**F.I.A.** **Cătălin Stoean**
**Laboratorul numărul 2**

# Noțiuni generale

Un program Prolog este alcătuit din **predicate**. Fiecare predicat este definit de *numele* său şi de un număr numit *aritatea* sa. Aritatea reprezintă un numar fix de argumente ale predicatului. Două predicate care au acelaşi nume, dar un număr diferit de argumente se consideră că sunt predicate diferite. Fiecare predicat dintr-un program este definit de existența uneia sau mai multor **clauze**.
Exemplu:

femeie(elena).

parinte(andrei, cristi).
parinte(andrei, elena).

În exemplul de mai sus, predicatul *parinte* are doua clauze, iar predicatul *femeie* are o singură clauză. O clauză poate fi un **fapt** sau o **regulă**. Un fapt are forma

nume_relatie(arg1, …, argN).

iar o regulă

nume_rel(arg1, …, argN) :- rel1(…), …, relM(…).

unde arg1, …, argN reprezintă argumentele. Ca argument, poate fi folosit orice **termen** al limbajului Prolog. Termenii de bază sunt:

- Întregii – numere pozitive sau negative
- Atomii – text care începe cu literă mică
- Variabilele – încep cu literă mare sau *underline* (_)
- Structuri compuse

*Întregii* în Prolog, sunt reprezentați de cifre (0 - 9).
*Atomii* sunt alcătuiți de obicei din litere şi cifre, având primul caracter o literă mică.
Exemplu de atomi:

salut
douaCuvinteAlaturate
un_atom
a2

Următoarele elemente nu reprezintă atomi:

nu-este-atom
5nu
_faraunderline
Litramare

Dacă avem caractere introduse între apostrofuri formăm atomi:
'acesta-este-atom'
'inca un atom'
'Atom'

*Variabilele* sunt asemănatoare atomilor, cu excepția că ele încep cu literă mare sau cu underline (_).
Exemple:

Variabila
_variabila
Alta_vaRiabila2

În sfărsit, *structurile compuse* sunt de forma următoare:

persoana(maria, inginer, adresa("Calea Bucuresti", "Bloc A10", "Ap 11")).

## Variabila anonimă

Variabilele se notează, după cum afirmam mai sus, cu şiruri de caractere care încep cu literă mare sau cu şiruri de caractere ce încep cu '_'. Acestea din urmă se numesc variabile anonime. Care este rolul lor? Vom observa, urmărind exemplul de mai jos.

Avem predicatele: *parinte(numeCopil, numePărinte), barbat(numeBărbat), femeie(numeFemeie)* şi vrem să afişăm pentru o persoană toți copiii şi toți nepoții. Pentru aceasta, definim predicatele *copii(+numePărinte, -numeCopil)* şi *nepoti(+numeBunic, -numeNepot)*.

Observație: Notația nume_predicat(+term1, -term2) reprezintă faptul că termenul *term1* este termen de intrare, iar *term2* este termen de ieşire.

Presupunem că avem introduse faptele parinte(…), barbat(…) şi femeie(…). Scriem în continuare regulile de calcul pentru afişarea copiilor şi a nepoților.

```
copii(Tata) :- parinte(Copil, Tata), tab(2), write_ln(Copil), fail.
nepoti(Bunic) :- parinte(Tata, Bunic), parinte(Nepot, Tata), tab(2), write_ln(Nepot), fail.
```

Putem testa aceste predicate şi observăm că afişează fiecare copiii, respectiv nepoții pentru persoana dată de noi. Spuneam însă că vrem să afişăm pentru o persoană atât copiii, cât şi nepoții în cadrul aceleiaşi interogări. Pentru aceasta, construim un nou predicat:

```
descendenti(X) :- write_ln('Copiii:'),  copii(X), write_ln('Nepotii:'), nepoti(X).
```

Ce observăm însă? Acesta nu afişează decât copiii! Am văzut că, individual, fiecare regulă funcționează: atunci care este problema? În rândurile următoare o să vedem de ce. Ei bine, într-adevăr funcționează separat dar pentru fiecare, după ce afişează elementele cerute, ne răspunde cu *No* datorită predicatului implicit *fail*. Pentru a putea folosi

predicatul *copii/1* în conjuncție cu alte predicate, nu trebuie decât să mai adăugăm o clauză *copii/1* care răspunde întotdeauna cu *Yes*:

```
copii(Tata) :- parinte(Copil, Tata), tab(2), write_ln(Copil), fail.
copii(OriceVariabila).
```

În acest caz, atunci când prima clauză va întoarce *No* (pentru că nu mai găseşte alți copii), sistemul va încerca cea de a doua clauză *copii/1*. Devreme ce argumentul celei de a doua clauze este o variabilă, sistemul va întoarce întotdeauna răspunsul *Yes*.

