SEIII - Logikprogrammierung

Übungsblatt 05

Nico Hahn 6990715 Hieu Nguyen 6632126

Aufgabe 1 - Unifikation

```
%?- h(g(F,k),g(k,F)) = h(g(m,H),g(H,m)).
% F = m,
% H = k.
% Das Praedikat h wird erfolgreich unifiziert, dann
% folgt die Unifikation von g(...). Aus der Unifikation
% ergibt sich die Variablenbindungen F = m, H = k.

% ?- d(x,d(y,d(z,nil))) = d(X,Y).
% X = x,
% Y = d(y, d(z, nil)).

% ?- m(X,c(g),h(X)) = m(t(r,s),c(u),h(g(T)),t).
% false.
% m wird zunaechst unifiziert. X wird an t(r,s) gebunden.
% Es schlaegt fehl bei der Unifikation von c(g) und c(u).

% ?- p(a,p(b,p(c,nil))) = p(X,p(Y,p(Z,nil))).
% X = a,
% Y = b,
% Z = c.
% p wird zunaechst erfolgreich unifiziert, X wird an a gebunden.
% Das 2. Praedikat p wird unifiziert, Y wird an b gebunden.
% Das 3. Praedikat p wird unifiziert, Z wird an c gebunden.

% ?- t(t(t(a,b),c),t(d,t(e,f))) = t(P,t(Q,R)).
% P = t(t(a, b), c),
% Q = d,
% R = t(e, f).
% Das erste t wird erfolgreich unifiziert.
% P wird bei der 2. Unifikation an t(t(a,b),c) gebunden.
% Bei der 3. Unifikation zw. t(d,t(e,f)) und t(Q,R) werden
% Q and d bebunden und R an t(e,f) gebunden

% ?- False = not(false).
% False = not(false).
% Hier findet nur eine Unifikation statt und False wird an not(false)
gebunden
```

Aufgabe 2

```
% 2.1
% peano2int(+Peano,-Int)
peano2int(0,0).
peano2int(s(X),Y) :- peano2int(X,Z), Y is Z + 1.

% 2.2 unmittelbarer Nachfolger
nachfolger(X,s(X)).
% oder
% nachfolger(0,s(0)).
% nachfolger(s(X),s(Y)) :- nachfolger(X,Y).

% 2.3 unmittelbarer Vorgaenger
vorgaenger(s(X),X).
% oder
% vorgaenger(X,Y) :- nachfolger(Y,X).

% 2.4 kleiner gleich
% lte(?X,?Y)
lte(0,0).
lte(0,s(_)).
lte(s(X),s(Y)) :- lte(X,Y).

% 2.5 Pruefe ob Peano2 = Peano1 * 2
% verdoppelt(?Peano1, ?Peano2)
verdoppelt(0,0).
verdoppelt(s(X),s(s(Y))) :- verdoppelt(X,Y).

% 2.6 Sub = Peano1 - Peano2
% sub(?Peano1,?Peano2,?Sub)
sub(X,0,X).
sub(s(X),s(Y),R) :- sub(X,Y,R).
% oder durch Wiederverwendung von add/3
% add(0,X,X).
% add(s(X),Y,s(R)) :- add(X,Y,R).
% sub(X,Y,Sub) :- add(Sub,Y,X).

% 2.7
% min(+X,+Y,-Min)
min(X,0,X).
min(0,X,X).
min(s(X),s(Y),s(Min)) :- min(X,Y,Min).
```

Aufgabe 3

```
?- consult('dateiverzeichnis.pl').
dirDepth(1,0).
dirDepth(DId,X) :- directory(DId,_,PId,_,_), dirDepth(PId,Y), X is Y + 1.
```

Aufgabe 4

```
?- consult('skigebiet.pl').
% 4.1
ist_erreichbar_rek(X,X,0).
ist_erreichbar_rek(Start,Ziel,Acc,Return) :-
  strecke(_,Start,Ziel,_,Laenge), Return is Acc + Laenge.
ist_erreichbar_rek(Start,Ziel,Acc,Return) :-
  strecke(_,Start,ZielX,_,Laenge),
  NewAcc is Acc + Laenge,
  ist_erreichbar_rek(ZielX,Ziel,NewAcc,Return).
ist_erreichbar(Start,Ziel,Return) :-
  ist_erreichbar_rek(Start,Ziel,0,Return).
```

Aufgabe 5

Bei der Wegplanung, dem Dateisystem, der Vorfahren-/Nachfahren-Beziehung in der Failiendatenbank werden Referenzen benutzt, um komplexe Logik bzw. Prädikate zu definieren. Referenz hier bedeutet die Variablenbindung zwischen unterschiedlichen Feldern in der Datenbank. Bei manchen Prädikatsdefinition wird einfach die Einträge in der Datenbank durchgesucht (wie bei der Familiendatenbank), bei anderen Prädikatsdefinition werden Konstrukte wie Rekursiv, Aggretation eingesetzt wie z.B. bei der Wegplanung. Eine weitere Gemeinsamkeit ist, dass die Einträge in der Datenbank als Knoten in einem Graphen sowie die Referenzen als Kanten betrachtet und Prädikaten werden dann definiert, um solche Graphen zu verarbeiten und daraus nützliche Informationen herzustellen.

Die Bedingung für eine terminierungssichere Prädikatsaufrufe ist zuerst die Prädikat selbst. Es muss so definiert dass es für eine "normale" bzw. gut definierte Datenbank terminierungssicher ist. Außerdem muss die Datenbank beim Prädikatsaufruf keine "Schleife" haben, sonst würde es bei vielen (insb. rekursiven) Prädikaten zu unendlichen Schleifen führen.

Beispiel für eine solche Datenbank - ein Prädikat, das die Wurzel des Verzeichnisbaums sucht, ist nicht terminierungssicher:

```
% directory(DId, ParentId)
directory(1, 3)
directory(2, 1)
directory(3, 2)
```