CMSC330 Spring 2014 Practice Problems 7 Solutions

1. Given the following set of clauses:

```
eats(alf, cats).
eats(mary, cheese).
eats(mary, bread).
```

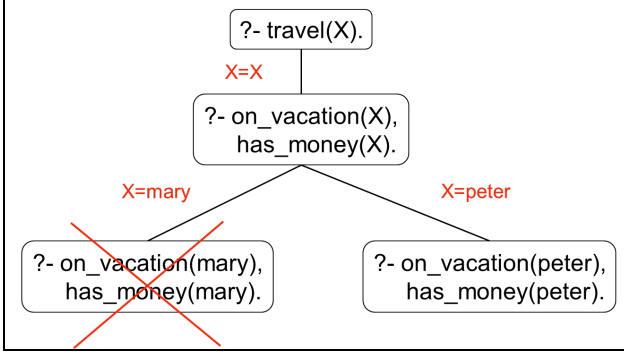
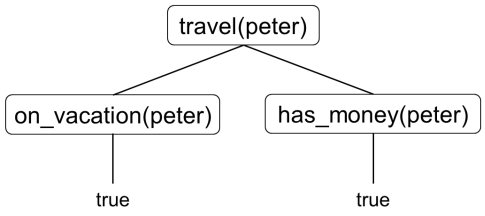
List all answers generated for the following queries

a.	?- eats(mary,cheese).	**Answers =**	**true.**
b.	?- eats(mary,cats).		**false.**
c.	?- eats(mary,X).		**X=cheese; X=bread.**
d.	?- eats(X,cats).		**X=alf.**
e.	?- eats(X,alf).		**false.**
f.	?- eats(X,Y).		**X=alf,Y=cats;** **X=mary,Y=cheese;** **X=mary,Y=bread.**

2. Given the following set of clauses:

```
travel(X) :- on_vacation(X), has_money(X).
on_vacation(mary).
on_vacation(peter).
has_money(peter).
```

   a. List all answers generated for ?- on_vacation(X).
      X=mary;
      X=peter.
   b. List all answers generated for ?- travel(X).
      X=peter.
   c. Draw the Prolog search tree for travel(X).
   d. Draw the Prolog clause tree for travel(peter).

Search Tree	Clause Tree
	

3. Given the following set of clauses:

```
foo([X], X).
foo ([_|T],X) :- foo (T,X).
```

a.	?- foo([1],1).	**Answers =**	**true.**
b.	?- foo([3],1).		**false.**
c.	?- foo([1,2,3],1).		**false.**
d.	?- foo([1,2,3],3).		**true.**
e.	?- foo([1,2,3],X).		**X=3.**
f.	?- foo([X,2,3],1).		**false.**
g.	?- foo([1,2,X],1).		**X=1.**
h.	?- foo([1,2|X],1).		**X=[1].**

4. Given a set of facts of form parent(name1,name2) where (name1 is the parent of name2):

   a. Define a predicate sibling(X,Y) which holds iff X and Y are siblings.
      sibling(X,Y) :- parent(Z,X), parent(Z,Y), not(X=Y).

   b. Define a predicate cousin(X,Y) which holds iff X and Y are cousins.
      cousin(X,Y) :- parent(Z,X), parent(W,Y), sibling(Z,W).

   c. Define a predicate grandchild(X,Y) which holds iff X is a grandchild of Y.
      grandchild(X,Y) :- parent(Z,X), parent(Y,Z).

   d. Define a predicate descendent(X,Y) which holds iff X is a descendent of Y.
      descendent(X,Y) :- parent(Y,X).
      descendent(X,Y) :- parent(Z,X), descendent(Z,Y).

5. Consider the following genealogical tree (and its graphical representation):

Genealogical Tree	Graphic Representation
parent(a,b). parent(a,c). parent(b,d). parent(b,e). parent(c,f).	a / \ b c / \ | d e f

List all answers generated by your definitions for the following queries:

a. ?- sibling (X,Y).
   X=b, Y=c;
   X=c, Y=b;
   X=d, Y=e;
   X=e, Y=d.

b. ?- cousin(X,Y).
   X=d, Y=f;
   X=e, Y=f;
   X=f, Y=d;
   X=f, Y=e.

c. ?- grandchild(X,Y).

   X=d, Y=a;
   X=e, Y=a;
   X=f, Y=a.

d. ?- descendent(X,Y).

   X=b, Y=a;
   X=c, Y=a;
   X=d, Y=b;
   X=e, Y=b;
   X=f, Y=c;
   X=d, Y=a;
   X=e, Y=a;
   X=f, Y=a.

