SEIII - Logikprogrammierung

Übungsaufgabe 07

Nico Hahn 6990715

Hieu Nguyen 6632126

Aufgabe 1

```
% Aufgabe 1.1
% Gibt eine Liste der Verzeichnisnamen auf dem Zugriffspfad zurueck
% zugriffspfad(+DirId, -Zugriffspfad)
?- consult('dateiverzeichnis.pl').
zugriffspfad(0, []).
zugriffspfad(DirId, ZP) :-
  directory(DirId,DirName,PId,_,_),
  zugriffspfad(PId, ParentZP),
  append(ParentZP, [DirName], ZP).

% Aufgabe 1.2
zugriffspfad_datei(FName, ZP) :-
  file(_,DirId,FName,_,_,_),
  zugriffspfad(DirId, DirZP),
  append(DirZP, [FName], ZP).
```

Aufgabe 2

```
% Aufgabe 2.1
?- consult('skigebiet.pl').
% ist_erreichbar_strecke(+Start,+Ziel,-Strecke)
ist_erreichbar_strecke(Start,Ziel,[Start,Ziel]) :-
  strecke(_,Start,Ziel,_,_).
ist_erreichbar_strecke(Start,Ziel,[Start|Strecke]) :-
  strecke(_,Start,ZielX,_,_),
  ist_erreichbar_strecke(ZielX,Ziel,Strecke).

% Aufgabe 2.2
% ist_erreichbar_seilbahn(+Start,+Ziel,-Strecke)
ist_erreichbar_seilbahn(Start,Ziel,[Start,Ziel]) :-
  strecke(_,Start,Ziel,_,_).
ist_erreichbar_seilbahn(Start,Ziel,[Start,Ziel]) :-
  lift(_,Start,Ziel,_).
ist_erreichbar_seilbahn(Start,Ziel,[Start|Strecke]) :-
  strecke(_,Start,ZielX,_,_),
  ist_erreichbar_seilbahn(ZielX,Ziel,Strecke).
ist_erreichbar_seilbahn(Start,Ziel,[Start|Strecke]) :-
  lift(_,Start,ZielX,_),
  ist_erreichbar_seilbahn(ZielX,Ziel,Strecke).
```

```
% Aufgabe 2.3
% ist_erreichbar_seilbahn_einmal(+Start,+Ziel,-Strecke)
ist_erreichbar_seilbahn_einmal(Start,Ziel,[Start,Ziel]) :-
  strecke(_,Start,Ziel,_,_).
ist_erreichbar_seilbahn_einmal(Start,Ziel,[Start,Ziel]) :-
  lift(_,Start,Ziel,_).
ist_erreichbar_seilbahn_einmal(Start,Ziel,[Start|Strecke]) :-
  strecke(_,Start,ZielX,_,_),
  ist_erreichbar_seilbahn_einmal(ZielX,Ziel,Strecke).
ist_erreichbar_seilbahn_einmal(Start,Ziel,[Start|Strecke]) :-
  lift(_,Start,ZielX,_),
  \+ member(Start, Strecke),
  ist_erreichbar_seilbahn_einmal(ZielX,Ziel,Strecke).
```

Aufgabe 3

```
% Aufgabe 3.1
% Ein Baum ist entweder ein Atom
% oder s(X,Y) mit X und Y wiederrum Baeume sind
baum(X) :- atom(X).
baum(s(X,Y)) :- baum(X), baum(Y).

% Aufgabe 3.2
% Tiefen von Wurzel zu den Blaettern bestimmen

% Ohne Endrekursiv
% tiefe_normal(+Baum, -Tiefe)
tiefe_normal(X, T) :- atom(X), T is 0.
tiefe_normal(s(X,_), T) :- tiefe_normal(X, T1), T is T1 + 1.
tiefe_normal(s(_,Y), T) :- tiefe_normal(Y, T1), T is T1 + 1.

% Mit Endrekursiv
% tiefe_end(+Baum, -Tiefe)
tiefe_rek(X, Acc, T) :- atom(X), T is Acc.
tiefe_rek(s(X,_), Acc, T) :-
  NeuAcc is Acc + 1,
  tiefe_rek(X, NeuAcc, T).
tiefe_rek(s(_,Y), Acc, T) :-
  NeuAcc is Acc + 1,
  tiefe_rek(Y, NeuAcc, T).
tiefe_end(X, T) :- tiefe_rek(X, 0, T).

% Aufgabe 3.3
% Maximale Tiefe eines Baumes bestimmen mit findall
% max_tiefe(+Baum, -Max)
max_tiefe(X, Max) :- findall(T, tiefe_end(X, T), Tiefen), max_member(Max,
Tiefen).
```

```
% Aufgabe 3.4
% Maximale Tiefe eines Baumes bestimmen mit Rekursion
% max_tiefe_normal(+Baum, -Max)
max_tiefe_normal(X, Max) :- atom(X), Max is 0.
max_tiefe_normal(s(X,Y), Max) :-
  max_tiefe_normal(X, Max1),
  max_tiefe_normal(Y, Max2),
  Max is max(Max1,Max2) + 1.

% Die Variante ohne Endrekursion fuehrt zu einer einfacheren Loesung.
% Sie ist einfach zu lesen und nachzuvollziehen.

% Aufgabe 3.5
% Prueft ob ein Baum balanciert oder nicht
% D.h. MaxHoehe - MinHoehe =< 1
% ist_balanciert(+Baum)
ist_balanciert(X) :-
  min_und_max_tiefe(X, Min, Max),
  Max - Min =< 1.
min_und_max_tiefe(X, Min, Max) :- atom(X), Min is 0, Max is 0.
min_und_max_tiefe(s(X,Y), Min, Max) :-
  min_und_max_tiefe(X, Min1, Max1),
  min_und_max_tiefe(Y, Min2, Max2),
  Min is min(Min1,Min2) + 1,
  Max is max(Max1,Max2) + 1.
```