- Waarom gebruiken we in Prolog de 'most general unifier' (mgu) en niet een unifier?
- Waarom werken we met failure-driven loop?
- Wat zijn de voor- en nadelen van laziness in Haskell?

24 augustus 2012

- Bespreek het verschil tussen open-ended lists en difference lists
- Bespreek het verschil tussen patterns en constructors
- Geef de definitie van een declaratieve programeertaal

19 januari 2012

- Bespreek het verschil tussen "open-ended" lists en verschillijsten (difference lists)
- Bespreek het verschil tussen pattern matching en guards.
- Wat is de functie van typeklassen in Haskell (en Mercury)

19 januari 2011

- Bespreek het verschil tussen unificatie en pattern matching.
- Wat doet backtracking in Prolog
- Wat zijn de voor en nadelen van luie uitvoering in Haskell?

12 januari 2009

- Bespreek het verschil tussen unificatie en pattern matching
- Wanneer doet prolog aan garbage collecting
- Hoe is in Haskell sprake van generisch programmeren

13 januari 2010

- Bespreek het verschil tussen unificatie en pattern matching.
- Bespreek meta-programmatie in Prolog.
- Wat zijn de voor en nadelen van luie uitvoering in Haskell?

14 januari 2010

- Bespreek het verschil tussen unificatie en pattern matching
- Hoe doet Prolog aan backtracking
- Wat is de functie van typeklassen in Haskell (en Mercury)

Oplossingen

Prolog

Belichting (18 januari 2018)

Je krijgt een rooster (denk doolhof) van 7x7 vakjes, waarin enkele muren geplaatst zijn. De bedoeling is om een aantal lampen in het rooster te plaatsen zodat alle vakjes verlicht zijn. Elke lamp belicht zijn volledige rij en kolom, tenzij er een muur in de weg staat. Op een muur kun je geen lamp zetten, en een lamp kan niet door een muur schijnen. Op sommige muren staat een "hint" zoals in Minesweeper. Die geeft aan op hoeveel naburige vakjes (d.w.z. degene die een zijde gemeenschappelijk hebben) er een lamp moet staan. Verder mag geen enkele lamp op een andere schijnen.

De datastructuur wordt voorgesteld als een verzameling feiten:

```
wall(X,Y)
hint(X,Y,AantalLampen)
```

Opdrachten:

- Schrijf een predicaat lit(X,Y, Lampen) dat aangeeft of het vakje op positie (X,Y) verlicht is door een van de meegegeven lampen. De lampen worden voorgesteld als Prolog-termen(light(X,Y))
- Schrijf een predicaat neighbours((X1,Y2), (X2, Y2)) dat (X2,Y2) unificeert met een buur van (X1,Y2) die in het bord ligt en geen muur is.
- Schrijf een predicaat forbidden(X,Y,Fulls) dat aangeeft of de positie (X,Y) zich naast een vakje bevindt uit de lijst Fulls. De betekenis van Fulls is een lijst met vakjes waarop een hint staat (d.i. er moeten N omringende lampen zijn) en waarbij er al N lampen rond dat vakje staan.
- Schrijf een predicaat light(Lights) dat de puzzel oplost. Het is aangewezen om hierbij te steunen op vraag 3. Als hulpmiddel kun je eerst de volgende predicaten schrijven:
  - light_hints (ariteit zelf te bepalen) dat het correcte aantal lampen rondom de vakjes met een hint zet.
  - light_remaining (ariteit zelf te bepalen) dat met een gedeeltelijke oplossing gegeven door light_hints de rest van de puzzel oplost.

Palindromen (15 januari 2014)

Een palindroom is een woord dat in beide richtingen hetzelfde leest. Zo zijn lepel, 1221 en stormrots palindromen. We willen een programma schrijven dat in staat is elk woord om te vormen tot een palindroom. We mogen elk teken veranderen in een ander teken om ons doel te bereiken. Hiervoor gelden volgende regels:

- Een letter veranderen kost €1
- Als je één letter vervangt moet je onmiddellijk alle voorkomens van deze letter vervangen.

Neem als voorbeeld [r,e,d,e]. We kunnen dit oplossen als volgt:

- Verander 'e' in 'r' -> [r,r,d,r].
- Verander 'd' in 'r' -> [r,r,r,r].

Met totale kost €3. Het is echter makkelijk in te zien dat dit niet de goedkoopste manier is. Als je 'r' en 'd' in 'e' veranderd krijg je de palindroom [e,e,e,e] met kost €2.

