```
student of(X,T):-follows(X,C), teaches(T,C).
follows(paul,computer science).
follows(paul, expert systems).
follows(maria,ai techniques).
teaches(adrian, expert_systems).
teaches(peter,ai techniques).
teaches(peter,computer science).
                        ?-student of(S,peter)
                    :-follows(S,C),teaches(peter,C)
                                      :-teaches(peter,ai techniques)
:-teaches(peter,computer science)
                    :-teaches(peter,expert systems)
             []
                                                     []
```

```
?-brother of(peter,B)
      brother of(X,Y):-brother of(Y,X).
                                                       :-brother_of(B,peter)
      brother_of(paul,peter).
                                                 :-brother_of(peter,B)
                                                                       :-brother_of(B,peter)
     ?-brother of(paul,B)
                                                             []
        :-brother_of(paul,Z),brother_of(Z,B)
:-brother_of(peter,B) :-brother_of(paul,Z1),brother_of(Z1,Z),brother_of(Z,B)
                                       brother_of(paul,peter).
  :-brother_of(peter,Z),brother_of(Z,B)
                                       brother_of(peter,adrian).
                                       brother of(X,Y):-brother of(X,Z),
                                                           brother_of(Z,Y).
```

Infinite SLD-trees

```
list([H|T]):-list(T).
                             ?-list(L).
    ?-list(L)
                                L = [];
                                L = [A];
          :-list(T1)
                                L = [A,B];
L = []
                :-list(T2)
     L = [A]
                      :-list(T3)
           L = [A,B]
```

list([]).

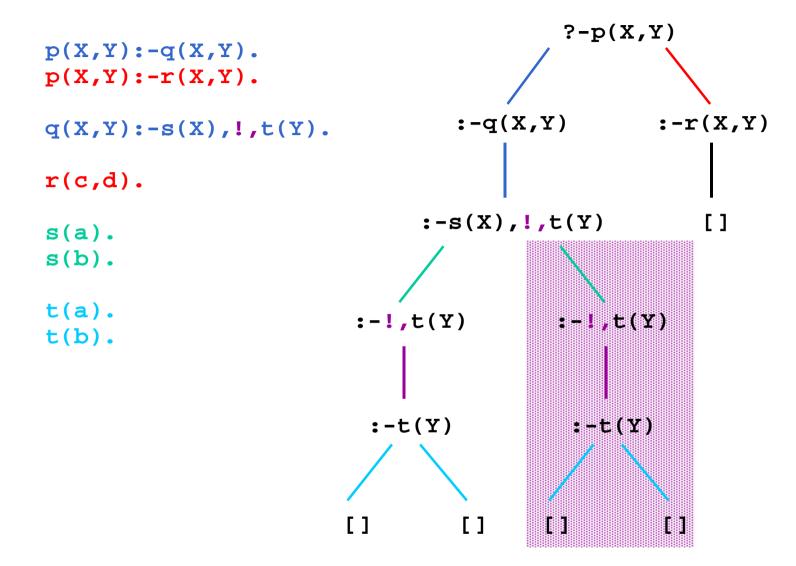
Exercise 3.2

```
plist([]).
                                          plist([H|T]):-
        ?-plist(L)
                                                p(H),plist(T).
                                          p(1). p(2).
               :-p(H1),plist(T1)
                                                ?-plist(L).
    L = []
                                                  L=[];
                                                  L=[1];
                                 :-plist(T1)
     :-plist(T1)
                                                  L=[1,1];
        :-p(H1),plist(T1)
                                      :-p(H1),plist(T1)
                                []
L = [1]
                   :-plist(T1) :-plist(T1)
                                                  :-plist(T1)
    :-plist(T1)
              L = [1,2] \cdot L = [2,1]
 L = [1,1]
                                            L = [2,2]
```



```
?-parent(john,C)
  parent(X,Y):-father(X,Y).
  parent(X,Y):-mother(X,Y).
  father(john,paul).
                                 :-father(john,C) :-mother(john,C)
  mother(mary,paul).
                                        []
      ?-parent(john,C)
:-father(john,C),! :-mother(john,C)
                                    parent(X,Y):-father(X,Y),!.
                                    parent(X,Y):-mother(X,Y).
                                    father(john,paul).
      :-!
                                    mother(mary,paul).
      []
```

