Laborator 4

Logică matematică și computațională

Găsirea soluțiilor

Introducere

Până acum, am implementat, în mare parte, predicate care reprezentau funcții, în sensul că intrarea funcției modelate de un asemenea predicat era reprezentată de primul argument (sau de primele argumente), iar ieșirea de ultimul argument (sau de ultimele argumente).

Acest mod de a programa era, însă, caracteristic programării funcționale, despre care se va vorbi la cursul omonim din anul II.

Forța Prolog-ului (și, în general, a programării logice) stă, de fapt, în lucrul cu predicate care nu reprezintă neapărat funcții, după cum se va vedea în acest laborator.

Exercițiul 1
Definiți un predicat `listaNelem/3` astfel încât, pentru orice `L`, `N`, `M`, `listaNelem(L,N,M)` este adevărat exact atunci când `M` este o listă cu `N` elemente care sunt toate elemente ale lui `L` (cu eventuale repetiții).

Interogați:

```
?- listaNelem([1,2,3],2,X).
?- listaNelem(L,1,[2]).
?- listaNelem(L,2,[2]).
?- listaNelem(L,2,[2,3]).
?- listaNelem(L,2,[2,3]), length(L,3).
?- length(L,3), listaNelem(L,2,[2,3]).
```

Toate soluțiile

Ce facem, însă, dacă vrem un predicat `listeNelem/3` astfel încât, pentru orice `L`, `N`, `LL`, `listeNelem(L,N,LL)` este adevărat exact atunci când `LL` este lista tuturor acelor `M` cu proprietatea că `listaNelem(L,N,M)`?

Există o soluție mai complexă care folosește doar conceptele introduse până acum, dar Prolog-ul ne furnizează și varianta:
`listeNelem(L,N,LL) :- bagof(M, listaNelem(L,N,M), LL).`

Pe lângă `bagof/3`, există mai multe asemenea „metapredicate”. Le vom descrie separat.

bagof/3

Interogați:

```
?- bagof((X,Y),
(member(X,[1,2,2,2,3]),member(Y,[0,1,2,3,4,5]),X<Y),L).
?- bagof(X,
(member(X,[1,2,2,2,3]),member(Y,[0,1,2,3,4,5]),X<Y),L).
?- bagof(X,
Y^(member(X,[1,2,2,2,3]),member(Y,[0,1,2,3,4,5]),X<Y),L).
```

În al doilea exemplu, variabila `Y` este *liberă*, așadar se caută soluții atât pentru ea, cât și pentru `L` (simultan). În al treilea exemplu, variabila `Y` este *cuantificată existențial*, în sensul că se caută „toți `X` astfel încât există `Y` astfel încât...".

setof/3

Interogați:

```
?- setof((X,Y),
(member(X,[1,2,2,2,3]),member(Y,[0,1,2,3,4,5]),X<Y),L).
?- setof(X,
(member(X,[1,2,2,2,3]),member(Y,[0,1,2,3,4,5]),X<Y),L).
?- setof(X,
Y^(member(X,[1,2,2,2,3]),member(Y,[0,1,2,3,4,5]),X<Y),L).
```

Comportamentul lui `setof/3` este similar cu cel al lui `bagof/3`, cu deosebirea că se încearcă eliminarea duplicatelor.

findall/3

Interogați:

```
?- findall(X,
(member(X,[1,2,2,2,3]),member(Y,[0,1,2,3,4,5]),X<Y),L).
?- findall((X,Y),
(member(X,[1,2,2,2,3]),member(Y,[0,1,2,3,4,5]),X<Y),L).
```

Comportamentul lui `findall/3` este similar cu cel al lui `bagof/3`, cu două deosebiri. În primul rând, semnul de cuantificare existențială nu mai este permis și, în același timp (sau chiar de aceea), orice variabilă așa-zis liberă va fi implicit cuantificată existențial.

