LABORATOR 1

Prolog (Sicstus, SWI) este un limbaj de programare simbolic, potrivit pentru probleme ce implica obiecte si relatii intre obiecte. Un program Prolog este o baza de cunostinte care poate fi interogata.

Sa consideram urmatorul program Prolog (salvat intr-un fisier text cu extensia **pl**):

```
parinte(ion,maria).
parinte(ana,maria).
parinte(ana,dan).
parinte(maria,elena).
parinte(maria,radu).
parinte(elena,nicu).
parinte(radu,gigi).
parinte(radu,dragos).
```

Pentru a interoga un program Prolog, acesta trebuie consultat.

```
consult('c:\\prolog\\pro.pl').
```

In fereastra de interogare introduceti urmatoarele cereri:

```
parinte(ana,maria).
parinte(ion,radu).
parinte(X,maria).
parinte(X,Y).
```

Obs.
1) constantele alfanumerice (atomi) incep cu litera mica;
2) variabilele incep cu litera mare;
3) raspunsurile se obtin in ordinea in care sunt puse informatiile in baza de cunostinte;
4) daca intrebarea contine variabile, Prolog gaseste obiectele particulare (instante) pentru care raspunsul este adevarat.

Pentru a gasi cine este bunicul lui radu, putem intreba

```
parinte(X,Y), parinte(Y,radu).
```

Reguli Prolog

O regula are forma generala $R:-C_1, C_2, \ldots, C_N$ si are semnificatia:

$\forall X_1, X_2, \ldots X_k$ variabile ce apar in regula, R este adevarat daca $C_1, C_2, \ldots, C_N$.

Exemplu

```
copil(X, Y):-parinte(Y, X).
```

adica $\forall$ X,Y daca Y este parintele lui X atunci X este copilul lui Y.

Plecand de la relatia parinte, se se scrie relatiile

```
frate(X,Y) (X este frate cu Y)
bunic(X,Y) (X este bunicul lui Y).
```

Pentru a defini o relatie predecesor, avem nevoie de o regula recursiva:

```
pred(X,Y):-parinte(X,Y).
pred(X,Z):-parinte(X,Y), pred(Y,Z).
```

Obs. Domeniul de valabilitate al unei variabile este clauza in care apare variabila respective.
Sa se scrie o alta varianta a relatiei predecesor.

In Prolog putem lucra cu structuri – mai multe componente combinate intr-un singur obiect si se poate descrie in general astfel:

functor(arg_1, arg_2, …, arg_N).

Obs.

1) functorul este atom (incepe cu litera mica);
2) argumentele pot fi structuri;
3) o structura este bine definite de functorul principal si de aritate (nr de argumente).

Exemple

data(1,octombrie,2005).
segment(punct(1,2),punct(6,7)).
punct(1,2) si punct(1,2,3) sunt structuri diferite

Unificarea (matching)

Este singura operatie permisa intre termenii Prolog.

Def Se numeste substitutie multimea

$\Theta=\{(X_i,t_i)_{i=1,n} \mid X_i$ variabile, t_i termeni Prolog si $X_i \neq X_j$ daca $i \neq j\}$

Daca T este un termen Prolog, notam TQ termenul obtinut din T prin inlocuirea simultana a tuturor aparitiilor X_i cu t_i unde $(X_i,t_i) \in \Theta$.

Def Un termen este complet instantiat daca nu contine nici o variabila.

Def T1 unifica cu T2 daca $\exists \Theta$ astfel incat $T1\Theta=T2\Theta$. Θ se numeste unificator.

Exemple

1) T1=f(X,a)
   T2=f(b,Y)
   $\Theta=\{(X,b), (Y,a)\}$
2) T1=X
   T2=f(g(Y,Z),b)
   $\Theta1=\{(X, f(g(Y,Z),b))\}$
   $\Theta2=\{(Y,a), (Z,c), (X,f(g(a,c),b))\}$

$\Theta1$ este mai generala decat $\Theta2$.

