A07: 11P


```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Aufgabenblatt 07 - 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:
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
?- [medien].
```

A1: 7P

```
%%% A1 %%%
%% 1.
%Ein Praedikat, das den Umsatz fuer jedes Produkt in einer vorgegebenen Ka egor
ie
%in einem vorgegebenen Jahr berechnet.

umsatz(Kategorie ID,Jahr ,Produkt ID,Umsatz) :-
  produkt( Produkt ID, Kategorie ID,   ,   ,   ,   ),
  verkauft( Produkt_ID, Jahr , Preis , Anzahl ),
  Umsatz is Preis * Anzahl.

%% 2.
% Ein Praedikat, das fur eine gegebene Kategorie ermittelt, ob sich der m t de
n
% betreffenden Produkten erzielte Umsatz in den letzten fuenf Jahren geste gert
% oder verringert hat. Ueberlegen Sie sich dafuer eine geeignete Heuristik zur
% Trendabschaetzung (z.B. Anstieg der Regressionsgeraden oder relative Hoe e
% des Umsatzes im letzten Jahr im Vergleich zum Mittelwert der letzten fue f
% Jahre.)

% Es wird der Umsatz aller Jahre aufsummiert und der Durchschnitt gebildet
% Wenn der Durchschnitt groesser als der aktuelle Umsatz (aus dem im Praed kats
aufruf angegebenen Jahr) ist,
% ist der Umsatz gestiegen und es wird 'true' ausgegeben. Wenn nicht, dann wird
 'false' ausgegeben.
umsatz_gesteigert(Kategorie ID, Jahr) :-
       umsatz_gesteigert_in_fuenf(Kategorie_ID, Jahr, 5).

umsatz_gesteigert_in_fuenf(Kategorie ID, Jahr, Jahre) :-
  findall(Umsatz ,
           (umsatz (Kategorie ID, Jahr1, , Umsatz) ,
            Jahr1 < Jahr,
            Jahr1 >= Jahr - Jahre),
          Umsaetze),
  umsatz (Kategorie ID, Jahr, , AktuellerUmsatz),
  sum list(Umsaetze , SummeUmsaetze),
  Durchschnittis (SummeUmsaetze+ AktuellerUmsatz) / (Jahre + 1),
  Durchschnitt< AktuellerUmsatz

% ?- umsatzGesteigert( 7 , 2012 , 5 ).
% -> false.
% ?- umsatzGesteigert(9, 2009, 5).
% -> true.
```

A1.3? -7P

```
%% 3.


%%% A2 %%%

?- [gleisplan].
```

A2: 4P

```
%% 1.
% Modifizieren Sie das Verbindungsprädikat aus Aufgabe 3.1 von Aufgabenbla t
% 4 so, dass die Liste der jeweils zu befahrenden Weichen ausgegeben werde
% kann. Achten Sie dabei darauf, dass in dieser Liste die Weichen stets in der
% Reihenfolge angegeben sind, in der sie zu befahren sind.
%
% verbindung1(?Gleis1, ?Gleis2, -Liste)
```



```
%
% Aufruf eines Hilfsprädikats, um das eigentliche Prädikat symmetrisch
% zu machen, sodass auch eine Verbindung von rechts nach links (auf dem
% Gleisplan) erkannt wird.
verbindung1(Gleis1, Gleis2, Liste) :-
       verbindungSym1(Gleis1, Gleis2, Liste).

verbindung1(Gleis1, Gleis2, Liste) :-
       verbindungSym1(Gleis2, Gleis1, Liste).

% Liegt eine direkte Verbindung vor, so wird der Name der Weiche in eine
% Liste geschrieben.
verbindungSym1(Gleis1, Gleis2, Liste) :-
       weiche(Weiche, Gleis1, Gleis2,  ),
       Liste = [Weiche].

% Liegt eine indirekte Verbindung vor, so wird der Name der ersten Weiche
% als Kopf in die Liste geschrieben und mit der Restliste nach weiteren
% Verbindungen gesucht.
verbindungSym1(Gleis1, Gleis2, Liste) :-
       weiche(Weiche, Gleis1, GleisMitte, _),
       Liste = [Weiche | Restliste],
       verbindungSym1(GleisMitte, Gleis2, Restliste).

% Tests %
%
% ?- verbindung1(a1, z5, Liste).
% Liste = [w11, w31] ;
% false.
%
% ?- verbindung1(a1, g4, Liste).
% false.

%% 2.