Pruning by means of cut



The effect of cut

```
?-parent(P,paul)
 parent(X,Y):-father(X,Y),!.
                                :-father(P,paul),!
                                                   :-mother(P,paul)
 parent(X,Y):-mother(X,Y).
 father(john,paul).
 mother(mary,paul).
                                                      : -!
       ?-parent(john,C)
                                   parent(X,Y):-father(X,Y),!.
:-father(john,C),!
                 :-mother(john,C)
                                   parent(X,Y):-mother(X,Y).
                                   father(john,paul).
                                   father(john,peter).
                                   mother(mary, paul).
                                   mother(mary, peter).
    []
```

Pruning away success branches

```
?-likes(A,B)
   likes(peter,Y):-friendly(Y).
   likes(T,S):-student of(S,T).
                                       :-friendly(B) :-student_of(B,A)
   student of(maria, peter).
   student of(paul,peter).
                                             Γ1
                                                      Г٦
                                                                Г٦
   friendly(maria).
                                           A=peter
                                                    A=peter
                                                               A=peter
                                           B=maria
                                                    B=maria
                                                               B=paul
likes(peter,Y):-!,friendly(Y). likes(T,S):-student of(S,T),!.
        ?-likes(A,B)
                                               ?-likes(A,B)
:-!,friendly(B) :-student of(B,A)
                                                    :-student of(B,A),!
                                       :-friendly(B)
 :-friendly(B)
                E 3
                          П
                                            Г1
              A=peter
                        A=peter
                                          A=peter
              B=maria
                        B=paul
                                          B=maria
                                                      Г٦
                                                                f 1
                                                    A=peter
                                                              A=peter
    A=peter
                                                    B=maria
                                                              B=paul
    B=maria
```

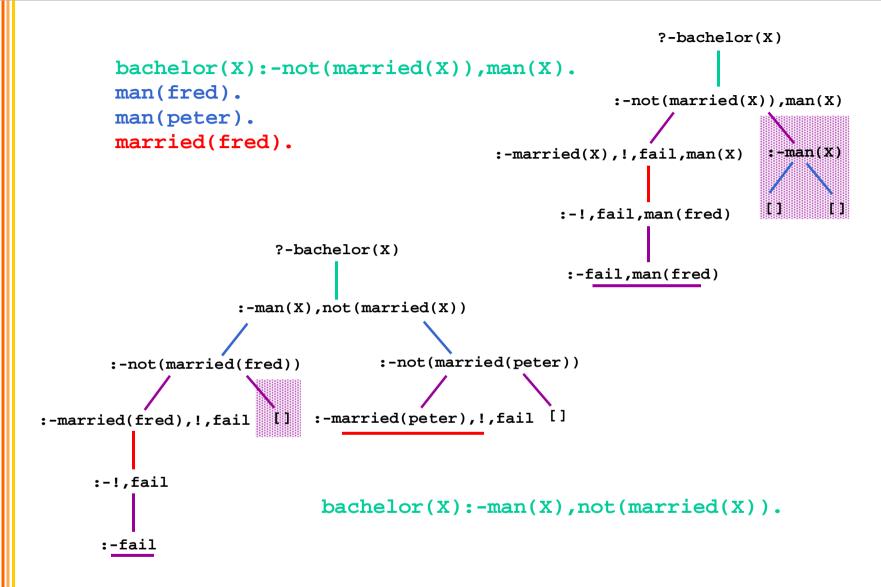
Exercise 3.3

?-p

```
p:-q,r.
p:-not(q),s.
s.
                                          :-not(q),s
                               :-q,r
not(Goal):-Goal,!,fail.
not(Goal).
                                  :-q,!,fail,s
                                                  :-s
                                                   []
                          p:-q,!,r.
:-q,!,r
                          p:-s.
                          S.
               []
```

```
?-p
p:-not(q),r.
p:-q.
q.
                                                    :-q
                                   :-not(q),r
r.
not(Goal):-Goal,!,fail.
not(Goal).
                            :-q,!,fail,r
                                             : -r
                                                    []
                             :-!,fail,r
                                              :-fail,r
```

:-not(q) fails



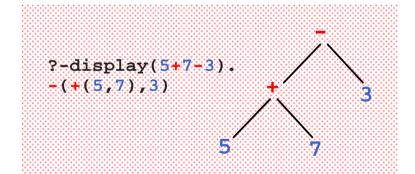
Prolog's not is unsound

```
p:-q,r,if_s_then_t_else_u.
p:-q,r,s,!,t.
                         if s then t else u:-s,!,t.
p:-q,r,u.
                         if s then t else u:-u.
q.
r.
                         q.
u.
                         r.
                                        ?-p
                         u.
                              :-q,r,if_s_then_t_else_u
:-q,r,s,!,t :-q,r,u
                               :-r,if_s_then_t_else_u
:-r,s,!,t
            :-r,u
                                :if s then t else u
 :-s,!,t
              :-u
                                 :-s,!,t
                                              :-u
               []
```



