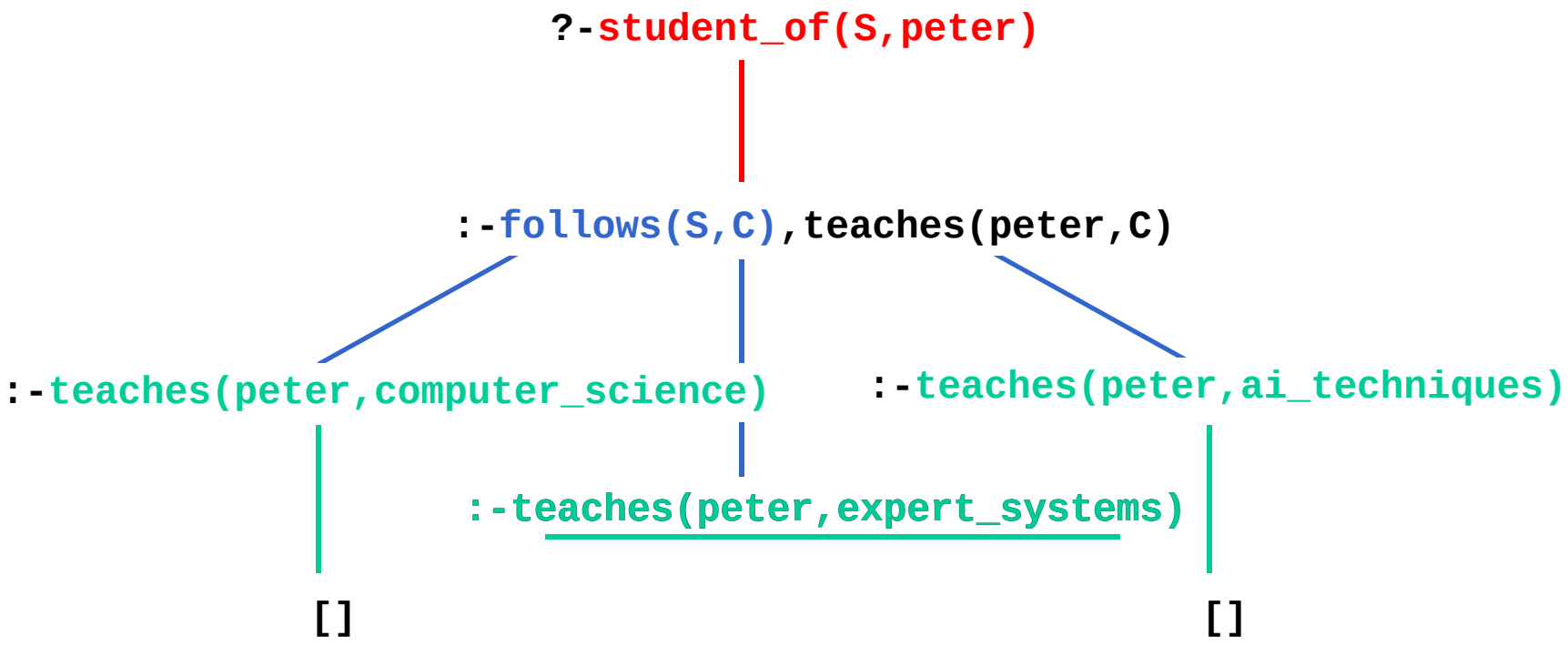
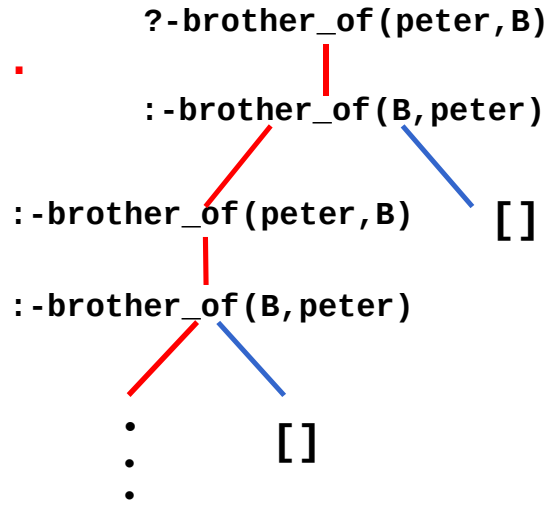
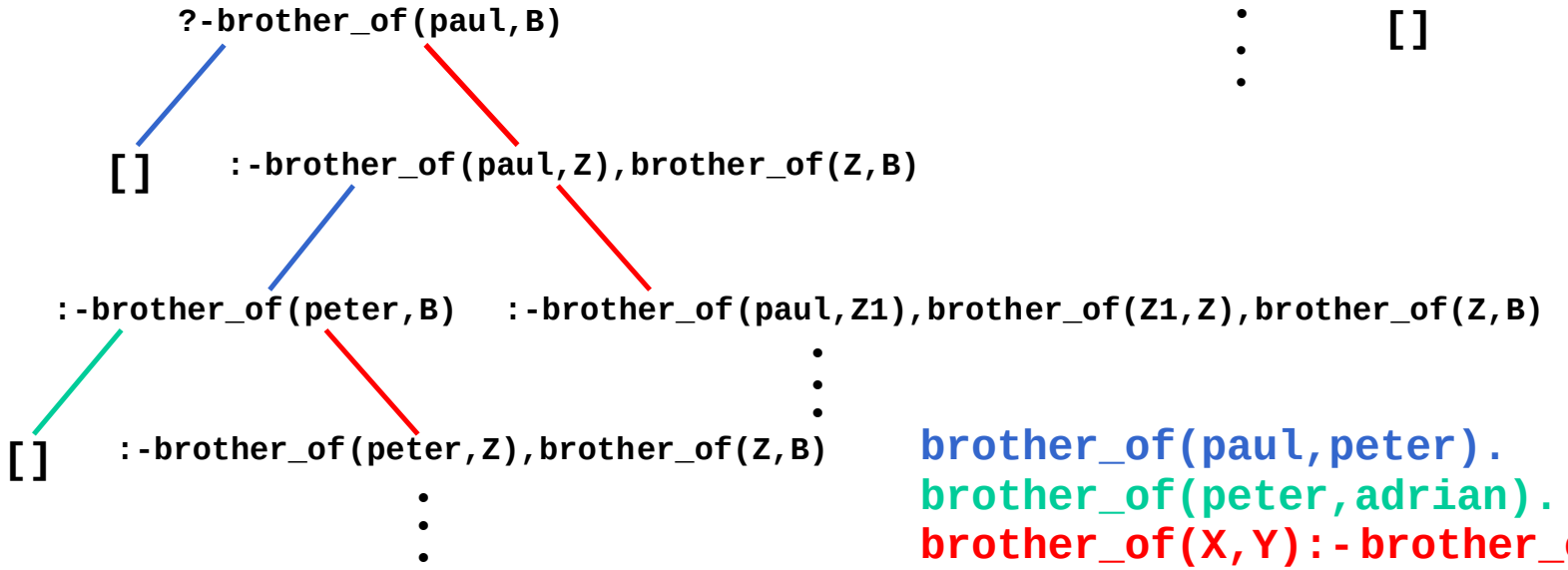
```
student_of(X,T):-follows(X,C),teaches(T,C).
follows(paul,computer_science).
follows(paul,expert_systems).
follows(maria,ai_techniques).
teaches(adrian,expert_systems).
teaches(peter,ai_techniques).
teaches(peter,computer_science).
```



