Opgave 3: Recursion, NAF and lists

Practicum Intelligente Systemen

maart 2014

1.a Schrijf voor de wumpus-wereld-agent een predikaat `plan(+X,+Y,?Z)` dat gegeven een `X` en `Y` coördinaat een lijst `Z` retourneert van acties die ondernomen dienen te worden om van (1,1) naar (X,Y) te komen (waar het goud ligt).
1.b Schrijf nu het predikaat `plan/3` dat gegeven een `X` en `Y` coördinaat een lijst retouneert van acties die ondernomen dienen te worden om van (1,1) naar (X,Y) te komen, op een *veilige* manier. De lijst `Z` is nu dus een plan om *veilig* van (1,1) naar (X,Y) te komen. Voor dit programma is het natuurlijk van belang om te definiëren wat "veilig" betekent. Voor deze opgave geven we de agent algehele kennis van de wereld, i.e. hij weet waar de gaten en de wumpus zich bevinden (dit gaan we in de volgende opgave veranderen). Gebruik de volgende definitie van het predikaat `veilig`:

```
veilig(X,Y) :- no_wumpus(X,Y), no_pit(X,Y).
no_wumpus(X,Y) :- \+ wumpus(X,Y).
no_pit(X,Y) :- \+ pit(X,Y).
```

Voeg aan je kennisbank een wereld toe waar geldt dat `pit(3,1), pit(3,3), pit(4,4)` en `wumpus(1,3)`. Deze wereld zullen we gebruiken om je programma in te testen. Je test je programma als volgt:

```
?- plan(2,3,L).

L = [goNorth, goEast, goNorth]
```

Je programma geeft waarschijnlijk meer dan één plan (na backtracken), en hoogstwaarschijnlijk zijn deze inefficiënt. Maar het gaat om het eerste plan dat je programma genereert[1]. Hint: Dit planningsprobleem is bijzonder geschikt voor een **backwards search** aanpak.

Een elementaire taak die lijsten en recursie vereist, is het sorteren van lijsten. In de volgende twee opgaven gaan we verschillende recursieve predikaten schrijven om een lijst te sorteren. De predikaten die we gaan schrijven zijn enkel bedoeld voor het sorteren van lijsten met getallen, alle andere soorten lijsten worden buiten beschouwing gelaten.

Op het college is reeds de meest simpele vorm van sorteren behandeld, namelijk Bubble Sort:

```
bubbleSort(List,Sorted) :-
          swap(List,List1), !,
          bubbleSort(List1,Sorted).
bubbleSort(Sorted,Sorted).
```

[1] Als je denkt dat je op een makkelijke manier wel alle efficiënte plannen kan genereren, dan is dat natuurlijk toegestaan. Dit kan je wellicht een bonus opleveren ;-) Maar: fouten die hierdoor in je programma worden geïntroduceerd worden natuurlijk wel gewoon fout gerekend. Efficiëntie proberen in te bouwen is dus niet geheel zonder risico.

```
swap([X,Y|Rest],[Y,X|Rest]) :-
          X > Y.
swap([Z|Rest],[Z|Rest1]) :-
          swap(Rest,Rest1).
```

2. Maak op basis van bovenstaand Bubble Sort algoritme een predicaat `colourSort/3` dat kleuren sorteert volgens een aangegeven volgorde. Bijvoorbeeld, wanneer je het programma vraagt `?- colourSort([z,r,g,g,w,z,z,r],[w,r,g,b,z],L)` dan moet het de eerste lijst sorteren in de volgorde van de elementen in de tweede lijst. Het antwoord dat je moet krijgen is dan `L = [w,r,r,g,g,z,z,z].`

3. Buble sort is inefficiënt. Eén van de meest efficiënte sorteeralgoritmen is Quick Sort. Dit sorteeralgoritme neemt een willekeurig element uit de lijst (bijv. het eerste), en splitst vervolgens de resterende lijst op in een lijst van elementen die kleiner en een lijst van elementen die groter zijn dan dit element. Nadat deze lijsten met kleinere en grotere elementen zelf gesorteerd zijn, worden de lijsten weer aan elkaar geplakt. Schrijf het predikaat `quickSort/2` dat gegeven een lijst de gesorteerde lijst teruggeeft.
(`quickSort(+X,?Y)`)

4. Een bekend Constraint Satisfaction Probleem, dat zeer geschikt is om opgelost te worden door middel van lijsten en recursie, is het acht-koninginnen-probleem. Het probleem bestaat eruit om acht koninginnen te plaatsen op een 8×8 schaakbord, zonder dat de koninginnen elkaar kunnen slaan. Bedenk dat een koningin in het schaken horizontaal, verticaal en ook diagonaal over het bord mag bewegen.
Een belangrijk aspect van het oplossen van dit probleem is het kiezen van een representatie voor het probleem. Aangezien koninginnen verticaal en horizontaal kunnen bewegen betekent dit dat elke rij en elke kolom van het schaakbord slechts één koningin kan bevatten. Hierdoor kunnen we het probleem reduceren tot een simpel permutatieprobleem, namelijk het vinden van een correcte volgorde van de lijst `[1,2,3,4,5,6,7,8]` (waarbij de positie van een getal in de lijst de x-coördinaat en het getal zelf de y-coördinaat van een koningin voorstelt) zodanig dat de koninginnen elkaar niet diagonaal kunnen slaan. Schrijf een predikaat `solution/1` dat een lijst teruggeeft met de posities van de koninginnen.
`solution(?X)`

Er bestaan 92 verschillende oplossingen voor een 8×8 schaakbord met 8 koniginnen. Om te checken of je programma alle oplossingen geeft kan je gebruik maken van de `findall/3` en `length/2` predikaten. Let wel, `findall/3` geeft een lijst van alle antwoorden die je programma geeft, dus ook dubbelen. Je kan echter door middel van `setof/3` testen of je programma echt 92 verschillende oplossingen vindt.

5. Bonusvraag: schrijf in Prolog een solver voor diagonaal-sudoku's (ook de nummers op de twee diagonalen moeten verschillend zijn). Kies zelf een geschikte representatie voor de sudoku's en test je programma op, bijvoorbeeld, de diagonaal-sudoku's uit De Volkskrant.

Inleveren

- Aan de hand van je antwoorden op de bovenstaande opgaven bepalen we het derde punt dat meetelt voor je algehele practicumcijfer.
- Inleveren dient te geschieden via 'submit', lever **ALLEEN** de (niet gezipte) .pl-file in!
- De deadline voor inleveren staat vast op 21-03-2014 om 23.59 uur.
- Lees de richtlijnen voor de programma's op **http://www.cs.uu.nl/docs/vakken/b3is/richtlijnen_2013_2014.pdf**
- Zorg ervoor dat de opgave voorzien is van zowel namen als studentnummers. Neem dit als commentaar op **bovenaan** in je .pl file. Commentaar kan in .pl files worden opgenomen door middel van het '%' symbool.
- **Zorg dat je op tijd inlevert, we accepteren geen overschreidingen van de deadline.**