A04: 16,5P


```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Aufgabenblatt 05 - SE3-LP WiSe 16/17
%
% Finn-Lasse Jörgensen 6700628 4joergen@informatik.uni-hamburg.de
% Fabian Behrendt 6534523 fabian.behrendt95@gmail.com
% Daniel Klotzsche 6535732 daniel_klotzsche@hotmail.de
%
% Wir sind bereit folgende Aufgaben zu präsentieren:
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

A1: 3P

```
%%% A1 %%%

% a(B, C) = a(m, p) ?
% a/2 = a/2
% B = m, C = p
% a(m, p) = a(m, p)
% Unifikation erfolgreich!

% s(1, 2) = s(P, P) ?
% s/2 = s/2
% P = 1
% Unifizieren fehlgeschlagen
% P wird bereits mit 1 belegt und kann nicht mehr mit 2 belegt werden!

% g(f(s, R),f(R, s)) = g(f(S, t(T)), f(t(t), S)) ?
% g/2 = g/2
% g(f/2, f/2) = g(f/2, f/2)
% S = s, R = t(T), t(T) = t(t)
% g(f(s, t(t)), f(t(t), s)) = g(f(s, t(t)), f( t(t), s))
% Unifikation erfolgreich!

% q(t(r,s),c(g),h(g(T)),t) = q(Y,c(f(r,T)),h(Y)) ?
% q/4 =/= q/3
% Unifizieren fehlgeschlagen
% q/4 und q/3 lassen sich nicht Unifizieren, weil sie nicht zusammen passen!

% true = not(not(True)) ?
% not(not(True) = true
% true = true
% Unifikation erfolgreich!

% True = not(false) ?
% not(false) = true, True = true
% true = true
% Unifikation erfolgreich
```

A2: 5,5P

```
%%% A2 %%%

%% Ein Praedikat, das eine Peano-Zahl in eine Integer-Zahl umwandelt.
%peanoToInt(?Peano, ?Int)
%Rekursion stoppt bei: 0 = 0
peanoToInt(0,0).
peanoToInt(s(N),X) :- peanoToInt(N,X1), X is X1+1.

%% Ein Praedikat, das zwei Peano-Zahlen im Hinblick auf die Relation "groesser
%% oder gleich" vergleicht und in allen Instanziierungsvarianten verwendbar
%% ist.
%
% Es gilt P1 >= P2.
% groesserOderGleichPeano(?P1, ?P2)
groesserOderGleichPeano(s(_), 0).
groesserOderGleichPeano(0,0).
groesserOderGleichPeano(s(P1), s(P2)) :-
```



```
        groesserOderGleichPeano(P1, P2).

%% Ein Praedikat, das den Quotienten und den gegebenenfalls verbleibenden
%% Rest bei der Division einer Peano-Zahl durch zwei berechnet.
%
```

A2.1c? -2P

```
%% Ein Praedikat max(?Peano1,?Peano2,?PeanoMax), das fur zwei PeanoZahlen
%% Peano1 und Peano2 deren Maximum als PeanoMax ermittelt.
%
% max(+Peano1, +Peano2, -PeanoMax)
max(Peano1, Peano2, PeanoMax) :-
        peanoToInt(Peano1, Int1),
        peanoToInt(Peano2, Int2),
        Int1 > Int2,
        PeanoMax= Peano1.

max(Peano1, Peano2, PeanoMax) :-
        peanoToInt(Peano1, Int1),
        peanoToInt(Peano2, Int2),
        Int2 > Int1,
        PeanoMax= Peano2.
```

-2P %es sollte mit Peano gerechnet werden

A2.1e?-2P

```
%% Ein Praedikat, das das Produkt von zwei Peano-Zahlen mit Hilfe der Russische
n
%% Bauernmultiplikation berechnet.
%

%% 2. Modifizieren Sie die im Skript angegebenen Praedikatsdefinitionen fuer lt
/2
%% und add/3, indem Sie Typtests fur die Argumentbelegungen hinzufuegen.
%% Wie aendert sich das Verhalten? Warum?
```

%es sollte mit peano(X) gestestet werden

```
lt(0, s(_)).
lt(s(X), s(Y)) :-
        nonvar(X),      % Prüfen, ob X an einen Wert gebunden wurde
        nonvar(Y),      % Prüfen, ob Y an einen Wert gebunden wurde
        lt(X, Y).

add(0, X, X).
add(s(X), Y, s(R)) :-
        nonvar(X),      % Prüfen, ob X an einen Wert gebunden wurde
        nonvar(Y),      % Prüfen, ob Y an einen Wert gebunden wurde
        var(R),         % Prüfen, ob R eine freie Variable ist
        add(X, Y, R).