Opdracht 1: Schrijf een predikaat allesgelijk(Seq1,Kost,Seq0) dat gegeven een sequentie Seq1 de goedkoopste sequentie Seq0 vindt met alle elementen gelijk, dus waarvan de Kost van de transformatie minimaal is. Indien er meerdere oplossingen met dezelfde minimale kost zijn, geef deze dan allemaal.

bv.

```
allesgelijk([r,e,d,e],Kost,Seq0).
  Kost = 2
  Seq0 = [e,e,e,e]

allesgelijk([a,b],Kost,Seq0).
  Kost = 1
  Seq0 = [a,a]
of
  Kost = 1
  Seq0 = [b,b]
```

Opdracht 2: Schrijf een predikaat palindroom(Seq1,Kost,Seq0) dat gegeven een sequentie Seq1 het goedkoopste palindroom Seq2 vindt. Merk hiervoor op dat elke sequentie kan opgedeeld worden in een aantal disjuncte sequenties die gelijk gesteld moeten worden. Bv. abyxzbzxxcb -> ab..b..cb, yx..xx en z..z. Indien er meerdere goedkoopste oplossingen zijn, geef ze dan allemaal.

bv.

```
palindroom([a,b,y,x,z,b,z,x,x,c,b],Kost,Seq0).
  Kost =  3
  Seq0 = [b,b,x,x,z,b,z,x,x,b,b]
```

Oplossing(en)

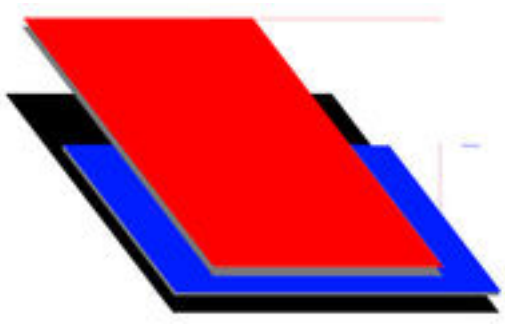
Jan heeft stenen (16 januari 2013)

Jan speelt graag met stenen. Hij heeft er een hele hoop die allemaal rechthoekig zijn. Elke steen heeft een andere kleur. Jan heeft ook een speelveld waarop hij de stenen kan leggen. Hij kan stenen op elkaar leggen. Niet alle stenen zijn evengroot. Zo kunnen we een speelveld voorstellen als volgt:

[[.,c,c,.], [a,c,c,a], [a,c,c,a]]

waarbij a en c kleuren zijn, en "." de afwezigheid van kleur aangeeft.

Hier een illustratie van het voorbeeld:



Vraag 1: Maak een predicaat "maakrechthoeken" om de posities van alle stenen als volgt te bepalen (voor het voorbeeld): [a-rhk(1,4,2,3),c-rhk(2,3,1,4)] waarbij een de waarden corresponderen met respectievelijk Index van Leftmost occurence,Rightmost occurence, lowesttopoccurence, highestbottomoccurence. (het assenstelsen van het veld is dus een standaard 2 dimensionaal x,y-vlak waarbij de y-as ondersteboven staat).

Vraag 2: Maak een predicaat "plan" dat de volgorde bepaalt waarin de rechthoeken gelegd zijn. Indien er meerdere mogelijkheden zijn, geef dan gesorteerd met @</2.

Oplossing(en)

Contradicties (01.09.2010)

Er worden multiple choice vragen gesteld. De persoon moet zijn voorkeur duidelijk maken. Zo verkiest jan (zie gegevens) c boven antwoorden a, b, d en e. Omdat mensen niet altijd consequent zijn, kunnen er tegenstrijdige antwoorden gegeven worden. Zo wordt in het voorbeeld i verkozen boven j, maar ook j boven i!

```
(opnieuw verzonnen)Gegevens:
*antwoord(jan,c,[a,b,d,e])
*antwoord(jan,i,[f,k,c,b])
*antwoord(jan,j,[i,f,h,e])
*antwoord(jan,k,[j,h,g,d])
```

1. Maak een predicaat meetegen(P,A) met als argumenten de persoon en een antwoordletter, dat teruggeeft of die antwoordletter in contradictie is met een andere antwoordletter. Hint: Hoe zou je in een graaf zien dat er een contradictie is?

2. Maak een predicaat tegen(P,Tegens) dat Tegens unificeert met een lijst van groepcontradicties. Zo'n groepcontradictie is een deelverzameling van alle antwoordletters, waarvan elk koppel een contradictie veroorzaakt. Zo'n deelverzameling bestaat dus uit minstens 2 elementen en zijn strikt verschillend (er mag geen antwoord in 2 groepcontradicties zitten). Voor het vb. krijg je dan:

```
Tegens = [[i,j,k]]
```

Oplossing(en)

Productie van mijnen (13 januari 2010)

Korte beschrijving: De productie van 2 mijnen hangt af van de voedsel leveringen die aan de mijn gedaan worden. Er zijn 3 soorten voedselleveringen. De productie van de mijn hangt af van hoeveel verschillende soorten er in de laatste 3 leveringen gebeurd zijn. Als er 1 soort is, produceert de mijn die dag 1 ton, als er 2 soorten voedsel geleverd werden in de laatste 3 leveringen, produceert de mijn 2 ton,...

vb:

- [1-v,2-v,3-v]=>tijdens levering 1 wordt type v geleverd, levering 2 type v levering 3 type v dus => 1+1+1=3ton totaal
- [1-v,2-g,3-f]=>1+2+3=6ton totaal
- [1-v,2-g,3-f,4-g]=>1+2+3+2(alleen de laatste 3leveringen bekijken)=8ton totaal

Als een mijn strict minder dan de helft aantal leveringen van de andere mijn ontvangen heeft krijgt deze mijn voorrang.

