Logic Programming

Exam 2021

Answers by
David De Potter

Question 1: Fail at the first hurdle

(a) The two atoms are different, so they don't unify.

(b) An atom doesn't unify with a complex term.

(c) The two complex terms have the same functor, but different arity, so they don't unify.

(d) The two complex terms have the same functor and arity, but their arguments differ: 0 doesn't unify with 1.

(e) The two complex terms have different functors, so they don't unify.

(f) The two complex terms have the same functor and arity, but the variable Ann is instantiated with the number 1, after which 0 can't be unified with Ann anymore.

(g) Both sides have the same complex term bea/1, but on the left-hand side we have a complex term with arity 1 as its argument, whereas on the right-hand side we have a complex term with arity 2 as its argument. Hence, unification is impossible.

(h) A complex term with arity 1 doesn't unify with an atom (an arbitrary sequence of characters between single quotes is treated as an atom).

(i) A complex term doesn't unify with a list.

(j) A list doesn't unify with an atom.

(k) A list containing two items (the atom bea, and the empty list) doesn't unify with a list of only one item.
The unification succeeds if we turn it into: [bea|[ ]] = [bea].

(l) The second item [bea] can't be unified with the atom bea.

(m) The second item [BEA] (a list containing a variable) can't be unified with the atom bea.

(n) On the left-hand side we have a list with another list as its only item, whereas on the right-hand side we have a list which contains a single atom. An atom can't be unified with a list.

(o) On the left-hand side we have a list of two items, whereas on the right-hand side we have a list containing three items, of which the third item may be anything: the tail of this list contains a single arbitrary item (anonymous variable).

(p) On the left-hand side we have a list with three atoms (ann, bea, cee), while on the right-hand side we have a list containing two atoms and a list containing one atom. The atom cee can't be unified with the list [cee].

(q) Both sides consist of a single list. The variable ANN can be unified with the atom ann. Likewise, the atom bea can be unified with the variable CEE. After that, however, there's nothing left to unify the tail [BEA|ANN] with.

(r) On the left-hand side we find a list of three items, whereas the right-hand side has a complex term with a list as its only argument. A complex term of arity 1 can't be unified with a list.

(s) A complex term can't be unfied with an atom.

(t) Both sides have the same functor with arity 2. The variable X can be unified with the atom d. However, a list of three items can't be unified with a list of four.

Question 2: Cutting it fine

The predicate pos/2 simply collects all the positive numbers from the input list. Zeros and negative numbers are ignored. For example, if we were to perform the query `?- pos([0,1,-3,5,-1,4],Out)`, then we'd get the answer: `Out=[1,5,4]`. We can also use the predicate to see if a certain list of positive numbers is indeed a complete list of all the positive numbers that occur in the same order in the first list. If that's the case, the answer would be simply `true`.

cls 1: This is the base clause, the predicate will succeed if at some point it can unify both lists with the empty list.

cls 2: This recursive clause verifies if the current head of the input list is positive and if so, this item is unified with the head of the output list.

cls 3: This is a recursive clause, which removes the head of the first list if it is a non-positive number. The output list remains unaltered.

Adding cuts at different positions.

cut pos 1: This is a red cut and a bad position. Now the predicate will only succeed if both lists are empty from the start. The cut has radically changed the predicate's meaning by discarding all other branches so that only the base clause determines the predicate's success.

cut pos 2: This is a green cut and a good position as it enhances efficiency. Once it's clear that the first list is empty, the cut ensures that the other open branches are discarded, which is good, because we already know L is empty at this point, so no need to check if it can be unified with a non-empty list.

cut pos 3: This is a green cut. This is a good position for the cut, as it doesn't change the intended meaning and it enhances efficiency by discarding the third option once it's been confirmed that $X > 0$, so that the option for $X \leq 0$ isn't checked anymore.

cut pos 4: This is a bad position for the cut because it's completely redundant and actually decreases the program's efficiency by making the predicate non-tail-recursive. Now there will be a long and completely useless goal stack of cuts to go through (goals which all succeed, because the cut always does) once the base clause of the predicate pos/2 is reached and recursion for this predicate has stopped.

