# *Limbajul PROLOG.*

## Lansare mediu PROLOG

Pentru dezvoltarea programelor, vom folosi implementarea SWI-PROLOG. Lansarea acestuia se face apăsând butonul *Start* si efectuând selectiile *Programs -> SWI Prolog -> SWI Prolog*. Pe ecran va apare fereastra aplicatiei si prompterul ? care arata ca programul asteapta comenzi.

**Comenzi uzuale:**

? – pwd.

Afiseaza directorul curent.

?-ls.

Afiseaza fisierele din directorul curent.

?-cd('cale').

Schimba directorul curent.

Exemplu:

?-cd("C:\\BIA\\Prolog").

Directorul curent devine c:\BIA\Prolog.

?-consult('numeFisier.pl').

Fişierul cu numele *numeFisier.pl* este încărcat în memorie.

?-halt.

Aceasta comanda încheie sesiunea de lucru curenta în SWI-Prolog.

Pentru a se afisa informatii cu privire la o comanda, apelam:

?-help(nume_comanda).

Exemplu:

?-help(halt).

**Important:**

1. Extensia implicită a unui fişier construit cu SWI-Prolog este **.pl**.
2. Fiecare enunț în Prolog se termina cu caracterul punct (.).
3. Enunțurile (întrebările) se introduc la prompterul ?- şi cele mai simple au structura sintactică identică cu cea a unui fapt.

## Entitățile limbajului PROLOG

Prolog este un limbaj logic, descriptiv, care permite specificarea problemei de rezolvat în termenii unor *fapte* cunoscute despre obiectele universului problemei şi a *relațiilor* existente între aceste *obiecte*. Execuția unui program Prolog constă în deducerea implicațiilor acestor fapte şi relații, programul definind astfel o mulțime de consecințe ce reprezintă înțelesul sau semnificația declarativă a programului.

Un program Prolog conține următoarele entități:

- *fapte* despre obiecte şi relațiile existente între aceste obiecte;
- *reguli* despre obiecte şi relațiile dintre ele, care permit deducerea (inferarea) de noi fapte pe baza celor cunoscute;
- *întrebări*, numite şi *scopuri*, despre obiecte şi relațiile dintre ele, la care programul răspunde pe baza faptelor şi regulilor existente.

### a) Fapte

Faptele sunt predicate de ordinul întâi de *aritate* ***n*** considerate adevărate (reprezinta cea mai simpla forma de predicat din Prolog). Ele stabilesc relații între obiectele universului problemei. Numărul de argumente ale faptelor este dat de *aritatea* (numărul de argumente) corespunzătoare a predicatelor.

**Exemple**:

| **Fapt:** | **Aritate:** |
|---|---|
| câine(bobi). | 1 |
| place(ion, ioana). | 2 |
| place(ion, ana). | 2 |
| frumoasă(ana). | 1 |
| bun(daniel). | 1 |
| deplasează(cub, camera1, camera2). | 3 |

Interpretarea particulară a predicatului şi a argumentelor acestuia depinde de programator. Ordinea argumentelor, odată fixată, este importantă

şi trebuie păstrată la orice altă utilizare a faptului, cu aceeaşi semnificație. Mulțimea faptelor unui program Prolog formează *baza de cunoştințe Prolog*. Asa cum vom arata in cele ce urmeaza, în *baza de cunoştinte* a unui program Prolog sunt incluse şi regulile Prolog.

## b) Scopuri

Obținerea consecințelor sau a rezultatului unui program Prolog se face prin fixarea unor scopuri care pot fi *adevărate* sau *false*, în funcție de conținutul *bazei de cunoştințe*. *Scopurile* sunt predicate pentru care se doreşte aflarea valorii de adevăr în contextul faptelor existente în baza de cunoştințe. Cum scopurile pot fi văzute ca întrebări, rezultatul unui program Prolog este răspunsul la o întrebare (sau la o conjuncție de întrebări). Acest răspuns poate fi afirmativ, **yes**, sau negativ, **no**. Un program Prolog, în cazul unui răspuns afirmativ la o întrebare, poate furniza şi alte informații din baza de cunoştințe.