SLD-tree

```
brother_of(X,Y):-brother_of(Y,X).
brother_of(paul,peter).
```

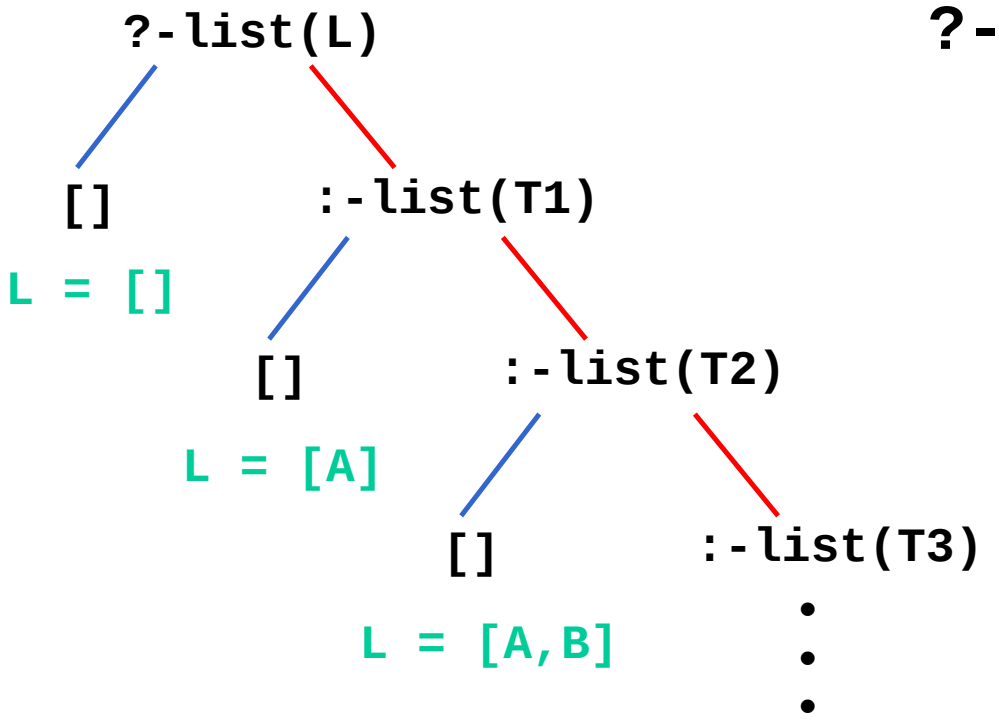





```
brother_of(paul,peter).
brother_of(peter,adrian).
brother_of(X,Y):-brother_of(X,Z),
                 brother_of(Z,Y).
```

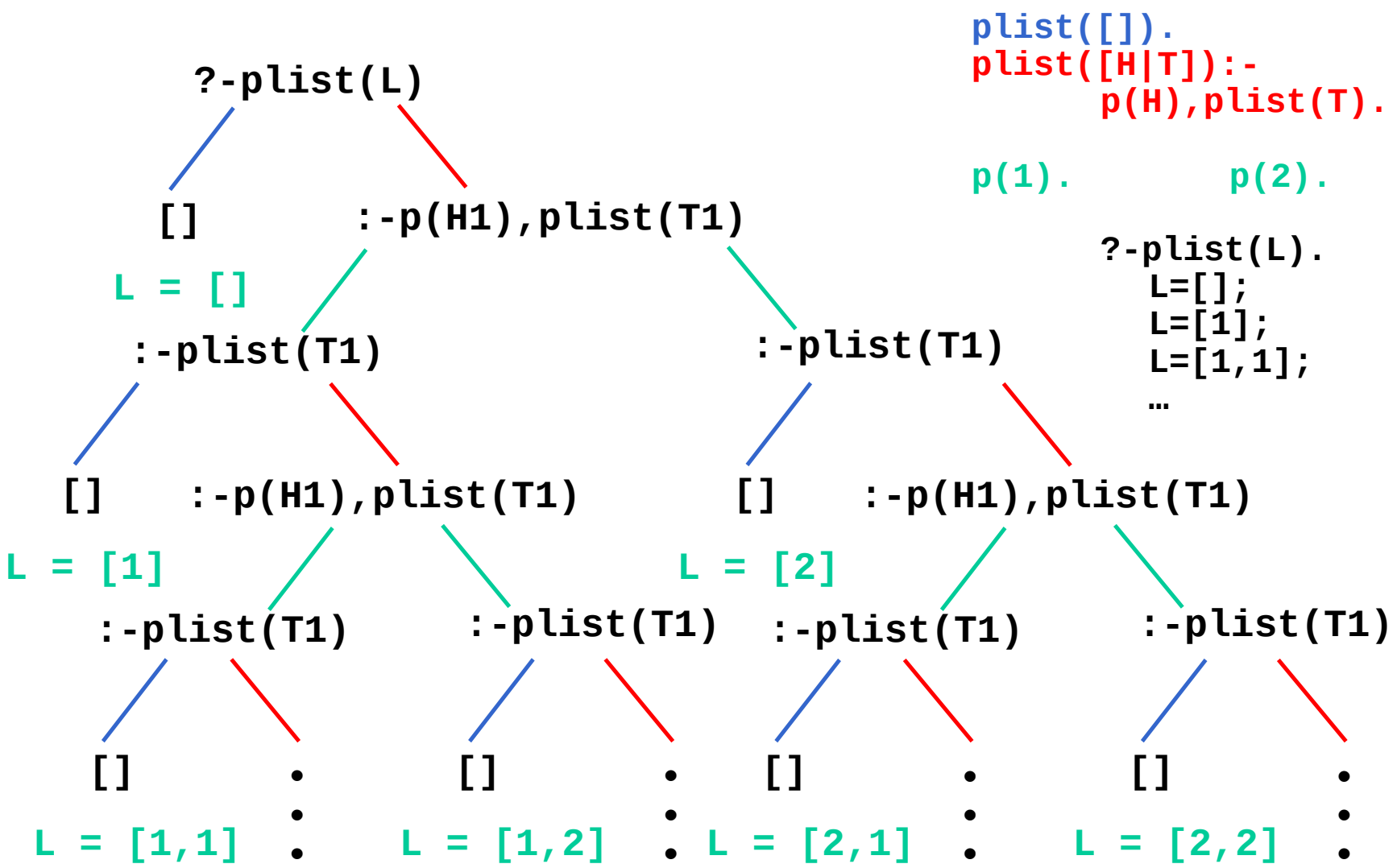
Infinite SLD-trees

```
list([]).
list([H|T]):-list(T).
```



