CS2203 / CSX3004 Programming Languages

Week 13 Worksheet

For the question 1 to 5, you should implement sets as lists, where each element of a set appears exactly once in its list , but in no particular order. Do not assume you can sort the lists. You can assume that input lists have no duplicate elements, and your solution must guarantee that output lists have no duplicate elements.

1. Define the `isMember` predicate so that `isMember(X,Y)` says that element x is a member of set Y. Do not use the predefined list predicates.

2. Define the `isUnion` predicate so that `isUnion(X,Y,Z)` says that the union of X and Y is Z. Do not use the predefined list predicates. Your predicate may choose a fixed order for Z. If you query `isUnion([1,2],[3],Z)`, it should find a binding for Z , but it does not need to succeed on both `isUnion([1],[2],[1,2])` and `isUnion([1,2], [2],[2,1])` . Your predicate does not need to work well when X or Y are unbound variables.

3. Define the `isIntersection` predicate so that `isIntersection(X,Y,Z)` says that the intersection of X and Y is Z. Do not use the predefined list predicates. Your predicate may choose a fixed order for Z. Your predicate need not work well when X or Y are unbound variables.

4. Define the `isEqual` predicate so that `isEqual(X,Y)` says that the sets X and Y are equal. Two sets are equal if they have exactly the same elements, regardless of the order in which those elements are represented in the set. Your predicate need not work well when X or Y are unbound variables.

5. Define the `isDifference` predicate so `isDifference(X,Y,Z)` says that the set Z contains the elements of X that do not also appear in Y. But unlike questions 1 – 4, you must make a relation that works no matter what order Z is in. Your predicate need not work well when X or Y are unbound variables.

6. Refer to the man-wolf-goat-cabbage solution below.

```
change(e,w).
change(w,e).

move([X,X,Goat,Cabbage], wolf, [Y,Y,Goat,Cabbage]) :-
    change(X,Y).
move([X,Wolf,X,Cabbage], goat, [Y,Wolf,Y,Cabbage]) :-
    change(X,Y).
move([X,Wolf,Goat,X], cabbage, [Y,Wolf,Goat,Y]) :-
    change(X,Y).
move([X,Wolf,Goat,C], nothing, [Y,Wolf,Goat,C]) :-
    change(X,Y).

oneEq(X,X,_).
oneEq(X,_,X).

safe([Man,Wolf,Goat,Cabbage]) :-
    oneEq(Man,Goat,Wolf),
    oneEq(Man,Goat,Cabbage).

solution([e,e,e,e],[]).
solution(Config,[Move|Rest]) :-
    move(Config,Move,NextConfig),
    safe(NextConfig),
    solution(NextConfig,Rest).
```

Try this query:

```
length(X, 7), solution([w,w,w,w], X).
```

Use the semicolon after each solution to make it print them all; that is, keep hitting the semicolon until it finally says `false`. As you will see, it finds the same solution more than once.

a. How many solutions does it print, and how many of them are distinct?
b. Modify the code to make it find only distinct solutions.