the actual list elements that will be used are unknown, Prolog generates "anonymous variables" of the form `_integer` (e.g., `_123`). For example, in this case we might generate

```
subsets(S,PS). ⇒  S = [], PS = [[]] ;
    S = [_737], PS = [[],[_737]] ;
    S = [_737,_739], PS = [[],[_739],[_737],[_737,_739]]
```

Does your solution to part (b) correctly handle the case in which both S and PS are not fixed (that is, **free**)? If so, you are done with this part. If not, extend your solution to part (b) to handle this case.

(d). Finally, happen that the `subsets` relation is used when S is free, but PS is bound to fixed value. In this case the relation must generate a value for S such that PS is S's power set. It may happen that no such S exists, in which case the relation should answer `no`. For example,

```
subsets(S,[[]]).     ⇒  S = []
subsets(S,[[],[a]]). ⇒  S = [a]
subsets(S,[[a],[]]). ⇒  S = [a]
subsets(S,[[a]]).    ⇒  no
```

Does your solution to part (c) correctly handle the case in which S is free and PS is fixed? If so, you are done with this part. If not, extend your solution to part (c) to handle this case. Note that since that there is no direct way to compute S from PS, you can use your solution to part (c) to "guess" possible solutions and match them with the known value of PS. Be careful though; in the case that no solution exists, you must be sure to stop guessing possible solutions when they become too large to possibly match the value of PS.

What to Hand In

Submit your solution electronically by placing eight files named `q1.pro`, `q2a.pro`, `q2b.pro`, `q3.pro`, `q4a.pro`, `q4b.pro`, `q4c.pro` and `q4d.pro` in your handin subdirectory: `~cs538-1/public/handin/proj3/your-login`. Each file should contain a comment of the form

```
% your name, your login
```