Question 3: Diversity matters

```
% true iff its argument is a list with at least 3 arguments and has
% elements that are all different

diversity(L) :- diversity([], L), L=[_,_,_|_].

diversity(Acc,[H|T]) :- \+ member(H,Acc), diversity([H|Acc],T).

diversity(_,[]).
```

Question 4: Tree analysis

```
% Database for question 4: tree analysis

rebmem(X,L) :- membre(L,X).

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

membre([H|_],H).
```

Question 5: The young ones

```
% The given database

age(ann, 20). age(joe, 44).
age(bob, 40). age(min, 27).
age(cai, 30). age(ned, 27).
age(deb, 42). age(pat, 33).
age(edo, 24). age(tod, 56).
```

(a) younger/2

```
% true iff X is younger than Y

younger(X, Y) :- age(X, A), age(Y, B), A < B.
```

(b) same_age/2

```
% true if the 1st argument unifies with a list of at least two people
% of the same age, and the 2nd argument unifies with the age of them

same_age(L,A) :- setof(X,(age(X,A)),L), length(L,Length), Length >= 2.

% alternative
same_age2(L, A) :- bagof(X,age(X,A),L), L=[_,_|_].
```

(c) oldest/1

```
% true if the 1st person is older than all other people in the KB

oldest(X) :- age(X,A), \+ (age(_,B), B > A).

% alternative
oldest2(X) :- age(X,A), findall(B,age(_,B),L), maxlist(L,A).

maxlist([H|T],Max) :- maxlist(T,H,Max).

maxlist([],Max,Max).

maxlist([H|T],Curr,Max) :- H > Curr, !, maxlist(T,H,Max).

maxlist([_|T],Curr,Max) :- maxlist(T,Curr,Max).
```

Question 6: Taking the train

```
% The given database

train(groningen,delfzijl).
train(groningen,leeuwarden).
train(groningen,assen).

direct(X,Y) :- train(X,Y).
direct(X,Y) :- train(Y,X).

route(X,Y,[X,Y]) :- direct(X,Y).
route(X,Y,[X|R]) :- route(Z,Y,R), direct(X,Z).
```

(a) There are seven clauses.

(b) We have four rules.

(c) The database contains three facts.

(d) Three predicates are defined: train/2, direct/2, route/3.

(e) The database doesn't have any dynamic predicates.

(f) There's one recursive predicate: route/3.

(g) There are no tail-recursive predicates in the database.

Question 7: A train of thought

The database on which the query is performed is the same as in question 6. From figure 1 it's clear that Prolog's answer to the query will be
`R = [delfzijl,groningen, assen]`.

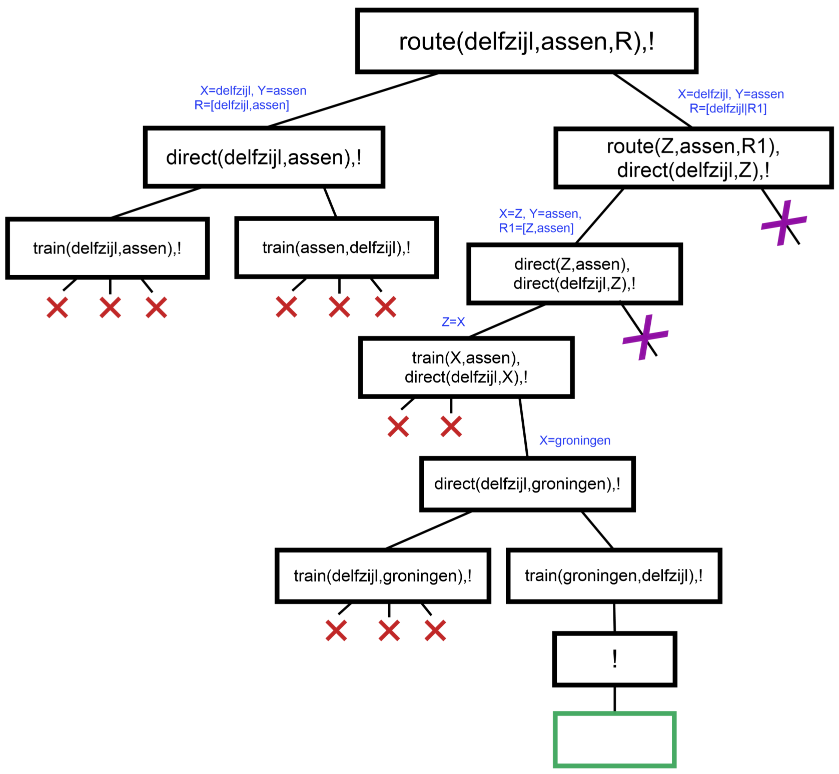

Figure 1: Search tree for *?- route(delfzijl,assen,R), !.*