Logisch programmeren 2012

Opdrachten Week 3

1

Maak de oefeningen bij hoofdstukken 4 en 6 van *LPN*. Check je antwoorden met de Oplossingen achter in het boek. Deze opdracht hoef je niet in te leveren.

2 Alle n-tupels

In deze opgave kijken we naar manieren om alle k^n woorden van lengte n te maken, gegeven een alfabet van k symbolen. Bijvoorbeeld: alle 2^n rijtjes die bestaan uit n bits (0 of 1). Voor $n = 3$ zijn dat $2^3 = 8$ rijtjes: $000, 001, 010, 011, 100, 101, 110, 111$.

Definieer een predicaat `word/3`. `word(N, Alfabet, Rijtje)` slaagt als `Rijtje` een lijst is met lengte `N`, gebouwd uit symbolen behorend tot een lijst `Alfabet`. Bijvoorbeeld:

```
?- word(3,[0,1],L),write(L),tab(2),fail.
[0,0,0]  [0,0,1]  [0,1,0]  [0,1,1]  [1,0,0]  [1,0,1]  [1,1,0]  [1,1,1]
false.
```

Zorg ervoor dat je antwoorden worden opgesomd in 'lexicografische volgorde' (geordend van klein naar groot zoals in een woordenboek). Je kan het ingebouwde `member/3` predicaat gebruiken.

3 Gray code

In situaties waar analoge naar digitale informatie moet worden omgezet (of omgekeerd) zijn er meer foutbestendige manieren om alle 2^n rijtjes van n bits te genereren. De binaire Gray code geeft een opsomming van de rijtjes waarbij er telkens slechts één bit verandert. De Gray code voor $n = 3$ is $000, 001, 011, 010, 110, 111, 101, 100$. Vergelijk die met de lexicografische volgorde hierboven.

3.1 Vertalen

Gegeven een geheel getal k met binaire representatie $(b_n \dots b_2 b_1 b_0)_2$ kan je op de volgende manier het corresponderende tupel $(a_n \dots a_2 a_1 a_0)_2$ van de Gray code berekenen:

$$\begin{aligned} a_j &= b_j \oplus b_{j+1} \quad \text{voor } n > j \geq 0 \\ a_n &= b_n \end{aligned} \tag{1}$$

waarbij $\oplus$ staat voor het exclusieve 'of':

$\oplus$	0	1
0	0	1
1	1	0

Definieer een predicaat `bin2gray/2` waarmee je deze omzetting uitvoert. De aanroep `bin2gray(Bin,Gray)` slaagt als `Bin` de binaire representatie is van een geheel getal en `Gray` het daarmee corresponderende tupel van bits in de Gray code. Hieronder als voorbeeld de omrekening van 5 in binaire notatie (101).

```
?- bin2gray([1,0,1],Gray).
Gray = [1, 1, 1] ;
false.
```

Hints. De recursieve definitie (1) zal je willen omzetten in Prolog lijstrecursie. Recursie op een Prolog lijst werkt van voor naar achter. Om volgens het recept (1) de bit a_j te vinden moet je 'achterstevoren' b_j met b_{j+1} vergelijken. Voor dit probleem is het handig om met een wrapper te werken:

```
bin2gray(Bin,Gray) :-
     reverse(Bin,RevBin),
     bin2gray_aux(RevBin,RevGray),
     reverse(RevGray,Gray).
```

Hier is `reverse/2` het ingebouwde predicaat om een lijst om te draaien. Exclusief 'of' ($\oplus$) in Prolog is het rekenpredicaat `xor`. Je kan dus constructies van de vorm `X is 0 xor 1` gebruiken.