Schrijf een predicaat dat de scenario bepaald met een maximum productie. maxproductie(Leveringen,Scenario).

vb:

- maxproductie([1-v,2-g,3-f,4-g,5-v,6-v,7-f],toestand(15,mine([7-f,6-v,4-g,3-f,1-v]),mine([5-v,2-g])).

Er was ook een bonusvraag indien je snel genoeg was.

Oplossing(en)

Leren werken in teamverband (15 januari 2009)

De visitatiecommisie vindt het heel belangrijk dat cw studenten in hun opleiding leren werken in teams. Het is echter opvallend dat cw studenten geneigd zijn om telkens in dezelfde teams te werken. De docenten willen vermijden en besluiten dat de studenten vanaf nu hun teams anders moeten kiezen: **twee studenten mogen gedurende heel hun opleiding maar 1 keer in hetzelfde team zitten**.

Wij zullen een Prolog programma schrijven voor de verdeling van de studenten over de teams. We gaan ervan uit dat er T teams zijn en dat in elk team S studenten zijn. De bedoeling is om uit te vissen hoeveel groepswerken er tijdens de opleiding van een student gepland kunnen worden zodat de nieuwe regel gerespecteerd wordt. Gemakshalve wordt verondersteld dat het goede studenten zijn en dat dus de groep van studenten gedurende de opleiding hetzelfde blijft.

Deel 1

Schrijf een predikaat verdeling(T,S,Verdeling) dat in het geval van T teams en S Studenten per team een mogelijke verdeling van de studenten geeft. Bijvoorbeeld voor verdeling(2,2,V) geeft dit V = [[1,2],[3,4]]. Om dit probleem in een redeljke tijd te kunnen oplossen willen we vermijden dat dezelfde verdeling verschillende keren gegeneeerd wordt. De verdeling [1,2],[3,4]] is in feite herzelfde als [[2,1],[3,4] en [[4,3], [2,1]], maar is duidelijk verschillend van [1,3],[2,4]]. (Personal note: bedoeld wordt dat [1,2,3] en [1,2,4] dezelfde teams zijn omdat 1 en 2 al bij elkaar gezeten hebben, de opgave was hier dus licht dubbelzinnig.)

In het reultaat van verdeling zijn daarom de elementen van de teams geordend van klein naar groter en is er ook een orde opgelegd aan de teams zelf. Deze worden geordend op hun kleinste element. [1,2][3,4] is ok, maar [3,4][1,2] niet.

```
? - verdeling(2,2,V).
V = [[1, 2], [3, 4]];
V = [[1, 3], [2, 4]];
V = [[1, 4], [2, 3]];

No
```

Deel 2

Schrijf een predikaat groepeer(T,S,W, Schedule) dat gegeven het aantal teams T het aantal studenten per team S en het aantal werkjes W een geldig schedule Schedule opstelt. Als er geen oplossing is dan faalt het predikaat.

```
? - groepeer(2,2,3,Schedule).
Schedule = [[[1,2],[3,4]], [[1,3],[2,4]], [[1,4],[2,3]]] ;

No

?- groepeer(4,3,3, Prt). %de vele [ en ] er zelf bijdenken
Prt = [[[1 2 3] [4 5 6] 7 8 9  10 11 12
 1 4 7  2 5 10  3 8 11  6 9 12
 1 5 8  2 4 12  3 9 10  6 7 11 ;

Prt = 1 2 3  4 5 6  7 8 9  10 11 12
 1 4 7  2 5 10  3 8 11  6 9 12
 1 5 9  2 6 11  3 7 12  4 9 10 ;

...

?- groepeer(2,3,2, Prt).
No.
```

Merk op dat ook in het antwoord de verdelingen opnieuw geordend zijn van klein naar groot. Voor sommige queries (bijv groepeer(4,3,3, P) kan je verschillende alternateive antwoorden krijgen).

Deel 3

Uiteindelijk kan je dan het aantal werkjes gaan bepalen. Schrijf een predikaat aantalwerkjes(T,S,W) dat gegeven het aantal teams T en het aantal studenten per team S het aantal werkjes bepaalt.