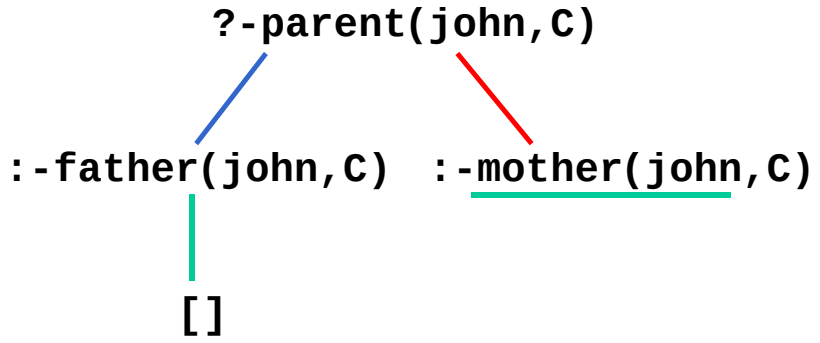

```
?-list(L).
   L = [];
   L = [A];
   L = [A,B];
   …
```

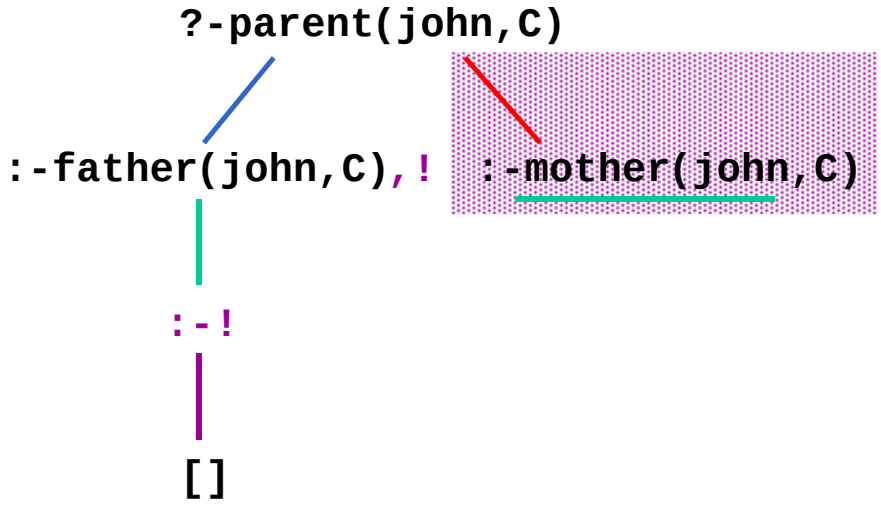
Exercise 3.2



Depth-first search

```
parent(X,Y):-father(X,Y).
parent(X,Y):-mother(X,Y).
father(john,paul).
mother(mary,paul).
```

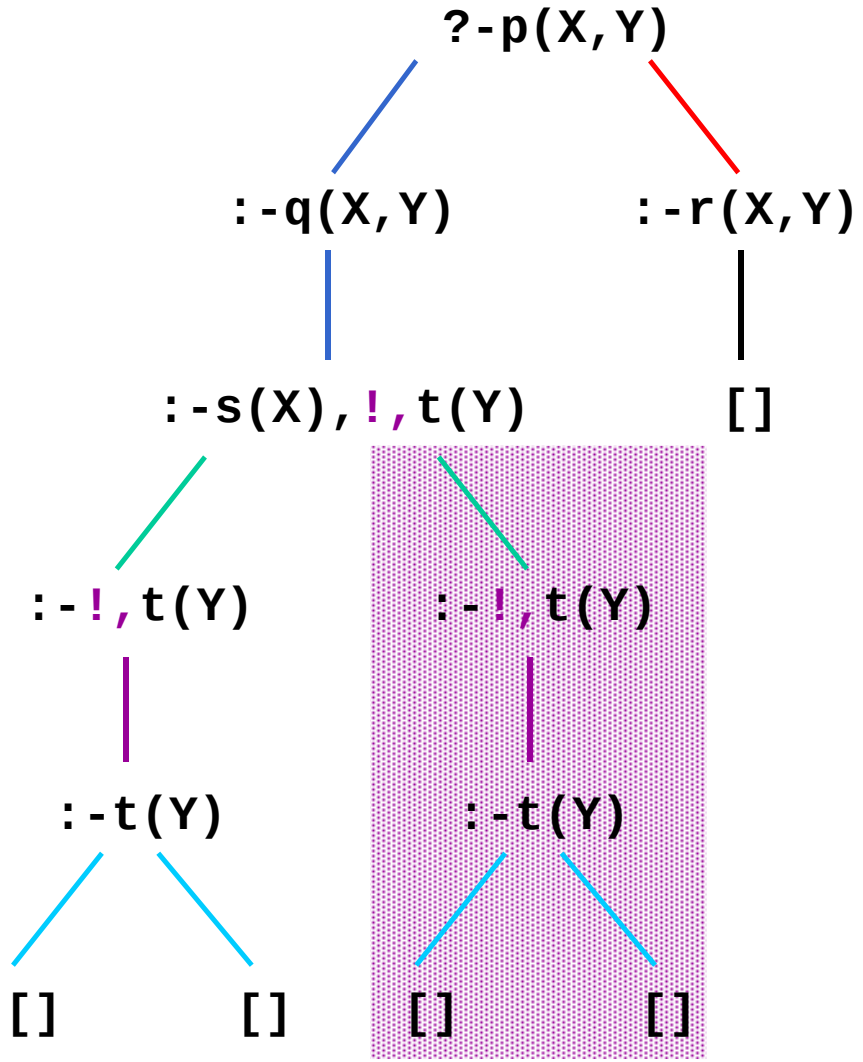





```
parent(X,Y):-father(X,Y),!.
parent(X,Y):-mother(X,Y).
father(john,paul).
mother(mary,paul).
```

