Logisch programmeren 2012

Week 6

Deze oefeningen gaan over het werken met verschillijsten en over het gebruikt van de DCG pijlnotatie `-->/2` waarmee de verschillijst argumenten automatisch ingevuld worden.

1

Een palindroom (of: spiegelwoord) is een rijtje dat je van links naar rechts of van rechts naar links kan lezen met hetzelfde resultaat. Een voorbeeld is de begroeting bij de eerste ontmoeting van Adam en Eva:

`[m,a,d,a,m,i,m,a,d,a,m]` ("Madam, I'm Adam")

Een recursieve definitie zou kunnen luiden: (i) het lege rijtje of één enkele letter is een palindroom, (ii) een palindroom is een rijtje waarvan de eerste en de laatste letter hetzelfde zijn, met daartussen een palindroom. Naieve code voor `palindroom/1` gebruikt `append/3`.

```
palindroom([]).
palindroom([_]).
palindroom([A|T]) :-
    append(Midden,[A],T),
    palindroom(Midden).
```

1.1 Opgave

Geef een definitie voor `palindroom/2` die van verschillijsten gebruik maakt, en waarbij `append/3` geen rol meer speelt. De wrapper is

```
palindroom(Lijst) :- palindroom(Lijst,[]).
```

1.2 Opgave

Schrijf je definitie voor `palindroom/2` uit §1.1 nu om naar de DCG pijlnotatie `-->/2`.

2

Hieronder de definitie van `post/3` voor het omzetten van formules in infixnotatie naar de Poolse postfixnotatie. De definitie maakt gebruik van verschillijsten.

```
:-op(400,fy,'~'). % negation
:-op(500,xfy,'=>'). % implication

prop(A) :- atom(A),
    atom_codes(A,[C]),
    between(97,122,C).

post(A,[A|L],L) :- prop(A).
post(~A,P,P1) :- post(A,P,['N'|P1]).
post(A/\B,P,P1) :- post(A,P,P0),post(B,P0,['K'|P1]).
post(A\/B,P,P1) :- post(A,P,P0),post(B,P0,['A'|P1]).
post(A=>B,P,P1) :- post(A,P,P0),post(B,P0,['C'|P1]).
```

2.1 Opgave

Schrijf de definitie om naar DCG formaat. Noem het predicaat `post_dcg` om het te onderscheiden van `post`. Denk erom dat goals die geen verschillijst argumenten hebben in de regels voor `'-->'` verschijnen tussen accolades.

Vergelijk de originele code voor `post/3`, en het resultaat van toevoegen van verschillijstargumenten aan je DCG definitie voor `post_dcg/3`:

```
?- listing(post/3), listing(post_dcg/3).
```

3 Opgave

Hieronder de `quicksort/3` definitie van Week 3 met verschillijsten.

```
quicksort( [], L, L).
quicksort( [X | Tail], L, L1)  :-
   split( X, Tail, Small, Big),
   quicksort( Small, L, [X | L0] ),
   quicksort( Big, L0, L1).
```

Schrijf de definitie om naar DCG formaat.

4

In het eerste deel van de cursus hebben we voorbeelden gezien van de accumulator hulpvariabele. Waar die hulpvariabele wordt gebruikt om een lijst op te bouwen, is de accumulatortechniek erg verwant met het gebruik van verschillijsten!

4.1 Opgave

Hieronder uit *LPN* de accumulator code voor een predicaat `accRev/3` waarmee je een lijst omkeert, en bijhorende wrapper.

```
rev(L,R) :- accRev(L,[],R).
accRev([H|T],A,R) :- accRev(T,[H|A],R).
accRev([],A,A).
```

Schrijf de definitie voor `accRev/3` om tot een DCG definitie voor een predicaat `rev/3`. Tip: de accumulator definitie heeft de verschillijst argumenten in de omgekeerde volgorde.

4.2 Opgave

Onze notatie voor getallen is een *positiestelsel*: een natuurlijk getal wordt uitgedrukt als een reeks van producten van machten van een grondtal b:

$$n_k \cdots n_2 n_1 n_0 = n_k b^k + \cdots + n_2 b^2 + n_1 b^1 + n_0 b^0$$

waarbij de n_i natuurlijke getallen kleiner dan het grondtal of nul zijn. Bijvoorbeeld: voor het grondtal 10 kan je 125 lezen als

$$125_{10} = 1 \times 10^2 + 2 \times 10^1 + 5 \times 10^0$$

Voor het omrekenen van een natuurlijk getal n_{10} (n in decimale notatie) naar m_b voor grondtal b is er een eenvoudig algoritme. Deel n door b: de rest is het *laatste* cijfer van m; voor het voorlaatste cijfer herhaal je met het naar beneden afgeronde resultaat van n gedeeld door b; ga zo door tot het resultaat van verder delen nul wordt.