Oplossing(en)

Logische gevolgen (12 januari 2009)

Opgave

De vraag bestond uit twee grote delen:

Deel 1: sluitingen

Schrijf een predicaat sluiting(S,F,Sluiting). Dit predicaat slaagt indien S een verzameling is, F een lijst van logische gevolgen ([a]>>[b,d] betekent uit a volgt b en d) en Sluiting de sluiting is van S onder F. Hiermee wordt bedoeld dat Sluiting alles is wat uit S volgt. Even enkele voorbeeldjes ter verduidelijking:

```
F1=[[a]>>[b,d],[b,d]>>[c]],
sluiting([a],F1,Sluiting).
--> Sluiting = [a,b,d,c]
```

```
F2=[[a]>>[b,d],[b,d]>>[c],[e]>>[f]],
sluiting([e],F2,Sluiting).
--> Sluiting = [e,f]
```

Deel 2: redundanties

Een implicatie wordt redundant genoemd (in een lijst van implicaties) indien ze het logisch gevolg is van een aantal andere van die implicaties. Bijvoorbeeld: als

```
F3=[[a]>>[b,d],[b,d]>>[c],[a]>>[c]]
```

dan is [a]>>[c] redundant want het gevolg van de andere twee. Schrijf een predicaat alleredundante(F,X) dat als eerste argument een lijst met implicaties krijgt en als tweede argument alle redundante uitspraken teruggeeft (samen met een minimale deelverzameling van F waaruit de redundantie volgt). Weer enkele voorbeeldjes:

```
F3=[[a]>>[b,d],[b,d]>>[c],[a]>>[c]],
alleredundante(F3,X).
-->X = [ redundant([a]>>[c],[[a]>>[b,d],[b,d]>>[c]]) ]
```

```
F4=[[a]>>[b],[b]>>[d],[a]>>[c],[a]>>[d],[c]>>[d]],
alleredundante(F4,X).
-->X= [redundant([a]>>[d],[[a]>>[b],[b]>>[d]]),redundant([a]>>[d],[[a]>>[c],[c]>>[d]])]
```

Oplossing(en)

Huffman-encodering (25 januari 2008)

Opgave

Bij een Huffman encodering krijgt elk symbool een bepaalde code toegekend. Deze code is zodanig opgesteld dat er geen enkele code een prefix is van een andere code. Bijvoorbeeld de code [0-[1,1],1-[0],2-[1,0]] is een Huffman code, de code [0-[0,1],1-[0],2-[1]] niet. Je krijgt een gecodeerd "bericht" waarvan de encoderingssleutel is verloren gegaan. De codes staan in volgorde van index, maar de 'scheidingen' tussen de codes zijn verdwenen. Zoek de mogelijke decoderingen. Bvb:

```
decodeer(3,[1,1,0,1,0],Res).
Res = [0-[1,1],1-[0],2-[1,0]];
No.
```

Immers, voor 0 kunnen we niet [1] nemen, want dan zou 1 ook moeten beginnen met [1] en dan is 0 een prefix van 1. Door analoge redeneringen blijkt dit de enige mogelijke toewijzing te zijn. Soms zijn er meerdere decoderingen mogelijk.

```
decodeer(2,[1,0,0],Res).
Res = [0-[1,0],1-[0]];
Res = [0-[1],1-[0,0]];
No.
```

Niet elke code is "even goed". Als we een bepaald symbool dikwijls gebruiken in de tekst is het beter daar een kleine encodering voor te hebben. Maak een predicaat beste/4 dat de best mogelijke encodering zoekt, gegeven de argumenten als in decodeer en een frequentietabel waarin staat hoe dikwijls een bepaald symbool voorkomt in de tekst. Voorbeeld:

```
Freqs = [0-1,1-5],beste(2,[1,0,0],Freqs,Res).
Res = [0-[1,0],1-[0]];

Freqs = [0-10,1-5],beste(2,[1,0,0],Freqs,Res).
Res = [0-[1],1-[0,0]];
```

een oplossing

alternatieve oplossing

oplossing mbv boom

Graadafstand

Opgave

Een graaf G is een stel knopen V met (niet-gerichte) bogen E ertussen - we maken dat soms expliciet door die graaf te noteren als G(V,E). De graad van een knoop v schrijven we als delta(v) en is het aantal bogen dat knoop v als eindpunt heeft.

Als G1(V1,E1) en G2(V2,E2) twee grafen zijn dan noemen we een bijectie f: V1->V2 een knoopbijectie tussen de twee grafen G1 en G2. Voor een gegeven knoopbijectie f tussen G1 en G2, definieren we de f-afstand tussen G1 en G2 als de som over alle knopen v van V1, van de absolute waarde van het verschil tussen de graad van v en de graad van f(v).

De graadafstand tussen twee grafen G1 en G2 met een gelijk aantal knopen, is dan gedefinieerd als het minimum van alle f-afstanden tussen G1 en G2. Schrijf een predicaat graadafstand/3 dat voor twee gegeven grafen G1 en G2 (als eerste en tweede argument - zie later hoe) het derde argument unificeert met de graadafstand tussen G1 en G2.