% Auf dem Bahnhof soll das Stellwerk automatisiert werden. Modifizieren
% Sie das Prädikat aus Aufgabenteil 1 so, dass auch die für die jeweilige
% Verbindung erforderlichen Weichenstellungen mit ausgegeben werden.
%
% verbindung2(?Gleis1, ?Gleis2, -Liste)
%
% Aufruf eines Hilfsprädikats, um das eigentliche Prädikat symmetrisch
% zu machen, sodass auch eine Verbindung von rechts nach links (auf dem
% Gleisplan) erkannt wird.
verbindung2(Gleis1, Gleis2, Liste) :-
       verbindungSym2(Gleis1, Gleis2, Liste).

verbindung2(Gleis1, Gleis2, Liste) :-
       verbindungSym2(Gleis2, Gleis1, Liste).

% Liegt eine direkte Verbindung vor, so wird der Name der Weiche und ihr
% Zustand in eine Liste geschrieben.
verbindungSym2(Gleis1, Gleis2, Liste) :-
       weiche(Weiche, Gleis1, Gleis2, Zustand),
       Liste = [Weiche, Zustand].

% Liegt eine indirekte Verbindung vor, so wird der Name der ersten Weiche
% und ihr Zustand als Kopf in die Liste geschrieben und mit der Restliste
% nach weiteren Verbindungen gesucht.
verbindungSym2(Gleis1, Gleis2, Liste) :-
       weiche(Weiche, Gleis1, GleisMitte, Zustand),
       Liste = [Weiche, Zustand | Restliste],
       verbindungSym2(GleisMitte, Gleis2, Restliste).

% Tests %
%
% ?- verbindung2(a1, z5, Liste).
% Liste = [w11, g, w31, g] ;
% false.
```

A2.3-8 -12P

```
%% 3.
```

```
% Sind in einem Bahnhof zwei parallele Gleisabschnitte durch einen dritten
% Gleisabschnitt miteinander verbunden, so werden die betreffenden Weichen
% zwangsgekoppelt, um Flankenfahrten auszuschließen. In diesem Falle stehen
% die beiden Weichen entweder so, dass zwei Züge gleichzeitig die parallelen
% Gleise befahren können, oder aber so, dass ein Zug von einem auf das andere
% Gleis wechseln kann.
%
% Erweitern Sie das Prädikat aus Aufgabenteil 2 so, dass auch die korrekte Stel
lung
% für die zwangsgekoppelten Weichen mit ausgegeben wird.
%
% verbindung2(?Gleis1, ?Gleis2, -Liste)
%
% Aufruf eines Hilfsprädikats, um das eigentliche Prädikat symmetrisch
% zu machen, sodass auch eine Verbindung von rechts nach links (auf dem
% Gleisplan) erkannt wird.
verbindung3(Gleis1, Gleis2, Liste) :-
        verbindungSym3(Gleis1, Gleis2, Liste).

verbindung3(Gleis1, Gleis2, Liste) :-
        verbindungSym3(Gleis2, Gleis1, Liste).

% Liegt eine direkte Verbindung vor, so wird der Name der Weiche und ihr
% Zustand in eine Liste geschrieben.
% Dabei wird - dem Modus entsprechend - der Zustand der gekoppelten Weiche
% an die Liste angefügt.
verbindungSym3(Gleis1, Gleis2, Liste) :-
        weiche(Weiche1, Gleis1, Gleis2, Zustand),
        (gekoppelt(Weiche1, Weiche2, Modus);
        gekoppelt(Weiche2, Weiche1, Modus)),
        (Modus = gegen,
        Zustand = g,
        Liste = [Weiche1, Zustand, Weiche2, a];
        Modus = gegen,
        Zustand = a,
        Liste = [Weiche1, Zustand, Weiche2, g];
        Modus = gleich,
        Liste = [Weiche1, Zustand, Weiche2, Zustand]).

% Liegt eine indirekte Verbindung vor, so wird der Name der ersten Weiche
% und ihr Zustand als Kopf in die Liste geschrieben und mit der Restliste
% nach weiteren Verbindungen gesucht.
verbindungSym3(Gleis1, Gleis2, Liste) :-
        weiche(Weiche1, Gleis1, GleisMitte, Zustand),
        (gekoppelt(Weiche1, Weiche2, Modus);
        gekoppelt(Weiche2, Weiche1, Modus)),
        (Modus = gegen,
        Zustand = g,
        Liste = [Weiche1, Zustand, Weiche2, a | Restliste];
        Modus = gegen,
        Zustand = a,
        Liste = [Weiche1, Zustand, Weiche2, g | Restliste];
        Modus = gleich,
        Liste = [Weiche1, Zustand, Weiche2, Zustand | Restliste]),
        verbindungSym3(GleisMitte, Gleis2, Restliste).

