Laboratorul 5 - Programare Logică și Funcțională

Seria 36

Martie 2024

Predicatul `numlist/3` primeşte două capete de interval `A` şi `B` şi returnează în al treilea argument toate elementele întregi din intervalul $[A, B]$, Dacă $A > B$, atunci întoarce `false`.

```
?- numlist(-2, 3, R).
R = [-2, -1, 0, 1, 2, 3]
```

Predicatul `findall/3`, descris de `findall(X, P, R)` găseşte toţi acei `X` care respectă proprietatea `P`, şi îi returnează în lista `R`. Predicatul `setof` are acelaşi comportament, dar asigură că rezultatul este strict o mulţime (nu conţine elemente duplicate).

1 Exerciţiul 1

1. Scrieţi un predicat `divisorsPairs/3` care determină toate perechile `(X, Y)`, astfel încât `X` şi `Y` sunt elemente ale listei determinată de primele două argumente ale predicatului, iar al treilea argument este lista perechilor cerute.

   ```
   ?- divisorsPairs(2, 6, LR).
   LR = [(2,2), (3,3), (4,2), (4,4), (5,5), (6,2), (6,3), (6,6)]
   ```

2. Scrieţi un predicat `oddIndexes/2` care să extragă toate elementele de pe poziţiile impare dintr-o listă, fără a utiliza recursia.

   ```
   ?- oddIndexes([a,b,c,d,e], LR).
   LR = [b, d]
   ```

2 Exerciţiul 2

În cadrul acestui exerciţiu, ne dorim să implementăm un algoritm *breadh-first search* (căutarea în lăţimea arborelui). Pentru a face acest lucru:

- definiţi o bază de cunoştinţe cu două predicate, `s/2` şi `objective/1`, pentru a defini funcţia *succesor* (muchia dintre două noduri) şi pentru a defini nodurile-obiectiv (cele la care vrem să ajungem, plecând de la un nod iniţial).

  ```
  s(1, 2).
  s(1, 3).
  s(2, 4).
  s(2, 5).
  s(3, 5).
  s(3, 4).
  objective(5).
  ```

- În această abordare putem căuta simultan în mai multe direcţii, în funcţie de cine este nodul curent şi cine sunt succesorii nodului curent. Astfel, ne dorim un predicat `extend/2`, care primeşte ca intrare un drum parţial şi returnează toate drumurile care se pot forma. **Important!** Drumurile se vor obţine în ordine inversă, astfel că `Head`-ul listei va fi mereu nodul curent. Utilizaţi predicatul `findall/3`.

  ```
  ?- extend([3, 1], L).
  L = [[5, 3, 1], [4, 3, 1]]
  ```

- Scrieţi un predicat `breadthfirst/2` care primeşte o listă de drumuri şi, a. dacă un drum a ajuns la nodul obiectiv, îl returnează, altfel extinde drumul, îl concatenează la celelalte deja existente şi reapelează recursiv.
- Scrieţi un predicat `solve/2` care primeşte un nod de start şi returnează toate drumurile (în ordine inversă) până la nodul obiectiv.

```
?- solve(1, R).
R = [[5, 2, 1], [5, 3, 1]]
```

3 Exerciţiul 3

Vrem să rezolvăm în Prolog următorul puzzle - **Zebra puzzle**. (`https://en.wikipedia.org/wiki/Zebra_Puzzle`).

Pentru fiecare personaj ştim următoarele:

- locuieşte într-o casă care are o anumită culoare;
- are o naţionalitate;
- are un anumit animal de companie;
- are o băutură preferată;
- fumează un anumit tip de ţigări.

Avem următoarele informaţii:

1. Sunt cinci case.
2. Englezul locuieşte în casa roşie.
3. Spaniolul are un câine.
4. În casa verde se bea cafea.
5. Ucraineanul bea ceai.
6. Casa verde este imediat în dreapta casei bej.
7. Fumătorul de *Old Gold* are melci.
8. În casa galbenă se fumează *Kools*.
9. În casa din mijloc se bea lapte.
10. Norvegianul locuieşte în prima casă.
11. Fumătorul de *Chesterfields* locuieşte lângă cel care are o vulpe.
12. *Kools* sunt fumate în casa de lângă cea în care se ţine calul.
13. Fumătorul de *Lucky Strike* bea suc de portocale.
14. Japonezul fumează *Parliaments*.
15. Norgenialul locuieşte lângă casa albastră.

Scopul este să determinăm naţionalitatea posesorului zebrei.
Pentru a putea rezolva, veţi defini următoarele:

1. Definiţi un predicat `right(X, Y)` care este adevărat când `X is Y + 1`.
2. Definiţi un predicat `left(X, Y)` care este adevărat când `Y is X + 1`.
3. Definiţi un predicat `near(X, Y)` care este adevărat când X este sau la stânga, sau la dreapta dreapta lui `Y`.

Veţi reprezenta casele prin:

```
house(Number,Nationality,Colour,Pet,Drink,Cigarettes)
```

iar soluţia va fi un predicat de forma:

```
solution(Street, ZebraOwner) :-
    Street = [
        house(1,_,_,_,_),
        house(2,_,_,_,_),
        house(3,_,_,_,_),
        house(4,_,_,_,_),
        house(5,_,_,_,_),
    ],
    member(house(_,english,red,_,_,_), Street),
    member(house(_,spanish,_,dog,_,_), Street),
    member(house(_,_,green,_,coffee,_), Street),
    ...,
    member(house(_,ZebraOwner,_,zebra,_,_), Street).
```

Completaţi toate informaţiile pentru a obţine soluţia.

4 Exerciţiul 4

Considerăm predicatele `tree/3` pentru reprezentarea arborilor binari, respectiv `nil/0` pentru reprezentarea arborelui vid. De exemplu, `tree(Anything, nil, nil)` este o frunză. Implementaţi:

- Cele trei tipuri de parcurgeri ale arborilor binari (in-ordine, pre-ordine, post-ordine).
- Scrieţi un predicat care să determine toate frunzele din arbore.