În al doilea rând, cele două metapredicate au un comportament diferit atunci când nu există soluții:

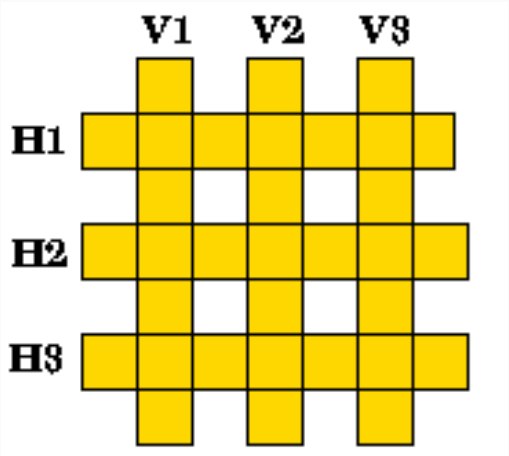
```
?- bagof(X, (member(X,[1,2,2,2,3]),0 is 1),L).
?- findall(X, (member(X,[1,2,2,2,3]),0 is 1),L).
```

Puzzle-uri simple

Exercițiul 2: cuvinte încrucișate
Șase cuvinte din engleză, anume:

abalone, abandon, anagram, connect, elegant, enhance

trebuie aranjate într-un puzzle de cuvinte încrucișate, ca în figură.



Exercițiul 2 (cont.)
Pornind de la faptele

```
word(abalone,a,b,a,l,o,n,e).
word(abandon,a,b,a,n,d,o,n).
word(anagram,a,n,a,g,r,a,m).
word(connect,c,o,n,n,e,c,t).
word(elegant,e,l,e,g,a,n,t).
word(enhance,e,n,h,a,n,c,e).
```

definiți un predicat `crosswd/6` care calculează toate variantele în care puteți completa grila. Primele trei argumente trebuie să fie cuvintele pe verticală, de la stânga la dreapta (V1, V2, V3), iar următoarele trei argumente trebuie să fie cuvintele pe orizontală, de sus în jos (H1, H2, H3).

Hint: Specificați că V1, V2, V3, H1, H2, H3 sunt cuvinte care au anumite litere comune. Unde este cazul, folosiți variabile anonime.

Exercițiul 3: labirint

Următoarele fapte (care se continuă pe slide-ul următor) descriu un labirint:

```
connected(1,2).
connected(3,4).
connected(5,6).
connected(7,8).
connected(9,10).
connected(12,13).
connected(13,14).
connected(15,16).
connected(17,18).
connected(19,20).
```

Exercițiul 3 (cont.)

```
connected(4,1).
connected(6,3).
connected(4,7).
connected(6,11).
connected(14,9).
connected(11,15).
connected(16,12).
connected(14,17).
connected(16,19).
```

Faptele indică ce puncte sunt conectate (din ce punct se poate ajunge într-un alt punct într-un pas).

Drumurile sunt cu sens unic (se poate merge pe ele doar într-o direcție).

De exemplu, se poate ajunge într-un pas de la 1 la 2, dar nu și invers.

Exercițiul 3 (cont.)

Adăugați un predicat `path/3` care indică dacă dintr-un punct se poate ajunge într-un alt punct, în mai mulți pași, cel de-al treilea argument reprezentând lista pașilor. Pe baza lui, construiți un predicat `pathc/2` care spune doar dacă dintr-un punct se poate ajunge într-un alt punct.

Interogați:

```
?- pathc(5,10).
?- path(5,10,L).
?- pathc(6,X).
?- path(6,X,L).
?- pathc(X,13).
```

Lucrul cu Prolog-ul pe desktop

Utilizarea SWI-Prolog

Pe lângă SWISH, putem lucra și în aplicația desktop SWI-Prolog. Programul propriu-zis se scrie într-un fișier, care se încarcă din consola programului, unde se vor scrie și interogările.