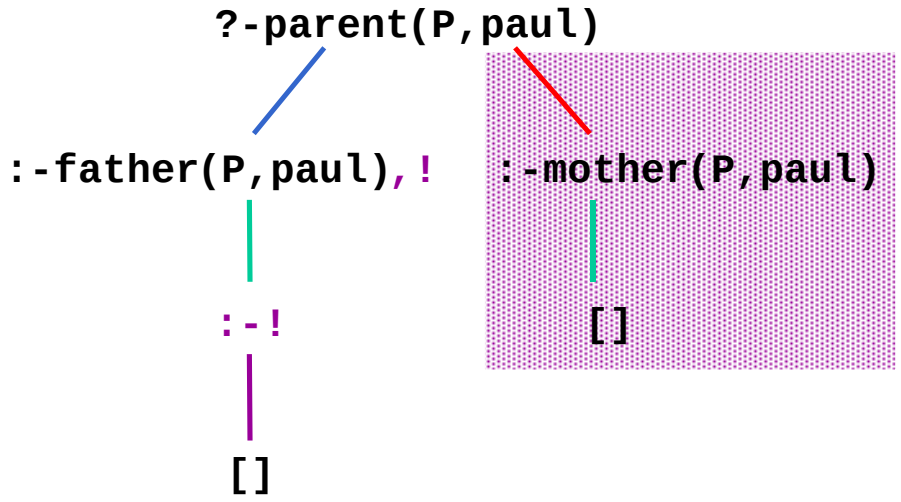
Pruning by means of cut

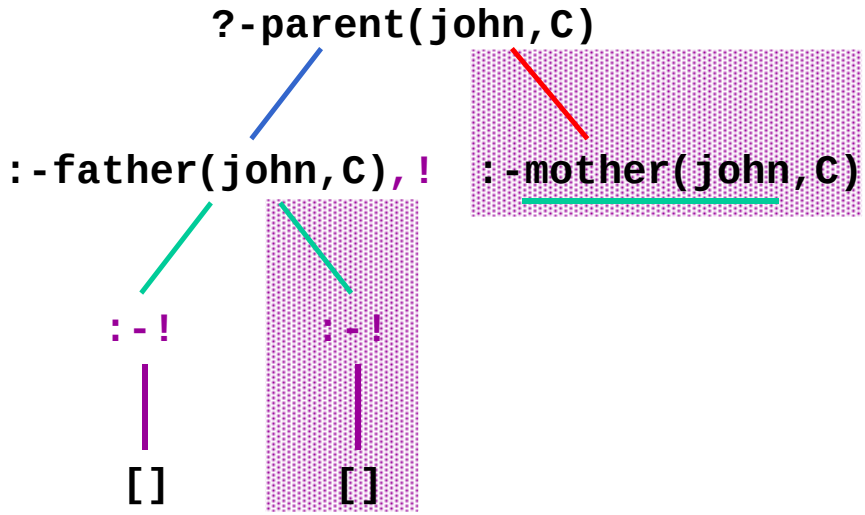
```
p(X,Y):-q(X,Y).
p(X,Y):-r(X,Y).

q(X,Y):-s(X),!,t(Y).

r(c,d).

s(a).
s(b).

t(a).
t(b).
```



The effect of cut

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

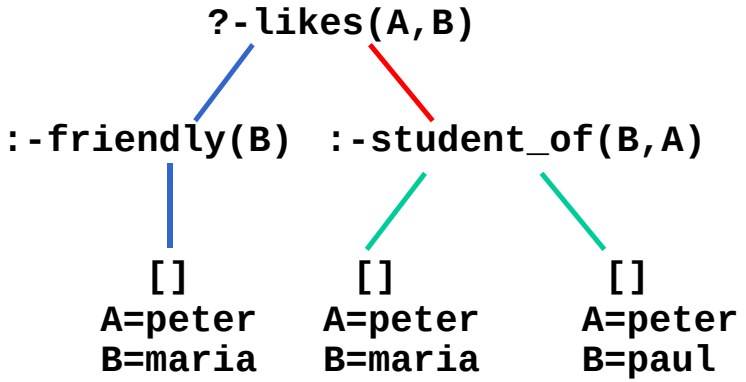




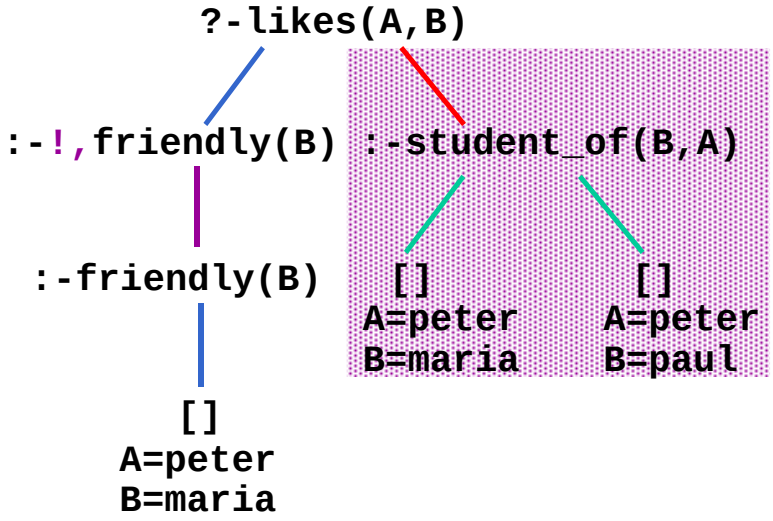

```
parent(X,Y):-father(X,Y),!.
parent(X,Y):-mother(X,Y).
father(john,paul).
father(john,peter).
mother(mary,paul).
mother(mary,peter).
```

Pruning away success branches

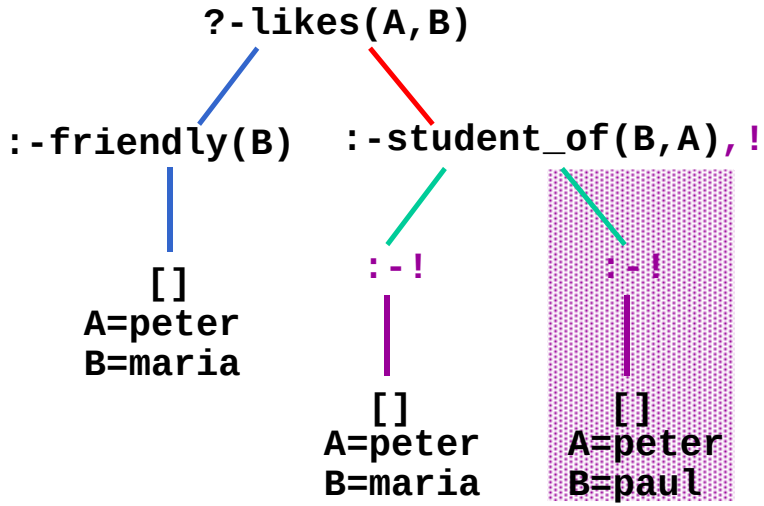
```
likes(peter,Y):-friendly(Y).
likes(T,S):-student_of(S,T).
student_of(maria,peter).
student_of(paul,peter).
friendly(maria).
```




```
likes(peter,Y):-!,friendly(Y).
```

