1. *Solution.* The following program solves the exercise:

```
% graph G
edge(a,b).
edge(a,c).
edge(b,d).
edge(c,d).
edge(d,e).
edge(f,g).

% connected(X,Y) is true if X is connected to Y in G
%
connected(X,X).
connected(X,Z) :-
        edge(X,Y),
        connected(Y,Z).
```

   We only show that the height of the SLD-tree is (grossly) bounded by the number of vertices n in the graph G. Suppose the height is strictly larger than n; then there exists a path in G whose length is larger than n. Contradiction to the assumption that G is acyclic. □

2. *Solution.* (a) The given goal order leads to non-termination, even if both arguments are complete lists. (b) The difference can be easiest seen from the queries given below:

```
:- \+ sublist([a,d],[a,b,c,d]).
:- subsequence([a,d],[a,b,c,d]).
```

   □

3. *Solution.* Consider the following program:

```
?- op(700,xfx,***).

constant(X) :- integer(X).
constant(X) :- atom(X).

normalise(Exp,Norm) :-
        normalise_ds(Exp,Norm *** 1).
```

```
normalise_ds(A*B, Norm *** Space) :-
        normalise_ds(A, Norm *** NormB),
        normalise_ds(B, NormB *** Space).
normalise_ds(A,(A * Space) *** Space) :-
        constant(A).
```

□

4. *Solution.* Consider the following program:

```
palindrome(Xs) :- palindrome(q0,Xs,[]).

palindrome(q0,[X|Xs],Ys) :- palindrome(q0,Xs,[X|Ys]).
palindrome(q0,[X|Xs],Ys) :- palindrome(q1,[X|Xs],Ys).
palindrome(q0,[_X|Xs],Ys) :- palindrome(q1,Xs,Ys).
palindrome(q1,[X|Xs],[X|Ys]) :- palindrome(q1,Xs,Ys).
palindrome(q1,[],[]).
```

□

5. *Solution.* Consider the following programs:

```
% prop/1 --> DCG that generates well-parenthesised propositional formulas
%
prop(p) --> "p".
prop(q) --> "q".
prop(r) --> "r".
prop(not(A)) --> "~", prop(A).
prop(and(A,B)) --> "(", prop(A), "&", prop(B), ")".
prop(or(A,B)) --> "(", prop(A), "|", prop(B), ")".
prop(impl(A,B)) --> "(", prop(A), "=>", prop(B), ")".

% prop2/1 --> DCG that generates propositional formulas using
% the standard precedence
%
atom(p) --> "p".
atom(q) --> "q".
atom(r) --> "r".
atom(A) --> "(", prop2(A), ")".
unary(not(A)) --> "~", unary(A).
unary(A) --> atom(A).
and(and(A,B)) --> unary(A), "&", and(B).
and(A) --> unary(A).
or(or(A,B)) --> and(A), "|", or(B).
or(A) --> and(A).
prop2(impl(A,B)) --> or(A), "=>", prop2(B).
prop2(A) --> or(A).
```

□

6. *Solution.*

statement	**yes**	**no**
An existential fact is a fact that contains existentially quantified variables.	☐	☑
Data is structured in logic programs to obtain for example (i) better modularity or (ii) better organisation of the data.	☑	☐
Almost all, but not all basic elements of a relation database model can be expressed in Prolog.	☐	☑
Consider the standard implementation of *member*/2. Then any call to *member* terminates iff the second argument is a complete list.	☑	☐
A Prolog clause is called *iterative* if it has one recursive call and zero or more calls to system predicates that appear before the recursive call.	☑	☐
A cut fixes all choices between (and including) the moment of matching the rule's head with parent goal and the cut. If backtracking should read the cut, then the cut succeeds and the execution is continued with the clause *after* the clause containing the cut.	☐	☑
?−**op**(180, xfy, [imp, =>]). asserts that the operators imp and => are binary right-associative operators.	☑	☐
The explicit constructor for difference structures should be removed, if time or space efficiency is an issue.	☑	☐
A meta-interpreter for a language is an interpreter for the language written in the language itself.	☑	☐
Prolog is the only language that allows the efficient manipulation of meta-interpreters.	☐	☑

□