Logisch programmeren 2012

Tentamen Deel 2

Bij elke opgave is het aantal punten dat je ermee kan verdienen opgegeven. Het totaal is 30 punten. Wie over een joker beschikt kan die inzetten voor een vraag ter waarde van 3 punten.

1 Structuur van termen (5 ptn)

Het ingebouwde `ground(Term)` slaagt als `Term` een Prolog term is die geen variabelen bevat. Definieer `ground/1`. Je kan de ingebouwde predicaten `atomic/1` en `compound/1` gebruiken om te testen met wat voor term je te doen hebt: `atomic/1` slaagt voor een atoom of een getal; `compound/1` voor een samengestelde term. Leg een samengestelde term met `=..` uiteen in zijn functor en de lijst van argumenten. Gebruik een hulppredicaat `ground_list/1` om te testen of elk van de argumenten variabele-vrij is.

Hieronder enkele voorbeeldaanroepen.

```
?- atomic(1).  ? -atomic(a).  ?-atomic(X).  ?- compound(X).  ?- compound(f(X)).
true.          true           false         false            true

?- f(a,g(X))=..List.
List = [f, a, g(X)].
```

2 DCG regels (5 ptn)

Definieer een predicaat `natrij/4` waarbij `natrij(Start,Eind,Lijst,Rest)` slaagt als de verschillijst `Lijst-Rest` een rij van natuurlijke getallen is oplopend van `Start` tot `Eind`. Bijvoorbeeld:

```
?- natrij(2,7,Rij,[]).     ?- natrij(7,2,Rij,[]).
Rij = [2,3,4,5,6,7]        false % 2 is kleiner dan 7
```

Geef je definitie in de vorm van een DCG die de verschillijstargumenten impliciet laat.

3 Lijstrecursie (3 ptn)

Definieer een predicaat `filter/3` waarbij `filter(N,In,Uit)` slaagt als `N` groter is dan nul en de lijst `Uit` het resultaat is van het verwijderen van de veelvouden van `N` uit `In`. Een voorbeeldaanroep:

```
?- filter(2,[1,2,3,4,5,6],L).
L = [1,3,5]
```

Je kan het ingebouwde rekenpredicaat `mod` gebruiken: `X is N mod M` bindt `X` aan de rest van delen van `N` door `M`.

4 Priemgetallen (3 ptn)

Een priemgetal is een natuurlijk getal dat geen delers heeft behalve 1 en zichzelf. Hieronder het algoritme van Eratosthenes om alle priemgetallen tot bij een natuurlijk getal n te vinden:

1. maak de rij van natuurlijke getallen van 2 tot n (de 'zeef');
2. verwijder het eerste (=kleinste) getal uit de zeef;
3. voeg dat getal toe aan de lijst priemgetallen;
4. verwijder alle veelvouden van dit getal uit de zeef;
5. herhaal stappen 2 tot 5 zolang de zeef niet leeg is.

Hieronder, als voorbeeld, het berekenen van de vier priemgetallen $p \leq 10$.

[2]	3	4	5	6	7	8	9	10	verwijder $2p$
	[3]	4	5	6	7	8	9	10	verwijder $3p$
			[5]	6	7	8	9	10	verwijder $5p$
					[7]	8	9	10	verwijder $7p$

Implementeer de Zeef van Eratosthenes. De wrapper gebruikt `natrij/4` om de zeef te initialiseren (Stap 1):

```
priem(N,PriemRij) :- natrij(2,N,Rij,[]),zeef(Rij,PriemRij).
```

Voor de Stappen 2 t/m 5 definieer je een predicaat `zeef/2`: `zeef(L,L1)` slaagt als `L1` de lijst van priemgetallen is die overblijft als je de niet-priemgetallen weglaat uit `L`. Het verwijderen van veelvouden in Stap 4 is voor rekening van `filter/3`.

(`natrij/4` en `filter/3` kan je beschouwen als ingebouwde predicaten voor deze opgave. Je cijfer voor deze opgave is dus niet afhankelijk van een al dan niet correcte oplossing voor `natrij/4` en `filter/3` in de eerdere opgaven.)

5 Dynamisch programmeren (3 ptn)

Met $\binom{n}{k}$ duiden we het aantal manieren aan om k elementen te kiezen uit een set van n alternatieven. Bijvoorbeeld: $\binom{3}{2} = 3$ want er zijn 3 manieren om 2 elementen uit een set van alternatieven $\{a, b, c\}$ te kiezen: $\{a, b\}$, $\{a, c\}$, $\{b, c\}$. De recursieve definitie $\binom{n}{k}$ heeft startwaarden $\binom{n}{0} = 1$ voor alle natuurlijke getallen n en $\binom{0}{k} = 0$ voor natuurlijke getallen $k > 0$; het recursieve geval voor $n, k > 0$ is