3.2 Gray code: recursieve definitie

In de vorige opgave bereken je voor een gegeven getal in binaire representatie het corresponderende tupel van de Gray code, zoals de aanroep hieronder laat zien.

```
?- word(3,[0,1],L),bin2gray(L,G),write((L,G)),nl,fail.
[0,0,0],[0,0,0]
[0,0,1],[0,0,1]
[0,1,0],[0,1,1]
[0,1,1],[0,1,0]
[1,0,0],[1,1,0]
[1,0,1],[1,1,1]
[1,1,0],[1,0,1]
[1,1,1],[1,0,0]
false.
```

Er is ook een recursieve definitie die rechtstreeks de reeks $n+1$-tupels van de Gray code berekent op basis van de reeks n-tupels. In de definitie (2) hieronder staat ε voor het lege rijtje. $0\Gamma_n$ is de reeks n-tupels Γ_n met vooraan een 0 toegevoegd aan elk rijtje van die reeks, $1\Gamma_n^R$ is de reeks Γ_n in omgekeerde volgorde, met vooraan een 1 toegevoegd aan elk rijtje van die omgekeerde reeks. De komma staat voor concatenatie.

$$\begin{aligned} \Gamma_0 &= \varepsilon \\ \Gamma_{n+1} &= 0\Gamma_n, 1\Gamma_n^R \end{aligned} \tag{2}$$

Zet (2) om in Prolog code: definieer een predicaat `gray/2`, waarbij `gray(N,Lijst)` voor een lengte `N` de reeks `Lijst` van Gray codes genereert. Bijvoorbeeld:

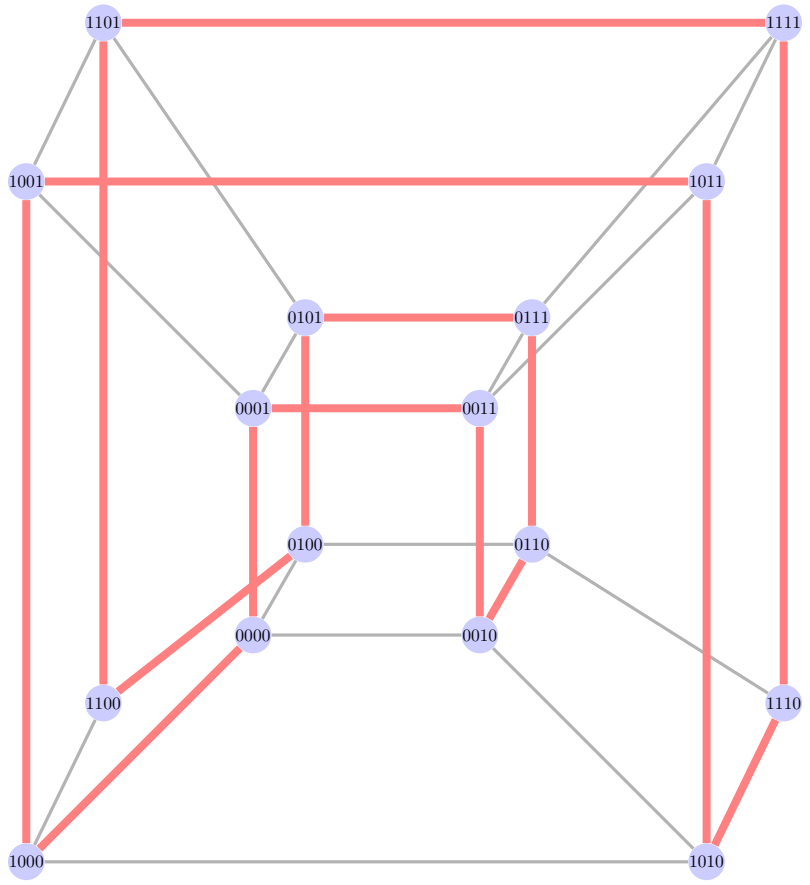
```
?- gray(3,Lijst).
Lijst = [[0,0,0],[0,0,1],[0,1,1],[0,1,0],[1,1,0],[1,1,1],[1,0,1],[1,0,0]].
```

Je definitie heeft, zoals gebruikelijk, een grensgeval en een recursief geval. Het grensgeval voor $n = 0$ vind je hieronder. Het recursieve geval mag je zelf uitwerken.

```
gray(0,[[]]).
gray(N,L) :- N>0, ...
```