Een graaf is gegeven door een rij feiten van de vorm:

```
graaf(8,a,b).
graaf(8,a,c).
graaf(8,a,d).
graaf(22,aa,bb).
graaf(22,aa,cc).
graaf(22,aa,dd).
graaf(15,a,b).
graaf(15,b,c).
graaf(15,a,d).
```

die betekenen: in graaf 8 is er een boog tussen a en b, ...

Elke boog wordt slechts 1 maal voorgesteld en voor het gemak beschouwen we alleen grafen zonder lussen (een lus is een boog van een knoop v naar v).

Het resultaat van een paar queries:

```
?- graadafstand(8,22,N).
N=0
```

```
?-graadafstand(15,22,N).
N=2
?-graadafstand(8,15,N).
N=2
```

Oplossingen

Winkelen

Opgave

Een bedrijf stelt orders op van de vorm

```
[zeep/10, prei/14]
```

met betekenis dat in deze order 10 eenheden zeep en 14 eenheden prei worden besteld. Het bedrijf heeft een aantal leveranciers, waarvan het voor elk product dat de leverancier aanbiedt de prijs kent - in de vorm:

```
prijs(delhaize, zeep, 8).
prijs(delhaize, prei, 10).
prijs(delhaize, zout, 6).
prijs(carrefour, prei, 9).
prijs(carrefour, soep, 19).
prijs(champion, zeep, 7).
prijs(champion, prei, 11).
prijs(champion, pinda, 6).
```

met betekenis: delhaize levert zeep aan 8 Euro per eenheid, enzovoort. Merk op dat niet elke leverancier dezelfde producten levert, en dat de prijzen varieren. Een order kan geplaatst worden bij 1 leverancier, en dan moet de order geplaatst worden bij de leverancier die de beste prijs heeft voor het hele order samen. Voor [zeep/10, prei/14] is dat delhaize, want carrefour kan niet alles leveren, de totaalprijs van delhaize is 10*8+14*10 == 220 en de totaalprijs van champion is 10*7+14*11 == 224.

Een order kan ook gesplitst worden over twee leveranciers (maar nooit meer dan twee) en als dat goedkoper is dan alles door 1 leverancier te laten leveren, dan moet die order gesplitst worden. In het voorbeeld is het beter om de zeep te laten leveren door champion en de prei door carrefour, want dat geeft de laagste totale kost.

Je schrijft een predicaat bestorder/2 dat als eerste argument een order krijgt en het tweede argument unificeert met een beschrijving van hoe dat order moet geplaatst worden. Voor het voorbeeld is dat:

```
?- bestorder([zeep/10, prei/14], Plaatsing).
Plaatsing = [zeep/champion, prei/carrefour]
```

De volgorde in Plaatsing is van geen belang. Als er meerdere beste orders zijn, dan geef je er maar 1. Een order kan willekeurig lang zijn.

Voor het gemak is er ook een feit dat aangeeft welke de leveranciers zijn; voor het voorbeeld:

```
leveranciers([delhaize,carrefour,champion]).
```

Je kan de gegeven feiten gebruiken om je programma te testen - je mag nieuwe feiten toevoegen voor je testen ook natuurlijk.

Bovenstaande is het eerste (en belangrijkste) stuk van je opdracht. Als je daarmee klaar bent, en je hebt nog tijd, schrijf je - gebaseerd op je predicaat bestorder/2 - een predicaat optimalorder/2, dat uit alle bestorders met dezelfde kost, het order kiest waarop de leverancier(s)s in het totaal het meeste reductie geven: een leverancier geeft reductie gebaseerd op het totaal aantal eenheden dat hij mag leveren, volgens een formule die per leverancier varieert en die gegeven wordt door feiten (voor het voorbeeld):

```
kortingspercentage(delhaize, E, max(0,min(10,(E-29)/10)) ).
kortingspercentage(carrefour, E, max(0,min(5,(E-10)/7)) ).
kortingspercentage(champion, E, max(0,min(11,(E-50)/9)) ).
```

Je roept die feiten op met tweede argument een totaal aantal eenheden dat bij die leverancier besteld wordt en als derde argument krijg je terug de formule waarmee je het kortingspercentage kan berekenen. Je programma moet een willekeurige (correcte) formule aankunnen.

een oplossing

Minimale snede

Opgave

Een transportnetwerk is gegeven: het is een samenhangende gerichte gewogen graaf met vanuit knoop a (de bron) alleen vertrekkende bogen en in z (de put) komen alleen maar bogen aan. Een voorbeeldtransportnetwerk is:

```
boog(a,b,1).
boog(a,c,1).
```

```
boog(b,c,1).
boog(c,z,3).
boog(b,z,1).
```

Je schrijft een predicaat mincut/2 dat voor een gegeven transportnetwerk argument 1 met een minimale snede unificeert en argument twee met de capaciteit van die minimale snede. Een snede in een transportnetwerk wordt bepaald door een partitie in twee disjuncte deelverzamelingen A en Z van de knopen van het transportnetwerk; bovendien zit a in A en z in Z.

