Aufgabe 7 (trenne)

Schreiben Sie ein Programm

```
trenne(+Eingabeliste, -ZahlenListe, -NichtzahlenListe)
```

das eine Eingabeliste in eine Zahlen- und eine Nichtzahlenliste trennt. Um festzustellen, ob ein Term eine Zahl ist, benutzen Sie das Prädikat `number(X)` von Prolog. Verwenden Sie außerdem das Prädikat `append(X, Y, XY)`, welches zwei Listen `X` und `Y` zu einer Liste `XY` verkettet, z.B.

Frage: `?- append([a], [b,c], XY).`
Antwort: `XY = [a,b,c]`

a) Einfach geht es so, dass die Elemente in den Ausgabelisten in der umgekehrten Reihenfolge wie in der Eingabeliste erscheinen, also z.B.

Frage: `?- trenne([a,2,b,4,1,d,c], Zahlen, Nichtzahlen).`
Antwort: `Zahlen = [1,4,2], Nichtzahlen = [c,d,b,a])`

```
% Loesung Aufgabe 7, 5. Juli 2007
% a) Ergebnisse in umgekehrter Reihenfolge:

trenneR([],[],[]).
trenneR([X|Xs],ZsX,NZs) :-
   number(X),
   trenneR(Xs,Zs,NZs),
   append(Zs,[X],ZsX).
trenneR([X|Xs],Zs,NZsX) :-
   not(number(X)),
   trenneR(Xs,Zs,NZs),
   append(NZs,[X],NZsX).
```

b) Können Sie es auch so, dass die Elemente in den Ausgabelisten in der gleichen Reihenfolge wie in der Eingabeliste erscheinen? Also, z.B.

Frage: `?- trenne([a,2,b,4,1,d,c],Zahlen,Nichtzahlen).`
Antwort: `Zahlen = [2,4,1], Nichtzahlen = [a,b,d,c])`

```
% b) Ergebnisse in gleicher Reihenfolge:
trenne([],[],[]).
trenne([X|Xs],[X|Zs],NZs) :-
   number(X), trenne(Xs,Zs,NZs).
trenne([X|Xs],Zs,[X|NZs]) :-
   not(number(X)), trenne(Xs,Zs,NZs).

% Oder: trenne/3 benutzen und die Ergebnisse umdrehen:
trenneR2(Liste,Zahlen,Nichtzahlen) :-
   trenne(Liste,Zs,NZs),
   reverse(Zs,Zahlen),
   reverse(NZs,Nichtzahlen).
```

Aufgabe 8 (weg)

a) Schreiben Sie ein Programm

`weg(X, Y)`

das prüft, ob ein Weg von X nach Y existiert. Die Datenbasis lautet:

```
kante(1,2).
kante(1,3).
kante(2,4).
kante(2,5).
kante(3,4).
kante(4,7).
kante(5,6).
kante(6,7).
```

b) Schreiben Sie ein Programm

`weg(X, Y, N)`

das prüft, ob ein Weg von X nach Y mit der Länge N existiert. Für das Zuweisen von Werten hat Prolog das Prädikat `is`. Zum Beispiel liefert die Frage `?- N is 4 + 5.` als Antwort: `N = 11`

c) Schreiben Sie ein Programm

```
weg(X, Y, N, L)
```

das prüft, ob ein Weg von X nach Y mit der Länge N existiert und in einer Liste `L = [xn,...,x1]` die Punkte `xi` aus {1,2,...,7} ausgibt, die auf dem jeweiligen Weg von `x0` nach `xn` (in der Reihenfolge `x1,x2,...`) besucht wurden (nach dem Startpunkt `x0`).

Hinweis: Um eine Liste umzudrehen, hat Prolog das Prädikat `reverse/2`.

Beispiel:

```
Frage:    ?- weg(1,7,N,L).
Antwort:  N = 3
          L = [2, 4, 7];

          N = 4
          L = [2, 5, 6, 7]
```

% Loesung Aufgabe 8c)

```
kante(1,2).
kante(1,3).
kante(2,4).
kante(2,5).
kante(3,4).
kante(4,7).
kante(5,6).
kante(6,7).

weg(X,Y) :- kante(X,Y).
weg(X,Y) :- kante(X,Z), weg(Z,Y).

weg(X,Y,1) :- kante(X,Y).
weg(X,Y,N) :- kante(X,Z), weg(Z,Y,K), N is K+1.

%A8 c
weg(X,Y,1,[Y]) :- kante(X,Y).
weg(X,Y,N,[Z|L]) :- kante(X,Z), weg(Z,Y,K,L), N is K+1.
weg4(X,Y,N,L) :- weg(X,Y,N,L1), reverse(L1,L).
```

Aufgabe 9 (trace)