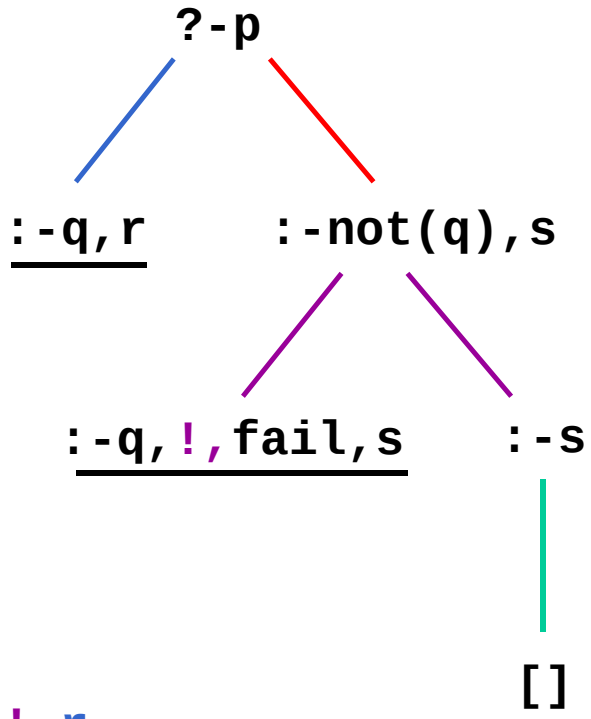


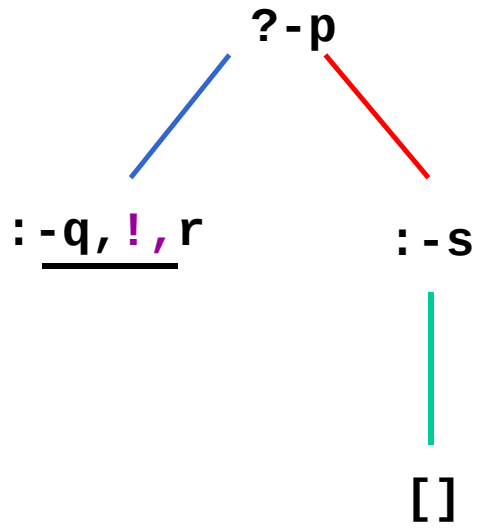

```
likes(T,S):-student_of(S,T),!.
```



Exercise 3.3

```
p:-q,r.
p:-not(q),s.
s.

not(Goal):-Goal,!,fail.
not(Goal).
```