Predicatul = spune daca 2 termeni Prolog unifica. In caz afirmativ, se da substitutia cea mai generala.

Sa se scrie urmatoarea intrebare

X=f(X).

Obs.

1) doi atomi unifica daca sunt identici;
2) o variabila poate unifica cu orice;
3) doua structuri unifica daca au acelasi functor principal, acelasi numar de argumente si argumentele ce se corespund unifica.

Aritmetica in Prolog

= = egalitate de termeni
1+2= = 2+1 no
=:= egalitate de valori numerice a doua expresii aritmetice
1+2=:=2+1 yes
\= = termeni diferiti
X is Y variabila X este instantiata cu valoarea Y.
X is 3+2.

Liste

O lista este o secventa de oricate articole separate prin virgula.

[1,2,3,4,5,a,b,c]

Lista vida se noteaza [].

O lista nevida se poate imparti in cap si coada [A|B]. Capul A este un singur element. Coada B este lista.

Putem pune in evidenta mai multe elemente la inceputul listei:

[A,B,C,…|T].

Exemplu

[1,2,3]=[1|[2,3]]=[1,2|[3]]=[1,2,3|[]].

Putem colecta intr-o lista elemente ce satisfac o anumita proprietate cu ajutorul a 3 predicate predefinite:

1) bagof(X,P,L) pune in lista L elementele X ce satisfac P. Daca nu exista nici un astfel de element raspunsul este no.

2) setof(X,P,L) la fel ca bagof dar elimina duplicatele iar lista rezultata este sortata

3) findall(X,P,L) daca nu exista nici un element care sa satisfaca P rezultatul este yes iar L=∅. Nu tine cont de variabilele care apar in P si nu apar in X.

Exemplu:

Presupunand ca avem relatii de tipul baiat(nume,varsta), se se calculeze suma varstelor baietilor din baza de cunostinte.

```
varstabaieti(L):-findall(Varsta,baiat(Nume,Varsta),L).
suma(S):-varstabaieti(L),suma(L,S).
suma([],0).
suma([H|T],S):-suma(T,S1), S is S1+H.
```

Eficienta programelor Prolog

Functia

$$f(x) = \begin{cases} 0, x \leq 3 \\ 2, x \in (3,6] \\ 4, x > 6 \end{cases}$$

poate fi implementata in Prolog prin urmatoarele reguli:

```
f(X,0):-X=<3.
f(X,2):-3<X, X=<6.
f(X,4):-6<X.
```

Daca punem intrebarea

```
f(1,Y),2<Y.
```

cautarea solutiei se face pe toate cele trei reguli desi logic ar fi trebuit sa se opreasca dupa prima regula. Putem preveni backtracking-ul cu ajutorul predicatului ! (*cut*). Programul devine astfel:

```
f(X,0):-X=<3,!.
f(X,2):-3<X, X=<6,!.
f(X,4):-6<X.
```

Daca X=<3 e adevarat nu se cauta mai departe (se taie backtracking-ul).

In regula 2 testul 3<X este redundant deoarece se ajunge aici doar daca nu s-a atins *cut*-ul din prima regula.

Deci programul poate fi scris:

```
f(X,0):-X=<3,!.
f(X,2):-X=<6,!.
f(X,4).
```

Exercitii

1) Sa se scrie predicatul max care calculeaza maximul dintre 2 valori.

2) Sa se scrie predicatele membru si concat.

3) Sa se calculeze suma alternata a elementelor unei liste.

4) Sa se elimine aparitia unui element dintr-o lista (aparitia tuturor elementelor).

5) Sa se inverseze o lista; sa se genereze toate permutarile elementelor unei liste.

6) Sa se afle numarul aparitiilor unui element intr-o lista.

7) Sa se insereze un element pe o anumita pozitie intr-o lista.

8) Sa se interclaseze doua liste ordonate crescator.