Prolog. Gekwoteerde zitting.

Patrick De Causmaecker

26/3/2020, 9:00 - 12:00

Er zijn vier vragen. Het symbool (*) duidt een (deel van een) vraag aan die je als moeilijker kan beschouwen en eventueel uitstellen voor het geval er op het einde tijd over is.

Je kan de documentatie bij SWI-Prolog gebruiken en de slides. Andere resources op het internet zijn niet toegelaten.

Als je klaar bent verwittig je de toezichter. Die zal je toelaten om je e-mail op te starten. Zend dan je antwoorden naar `patrick.decausmaecker@kuleuven.be`

1 Nauwkeurig Rekenen (0.8/3)

Stel positieve gehele getallen van willekeurige grootte in binaire representatie voor als lijsten van atomen 0 en 1.

1. Schrijf het predicaat (0.4/3)

   $som(A, B, C)$

   dat slaagt als C de voorstelling is van de som van A en B waarbij A en B volledig geunificeerde lijsten zijn. Bijvoorbeeld

```
?- A = [1,0,1],B = [1,1,0],C = [_,_,_,_],som(A,B,C).
A = [1, 0, 1],
B = [1, 1, 0],
C = [1, 0, 1, 1] ;
false.
?- A = [1,0,1],B = [0,1,0,0],C = [_,_,_,_],som(A,B,C).
A = [1, 0, 1],
B = [0, 1, 0, 0],
C = [1, 0, 0, 1] ;
false.
?- A = [1,0,1],B = [0,1,1,0],C = [_,_,_,_],som(A,B,C).
A = [1, 0, 1],
B = [0, 1, 1, 0],
C = [1, 0, 1, 1] ;
false.
?- A = [1,0,1],B = [0,1,1,0],C = [_,_,_],som(A,B,C).
false.
```

   Het laatste voorbeeld faalt omdat de te kleine lijst C het resultaat niet kan voorstellen. Het tweede voorbeeld laat zien dat de eerste (meest beduidende bit) ook nul mag zijn. C moet minstens evenveel elementen kunnen bevatten als de langste van A en B. Vandaar:

```
?- A = [0,1],B = [0],C = [_],som(A,B,C).
false.
?- A = [0,1],B = [0],C = [_,_],som(A,B,C).
```

```
A = C, C = [0, 1],
B = [0] ;
false.
```

2. (*) Schrijf het predicaat (0.4/3)

   $somver(A, B, C)$

   dat een versatile versie is van *som*. *somver* laat een willekeurig aantal niet geünificeerde termen in elk van de lijsten toe en slaagt voor alle unificaties waarvoor de som van A en B gelijk is aan C. Eventuele inleidende nullen in A, B of C mogen hierbij worden weggelaten. Bijvoorbeeld

```
?- A = [0,1],B = [0],C = [_],somver(A,B,C).
A = [0, 1],
B = [0],
C = [1] ;
false.
?- A = [0,1],B = [1],C = [_],somver(A,B,C).
false.
```

   Hieronder nog een paar voorbeelden waarbij willekeurige termen niet geünificeerd zijn.

```
?- A = [1,_],B = [_],C = [1,0],somver(A,B,C).
A = C, C = [1, 0],
B = [0] ;
false.
?- A = [_,_],B = [_],C = [1,0],somver(A,B,C).
A = C, C = [1, 0],
B = [0] ;
A = [0, 1],
B = [1],
C = [1, 0] ;
false.
?- A = [_],B = [_],C = [_,_],somver(A,B,C).
A = B, B = [0],
C = [0, 0] ;
A = [0],
B = [1],
C = [0, 1] ;
A = [1],
B = [0],
C = [0, 1] ;
A = B, B = [1],
C = [1, 0].
?- A = [_,_,_,_,_],B = [_,_,_,_,_],C = [1,0,0,0,0],findall(_,somver(A,B,C),ALL),length(ALL,CT).
A = [_7498, _7504, _7510, _7516, _7522],
B = [_7534, _7540, _7546, _7552, _7558],
C = [1, 0, 0, 0, 0],
ALL = [_8798, _8792, _8786, _8780, _8774, _8768, _8762, _8756, _8750|...],
CT = 17.
```

2 Boom met een accu (0.8/3)

Gegeven zijn termen

1. $val(X)$ die een waarde voorstelt, X unificeert met een atoom

2. $plus(X, Y)$ die de som ('+') van twee uitdrukkingen voorstelt
3. $maal(X, Y)$ die het product ('x') van twee uitdrukkingen voorstelt

We kunnen we een uitdrukking construeren als bijvoorbeeld

$$plus(plus(val(a), val(b)), maal(val(c), plus(val(d), val(e))))$$

Gevraagd is

1. Een predicaat (0.2/3)

   $infix(U, L)$

   dat slaagt als U een uitdrukking is zoals hierboven gedefinieerd en L een lijst is die overeenkomt met de infix notatie voor deze uitdrukking. Bijvoorbeeld

   ```
   ?- infix(plus(plus(val(a),val(b)),maal(val(c),plus(val(d),val(e)))),R),print(R).
   [a,+,b,+,c,x,d,+,e]
   R = [a, +, b, +, c, x, d, +, e].
   ```

2. Een predicaat (0.2/3)

   $infixprior(U, L)$

   dat slaagt zoals infix, maar haakjes introduceert om de prioriteit van de vermenigvuldiging t.o.v. de optelling in rekening te brengen. Bijvoorbeeld

   ```
   ?- infixprior(plus(plus(val(a),val(b)),maal(val(c),plus(val(d),val(e)))),R),print(R).
   [a,+,b,+,c,x,'(',d,+,e,')']
   R = [a, +, b, +, c, x, '(', d, +|...].
   ```

3. (*) Een predicaat (0.2/3)

   $infixaccu(U, L)$

   dat werkt zoals $infix$ maar gebruik maakt van een accumulator.