De capaciteit van een snede is de som van de capaciteiten van de bogen die van A naar Z lopen. De snede geef je door de term snede(<knopen van A>, <knopen van Z>)

Voor het voorbeeld:

```
?- mincut(X,Y).
Y = snede([a],[z,b,c])
X = 2
Yes
```

Als er meer dan 1 minimale snede is, dan mag je zelf kiezen welke je teruggeeft.

Het vervolg van deze oefening bestaat uit het teruggeven van een minimale snede met zo weinig mogelijk gesneden bogen.

```
boog(a,b,1).
boog(a,c,1).
boog(b,d,1).
boog(c,d,1).
boog(d,z,2).
```

heeft als minimale snedes snede([a],[b,c,d,z]) en snede([a,b,c,d],[z]). In de eerste worden de bogen (a,b) en (a,c) gesneden; in de tweede enkel (d,z). Bijgevolg is de tweede het antwoord voor kleinste_mincut/2:

```
?- kleinste_mincut(X,Y).
Y = snede([a,b,c,d],[z])
X = 2
Yes
```

een oplossing

Handelsreiziger verplaatsen

Opgave

feiten: Een predikaat dat stelt dat 2 steden buren van elkaar zijn en een predikaat dat stelt dat een stad een hotel heeft.

```
buurvan(a,b)   (a is buur van b, maar ook b is buur van a)
heefthotel(a)
```

Een bedrijf heeft handelsreizigers en die doen steden aan. 's avonds bellen ze naar het bedrijf en vragen vanuit de stad waar ze nu zijn of er een buurstad is die een hotel heeft. Zo ja, dewelke.

Maak verplaats([persoon,plaats],moves) waarbij moves moet geunificieerd worden met een mogelijke verplaatsing vanuit 'plaats' naar een buurstad met een hotel.

```
?- verplaats([(ik,leuven),(jij, gent)],X)
X = [(ik,leuven,brussel),(jij,leuven,ietsbuitengent)]
```

Maak verplaats_beste die met dezelfde argumenten, de weg geeft naar een hotel in de minste moves.

een oplossing

Keigrafen (juni 2005)

Opgave

(Ik probeer hier de vraag te reconstrueren ...)

We werken met knopen in een ongerichte grafe. Elk knooppunt bevat geen, 1 of meer keien. Je kunt keien van knooppunt als volgt verplaatsen:

1. de bron bevat n keien (n > 0)
2. je verplaatst x keien (1 <= x <= n)
3. de bron verliest deze x keien
4. het doel ontvangt x - 1 keien (1 keien gaat dus verloren!)

Een knoop is kei_bereikbaar als je na een eindig aantal (0 of meer) verplaatsingen van keien 1 of meer keien in de knoop hebt.

Een grafe is kei_goed als alle knopen kei_bereikbaar zijn.

Een grafe is kei_nijg als alle knopen TEGELIJK kei_bereikbaar zijn (en je met andere woorden na een eindig aantal verplaatsingen van keien in elke knoop minstens 1 kei hebt).

Opgave is simpel: schrijf prolog-code die deze eigenschappen berekent.

een oplossing

N-queens

Opgave

Schrijf een genereer-en-test programma voor het **N-queens probleem**, ttz zet N koninginnen op een NxN schaakbord, zodat geen enkele koningin door een andere geslagen kan worden. Je mag gelijk welke voorstelling van een bord gebruiken, maar je levert als resultaat een lijst van rijen waarin opeenvolgende koninginnen staan: bv. nqueens(4, L) -> L = [2, 4, 1, 3] Deze oplossing zegt dat de koningin in kolom 1 op rij 2 staat, in kolom 2 op rij 4, ...

Te vinden oplossingen: [2, 4, 1, 3] ** [3, 1, 4, 2]

_____ ** _____

|__|K_|__|__| ** |__|__|K_|__|

|__|__|__|K_| ** |K_|__|__|__|

|K_|__|__|__| ** |__|__|__|K_|

|__|__|K_|__| ** |__|K_|__|__|

een oplossing

Celautomaat (19 juni 2006)

Opgave

Een cellulaire automaat is oneindig lang en bestaat uit een rij zwarte (aangeduid door x) en witte cellen (aangeduid door o). Alle cellen voor en achter degene die we in het geheugen hebben, zijn automatisch wit.

Een cellulaire automaat kan overgaan naar een volgende generatie. Hierbij is de nieuwe toestand van een cel enkel afhankelijk van de toestand in de huidige generatie van de linker-, rechter- en huidige cel. Hierbij wordt gebruik gemaakt van regels: [X,Y,Z,N] betekent dat een cel met waarde Y de waarde N krijgt als de linker - en rechtercel respectievelijk waardes X en Z hadden.

