Logikprogrammieren Hausaufgaben

Blatt 3

Morten Seemann 6945442, Tore Wiedenmann 6488837

Aufgabe 1: Reimplementieren Sie die folgenden eingebauten Prädikate mit Hilfe von rekursiven Definitionen und vergleichen Sie das Verhalten Ihrer Implementierungen mit dem der eingebauten Prädikate:

numlist

```
%numlist(+Low, +High, -List)
% List is a list [Low, Low+1, ... High]. Fails if High < Low.
my_numlist(L,L,[L]).
my_numlist(L,H,List):- L<H,
                       L1 is L+1,
                       List=[L|X],
                       my_numlist(L1,H,X).
```

Sowohl das Eingebaute Prädikat als auch unseres benötigt eine Instanziierung der Ersten beiden Variablen. Sie können also als Erzeuger der Liste von L bis H benutzt werden oder als Test ob eine solche Liste korrekt ist. Nicht jedoch als Anfrage aus welchen Zahlen diese erzeugt wurde.

nth1

```
% nth1(?Index, ?List, ?Elem)
% Is true when Elem is the Index'th element of List. Counting starts at 1.
my_nth1(1,[H|_],H).
my_nth1(Index,[_|T],Element):-Index>1,
                              I1 is Index-1,
                              my_nth1(I1,T,Element).
```

Hier verhalten sich ebenfalls beide Prädikate gleich. Sie können beide als boolsche Funktion genutzt werden oder aber eine uninstanziierte Liste mit einem Gegebenen Element an gegebener Stelle als Antwort geben.

ord union

```
% ord_union(+Set1, +Set2, ?Union)
%Union is the union of Set1 and Set2

my_ord_union(List1,[],List1).
my_ord_union(List1, [H|T], [H|Out]):-my_ord_union(List1,T,Out).
```

Unser Prädikat schreibt stets die Elemente des Zweiten Arguments zuerst in die Union, in der gleichen Reihenfolge wie im Input. Das Offizielle Prädikat folgt irgendeiner mir nicht einsichtlichen(siehe Bsp) Logik um die Elemente der Listen zu vergleichen und erzeugt daraus eine umsortierte Union beider Listen. Beide Prädikate können sowohl eine Union erzeugen als auch herausfinden aus was für listen eine angegeben Union enstanden sein könnte.

```
%BSP
?- ord_union([aa1,a1,b1],[b2,a,a3,aa,aa2],L).
L = [aa1, a1, b1, b2, a, a3, aa, aa2].
% Hier funktioniert es wie unser Praedikat, und


```

```
% meistens sortiert er zahlen vor buchstaben,
?- ord_union([a,b,c],[1,2,3],L).
L = [1, 2, 3, a, b, c].

?-  ord_union([a,a1,d1],[aa1,11,a],L).
L = [a, a1, aa1, 11, a, d1].
%aber hier hat es auf einmal mittendrin aufgegeben?!

%Die Dokumentation ist uebrigens auch nicht gerade hilfreich...
% "Union is the union of Set1 and Set2"
```

Aufagbe 2: Gegeben sei eine binäre Zahlrepräsentation als inverse Liste, bei der die niedrigwertigste Binärstelle am Listenanfang steht.

```
%Den Brauchen wir ein paar mal fuer das zweite Praedikat, deshalb vorweg 5.)
umrechner([],X,_,X).
umrechner([H|T],Akk,Count,N):- Akk1 is Akk + (2**Count)*H,
                               Count1 is Count +1,
                               umrechner(T,Akk1,Count1,N).

%Binary -> Zahl konvertierer
convert(0,[]).
convert(X,N):- umrechner(X,0,0,N).

%Zahl -> Binary mit LSB als erstes.
convert2(0,[]).
convert2(N,Bin):- Odd is N mod 2,
                  Divisor is N//2,
                  convert2(Divisor,Tail),
                  Bin=[Odd|Tail].

%Test (convert2(X,BinX),convert(BinX,X) = True ) gdw wenn die funktionen korrekt sind.
?- umrechner2(27,X),convert(X,N).
X = [1, 1, 0, 1, 1],
N = 27 .
```

1.) Implementieren Sie zwei Prädikate, die fur eine gegebene Binärzahl ermitteln, ob diese gerade bzw. ungerade ist.

```
%Variante 1
is_odd([1|_]).
%Variante 2
is_odd2(X):- convert(X,Y), 1 is Y mod 2.
```