Je kan opnieuw het ingebouwde `reverse/2` gebruiken voor het omkeren van een lijst. En je zal een hulpdefinitie `prefix/3` moeten vinden, waarbij `prefix(El,L,L1)` slaagt als `L1` de elementen van lijst `L` bevat met telkens vooraan `El` toegevoegd.

```
?- prefix(1,[[1,0],[1,1],[0,1],[0,0]],L).
L = [[1, 1, 0], [1, 1, 1], [1, 0, 1], [1, 0, 0]].
```



Figuur 1: Hamilton pad in de 4-kubus

4 Torens van Hanoi

De Gray code heeft vele verrassende toepassingen. Zo bepaalt de Gray sequentie van n-tupels een Hamiltoniaanse wandeling door een n-dimensionele hyperkubus: een pad dat alle knopen precies één keer bezoekt alvorens terug te keren naar het startpunt. Figuur 4 geeft een illustratie voor $n = 4$ met het pad $0000, 0001, 0011, 0010, \ldots$

De Gray code levert ook de sleutel voor het oplossen van de bekende 'Torens van Hanoi' puzzel. Die puzzel speel je met een aantal schijven van oplopende grootte verdeeld over drie stapels (links l, midden m, rechts r). In de beginsituatie liggen alle schijven op stapel l geordend naar grootte, met de grootste onderaan. De taak is om de schijven met dezelfde ordening op stapel r te krijgen. Je mag telkens één schijf verplaatsen, maar een grotere schijf mag nooit op een kleinere gelegd worden. Een optimale oplossing, voor n schijven, heeft $2^n - 1$ stappen.

Je kan de oplossing voor n schijven op de volgende manier aflezen van de reeks n-tupels van de Gray code.

1. De bit die verandert tussen twee opeenvolgende n-tupels geeft aan welke schijf je moet verplaatsen. De meest linkse bit is daarbij de grootste schijf, de meest rechtse de kleinste.
2. Behalve voor de kleinste schijf is de stapel waar die naartoe moet uniek bepaald door de voorwaarde dat een schijf nooit op een kleinere gelegd mag worden.
3. Bij verplaatsing van de kleinste schijf zijn er telkens twee mogelijke bestemmingen. Je kan de goede bestemming kiezen door een modulo-3 klok bij te houden. In het geval van een oneven aantal schijven is dat $rmlrml\ldots$; in het geval van een even aantal schijven $mrlmrl\ldots$

4.1 De torens in Prolog

Voor deze opdracht zet je de hierboven gegeven strategie om in Prolog. Je hoofdpredicaat wordt `towers/5`, met de volgende argumenten:

`towers(N,Gray,Acc,Stappen,Klok)`

`N`: het aantal schijven

`Gray`: de Gray sequentie voor `N`, m.a.w. `gray(N,Gray)`

`Acc`: een accumulator lijst om de opeenvolgende spelsituaties bij te houden

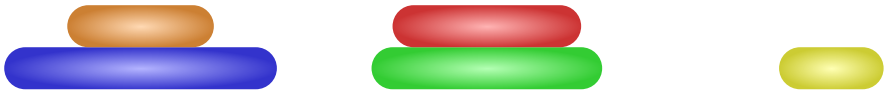
`Stappen`: de uiteindelijke reeks van stappen

`Klok`: het controle argument waarmee je de bestemming van de kleinste schijf bijhoudt (`[r,m,l]` voor oneven `N`, `[m,r,l]` voor even `N`)

Om je op weg te helpen, vind je hieronder alvast de definitie van `towers/5`.

```
towers(N,[H0,H1|T],[[L,M,R]|Rest],Result,Control) :-
    move(N,H0,H1,L,M,R,L1,M1,R1,Control,Control1),
    towers(N,[H1|T],[[L1,M1,R1],[L,M,R]|Rest],Result,Control1).
towers(_,[_],A,A,_).
```

Het grensgeval bereik je na $2^n - 1$ stappen als je bij het laatste tupel van de Gray sequentie bent aanbeland. De accumulator is dan de lijst van stappen die een oplossing vormen, met de laatste stap eerst. Je zal die lijst dus nog omkeren met je wrapper predicaat.