?-
$$X = 9$$

?-
$$X is 5*3+7/2$$
.
 $X = 18.5$



$$?-X = 5+7-3.$$

 $X = 5+7-3$

$$?-9 = 5+7-3.$$

$$?-9 = X+7-3.$$

$$?-X = Y+7-3.$$

 $X = _947+7-3$
 $Y = _947$

```
zero(A,B,C,X):-
  X is (-B + sqrt(B*B - 4*A*C)) / 2*A.
zero(A,B,C,X):-
  X is (-B - sqrt(B*B - 4*A*C)) / 2*A.
```

Prolog does not check for circular bindings

```
?-X = f(X).
X = f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f)
Error: term being written is too deep
```

This may lead to unsound behaviour

```
strange:-X=f(X).
?-strange.
Yes
```

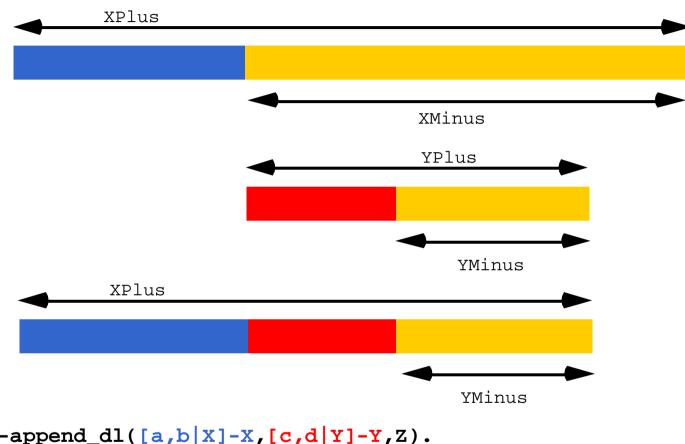
Occur check

```
length([H|T],N1):-length(T,M1),
?-length([a,b,c],N)
                                          N1 is M1+1
                            \{H->a, T->[b,c], N1->N\}
                        length([H|T],N2):-length(T,M2),
 :-length([b,c],M1),
                                          N2 is M2+1
   N is M1+1
                             \{H->b, T->[c], N2->M1\}
                       length([H|T],N3):-length(T,M3),
  :-length([c],M2),
                                          N3 is M3+1
    M1 is M2+1,
    N is M1+1
                             \{H->c, T->[], N3->M2\}
   :-length([],M3),
    M2 is M3+1,
                                  length([],0)
    M1 is M2+1,
    N is M1+1
                            \{M3->0\}
   :-M2 is 0+1,
     M1 is M2+1,
     N is M1+1
                                              length([],0).
             \{M2->1\}
                                              length([H|T],N):-
    :-M1 is 1+1,
     N is M1+1
                                                        length(T,M),
             \{M1->2\}
                                                        N is M+1.
     :-N is 2+1
             \{N->3\}
         []
```

```
length_acc([H|T],N10,N1):-N11 is N10+1,
?-length acc([a,b,c],0,N)
                                                     length acc(T,N11,N1)
                         \{H->a, T->[b,c], N10->0, N1->N\}
:-N11 is 0+1.
  length_acc([b,c],N11,N)
          {N11->1}
                            length_acc([H|T],N20,N2):-N21 is N20+1,
:-length_acc([b,c],1,N)
                                                     length acc(T,N21,N2)
                         \{H->b, T->[c], N20->1, N2->N\}
:-N21 is 1+1,
  length acc([c],N21,N)
           {N21->2}
:-length acc([c],2,N)
                           length_acc([H|T],N30,N3):-N31 is N30+1,
                                                     length acc(T,N31,N3)
                         \{H->c, T->[], N30->2, N3->N\}
:-N31 is 2+1,
  length_acc([],N31,N)
                                               length acc([],N,N).
           {N31->3}
                                               length_acc([H|T],N0,N):-
:-length_acc([],3,N)
                        length_acc([],N,N)
                                                        N1 is N0+1,
            \{N->3\}
                                                        length acc(T,N1,N).
          []
```

Exercise 3.11

append_dl(XPlus-XMinus,YPlus-YMinus,XPlus-YMinus):-XMinus=YPlus.