Comenzi utile:

- `pwd.` pentru a afla directorul curent;
- `[fisier].` pentru încărcarea fișierului `fisier.pl` (el trebuie reîncărcat la fiecare modificare);
- `;` pentru a se genera următoarea soluție;
- *enter* pentru a nu mai căuta altă soluție;
- CTRL+C și a pentru oprirea unei căutări în desfășurare.

Puzzle-ul *Countdown*

Exercițiul 4: cel mai lung cuvânt
Acest exemplu provine din
Ulle Endriss, *Lecture Notes – An Introduction to Prolog Programming.*
și a mai fost folosit în trecut în laboratoare de programare logică în FMI.

Countdown este un joc de televiziune popular în Marea Britanie în care jucătorii trebuie să găsească un cuvânt cât mai lung cu literele dintr-o mulțime dată de nouă litere.

Să încercăm să rezolvăm acest joc cu Prolog!

Exercițiul 4 (cont.)
Concret, vom încerca să găsim o soluție optimă pentru următorul joc:

Primind o listă cu litere din alfabet (nu neapărat unice),
trebuie să construim cel mai lung cuvânt format din literele date
(pot rămâne litere nefolosite).

Vom rezolva jocul pentru cuvinte din limba engleză.

Scorul obținut este lungimea cuvântului găsit.

Exercițiul 4 (cont.)
Scopul final este de a construi un predicat în Prolog `topsolution/2`, cu următorul comportament:

dându-se o listă de litere în primul său argument, trebuie să returneze în al doilea argument o soluție cât mai bună, adică un cuvânt din limba engleză de lungime maximă care poate fi format cu literele din primul argument.

```
?- topsolution([r,d,i,o,m,t,a,p,v],Word).
Word = dioptra
```

Exercițiul 4 (cont.)
Începeți prin a descărca fișierul `words.pl` în același director cu fișierul programului vostru.

Acest fișier conține o listă cu peste 350.000 de cuvinte din limba engleză, de la `a` la `zyzzyva`, sub formă de fapte.

Scrieți `[words].` în consolă pentru a încărca aceste fapte.

Exercițiul 4 (cont.)
Predicatul predefinit din Prolog `atom_chars(Atom,CharList)` descompune un atom într-o listă de caractere.

Folosiți acest predicat pentru a defini un predicat `word_letters/2` care transformă un cuvânt (i.e, un atom în Prolog) într-o listă de litere.
De exemplu:

```
?- word_letters(hello,X).
X = [h,e,l,l,o]
```

Ca o paranteză, observați că puteți folosi acest predicat pentru a găsi cuvinte în engleză de 45 de litere:

```
?- word(Word), word_letters(Word,Letters),
   length(Letters,45).
```

Exercițiul 4 (cont.)
Mai departe, scrieți un predicat `cover/2` care, primind două liste, verifică dacă a doua listă „acoperă” prima listă (i.e., verifică dacă fiecare element care apare de k ori în prima listă apare de cel puțin k ori în a doua listă).

De exemplu

```
?- cover([a,e,i,o], [m,o,n,k,e,y,b,r,a,i,n]).
true

?- cover([e,e,l], [h,e,l,l,o]).
false
```

Exercițiul 4 (cont.)
Scrieți un predicat `solution/3` care primind o listă de litere ca prim argument și un scor dorit ca al treilea argument, returnează prin al doilea argument un cuvânt cu lungimea egală cu scorul dorit, „acoperit” de lista respectivă de litere.

De exemplu

```
?- solution([g,i,g,c,n,o,a,s,t], Word, 3).
Word = act
```

Exercițiul 4 (cont.)
Implementați acum predicatul `topsolution/2`.

Testați, de exemplu, predicatul definit pe mulțimea de litere:

```
[y,c,a,l,b,e,o,s,x]
```

Aceasta este una dintre listele de litere folosite în ediția de *Countdown* din 18 decembrie 2002 din Marea Britanie, în care Julian Fell a obținut cel mai mare scor din istoria concursului. Pentru lista de mai sus, el a găsit cuvântul *cables*, câștigând astfel 6 puncte.

Poate programul vostru să bată acest scor?