6. Given the following set of clauses:

```
jedi(luke).
jedi(yoda).
sith(vader).
sith(maul).
fight(X,Y) :- jedi(X), sith(Y).
fight(X,Y) :- sith(X), X\=Y, sith(Y).
fight(X,Y) :- jedi(X), !, jedi(Y).
```

List all answers generated for the following queries

a.	?- fight(luke,yoda).	**Answers =**	**true.**
b.	?- fight(luke,vader).		**true.**
c.	?- fight(vader,yoda).		**false.**
d.	?- fight(vader,maul).		**true.**
e.	?- fight(luke,X).		**X=vader; X=maul; X=luke; X=yoda.**
f.	?- fight(vader,X).		**false.**
g.	?- fight(X,yoda).		**X=luke.**
h.	?- fight(X,maul).		**X=luke; X=yoda; X=vader.**
i.	?- fight(X,Y).		**X=luke, Y=vader;** **X=luke, Y=maul;** **X=yoda, Y=vader;** **X=yoda, Y=maul;** **X=luke, Y=luke;** **X=luke, Y=yoda.**

7. Given the following set of clauses, what is the output for foo([3,1,2,0],R), if any?

Part	Code	Answer
A	foo([H|_], H). foo([_|T],X) :- foo(T,X).	**R=3; R=1; R=2; R=0.**
B	foo([_|T],X) :- foo(T,X). foo([H|_], H).	**R=0; R=2; R=1; R=3.**
C	foo([H|_], H) :- H > 1. foo([_|T],X) :- foo(T,X).	**R=3; R=2.**
D	foo([_|T],X) :- foo(T,X). foo([H|_], H) :- H > 1.	**R=2; R=3.**
E	foo([H|_], H) :- H > 1, !. foo([_|T],X) :- foo(T,X).	**R=3.**
F	foo([_|T],X) :- foo(T,X). foo([H|_], H) :- H > 1, !.	**R=2; R=3.**
G	foo([H|_], H). foo([_|T],X) :- X > 1, foo(T,X).	**R=3; error (X not instantiated).**
H	foo([_|T],X) :- X > 1, foo(T,X). foo([H|_], H).	**error (X not instantiated).**
I	foo([H|_], H). foo([_|T],X) :- foo(T,X), X > 1.	**R=3; R=2.**
J	foo([_|T],X) :- foo(T,X), X > 1. foo([H|_], H).	**R=2; R=3.**
K	foo([H|_], H). foo([_|T],X) :- foo(T,X), !, X > 1.	**R=3.**
L	foo([_|T],X) :- foo(T,X), !, X > 1. foo([H|_], H).	**false.**

8. Define a predicate reverse(L,K) which holds if and only if the list K is the reverse of the list L.

Naive, inefficent (quadratic) solution:

```
naive_reverse([ ],[ ]).
naive_reverse([X|L],K) :- naive_reverse(L,M), concat(M,[X],K).
```

Fast (linear), tail-recursive solution:

```
fast_reverse(L,K) :- revHelper(L,K,[ ]).
revHelper([ ],K,K).
revHelper([X|L],K,M) :- revHelper(L,K,[X|M]),
```

9. Define a predicate add_up_list(L,X) which, given a list of integers L, returns a list of integers in which each element is the sum of all the elements in L up to the same position. Example:

```
?- add_up_list([1,2,3,4],X).
X = [1,3,6,10].
```

add_up_list(L,K) :- addHelper(L,K,0).
addHelper([ ],[ ],_).
addHelper([X|L],[Y|K],Z) :- Y is Z+X, addHelper(L,K,Y).

10. Consider the following Prolog predicate definition

```
remove_at(X,[X|Xs],1,Xs).
remove_at(X,[Y|Xs],K,[Y|Ys]) :- K1 is K - 1, remove_at(X,Xs,K1,Ys).
```

It works for queries like

```
?- remove_at(X,[a,b,c,d],2,R).
X = b,
R = [a,c,d].
```

However, it throws an exception for queries like

```
?- remove_at(c,[a,b,c,d],V,R).
ERROR: remove_at/4: Arguments are not sufficiently instantiated
```

Modify the predicate definition to make it work for the above query.

remove_at(X,[X|Xs],1,Xs).
remove_at(X,[Y|Xs],K,[Y|Ys]) :- remove_at(X,Xs,K1,Ys), K is K1 + 1.

11. Write the prolog predicate flatten(L,R) that flattens a list of lists in L to a single list R.

The equivalent OCaml function is given by

```
let rec flatten l = match l with
[ ] | [[ ]] -> [ ]
| [ ]::t -> flatten t
| [h]::t -> h::flatten t
| ((h1::t1)::t) -> h1::flatten(t1::t);;
```

flatten([ ],[ ]) :- !.
flatten([[ ]],[ ]) :- !.
flatten([[ ]|T2],T) :- flatten(T2,T), !.
flatten([[H]|T2],[H|T]) :- flatten(T2,T), !.
flatten([[H1|T1]|T2],[H1|T]) :- flatten([T1|T2],T).