Er bestaat de zogenaamde "javel-regel": [o,o,o,o]. Deze garandeert dat de oneindige stukken witte band voor en achter het stuk band in het geheugen wit blijven.

Voorbeeld: *... o x o ...* wordt *... o x x o o ...* onder de regels *[o, o, x, x], [o, x, o, x], [x, o, o, o]* en de javelregel

1. Schrijf een predicaat *volgendegen(Seq, Regels, Volgende)* dat een volgende generatie berekent van *Seq* met de gegeven regels indien mogelijk en anders faalt. **Hint:** voeg 2 witte cellen toe voor en achter je invoer.

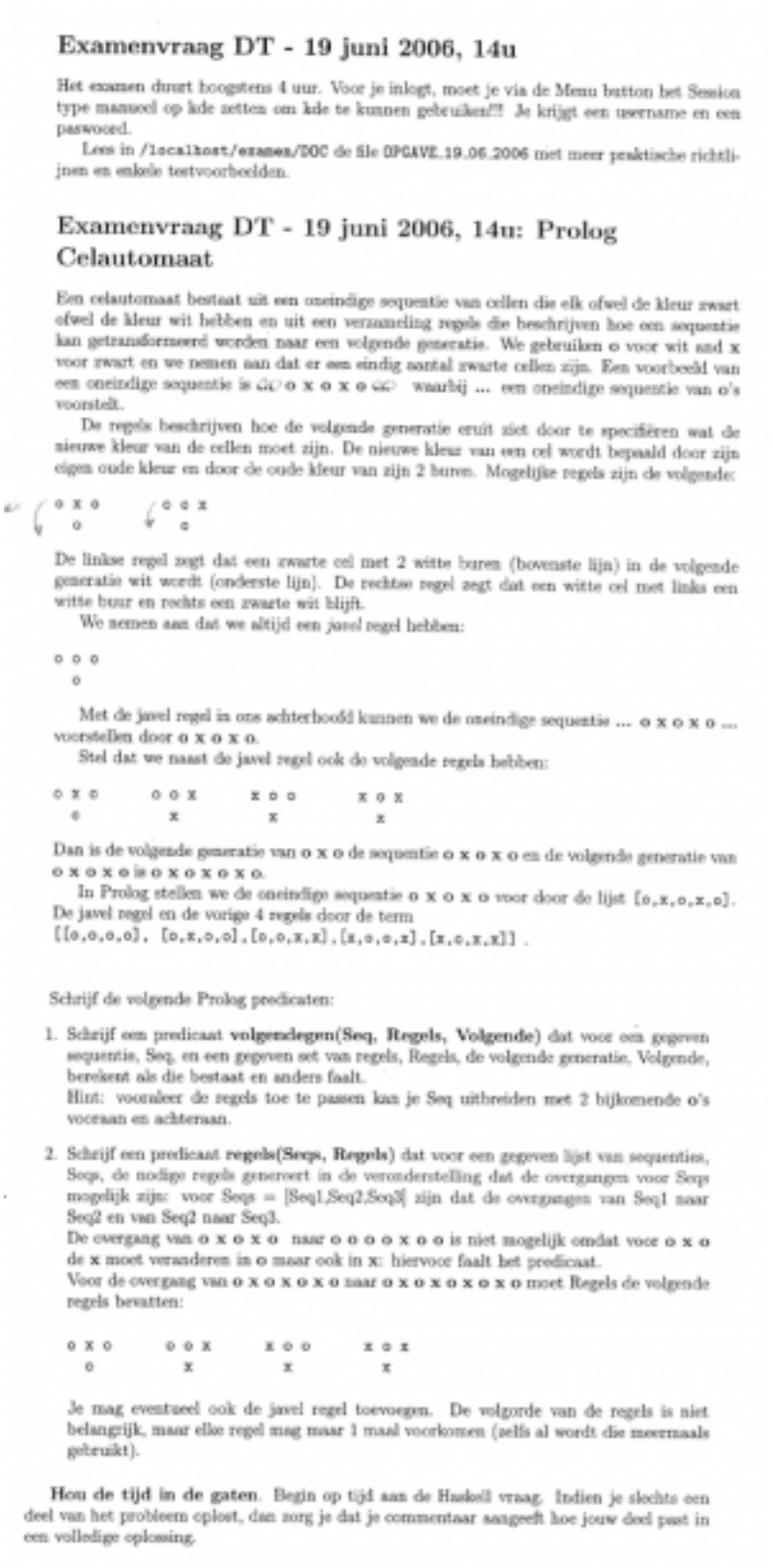
2. Schrijf een predicaat *regels(Sequenties, Regels)* Hierbij is *Sequenties* een lijst van achtereenvolgende generaties van de automaat. **Regels** moet geunificeerd worden met de regels die nodig zijn voor deze transformaties indien dit mogelijk is. De overgang van *o x o x o* naar *o o o o x o o* is niet mogelijk omdat voor *o x o* de x zowel in een o als een x moet veranderen. De regels blijven voor elke generatie-overgang dezelfde. De lijst van regels mag geen dubbels bevatten.