În această a doua clauză, pe noi nu ne interesează ce variabilă folosim. Pentru asemenea situații, folosim *variabila anonimă* care este notata prin underline (_) sau printr-un şir de caractere care începe cu underline:

```
copii(_).
```

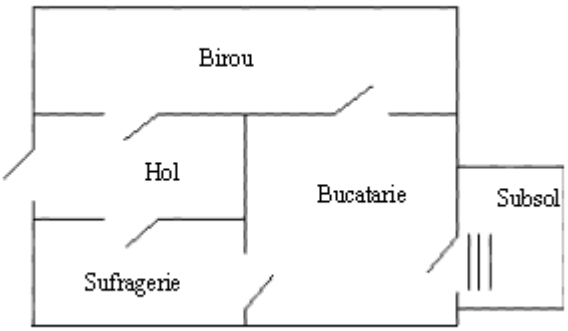
Astfel, obținem următorul program:

```
copii(Tata) :- parinte(Copil, Tata), tab(2), write_ln(Copil), fail.
copii(_).

nepoti(Bunic) :- parinte(Tata, Bunic), parinte(Nepot, Tata), tab(2), write_ln(Nepot), fail.
nepoti(_).

descendenti(X) :- write_ln('Copiii:'),  copii(X), write_ln('Nepotii:'), nepoti(X).
```

Observație: nume_predicat/n reprezintă faptul că nume_predicat este un predicat de aritate n.

## Exercițiu

Presupunem că avem următoarea schemă a unei case:




În fiecare cameră se găsesc următoarele lucruri:

- în birou: masă, computer;
- în hol: cuier;
- în sufragerie: televizor, bibliotecă;

- în bucătărie: aragaz, frigider, biscuiți, mere;
- la subsol: maşina de spălat, rufe murdare.

Definiți legăturile dintre camere (eventual printr-un predicat numit *uşă*), locurile unde sunt situate diferitele obiecte, locul unde se află subiectul. La un moment dat, dacă apelez un predicat numit *start* sistemul să spună care este camera în care se află subiectul, care sunt camerele în care poate merge şi care sunt lucrurile pe care le poate vedea în camera în care se află.
Exemplu:?- start.

Esti in bucatarie
Poti vedea:
 aragaz
 frigider
 biscuiti
 mere

Poti mege in:
 birou
 subsol
 sufragerie

## Operații matematice

Pentru evaluarea unei expresii aritmetice, în Prolog folosim predicatul predefinit *is*. Sintaxa acestui operator este următoarea:

X is <expresie aritmetică>.

Variabila X va lua valoarea expresiei aritmetice. Important este faptul că variabila X nu trebuie să aibă anterior o valoare asignată. Expresia aritmetică arată ca o expresie aritmetică din orice alt limbaj de programare.
Exemplu:

? – X is 2 + 2.
X = 4

? – X is 3 * (4 + 1).
X = 15

În afară de predicatul *is*, putem folosi operatori pentru compararea numerelor: <, >, =<, >=, ==, =\=. Ultimii doi operatori sunt pentru a verifica egalitatea dintre două numere, respectiv pentru a verifica dacă două numere sunt diferite.
Exemplu:

? – X is 2 + 2, X > 3.
X = 4
Yes

Operatorii pot fi folosiți şi în cadrul regulilor:

suma(N1, N2, S) :- S is N1 + N2.

## Important

Trebuie să înțelegem faptul că scrierea ”2+3” în Prolog nu reprezintă o instrucțiune care păstrează rezultatul acestei adunări; ea reprezintă mai degrabă ”adunarea lui 2 cu 3”. Astfel, ”2+3” este un termen diferit de ”4+1” şi bineînțeles diferit de ”5*1”. Pentru a înțelege aceasta, iată următorul exemplu:

```
numar(3).
numar(4).
numar(5).
```

Presupunem că avem aceste fapte într-un program Prolog. Dacă adresăm interogarea:

```
? – numar(2 + 1).
```

ni se va răspunde cu *No* pentru că termenul 2 + 1 nu poate fi *unificat* cu vreunul din termenii din baza de cunoştințe. La o interogare de forma :

```
? – X is 2 + 1, numar(X).
```

vom primi următorul răspuns:

```
X = 3
Yes
```

În acest caz, s-a realizat întâi calculul, (datorită predicatului *is*)iar apoi a fost realizată interogarea numar(3).

## Interogări

Introduceți următoarele interogări şi asigurați-vă că înțelegeți de ce primiți rezultatele oferite de sistem în fiecare caz:

- N is 1+1.
- N is 1+1, P is N*2, Q is P+Q.
- N is X+1.
- I is I+1.
- I is 6, I is I+1.
- I is 6, J is I+1.

## Exerciții

1. Definiți un predicat care să adauge o unitate la un număr dat.
2. Maximul a două numere.
3. Maximul a trei numere.
4. Valoarea absolută a unui număr.
5. Funcția f(x) care este x-1 dacă x > 0 şi 0 altfel.