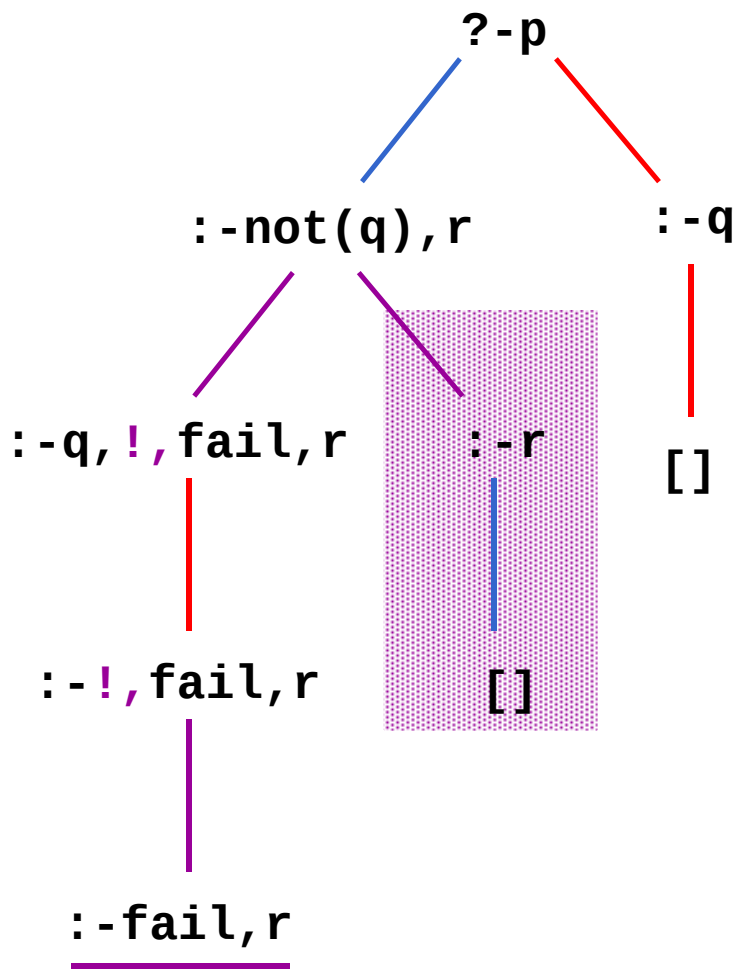




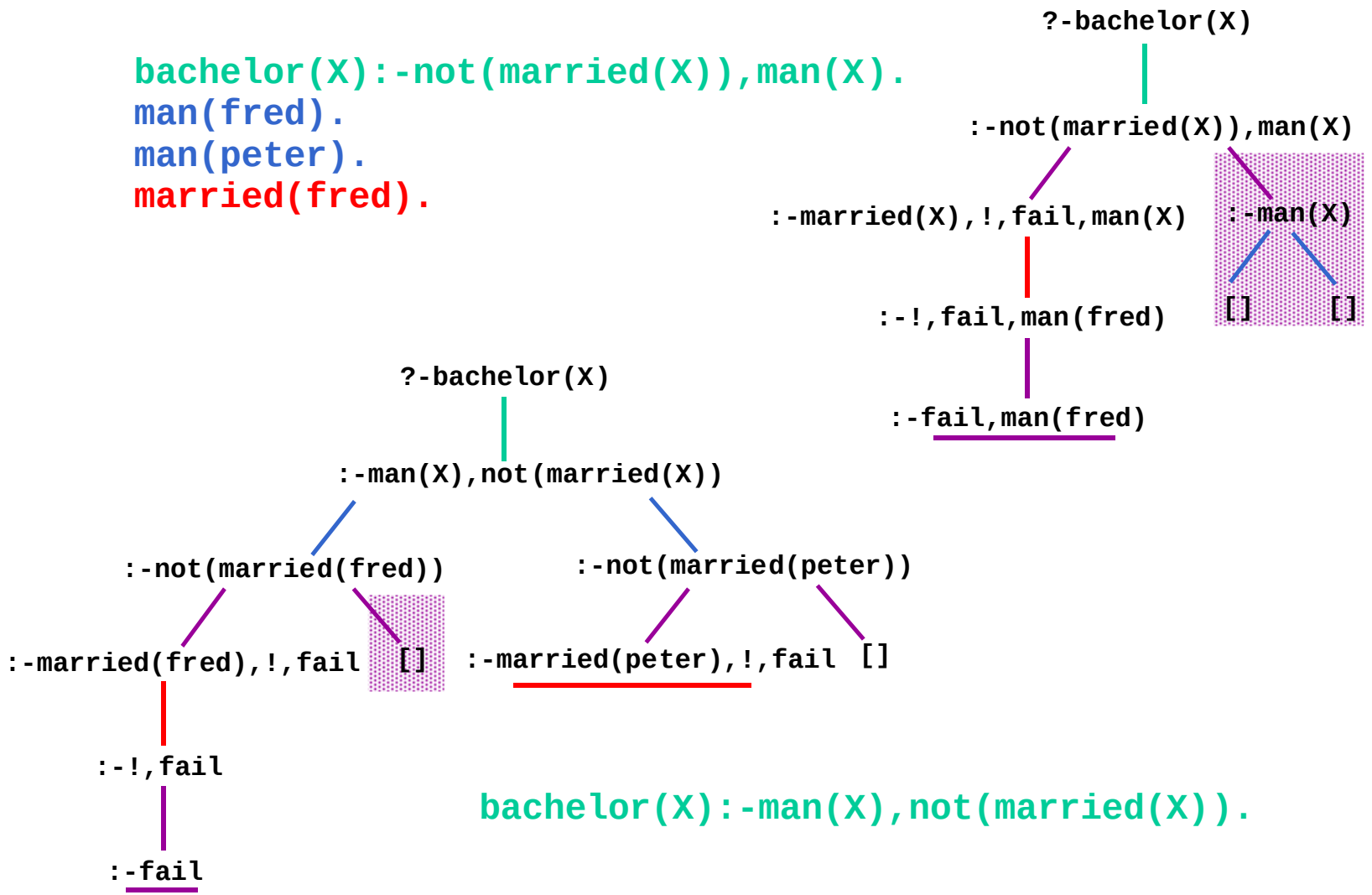

```
p:-q,!,r.
p:-s.
s.
```

not vs. cut

```
p:-not(q),r.
p:-q.
q.
r.

not(Goal):-Goal,!,fail.
not(Goal).
```

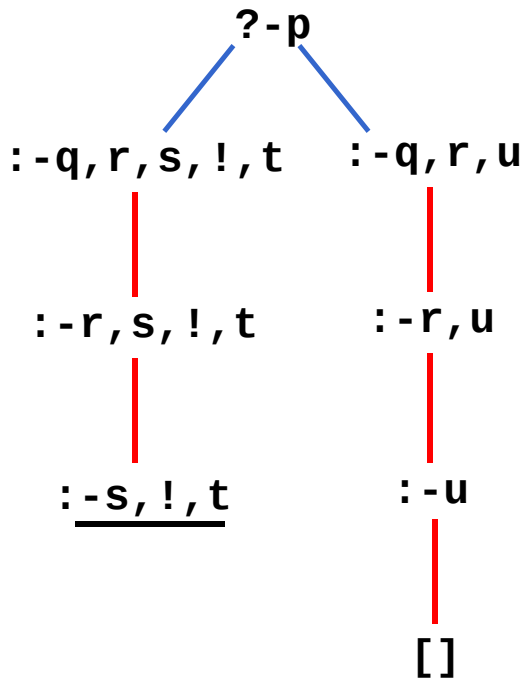


:-not(q) fails

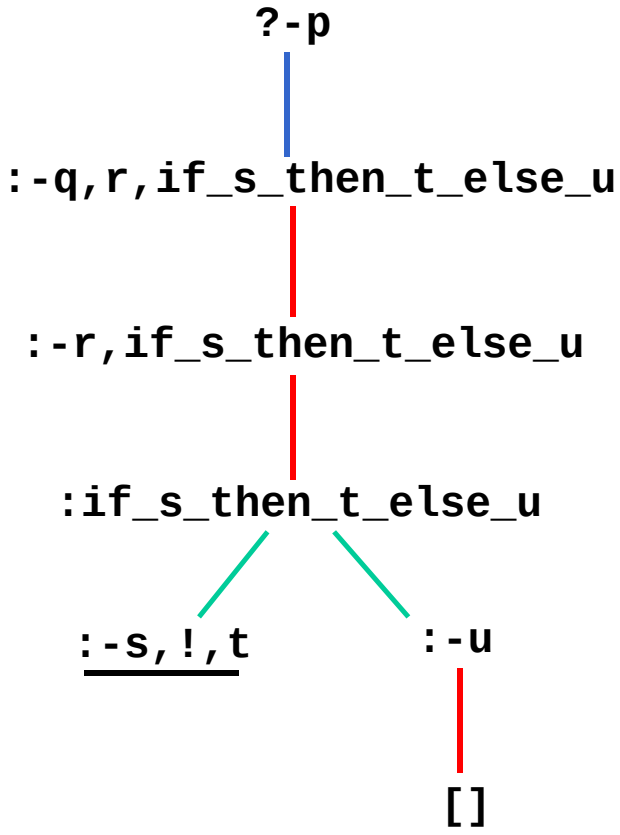


Prolog's `not` is unsound

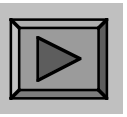
```
p:-q,r,s,!,t.
p:-q,r,u.
q.
r.
u.
```




```
p:-q,r,if_s_then_t_else_u.
if_s_then_t_else_u:-s,!,t.
if_s_then_t_else_u:-u.
q.
r.
u.
```



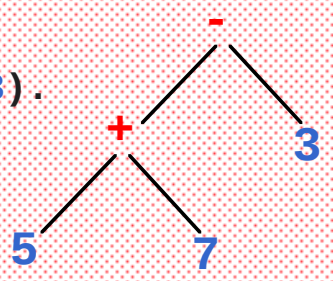
Exercises 3.7-8



```
?-X is 5+7-3.
  X = 9

?-9 is 5+7-3.
  Yes

?-9 is X+7-3.
  Error in arithmetic expression

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

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

?-9 = 5+7-3.
  No

?-9 = X+7-3.
  No

?-X = Y+7-3.
  X = _947+7-3
  Y = _947
```

```
?-display(5+7-3).
-(+(5,7),3)
```



Prolog arithmetic vs.unification

```
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.
```