% Tests %
%
% ?- verbindung3(a1, z5, Liste).
% Liste = [w11, g, w12, a, w31, g, w32, g] ;
% false.
%
% ?- verbindung3(z1, z5, Liste).
% false.
%
% Dieser Test schlägt fehl, da in dieser Implementation nur Weichen
% berücksichtigt werden, welche tatsächlich gekoppelt sind.
% Für alle anderen Weichen, die auf der Strecke befahren werden,
% liefert der Aufruf natürlich false.
```



```
%% 4.
% Ein Prädikat, welches prüft, ob ein Element in einer Liste enthalten ist
%
% istEnthalten(?Element, +Liste)
istEnthalten(Element, Liste) :-
        Liste = [Element | _].

istEnthalten(Element, Liste) :-
        Liste = [ Kopf | Restliste],
        istEnthalten(Element, Restliste).

% Der Zug fährt von links nach rechts uber eine Weiche und hat sein Ziel
% erreicht.
verbindung(Gleis1, Gleis2, Liste) :-
        weiche( _, Gleis1, Gleis2, _),
        Liste = [Gleis2].

% Der Zug fährt von rechts nach links uber eine Weiche und hat sein Ziel
% erreicht.
verbindung(Gleis1, Gleis2, Liste) :-
        weiche( _, Gleis2, Gleis1, _),
        Liste = [Gleis1].

% Der Zug fährt von links nach rechts uber eine Weiche, hat ein zuvor noch
% nicht verwendetes Gleis erreicht und setzt seine Fahrt in der gleichen
% Richtung fort.
verbindung(Gleis1, Gleis2, Liste) :-
        weiche( _, Gleis1, GleisMitte _),
        gleis(GleisMitte _, _),
        \+ istEnthalten(GleisMitte, Liste),
        Liste = [GleisMitte | Restliste],
        verbindung(GleisMitte, Gleis2, Restliste).

% Der Zug fährt von rechts nach links uber eine Weiche, hat ein zuvor noch
% nicht verwendetes Gleis erreicht und setzt seine Fahrt in der gleichen
% Richtung fort.
verbindung(Gleis1, Gleis2, Liste) :-
        weiche( _, Gleis2, GleisMitte _),
        gleis(GleisMitte _, _),
        \+ istEnthalten(GleisMitte, Liste),
        Liste = [GleisMitte | Restliste],
        verbindung(GleisMitte, Gleis1, Restliste).

% Der Zug fährt von links nach rechts uber eine Weiche, hat ein Gleis er-
% reicht und setzt seine Fahrt in umgekehrter Richtung fort.
verbindung(Gleis1, Gleis2, Liste) :-
        weiche( _, Gleis1, GleisMitte _),
        gleis(GleisMitte _, _),
        Liste = [GleisMitte | Restliste],
        verbindung(Gleis2, GleisMitte, Restliste).

% Der Zug fährt von rechts nach links uber eine Weiche, hat ein Gleis er-
% reicht und setzt seine Fahrt in umgekehrter Richtung fort.
verbindung(Gleis1, Gleis2, Liste) :-
        weiche( _, Gleis2, GleisMitte _),
        gleis(GleisMitte _, _),
        Liste = [GleisMitte | Restliste],
        verbindung(Gleis1, GleisMitte, Restliste).

% Der Zug fährt von links nach rechts uber eine Weiche, hat noch kein Glei
% erreicht und setzt seine Fahrt in der gleichen Richtung fort.
verbindung(Gleis1, Gleis2, Liste) :-
        weiche( _, Gleis1, GleisMitte _),
        \+ gleis(GleisMitte _, _),
        Liste = [GleisMitte | Restliste],
        verbindung(GleisMitte, Gleis2, Restliste).

% Der Zug fährt von rechts nach links uber eine Weiche, hat noch kein Glei
% erreicht und setzt seine Fahrt in der gleichen Richtung fort.
verbindung(Gleis1, Gleis2, Liste) :-
```

```
        weiche( , Gleis2, GleisMitte, ),
        \+ gleis(GleisMitte, , ),
        Liste = [GleisMitte | Restliste],
        verbindung(GleisMitte, Gleis1, Restliste).

%% 5.

%% 6.

%% 7.
```