% Durch Einfügen von Typtests müssen die Prädikate nun mit bestimmten Argumente
n
% aufgerufen werden.
% So darf lt(+,+) nicht mehr unterspezifiziert aufgerufen werden,
% add(+,+,-) hingegen muss in seinem letzten Argument unterspezifiziert sein, w
ährend
% seine ersten beiden Argumente angegeben sein müssen.
```

-0,5P %Prolog kann keine unendlichen Ergebnisse mehr darstellen

A3: 3P

Doku?-1P

Tests?-1P

```
%%% A3 %%%

uebergeordnet(Kategorie, Ueberkategorie) :- sub(Kategorie, _, Ueberkategorie).
uebergeordnet(Kategorie, Ueberkategorie) :-
    sub(Kategorie, _, X),
    uebergeordnet(X, Ueberkategorie).

ebene_von(Ebene, Kategorie) :- sub(Kategorie, Ebene, _).
ebene_von(Ebene, Kategorie) :-
    reich(Kategorie),
    Ebene = reich.
```

```
uebergeordnet(Kategorie, Ebene, Ueberkategorie) :-
    (
        sub(Kategorie, _, Ueberkategorie);
        (
            sub(Kategorie, _, X),
            uebergeordnet(X, _, Ueberkategorie)
        )
    ),
ebene_von(Ebene, Ueberkategorie).
```

A4:5P

```
%%% A4 %%%
```

Doku?-1P

```
?- [gleispla].

%% 1.
% Das Prädikat verbindung(Gleis1, Gleis2) gibt an, ob eine Verbindung zwischen
% Gleis1 und Gleis 2 besteht.
% Gleis1 hat eine direkte oder eine indirekte Verbindung zu Gleis2,
% ohne dabei die Fahrtrichtung wechseln zu müssen.
%
% verbindung(?Gleis1, ?Gleis2, -Anzahl)
verbindung(Gleis1, Gleis2, Anzahl) :-          %
        verbindungSym(Gleis1, Gleis2, Anzahl). % Erzeugen der

                %
verbindung(Gleis1, Gleis2, Anzahl) :-          % symmetrischen Hülle
        verbindungSym(Gleis2, Gleis1, Anzahl). %

verbindungSym(Gleis1, Gleis2, Anzahl) :-
        weiche(_, Gleis1, Gleis2, _),
        Anzahl is 1.

verbindungSym(Gleis1, Gleis2, Anzahl) :-
        weiche(_, Gleis1, GleisMitte, _),
        verbindungSym(GleisMitte, Gleis2, Anzahl1),
        Anzahl is Anzahl1+1.

%% 2.
% Hilfsprädikat zur Berechnung des Ankunftsgleises unter Berücksichtigung
% der minimalen Anzahl an Weichen, die durchfahren werden.
%
% berechneMinWeichenAnkunft(?Gleis1, ?Gleis2, -Min)
berechneMinWeichenAnkunft(Gleis1, Gleis2, Min) :-
        findall(AnzahlWeichen,
                        (gleis(Gleis2, _, b), verbindung(Gleis1, Gleis2, Anzahl
Weichen)),
                        ListeAnzahl),
        min_list(ListeAnzahl, Min).

% Das Prädikat ankunft(Von_Ort, Gleis) gibt an, auf welchem Gleis ein Zug, der
% aus Von_Ort kommt, einfährt.
% Hierbei ist die Anzahl der durchfahrenen Gleise minimal.
%
% ankunft(?Von_Ort, -Gleis)
ankunft(Von_Ort, Gleis) :-
        einfahrt(Ankunftsgleis, Von_Ort),
        berechneMinWeichenAnkunft(Ankunftsgleis, Gleis, Min),
        gleis(Gleis, _, b),
        verbindung(Ankunftsgleis, Gleis, Min).

% Hilfsprädikat zur Berechnung des Abfahrtsgleises unter Berücksichtigung
% der minimalen Anzahl an Weichen, die durchfahren werden.
%
% berechneMinWeichenAbfahrt(?Gleis1, ?Gleis2, -Min)
berechneMinWeichenAbfahrt(Gleis1, Gleis2, Min) :-
```



```
        findall(AnzahlWeichen,
                        (gleis(Gleis1, _, b), verbindung(Gleis1, Gleis2, Anzahl
Weichen)),
                        ListeAnzahl),
        min_list(ListeAnzahl, Min).

% Das Prädikat abfahrt(Nach_Ort, Gleis) gibt an, auf welchem Gleis ein Zug, der
% nach Nach_Ort fährt, abfahren muss.
% Hierbei ist die Anzahl der durchfahrenen Gleise minimal.
%
% abfahrt(?Nach_Ort, -Gleis)
abfahrt(Nach_Ort, Gleis) :-
        ausfahrt(Abfahrtsgleis, Nach_Ort),
        berechneMinWeichenAbfahrt(Gleis, Abfahrtsgleis, Min),
        gleis(Gleis, _, b),
        verbindung(Gleis, Abfahrtsgleis, Min).

%%% A5 %%%
```