Exercise 3.9

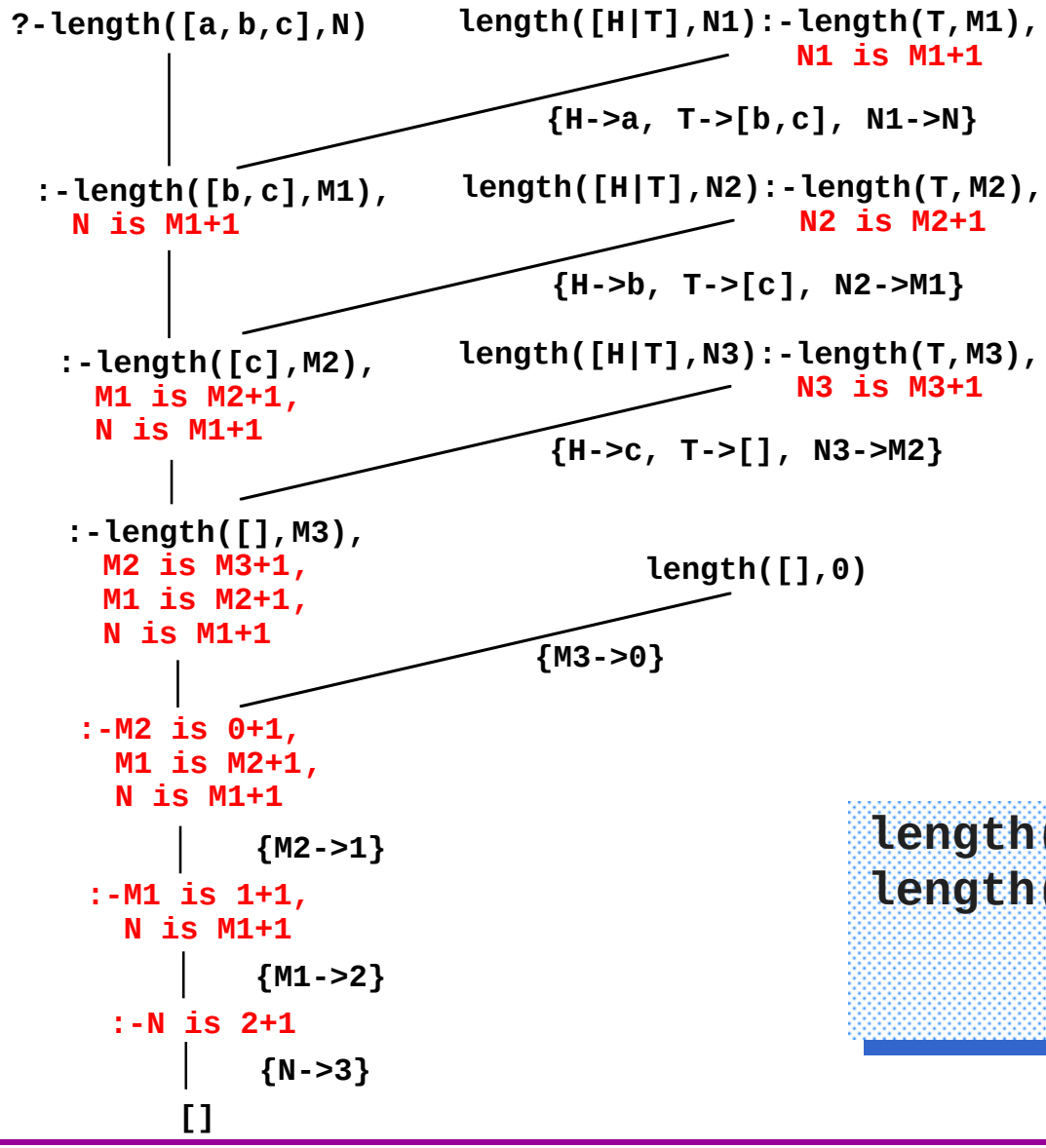
- Prolog does not check for circular bindings

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

- This may lead to unsound behaviour

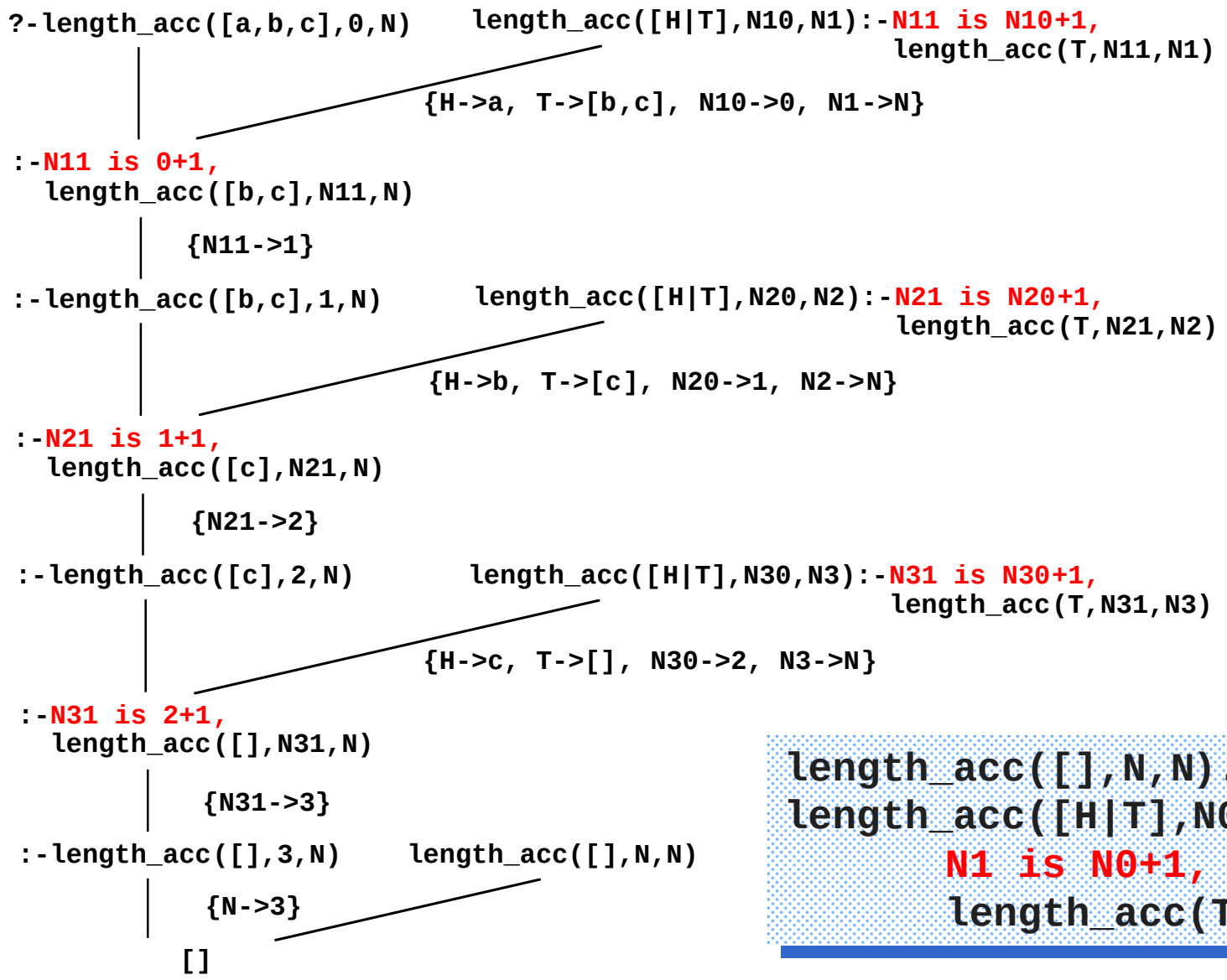
```
strange:-X=f(X).

?-strange.
  Yes
```

Occur check




```
length([],0).
length([H|T],N):-
        length(T,M),
        N is M+1.
```