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

In Prolog:

```
binom(_,0,1).
binom(0,K,0) :- K>0.

binom(N,K,B) :- N>0,K>0,
    N1 is N-1, K1 is K-1,
    binom(N1,K1,B1),
    binom(N1,K,B2),
    B is B1+B2.
```

De definitie van `binom/3` is bijzonder inefficiënt: in de recursieve aanroep worden eerder berekende waarden voor `binom` telkens weer herberekend. Dat probleem kunnen we verhelpen met dynamisch programmeren. Neem aan dat we over een dynamisch predicaat `binom_found/3` beschikken, waarin we gevonden `binom` waarden opslaan met `assert/1`. Het recursieve geval voor `binom/3` passen we aan tot:

```
binom(N,K,B) :- N>0,K>0,
    N1 is N-1, K1 is K-1,
    binom_memo(N1,K1,B1),
    binom_memo(N1,K,B2),
    B is B1+B2.
```

`binom_memo/3` kijkt eerst of er een oplossing te vinden is bij de data voor `binom_found/3`. Als dat niet zo is, roept `binom_memo/3` het predicaat `binom/3` aan, en slaat de nieuwgevonden oplossing op als feit voor `binom_found/3`.

Geef de definitie voor `binom_memo/3`.

6 Zoeken: lengte van een pad (5 ptn)

Beschouw de toestandsruimte van Figuur 1. We duiden een knoop aan met een term `X/Y`, waarbij `X` op de horizontale en `Y` op de verticale as ligt. De knoop linksonder is dus `0/0`, de knoop rechtsboven `3/3`, enzovoort. De definitie voor `knoop/2` legt vast wat de knopen van deze toestandsruimte zijn.

```
knoop(X,Y) :- between(0,3,X),between(0,3,Y).
```

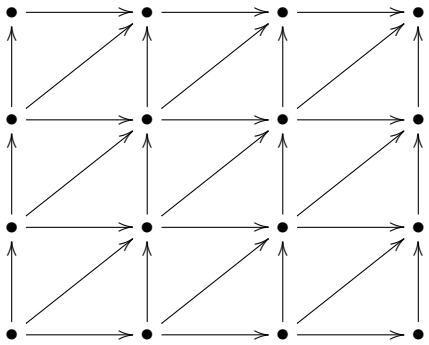
De knopen zijn verbonden met horizontale, verticale en diagonale overgangen. De lengte van een horizontale stap is 4, van een verticale 3, en van een diagonale (met dank aan Pythagoras) 5.

In deze opgave definieer je twee predicaten `s/3` en `g/2`. `s/3` geeft de lengte van de 1-staps overgangen, `g/2` de gesommeerde lengte voor een pad van 1-staps overgangen.

`s/3.` `s(From,To,L)` slaagt als `From` en `To` knopen zijn in de toestandsruimte die één stap van elkaar verwijderd zijn en `L` de afstand is van `From` naar `To`. Hieronder voorbeelden.

```
?- s(0/0,1/1,L).     ?- s(0/1,1/0,L).          ?- s(3/3,3/4,L).
L = 5.               false % geen verbinding   false % 3/4 buiten de graaf
```

`g/2` `g(Pad,Lengte)` slaagt als `Lengte` de totale lengte is van de startknoop tot de actuele knoop. `Pad` is een lijst van knopen `X/Y`. De actuele knoop is het eerste element van `Pad`, de startknoop het laatste. Gebruik `s/3` in je definitie van `g/2`. Een voorbeeldaanroep:



Figuur 1: Toestandsruimte

```
?- g([2/1,1/1,0/0],Lengte).
Lengte = 9
```

7 Heuristiek: Manhattan distance (3 ptn)

Bij best first zoeken maakt de heuristische functie h een schatting van de kortste route van een knoop naar de doelknoop. We willen een *toelaatbare* heuristische functie. Dat wil zeggen dat $0 \leq h(n) \leq h^*(n)$, waar $h^*(n)$ de daadwerkelijk kortste afstand is tussen n en de doelknoop. De heuristische functie is beter naarmate hij de echte korste afstand tussen n en doel dichter benadert. Maar hij mag niet over het doel heenschieten.

Definieer de 'Manhattan distance' tussen twee punten in Fig 1: de afstand in termen van horizontale en verticale stappen. Voorbeeld: `manhattan(0/0,2/1,11)` slaagt, want een horizontale stap heeft lengte 4, een verticale lengte 3.

Laat aan de hand van een voorbeeld zien dat de 'Manhattan distance' *niet* een toelaatbare invulling voor h is. Stel dat `3/2` de doelknoop is. Geef een knoop waarvoor de 'Manhattan distance' naar `3/2` groter is dan de werkelijk kortste afstand in de toestandsruimte.

8 Een perfecte heuristiek (3 ptn)

Geef een optimale definitie voor `h/3`. `h(From,To,H)` geeft `H` als de geschatte kortste afstand van `From` naar `To`. Een optimale `h/3` betekent dat de *geschatte* kortste afstand naar het doel gelijk is aan de *werkelijk* kortste afstand.

□