- Voorbeelden:
  - *regels([[o, x, x], [o, o, o, x, o], [o, o, o, o, o, o, o]], X)* geeft

[[x, x, o, x], [o, x, x, o], [o, x, o, o], [o, o, o, o]] (Opmerking: ik denk dat hier enkele regels ontbreken? Volgens mij moeten deze er nog bij: [o, o, x, o], [x, o, o, o], [o, x, o, o])

  - *regels([[o, x, o, x, o], [o, o, o, x, o]], X)* heeft geen geldig resultaat, omdat er geen consistente set regels voor deze overgang bestaat.

een oplossing



Examenvraag DT - 19 juni 2006, 14u

Het examen duurt hoogstens 4 uur. Voor je inlogt, moet je via de Menu button het Session type manueel op kde zetten om kde te kunnen gebruiken!?! Je krijgt een username en een paswoord.

Log in /localhost/examen/DOC de file OPGAVE.19.06.2006 met meer praktische richtlijnen en enkele testvoorbeelden.

Examenvraag DT - 19 juni 2006, 14u: Prolog

Celautomaat

Een celautomaat bestaat uit een oneindige sequentie van cellen die elk ofwel de kleur zwart ofwel de kleur wit hebben en uit een verzameling regels die beschrijven hoe een sequentie kan getransformeerd worden naar een volgende generatie. We gebruiken o voor wit and x voor zwart en we nemen aan dat er een eindig aantal zwarte cellen zijn. Een voorbeeld van een oneindige sequentie is ... o o o x o x o o o ... waarbij ... een oneindige sequentie van o's voorstelt.

De regels beschrijven hoe de volgende generatie eruit ziet door te specifiëren wat de nieuwe kleur van de cellen moet zijn. De nieuwe kleur van een cel wordt bepaald door zijn eigen oude kleur en door de oude kleur van zijn 2 buren. Mogelijke regels zijn de volgende:

```
o x o      o o x
  o          o
```

De linkse regel zegt dat een zwarte cel met 2 witte buren (bovenste lijn) in de volgende generatie wit wordt (onderste lijn). De rechtse regel zegt dat een witte cel met links een witte buur en rechts een zwarte wit blijft.

We nemen aan dat we altijd een *javel* regel hebben:

```
o o o
  o
```

Met de javel regel in ons achterhoofd kunnen we de oneindige sequentie ... o x o x o ... voorstellen door o x o x o.

Stel dat we naast de javel regel ook de volgende regels hebben:

```
o x o      o o x      x o o      x o x
  o          x          x          x
```

Dan is de volgende generatie van o x o de sequentie o x o x o en de volgende generatie van o x o x o is o x o x o x o.

In Prolog stellen we de oneindige sequentie o x o x o voor door de lijst [o,x,o,x,o]. De javel regel en de vorige 4 regels door de term
[[o,o,o,o], [o,x,o,o],[o,o,x,x],[x,o,o,x],[x,o,x,x]] .

Schrijf de volgende Prolog predicaten:

1. Schrijf een predicaat **volgendegen(Seq, Regels, Volgende)** dat voor een gegeven sequentie, Seq, en een gegeven set van regels, Regels, de volgende generatie, Volgende, berekent als die bestaat en anders faalt.
   Hint: vooraleer de regels toe te passen kan je Seq uitbreiden met 2 bijkomende o's vooraan en achteraan.

2. Schrijf een predicaat **regels(Seqs, Regels)** dat voor een gegeven lijst van sequenties, Seqs, de nodige regels genereert in de veronderstelling dat de overgangen voor Seqs mogelijk zijn: voor Seqs = [Seq1,Seq2,Seq3] zijn dat de overgangen van Seq1 naar Seq2 en van Seq2 naar Seq3.
   De overgang van o x o x o naar o o o o x o o is niet mogelijk omdat voor o x o de x moet veranderen in o maar ook in x: hiervoor faalt het predicaat.
   Voor de overgang van o x o x o x o naar o x o x o x o x o moet Regels de volgende regels bevatten:

```
o x o      o o x      x o o      x o x
  o          x          x          x
```

Je mag eventueel ook de javel regel toevoegen. De volgorde van de regels is niet belangrijk, maar elke regel mag maar 1 maal voorkomen (zelfs al wordt die meermaals gebruikt).

Hou de tijd in de gaten. Begin op tijd aan de Haskell vraag. Indien je slechts een deel van het probleem oplost, dan zorg je dat je commentaar aangeeft hoe jouw deel past in een volledige oplossing.

Haskell

E-Mailserver (18 januari 2018)