Laboratorul 4 FLP

Recapitulare

Laboratorul 1

- sintaxa limbajului Prolog;
- cum arata un program in Prolog;
- facts, reguli de derivare.
- Exemple: is_bigger, arbore_genealogic cu female/1, male/1, parent/2.

Laboratorul 2

- recursivitate si liste.

Laboratorul 3

- cum raspunde Prolog intrebarilor;
    - algoritmul de unificare.
    - algoritmul de unificare lucreaza cu trei operatii: SCOATE, DESCOMPUNE si REZOLVA.
    - SCOATE - elimina egalitatile redundante din lista de rezolvat;
    - DESCOMPUNE - unifica argumentele pentru doi termeni identici din punct de vedere structural;
    - ex: $f(t_1, t_2,... t_n)=f(t_1', t_2', ... t_n')$;
    - REZOLVA: daca am egalitati de forma $x = t$ sau $t = x$, unde t nu contine x, atunci pot adauga in multimea solutiei egalitatea $x = t$, iar toate aparitiile lui x vor fi substituite cu t in restul solutiei si in restul listei de rezolvat.

Laboratorul 4

In acest laborator vom rezolva doua probleme, **Zebra puzzle** si **Countdown**.

Zebra Puzzle

Avem 5 case cu 5 atribute fiecare. + Numarul casei (de la 1 la 5) + Nationalitatea celui care locuieste in casa + Culoare + Animalul de companie + Bautura preferata + Ce tigari fumeaza

```
casa(Numar,Nationalitate,Culoare,Animal,Bautura,Tigari).
```

Trebuie sa determinam ce nationalitate are posesorul zebrei, stiind un set de constrangeri.

```
la_dreapta(X, Y) :- X is Y + 1.
la_stanga(X, Y) :- la_dreapta(Y, X).
langa(X, Y) :- la_dreapta(X, Y) ; la_stanga(X, Y).

solutie(Strada, PosesorZebra) :-
    Strada = [
        casa(1,_,_,_,_,_),
        casa(2,_,_,_,_,_),
        casa(3,_,_,_,_,_),
        casa(4,_,_,_,_,_),
        casa(5,_,_,_,_,_)
    ],
    member(casa(_,englez,rosie,_,_,_), Strada),
    member(casa(_,spaniol,_,caine,_,_), Strada),
    member(casa(_,_,verde,_,cafea,_), Strada),
    member(casa(_,ucrainean,_,_,ceai,_), Strada)
    member(casa(A,_,verde,_,_,_), Strada),
    member(casa(B,_,bej,_,_,_), Strada),
    la_dreapta(A, B),
    member(casa(_,_,_,melci,_,'Old Gold'), Strada),
    member(casa(C,norvegian,_,_,_,_), Strada),
    member(casa(D,_,albastra,_,_,_), Strada),
    langa(C, D),
    ...
    member(casa(_,PosesorZebra,_,zebra,_,_),Strada).
```

Pentru a rula, intrebarea pentru Prolog este

```
?- solutie(Strada, PosesorZebra).
Strada = ...
PosesorZebra = japonez
```

Tema: de trimis codul integral pentru aceasta problema, pe *bogdan.macovei.fmi@gmail.com*.

Countdown

Fiind data o lista de litere, sa se gaseasca cel mai lung cuvant din limba engleza care se poate forma folosind toate (sau o parte) dintre aceste litere.

```
% word/1 - cuvintele din limba engleza
word(hello).
word(something).

% cum includem fisierul in fisierul nostru
:- include('words.pl').
```

Predicate utile:

```
% atom_chars/2
% atom_chars(Word, Letters) are ca efect
%  obtinerea listei de litere in Letters
%  pentru cuvantul Word
% atom_chars(hello, X).
% X = [h, e, l, l, o].
```

```
% select/3
% select(+Elem, +List, -ListR).
% elimina prima aparitie a lui Elem in List
% si intoarce rezultatul in ListR
% select(8, [1,3,8,5,8,9], LR).
% LR = [1,3,5,8,9].
```

Solutie:

Prima solutie ar fi sa generam toate cuvintele posibile, utilizand lista de litere primita. L - lista de litere $\exists \sigma\in S_{lenght(L)}$ astfel incat $\sigma(L) = w \in KB$.

Pot exista litere in lista pe care sa nu le folosesc. Inseamna ca pot genera toate permutarile luand o litera din lista, doua litere, ..., n litere din lista.

$Nr = \sum_{i}A_{N}^{i} > \sum_{i}C_{N}^{i}=2^N$

Un prim pas ar fi sa verific daca o lista de litere acopera o alta lista de litere. *cover/2*

cover([b,a,e,s,c],[c,a,b,l,e,s]). [b | [a, e, s, c]] [c, a, b, l, e, s] => [c, a, l, e, s] [a, e, s, c] [c, a, l, e, s]

```
 % returneaza True daca
% prima lista poate acoperi a doua lista
:- include('words.pl').

cover([], _).
cover([H | T], L) :-
    select(H, L, R),
    cover(T, R).

solution(ListLetters, Word, Score) :-
    word(Word),
    atom_chars(Word, Letters),
    length(Letters, Score),
    cover(Letters, ListLetters).

search_solution(_, 'no solution', 0).
search_solution(ListLetters, Word, X) :-
    solution(ListLetters, Word, X).
search_solution(ListLetters, Word, X) :-
    Y is X - 1,
    search_solution(ListLetters, Word, Y).

topsolution(ListLetters, Word) :-
    length(ListLetters, Score),
    search_solution(ListLetters, Word, Score).

?- topsolution([y,c,a,l,b,e,o,s,x], X).
```