4. (*) Een predicaat (0.2/3)

   $infixprioraccu(U, L)$

   dat werkt zoals $infixprior$ maar gebruik maakt van een accumulator.

3 Telt er iets op? (0.8/3)

Gegeven een verzameling getallen en gegeven een som, is er een deelverzameling van deze getallen waarvan de som overeenkomt? Bijvoorbeeld, voor de verzameling $\{1, 2, 3, 4, 5, 6, 7\}$ en de som 11 is het antwoord positief want $1 + 3 + 7 = 11$ (en ook $2 + 4 + 5 = 11$ maar er is niet gevraagd hoeveel verzamelingen voldoen). Gevraagd is

1. Een predicaat (0.2/3)

   $subsetsum(Set, Som)$

   dat slaagt als er in de verzameling getallen voorgesteld door Set een deelverzameling is waarvan de som gelijk is aan het getal Som. Je mag veronderstellen dat Set een lijstvoorstelling van een dergelijke set is.

2. Een predicaat (0.5/3)

   $subsetsum\_recurs(Set, Som)$

   dat onder dezelfde voorwaarden slaagt maar dat geïmplementeerd is door recursief een element uit Set te verwijderen en het één keer niet in de deelverzameling op te nemen en één keer wel. Als je oplossing voor Vraag 1 dit precies zo doet hoef je deze vraag niet op te lossen.

3. (*) Een predicaat (0.3/3)

   $subsetsum\_memo(Set, Som)$

   dat dezelfde recursieve methode gebruikt als in Vraag 2 maar een vorm van memoïzatie gebruikt om het veelvuldig beantwoorden van dezelfde vraag efficiënter te behandelen.

We kunnen het gedrag van de predicaten bekijken via time/1:

```
?- time(subsetsum([1,2,3,4,5,6,9],10)).
% 45 inferences, 0.000 CPU in 0.000 seconds (68% CPU, 1551724 Lips)
true.
?- time(subsetsum([1,2,3,4,5,6,9],10)).
% 45 inferences, 0.000 CPU in 0.000 seconds (68% CPU, 1607143 Lips)
true.
?- time(subsetsum_memo([1,2,3,4,5,6,9],10)).
% 81 inferences, 0.000 CPU in 0.000 seconds (87% CPU, 910112 Lips)
true.
?- time(subsetsum_memo([1,2,3,4,5,6,9],10)).
% 1 inferences, 0.000 CPU in 0.000 seconds (55% CPU, 58824 Lips)
true.
?- time(subsetsum_memo([1,2,3,4,25,26,39],10)).
% 329 inferences, 0.000 CPU in 0.000 seconds (83% CPU, 2550388 Lips)
true.
?- time(subsetsum_memo([1,2,3,4,25,26,39],10)).
% 1 inferences, 0.000 CPU in 0.000 seconds (73% CPU, 76923 Lips)
true.
```

4 Bevroren dominosteentjes (0.6/3)

We gebruiken de term $d(X, Y)$ om een dominosteen te beschrijven die aan de bovenkant X ogen heeft en aan de onderkant Y ogen. Voor de eenvoud nemen we aan dat dominostenen in een rijtje worden gelegd, waarbij steeds de onderkant van de eerste dominosteen evenveel ogen heeft als de bovenkant van de tweede dominosteen. Dit is bijvoorbeeld een geldig rijtje:

$$d(1,3), d(3,2), d(2,1), d(1,3), d(3,4)$$

Gegeven is nu een lijst van dominostenen en een tafel met een aantal vrije posities. De vraag is of de tafel volledig bedekt kan worden. We doen dat met het predicaat $legze/2$:

```
?- T = [_,_],legze(T,[d(1,2),d(1,3),d(2,3),d(3,4),(d1,4)]).
T = [d(1, 2), d(2, 3)] ;
T = [d(1, 3), d(3, 4)] ;
T = [d(2, 3), d(3, 4)] ;
false.
?- T = [_,_,_],legze(T,[d(1,2),d(1,3),d(2,3),d(3,4),(d1,4)]).
T = [d(1, 2), d(2, 3), d(3, 4)] .
?- T = [_,_,_],legze(T,[d(1,2),d(1,3),d(2,3),d(3,4),(d1,4)]).
T = [d(1, 2), d(2, 3), d(3, 4)] ;
false.
```

```
?- T = [_,_,_],legze(T,[d(1,2),d(1,3),d(2,3),d(4,2),d(3,4),(d1,4)]).
T = [d(1, 2), d(2, 3), d(3, 4)] ;
T = [d(1, 3), d(3, 4), d(4, 2)] ;
T = [d(2, 3), d(3, 4), d(4, 2)] ;
T = [d(4, 2), d(2, 3), d(3, 4)] ;
T = [d(3, 4), d(4, 2), d(2, 3)] ;
false.
?- T = [_,_,_,_],legze(T,[d(1,2),d(1,3),d(2,3),d(4,2),d(3,4),(d1,4)]).
T = [d(1, 2), d(2, 3), d(3, 4), d(4, 2)] ;
T = [d(1, 3), d(3, 4), d(4, 2), d(2, 3)] ;
false.
?- T = [_,_,_,_,_],legze(T,[d(1,2),d(1,3),d(2,3),d(4,2),d(3,4),(d1,4)]).
false.
```

1. Schrijf een versie van het predicaat $legze/2$ die gebruik maakt van $freeze/2$. (0.6/3)
2. Enkel als het niet lukt met $freeze/2$ in 1, schrijf een versie van het predicaat $legze/2$ die geen gebruik maakt van $freeze/2$. (0.3/3)