Verwenden Sie den eingebauten Tracer, um festzustellen, in welcher Reihenfolge Prolog welche Aufrufe von `weg/4` und welche von `kante/2` macht, wenn Sie mit Ihrem Programm zu Aufgabe 8 die Frage

```
?- weg(2,7,N,L).
```

stellen. Genauer gesagt, machen Sie folgendes:

a) Schauen Sie mit `?- apropos(trace).` oder `?- help(trace).`, wie man Prolog sagt, von welchen Prädikaten es Informationen über
- Aufrufen (call) des Prädikats
- Benutzen der nächsten passenden Klausel des Prädikats (redo)
- Finden einer Lösung (exit)
- Scheitern einer Beweissuche für den Aufruf (fail)

ausgeben soll.

b) Sagen Sie Prolog, es soll
- für `weg/4` die Aufrufe, das Benutzen der nächsten Klausel, und das Finden einer Lösung,
- für `kante/2` nur die Aufrufe und die gefundenen Lösungen

melden.

%Loesung Aufgabe 9b)
trace(weg/4,[call,redo,exit]), trace(kante,[call,exit]).

Aufgabe 10

a) Schreibe ein Prolog-Praedikat,

```
element(+Atomarer Term,+Liste von atomaren Termen)
```

mit dem man feststellen kann, ob ein Term (hier: = Atomarer Term oder Liste) in einer Liste von atomaren Termen vorkommt.

Sie können annehmen, daß die Eingabeliste nur atomare Terme enthaelt.

```
% A10 a)
% element(+atomic term,-list of atomic terms).

element(X,[X|_Atome]) :- atomic(X).
element(X,[_|Atome]) :- element(X,Atome).

/* Beachte: das liefert bei der Frage ?- element(X,[a,b,a]) zweimal
   die Antwort X=a. Aber hierbei hat die Eingabe einen nicht-atomaren
   Term, die Variable X. Und da war nicht genau gesagt, ob dieselbe
   Lösung mehrfach genannt werden darf.

   Wenn man erzwingen will, daß nur Anfragen mit atomaren Eingabetermen
   gestellt werden, kann man es bei diesen W-Fragen (statt Ja/Nein-Fragen)
   scheitern lassen:

   element(X,[Atom|Atome]) :-
       ( atomic(X), X=Atom          % Mit: (If -> Then ; Else)
       -> true
       ; element2(X,Atome) ).

*/
```

b) Schreiben Sie ein Praedikat

```
difference(+Liste1,+Liste2,-Ergebnis),
```

das bei Eingabe zweier Listen von atomaren Termen, Liste1 und Liste2, im dritten Argument als Ergebnis eine Liste der Elemente ausgibt, die in Liste1 vorkommen, aber nicht in Liste2.

% Loesung A10 b)

```
lt(X,Y):-var(X);var(Y).
lt(X,Y):-nonvar(X),nonvar(Y),X<Y.

difference([],S,[]).
difference(S,[],S) :- S\=[].
difference([X|TX],[X|TY],TZ):-
  difference(TX,TY,TZ).
difference([X|TX],[Y|TY],[X|TZ]):-
  lt(X,Y),
  difference(TX,[Y|TY],TZ).
difference([X|TX],[Y|TY],TZ):-
  lt(Y,X),
  difference([X|TX],TY,TZ).
```

c) Schreibe ein Praedikat

```
insert(+Liste atomarer Terme, +atomarer Term,-erweiterte Liste)
```

das einen Term in eine Liste einfügt, wenn er darin noch nicht vorkommt.

Hinweis: Vergleiche den Term mit dem ersten Element der Liste ...

```
%Loesung A10 c)
%Nach dem Semikolon sind also keine erfolgreichen Lösungen mehr vorhanden.
%Dennoch haben wir damit ein Backtracking und somit unnötige Bearbeitungszeit
   ausgelöst.
%Um das zu vermeiden, können wir das cut-Symbol wie folgt verwenden:

insert(X,[ ],[X]) :- !.
insert(X,[H|T],[X,H|T]) :- not(X = H), !.
insert(X,[H|T],[H|T1]) :- X = H, !, insert(X,T,T1).

% Das cut-Symbol in der dritten Klausel ist nicht unbedingt notwendig, da es ohnehin die
   letzte Klausel ist.
% Alle
```

d) Schreibe ein Praedikat

```
union(+Liste1, +Liste2,-erweiterte Liste)
```

das die beiden Listen `Liste1` und `Liste2` zu einer erweiterten Liste vereinigt.

```
%Loesung A10 d)
element(X,[X|_Atome]) :- atomic(X).
element(X,[_|Atome]) :- element(X,Atome).

union([],S,S).
union(S,[],S):-S\=[].
union([X|TX],[X|TY],[X|TZ]):-
  union(TX,TY,TZ).
union([XX|TX],[YY|TY],[XY|TZ]):-
  XX \= YY,
  not(element(XX,TY)),
  union(TX,[YY|TY],TZ).
union([XX|TX],[YY|TY],[YY|TZ]):-
  XX \= YY,
    not(element(XX,TY)),
  union([XX|TX],TY,TZ).
```