Logisch programmeren 2012

Week 2: werken met accumulatoren

1. Iteratieve programma's zonder staartrecursie

Hieronder de recursieve definitie van vermenigvuldiging voor natuurlijke getallen in successor notatie.

$$\begin{aligned} 0 \cdot a &= 0 \\ s(a) \cdot b &= (a \cdot b) + b \end{aligned}$$

We kunnen deze definitie opschrijven in de vorm van een prolog programma voor times/3, waarbij we add/3 gebruiken.

```
times(0,_,0).
times(s(X),Y,Z) :- times(X,Y,W),add(W,Y,Z).
```

Dit programma definieert vermenigvuldigen als herhaald optellen. Het programma is niet erg efficiënt. De recursieve aanroep van times/3 is de eerste body literal: er wordt niets uitgerekend tot times/3 bij het grensgeval times(0,_,0) is aanbeland.

2. Accumulatoren en staartrecursie

Voor een iteratieve programmeerstijl kan je in prolog gebruik maken van een accumulator variabele, die de tussentijdse resultaten van de iteratie doorgeeft. Zie het programma voor times/4 hieronder. times(X,Y,Acc,Product) rekent het Product uit van X en Y met behulp van accumulator Acc.

```
times(s(X),Y,A,Z) :- add(Y,A,A1),times(X,Y,A1,Z).
times(0,_,A,A).
```

Voor de initiële aanroep van times/4 stellen we de accumulator gelijk aan 0. Elke stap van de recursie maakt een tussentijdse som aan en geeft die door via de accumulator variabele. Als het grensgeval wordt bereikt (vermenigvuldigen met zero) is de accumulator gelijk aan de gezochte eindwaarde van het product.

Je kan een *wrapper* maken, waarmee je times/3 definieert in termen van times/4. De wrapper definitie vult de goede startwaarde voor de accumulator alvast in.

```
times(X,Y,Product):-times(X,Y,0,Product).
```

3. Peano naar decimaal

Nog een simpel voorbeeld: omrekenen van Peano naar decimale notatie.

Eerst een linksrecursieve versie.

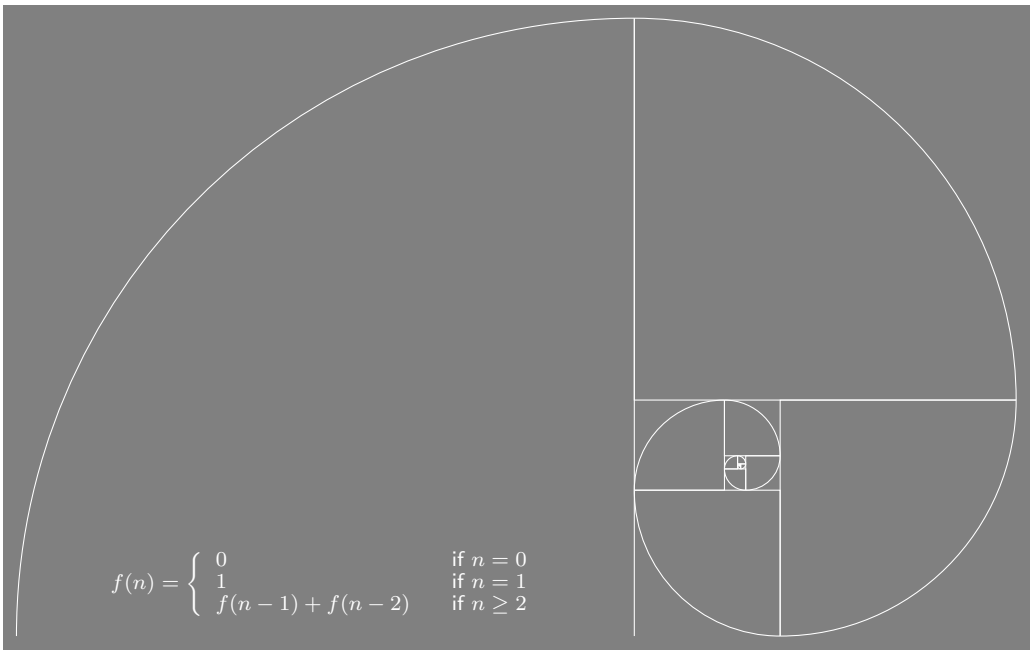
```
s2d(0,0).
s2d(s(N),D) :- s2d(N,D0), D is D+1.
```

Nu een staartrecursieve versie met accumulator, en wrapper.

```
s2d(Peano,Dec) :- s2d(Peano,0,Dec).
```

```
s2d(0,A,A).
s2d(s(N),A,D) :- A1 is A+1, s2d(N,A1,D).
```

4. Fibonacci spiraal



5. Fibonacci getallen

De Fibonacci getallen vormen de volgende reeks: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377,... Na de beginwaarden 0 en 1 is elk volgend getal in de reeks de som van de vorige twee.

$$F_0 = 0 \quad ; \quad F_1 = 1 \quad ; \quad F_n = F_{(n-1)} + F_{(n-2)}$$

De code voor fibo/2 zet de definitie om in Prolog. We schrijven fibo(N,F) als F het n-de Fibonacci getal is.

```
fibo(0,0).
fibo(1,1).
fibo(N,F) :- N > 1,
  N1 is N-1, N2 is N-2,
  fibo(N1,F1), fibo(N2,F2),
  F is F1+F2.
```

Door de dubbele recursie is de code hopeloos inefficiënt. Overtuig je daarvan met trace/0. Probeer fibo(15,F).

6. Fibonacci getallen: accumulator versie

fib(N,F1,F2,F) slaagt als F het N-de Fibonacci getal is; F wordt berekend met behulp van accumulator argumenten voor het vorige (F1) en voor-vorige (F2) Fibonacci getal.

```
fib(0,0).
fib(1,1).
fib(N,F) :- N > 1,
   fib(N,1,0,F).    % beginwaarde voor de accumulatoren

fib(2,F1,F2,F) :-   % grensgeval voor fib/4: N=2
   F is F1 + F2.
fib(N,F1,F2,F) :- N > 2,
   N1 is N-1,
   NF1 is F1 + F2,
   fib(N1,NF1,F1,F).
```

Vergelijk m.b.v. trace/0 de werking van fib/2 met die van fibo/2. Probeer bijvoorbeeld fib(1000,F).