Logikprogrammieren Hausaufgaben

Blatt 3

Morten Seemann 6945442, Tore Wiedenmann 6488837

Aufgabe 1: Definieren Sie ein Prädikat, das Pi mit einer vorgegebenen Anzahl von Rekursionsschritten approximiert. Implementieren Sie Ihr Prädikat in zwei Varianten.

1.)

```
%Beim Rekursiven Abstieg
leibnitz(1,1).
leibnitz(N,PI) :- N>1,
                  N1 is N-1,
                  leibnitz(N1,P1),
                  PI is P1 + ((-1)^(N+1))/(2*N -1).

pi(N,PI) :- leibnitz(N,X),PI is 4*X.

%Beim Rekursiven Aufstieg
leibnitz_er(N,PI) :- leibnitz_end(N,1,PI).
leibnitz_end(1,X,X).
leibnitz_end(N,AKK,PI) :- N>1,
                          N1 is N-1,
                          AKK1 is AKK + ((-1)^(N+1))/(2*N -1),
                          leibnitz_end(N1,AKK1,PI).

pi_er(N,PI) :- leibnitz_er(N,X),PI is 4*X.

%Letzteres ist die Endrekursive Variante, da der Rekursive-Aufruf als
   letztes im Rekursionsschrit passiert.
```

2.)Vergleichen Sie Ihre Definitionen im Hinblick auf die Verständlichkeit und das Berechnungsverhalten.

Ich finde das die Nicht-endrekursive Variante grundsätzlich verständlicher ist. Bei dieser ist deutlicher die Leibnitz-Reihe zu sehen. Die notwendigkeit des wrappers und und die Abbruchbedingung der Endrekursiven Struktur in Prolog nehmen m.M.n. der Endrekursion die Eleganz. (Rechenzeit Vergleich in 3.))

3.)Implementieren Sie eine alternative Variante zur Berechnung von Pi mit Hilfe der Wallis'schen Formel:

```

wallisch(1,4/3).
wallisch(N,PI):- N>1,
                 N1 is N-1,
                 wallisch(N1,P1),
                 X is (2*N)/(2*N - 1),
                 Y is (2*N)/(2*N +1),
                 PI is P1*X*Y.

pi_wall(N,PI) :- wallisch(N,X),PI is 2*X.


```

```
wall_end(1,X,X).
wall_end(N,AKK,PI):- N>1,
                     N1 is N-1,
                     X is (2*N)/(2*N - 1),
                     Y is (2*N)/(2*N +1),
                     AKK1 is AKK*X*Y,
                     wall_end(N1,AKK1,PI).

pi_wall_er(N,PI):- wall_end(N,4/3,X),PI is X*2.
```

Vergleichen Sie Genauigkeit und Rechenzeitbedarf mit den Implementationen aus Teilaufgabe

```
?- time(pi_wall(100000,PI)).
% 499,997 inferences, 17.797 CPU in 17.892 seconds (99% CPU, 28095 Lips)
PI = 3.141584799656923 .

?- time(pi(100000,PI)).
% 299,998 inferences, 6.031 CPU in 6.078 seconds (99% CPU, 49741 Lips)
PI = 3.1415826535897198 .

?- time(pi_er(100000,PI)).
% 299,999 inferences, 0.219 CPU in 0.218 seconds (100% CPU, 1371424 Lips)
PI = 3.1415826535896922 .

?- time(wall_er(100000,PI)).
% 499,996 inferences, 0.328 CPU in 0.334 seconds (98% CPU, 1523797 Lips)
PI = 3.14158479965707 .
```

Aufgabe 2: Stromorientierte Verarbeitung 1.) Implementieren Sie auf der Basis des Rekursionsschemas fur die stromorientierte Verarbeitung inkrementelle Varianten der Prädikate aus Aufgabe 1.

```
%Leibnitz
leib_pi_incr_viertel(1,1).
leib_pi_incr_viertel(N,PI):- leib_pi_incr_viertel(N1,P1),
                     N is N1+1,
                     PI is (P1 + ((-1)^(N+1))/(2*N -1)).

leib_pi_incr(N,PI):- leib_pi_incr_viertel(N,X),PI is 4*X.

%Wallis
wall_pi_incr_halbe(1,4/3).
wall_pi_incr_halbe(N,PI):- wall_pi_incr_halbe(N1,P1),
                           N is N1+1,
                           X is (2*N)/(2*N - 1),
                           Y is (2*N)/(2*N +1),
                           PI is P1*X*Y.

wall_pi_incr(N,PI):- wall_pi_incr_halbe(N,X),PI is 2*X.
```

2.) Analysieren Sie das Konvergenzverhalten der beiden Berechnungsvorschriften für Pi:

Sowohl die Leibnitzformel als auch das Wallische Produkt sind nicht sehr gut zur Approximation geeignet. Ersteres konvergiert ungefähr logarithmisch während letzteres fast linear im Fehlerterm gegen 0 konvergiert. Insgesammt ist das Wallische Produkt stets etwas schneller als die Leibnitzformel.

Aufgabe 3 : Wir betrachten rekursiv eingebettete Strukturen (Binärbäume), die nur aus zweistelligen Strukturen mit dem Funktor s bzw. aus Namen zusammengesetzt sind.

```
% 1.) Definieren Sie einen Typtest fuer derartige Strukturen
s(a,b).
s(X,s(Y,Z)):- atom(X),s(Y,Z).
s(s(X,Y),Z):- s(X,Y),atom(Z).
s(s(X,Y),s(V,W)):- s(X,Y),s(V,W).

%TESTS
?- s(s(s(a,b),b),c).
true .
?- s(s(a,b),c).
true .
```

2.) Definieren Sie ein nicht-endrekursives und ein endrekursives Prädikat, das für einen Binärbaum die lokalen Einbettungstiefen auf allen Pfaden vom Wurzelknoten zu den einzelnen Blattknoten als alternative Variablenbindungen ermittelt.

```
%Nicht-endrekursiv

%Blaetter haben Rest-tiefe 0
tiefe(X,0):- atom(X).
%Bei  Baum = s(X,Y) ist die Tiefe von der Wurzel:
% 1 plus die Tiefe, die von X
tiefe(s(X,_),N):- tiefe(X,N1),N is 1+N1.
% oder von Y aus erreicht werden kann.
tiefe(s(_,Y),N):- tiefe(Y,N1),N is 1+N1.

%endrekursiv
tiefe_er(X,N,N):-atom(X).

tiefe_er(s(X,_),AKK,N):- AKK1 is AKK+1,
                         tiefe_er(X,AKK1,N).
tiefe_er(s(_,Y),AKK,N):- AKK1 is AKK+1,
                         tiefe_er(Y,AKK1,N).
tiefe2(X,N):- tiefe_er(X,0,N).

%TESTS
?- tiefe(s( s(a,b),s(b,s(a,b))),N).
N = 2 ;
N = 2 ;
N = 2 ;
N = 3 ;
N = 3.

?- tiefe2(s( s(a,b),s(b,s(a,b))),N).
N = 2 ;
N = 2 ;
N = 2 ;
N = 3 ;
N = 3.
```

3.) Verwenden Sie das Prädikat aus Teilaufgabe 2, um die maximale Einbettungstiefe eines Binärbaums zu ermitteln.

```
max_tiefe(X,Max):- findall(N,tiefe(X,N),L),max_list(L,Max).

%TESTS
?- max_tiefe(s( s(a,b),s(b,s(a,b))),N).
N = 3.
```