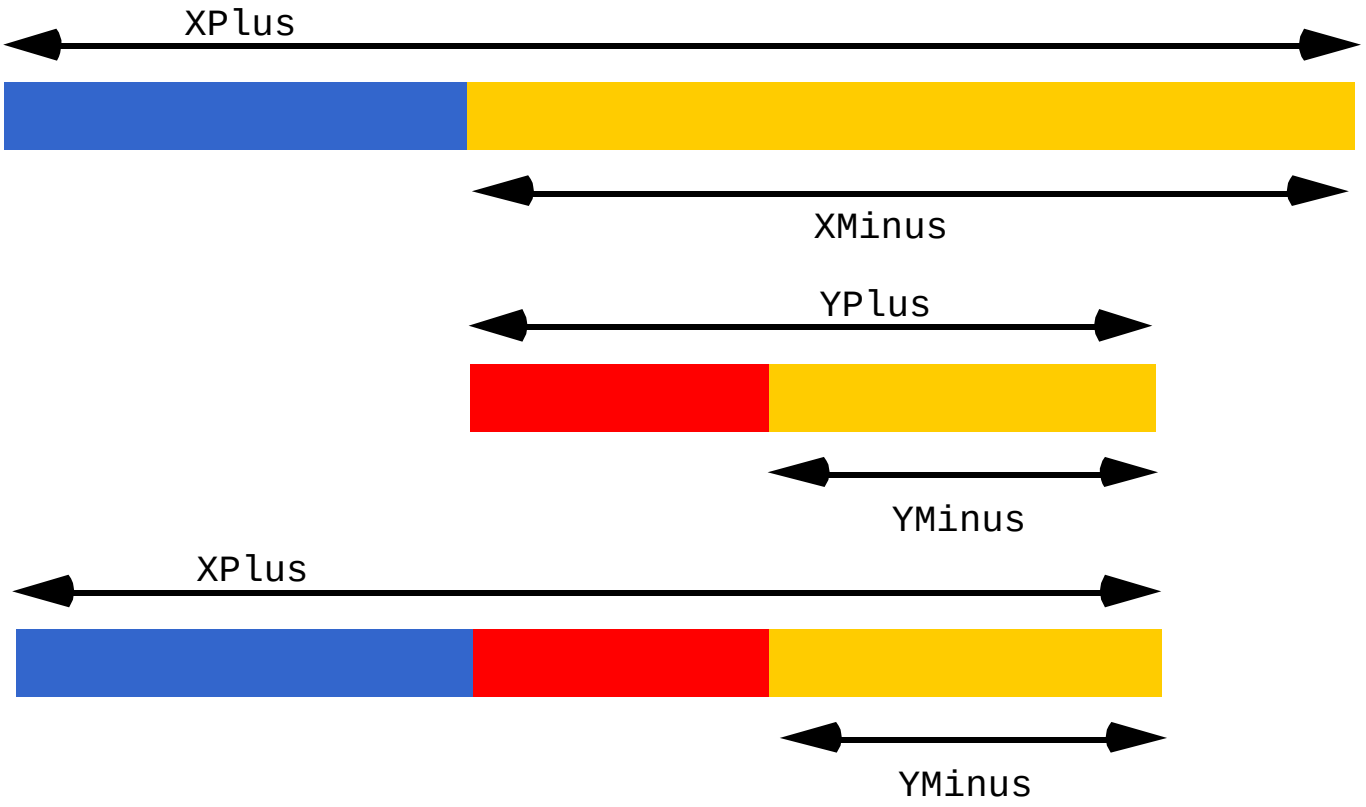
Exercise 3.10




```
length_acc([],N,N).
length_acc([H|T],N0,N):-
        N1 is N0+1,
        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

```
parent(john,peter).
parent(john,paul).
parent(john,mary).
parent(mick,davy).
parent(mick,dee).
parent(mick,dozy).
```

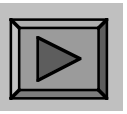
```
?-findall(C,parent(john,C),L).

  L = [peter,paul,mary]
```

```
?-findall(C,parent(P,C),L).

  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]
```

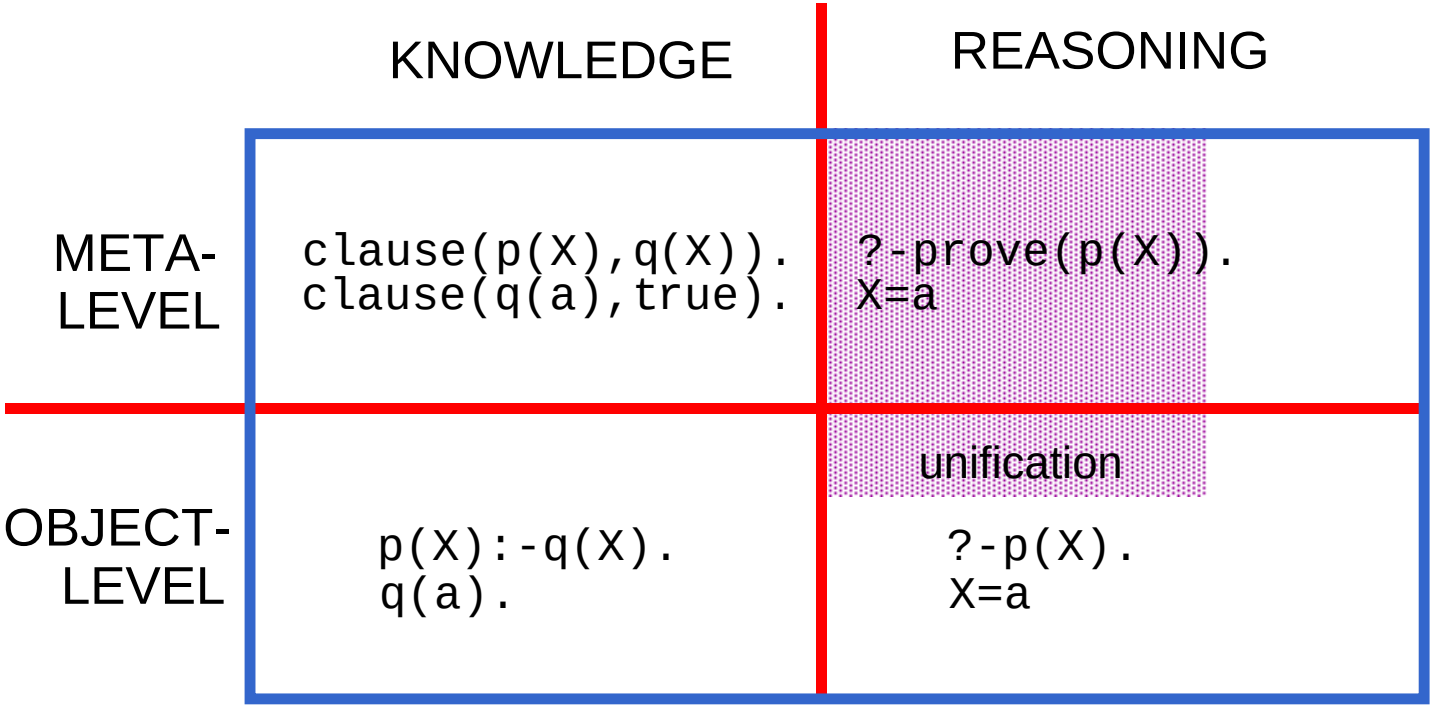
Second-order predicates



```
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



Meta-level vs. object-level

- Write down declarative specification

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

- 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).
```

☞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).
```

Writing a sort predicate

☞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 an insert predicate

☞ Write down declarative specification

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

☞ Write down skeleton

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

- 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].
```

- 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).
```

Writing an insert predicate