2.) Implementieren Sie zwei Prädikate zum Verdoppeln des Wertes einer Binärzahl bzw. zum Halbieren, falls der Wert der Binärzahl geradzahlig ist.

```
%Variante 1
double(X,[0|X]).
%Variante 2
double2(0,0).
double2(X,Y):- convert(X,A), B is 2*A, convert2(B,Y).
```

3.) Implementieren Sie ein Prädikat, das eine ungerade Binärzahl größer als eins durch Subtraktion von eins geradzahlig macht.

```
minus_eins([1],[]).
minus_eins([1|T],[0|T]).
```

4.) Implementieren Sie ein Prädikat, mit dem man testen kann, ob eine Binärzahl vorliegt. Vernachlässigen Sie dabei (vorerst) das Verbot führender Nullen.

```
%Abbruchbedingung
is_binary([]).
%X ist binary iff  X ist in {1,0}^n
is_binary([H|T]):- H=1,is_binary(T).
is_binary([H|T]):- H=0,is_binary(T).
```

5.) Definieren Sie ein funfstelliges Prädikat zur Addition zweier Einzelbits (Volladdierer).

```
%Logic Gates
or(A,B,1):- A+B>0.
or(A,B,0):- not(or(A,B,1)).

and(A,B,1):- A+B=:=2.
and(A,B,0):- not(and(A,B,1)).

xor(A,B,1):- A+B=:= 1.
xor(A,B,0):- not(xor(A,B,1)).

%Halbaddierer
halbaddierer(X,Y,C,S):- and(X,Y,C),xor(X,Y,S).

%Volladdierer
volladdierer(X,Y,Cin,Cout,S):- halbaddierer(X,Y,C1,S1),
                               halbaddierer(Cin,S1,C2,S),
                               or(C1,C2,Cout).
```

7.) Implementieren Sie auf der Basis des Prädikats aus Teilaufgabe 6 ein Prädikat zur Addition beliebig großer Binärzahlen in Listenrepräsentation.

```
%Wenn der letzte Carry 1 ist muessen wir noch eine Stelle ergaenzen
   Ansonsten sind wir fertig.
add([],[],0,[]).
add([],[],1,[1]).
%Wir Addieren die Bits an der Aktuellen Stelle und geben den Cout weiter
   and die naechste.
add([H1|T1],[H2|T2],Car1,SUM):- volladdierer(H1,H2,Car1,Car2,Out),
                                add(T1,T2,Car2,Out2),
                                SUM=[Out|Out2].

%Wir addieren und starten mit Carry 0.
%Addiert zwei Gleich lange binaerzahlen.
addiere(X,Y,S):- length(X,N),length(Y,N),add(X,Y,0,S).
%Wenn die Eine laenger ist als die andere Stocken wir mit ausreichnend 0'
   en am ende auf und addieren dann
addiere(X,Y,S):- length(X,N1),length(Y,N2),
                 N1>N2,
                 Diff is N1-N2,
                 aufstocken(Y,Diff,Yneu),
                 addiere(X,Yneu,S).
addiere(X,Y,S):-length(X,N1),length(Y,N2),N2>N1,addiere(Y,X,S).
```

```
19
20 aufstocken(X,0,X).
21 aufstocken(X,N,E):- N>0,
22                      N1 is N-1,
23                      append(X,[0],X1),
24                      aufstocken(X1,N1,E).
```

8.) Definieren Sie ein Prädikat zur Multiplikation von Binärzahlen nach dem Prinzip der russischen Bauernmultiplikation.

```
1  rusbaumult(X,[1],X).
2  rusbaumult(X,Y,E):- convert(Y,N),
3                      N>1,
4                      is_odd(Y),
5                      minus_eins(Y,Yneu),
6                      rusbaumult(X,Yneu,E1),
7                      addiere(E1,X,E).
8
9  rusbaumult(X,Y,E):- convert(Y,N),
10                     N>1,
11                     not(is_odd(Y)),
12                     double(X,Xneu),
13                     double(Yneu,Y),
14                     rusbaumult(Xneu,Yneu,E).
```

9.)

```
1 rus2(X,[1],X).
2 rus2(X,[1|T],E):- rus2(X,[0|T],E1),
3                   addiere(E1,X,E).
4 rus2(X,[0|T],E):- rus2([0|X],T,E).
```