Definieer een predicaat `powers/3` waarmee je dit algoritme implementeert. Enkele voorbeeldaanroepen:

```
?- powers(11,10,Powers). % 11 basis 10
Powers = [1, 1]. % 11 = 1*(10^1) + 1*(10^0)

?- powers(11,2,Powers). % 11 basis 2
Powers = [1, 0, 1, 1]. % 1*(2^3) + 0*(2^2) + 1*(2^1) + 1*(2^0)
```

Maak gebruik van een accumulator, zoals je dat voor de definitie van het omkeren van een lijst hebt gedaan. Definieer dus `powers/4` en gebruik de wrapper hieronder:

```
powers(N,Base,Powers) :-  % wrapper
    N > 0,
    powers(N,Base,[],Powers). % beginwaarde accumulator: []
powers(0,_,[0]).
```

4.3 Opgave

Schrijf nu je definitie voor `powers/4` om naar de DCG notatie met `'-->'`. Zoals bij `rev/3` heeft de accumulator definitie de verschillijst argumenten in de omgekeerde volgorde.

5

Schrijf een DCG die de dd/mm/jjjj notatie voor een datum vertaalt in het Nederlands. Bijvoorbeeld: 30/6/1954 wordt 'dertig juni negentienhonderdvierenvijftig'. Hoofdpredicaat wordt `datum/5`: in `datum(Dag,Maand,Jaar,Lijst,Rest)` staan de argumenten `Dag`, `Maand`, `Jaar` voor de numerieke datumaanduiding, `Lijst, Rest` is een verschillijst voor de vertaling in het Nederlands.

Het is handig om `Dag` en `Jaar` om te zetten naar de lijstnotatie van `powers/3` van de vorige opdracht. De wrapper kan er dan zo uitzien:

```
datum(Dag,Maand,Jaar,NL) :-
    powers(Dag,10,DD),
    powers(Jaar,10,YYYY),
    datum(DD,Maand,YYYY,NL,[]).
```

In je definitie voor `datum/5` zal je een aantal hulppredicaten willen gebruiken voor de verwerking van de dag-, maand- en jaaraanduidingen. Schrijf die hulppredicaten ook in DCG formaat. Mogelijke deelqueries:

```
?- dag([1,9],L,[]).
L = [negen,tien].

?- maand(6,L,[]).
L = [juni].

?- jaar([1,9,4,5],L,[]).
L = [negen,tien,honderd,vijf,en,veertig].
```

Hints Een aantal getallen zal je als *feiten* willen coderen: 0..9, 10..14, tientallen, honderd, duizend. Voor de regels zijn er algemene gevallen, en uitzonderingen: 1302 wordt 'dertienhonderd en twee' maar 2002 niet 'twintig honderd en twee'. Gebruik voor het inperken van de gevallen waarop een regel van toepassing is niet de vergelijkingspredicaten `>,<` maar het ingebouwde `between/3` (voorbeeld: `between(2,9,N)` voor het vinden van waarden voor `N` in het interval 2..9). Met `between/3` blijft je code voor `datum/5` omkeerbaar:

```
?- datum([8],6,[1,9,5,4],NL,[]).
NL = [acht,juni,negentien,honderd,vier,en,vijftig].
?- datum(D,M,J,[acht,juni,negentien,honderd,vier,en,vijftig],[]).
D=[8], M=6, J=[1,9,5,4].
```

Testen Met `test/0` hieronder kan je je code voor `datum/4` testen. Stuur de uitvoer van `test/0` mee met je uitwerking (tussen `/*` ... `*/` commentaartekens).

```
test :- d(D,M,J),
    (datum(D,M,J,NL)
    ->
    format("~d/~d/~d: ~w\n",[D,M,J,NL])
    ;
    format("~d/~d/~d: geen vertaling\n",[D,M,J])),
    fail.
test.

% data
d(24,8,79).
d(25,12,800).
d(26,6,1321).
d(11,7,1302).
d(5,5,1945).
d(12,4,1961).
d(20,12,2006).
d(14,7,1789).
d(25,8,1830).
d(22,2,1900).
```

□