Figuur 2: Een spelsituatie voor $n = 5$

De recursieve regel kijkt naar twee opeenvolgende Gray code tupels `H0, H1` en naar de laatste spelsituatie `[L,M,R]`, waar `L, M, R` lijsten zijn met de inhoud van de linker, middelste en rechter stapel. Het predicaat `move/11` voert dan een zet uit. Vergelijking van de tupels `H0, H1` leert in welke bit ze verschillen: dat bepaalt welke schijf er verplaatst moet worden. (We nummeren schijven `1, 2, 3, ...` van klein naar groot.) De inhoud van de stapels `L, M, R` wordt omgezet in `L1, M1, R1`. Als schijf `1` (de kleinste) moet bewegen, inspecteer je het controle argument om de bestemming te weten; je roteert de klok een positie vooruit, voor eventuele volgende verplaatsingen van schijf `1`. Voor schijven groter dan `1` blijft het controle argument ongewijzigd: hun bestemming is uniek bepaald omdat ze niet op een kleinere schijf terecht mogen komen.

De routine `move/11` ga je zelf uitwerken. Schrijf ook een wrapper `towers/2`. Het predicaat `towers(N,Oplossing)` geeft de lijst van stappen `Oplossing` voor een spel met `N` schijven. De wrapper moet de goede beginwaarden geven voor de argumenten van `towers/5`, en het resultaat van `towers/5` omkeren om de oplossing in de goede volgorde te geven. Voor de 3 schijven puzzel moet de beginaanroep van `towers/5` er zo uit komen te zien:

```
towers(3,[[0,0,0],[0,0,1]|T],[[[1,2,3],[],[]]|Rest],Result,[r,m,l])
```

4.2 Visualisatie

In het startbestand vind je een predicaat `tikz_moves/1` om de puzzel te visualiseren. Bij een aanroep

```
?- towers(5,Oplossing),tikz_moves(Oplossing).
```

schrijft `tikz_moves/1` een bestand `towers.tex` naar je werkfolder. Met het commando `pdflatex towers` (in Terminal) kan je dat opmaken tot `towers.pdf`: een filmpje van je oplossing. In Figuur 2 zie je een situatie uit een spel met vijf schijven. (Zes schijven is het maximum voor de visualisatie.)

5 Incomplete datastructuren

5.1 Quicksort

Op de slides van donderdag vind je Prolog code voor `quicksort/3` die van verschillijsten gebruik maakt. Om de code werkend te krijgen moet er nog een predicaat `split/4` gegeven worden: `split(Sleutel,Lijst,Kleiner,GroterOfGelijk)` splitst `Lijst` in een lijst van elementen kleiner dan `Sleutel` en een lijst van elementen groter of gelijk aan `Sleutel`.

Definieer `split/4`. Maak gebruik van alfanumerieke vergelijking (`@<` enzovoort).

5.2 Pools

De Poolse notatie voor formules van de propositielogica schrijft de connectieven voor conjunctie, disjunctie, implicatie, negatie in prefixvorm. Een formule kan dan geschreven worden als een rijtje van symbolen; je hebt (als je de plaatsigheid van de connectieven kent) geen haakjes nodig om te disambigueren. Hieronder een vertaalsleutel (de gebruikte letters voor de connectieven zijn afgeleid van hun Poolse namen):

		POOLS
negatie	$\neg\phi$	Nϕ
conjunctie	$\phi \wedge \psi$	K$\phi\psi$
disjunctie	$\phi \vee \psi$	A$\phi\psi$
implicatie	$\phi \rightarrow \psi$	C$\phi\psi$

In Prolog schrijven we voor de binaire operaties `(Phi,Psi)`, `(Phi;Psi)`, en (let op) `(Psi:-Phi)`. Laten we negatie schrijven als `-(Phi)`.

Definieer een predicaat `pools/3` waarmee je logische formules (Prolog syntax) omzet in Poolse notatie. Gebruik atomen `p, q, r, ...` voor de propositieletters. De aanroep `pools(Formule,Lijst,Verschil)` slaagt als `Lijst, Verschil` een verschillijst paar is voor de vertaling van `Formule`. Hieronder de wrapper voor `pools/2` en een voorbeeld.

```
pools(Formule,Lijst) :- pools(Formule,Lijst,[]).

?- pools((p,(q;r)),L).
L = ['K', p, 'A', q, r].
```

□