?-append_dl([a,b|X]-X,[c,d|Y]-Y,Z). X = [c,d|Y], Z = [a,b,c,d|Y]-Y

Difference lists

```
?-findall(C,parent(john,C),L).
parent(john, peter).
parent(john,paul).
                        L = [peter,paul,mary]
parent(john,mary).
parent(mick,davy).
parent(mick,dee).
                       ?-findall(C,parent(P,C),L).
parent(mick,dozy).
                        L = [peter,paul,mary,davy,dee,dozy]
?-bagof(C,parent(P,C),L).
 P = john
  L = [peter,paul,mary];
  P = mick
 L = [davy,dee,dozy]
?-bagof(C,P^parent(P,C),L).
 L = [peter,paul,mary,davy,dee,dozy]
```

Second-order predicates



```
prove(true):-!.
prove((A,B)):-!,
   prove(A),
   prove(B).
prove(A):-
   /* not A=true, not A=(X,Y) */
   clause(A,B),
   prove(B).
                         prove_r(true):-!.
                         prove_r((A,B)):-!,
                             clause(A,C),
                             conj_append(C,B,D),
                             prove_r(D).
                         prove_r(A):-
                             /* not A=true, not A=(X,Y) */
                             clause(A,B),
                             prove_r(B).
```

Prolog meta-interpreters

KNOWLEDGE

REASONING

META-LEVEL

clause(p(X), q(X)). clause(q(a), true). ?-prove(p(X)). X=a

OBJECT-LEVEL

p(X):-q(X). q(a).

unification

?-p(X). X=a

Write down declarative specification

```
% partition(L,N,Littles,Bigs) <- Littles contains numbers
% in L smaller than N,
Bigs contains the rest</pre>
```

Identify recursion and 'output' arguments

Write down skeleton

```
partition([],N,[],[]).
partition([Head|Tail],N,?Littles,?Bigs):-
    /* do something with Head */
    partition(Tail,N,Littles,Bigs).
```

Logic programming methodology

Complete bodies

```
partition([],N,[],[]).
partition([Head|Tail],N,?Littles,?Bigs):-
    Head < N,
    partition(Tail,N,Littles,Bigs),
    ?Littles = [Head|Littles],?Bigs = Bigs.
partition([Head|Tail],N,?Littles,?Bigs):-
    Head >= N,
    partition(Tail,N,Littles,Bigs),
    ?Littles = Littles,?Bigs = [Head |Bigs].
```

Fill in 'output' arguments

```
partition([],N,[],[]).
partition([Head|Tail],N,[Head|Littles],Bigs):-
    Head < N,
    partition(Tail,N,Littles,Bigs).
partition([Head|Tail],N,Littles,[Head|Bigs]):-
    Head >= N,
    partition(Tail,N,Littles,Bigs).
```

Methodology

Write down declarative specification

```
% sort(L,S) <- S is a sorted permutation of list L
```

Write down skeleton

```
sort([],[]).
sort([Head|Tail],?Sorted):-
   /* do something with Head */
sort(Tail,Sorted).
```

Complete body (auxiliary predicate needed)

```
sort([],[]).
sort([Head|Tail],WholeSorted):-
    sort(Tail,Sorted),
    insert(Head,Sorted,WholeSorted).
```

Writing a sort predicate

Write down declarative specification

```
% insert(X,In,Out) <- In is a sorted list, Out is In
with X inserted in the proper place</pre>
```

Write down skeleton

```
insert(X,[],?Inserted).
insert(X,[Head|Tail],?Inserted):-
   /* do something with Head */
   insert(X,Tail,Inserted).
```

Writing an insert predicate

Complete bodies

```
insert(X,[],?Inserted):-?Inserted=[X].
insert(X,[Head|Tail],?Inserted):-
   X > Head,
   insert(X,Tail,Inserted),
   ?Inserted = [Head|Inserted].
insert(X,[Head|Tail],?Inserted):-
   X =< Head,
   ?Inserted = [X,Head|Tail].</pre>
```

Fill in 'output' arguments

```
insert(X,[],[X]).
insert(X,[Head|Tail],[X,Head|Tail]):-
   X =< Head.
insert(X,[Head|Tail],[Head|Inserted]):-
   X > Head,
   insert(X,Tail,Inserted).
```