**Exemplu:**

Considerând baza de cunoştințe specificată anterior, se pot pune diverse întrebări, cum ar fi:

**?-** place(ion, ioana).
**Yes** **%** deoarece acest fapt există în baza de cunoştințe
**?-** caine(bobi).
**Yes** **%** deoarece şi acest fapt există în baza de cunoştințe
**?-** papagal(ion).
N**o** **%** deoarece acest fapt **nu** există în baza de cunoştințe
**?-** bun(robert).
**No** **%** deoarece acest fapt **nu** există în baza de cunoştințe

În exemplele prezentate până acum, argumentele faptelor şi întrebărilor au fost obiecte particulare, numite şi *constante sau atomi simbolici*. Predicatele Prolog, ca orice predicate în logica cu predicate de *ordinul I*, admit ca argumente şi obiecte generice numite *variabile*. În Prolog, prin convenție, numele argumentelor *variabile* începe cu *literă mare* iar numele *constantelor* simbolice începe cu *literă mică*. O *variabilă* poate fi *instanțiată* (legată) dacă există un obiect asociat acestei variabile, sau *neinstanțiată* (liberă) dacă nu se ştie încă ce obiect va desemna variabila.

La fixarea unui scop Prolog care conține variabile, acestea sunt neinstanțiate, iar sistemul încearcă satisfacerea acestui scop căutând printre faptele din baza de cunoştințe un fapt care se poate identifica cu scopul, printr-o instanțiere adecvată a variabilelor din scopul dat. Este vorba de fapt

de un proces de unificare a predicatului scop cu unul din predicatele fapte existente în baza de cunoştințe.

La încercarea de satisfacere a scopului, căutarea se face întotdeauna pornind de la începutul bazei de cunoştințe. Dacă se întâlneşte un fapt cu un simbol predicativ identic cu cel al scopului, variabilele din scop se instanțiază conform algoritmului de unificare şi valorile variabilelor astfel obținute sunt afişate ca răspuns la satisfacerea acestui scop.

**Exemple**:

**?-** caine(CineEste).

**CineEste = bobi**

**?-** deplaseaza(Ce, DeUnde, Unde).

**Ce = cub, DeUnde = camera**1**, Unde = camera**2

**?-** deplaseaza(Ce, Aici, Aici).

**no**

În cazul în care există mai multe fapte în baza de cunoştințe care unifică cu întrebarea pusă, deci există mai multe răspunsuri la întrebare, corespunzând mai multor soluții ale scopului fixat, limbajul Prolog procedează astfel. Prima soluție este dată de prima unificare şi există atâtea soluții câte unificări diferite există. La realizarea primei unificări se marchează faptul care a unificat şi care reprezintă prima soluție. La obținerea următoarei soluții, căutarea este reluată de la marcaj în jos în baza de cunoştințe. Obținerea primei soluții este de obicei numită *satisfacerea scopului* iar obținerea altor soluții, *resatisfacerea scopului*. La satisfacera unui scop căutarea se face întotdeauna de la începutul bazei de cunoştințe. La resatisfacerea unui scop, căutarea se face începând de la marcajul stabilit de satisfacerea anterioară a acelui scop.

Sistemul Prolog, fiind un sistem interactiv, permite utilizatorului obținerea fie a primului răspuns, fie a tuturor răspunsurilor. În cazul în care, după afişarea tuturor răspunsurilor, un scop nu mai poate fi resatisfăcut, sistemul răspunde **no**.

**Exemple**:

**?-** place(ion, X).
**X = ioana**;  % cerem o nouă soluție tastând caracterul ";" şi Enter
**X = ana**;
**no**
**?-** place(Cine, PeCine).
**Cine = mihai, PeCine = maria**;
**Cine = mihai, PeCine = ana**;
**no**

Aşadar, există două soluții pentru scopul **place(ion, X)** şi tot două soluții pentru scopul **place(Cine, PeCine)**, considerând tot baza de cunoştințe prezentată în secțiunea a).

## c) Reguli

O regulă Prolog exprimă un fapt care depinde de alte fapte şi este de forma:

$$\mathbf{S :- S_1, S_2, \ldots S_n.}$$

cu semnificația prezentată la începutul acestui laborator. Fiecare $\mathbf{S_i}$, $\mathbf{i} = 1,\mathbf{n}$ şi **S** au forma faptelor Prolog, deci sunt predicate, cu argumente constante, variabile sau structuri. Faptul **S** care defineşte regula, se numeşte *antet* de regulă, iar $\mathbf{S_1}$, $\mathbf{S_2}$, …$\mathbf{S_n}$ formează *corpul* regulii şi reprezintă conjuncția de scopuri care trebuie satisfăcute pentru ca antetul regulii să fie satisfăcut.

Fie următoarea bază de cunoştințe Prolog:

- Ana este frumoasă. *frumoasa(ana).* % Ipoteza 1
- Ion este bun. *bun(ion).* % Ipoteza 2
- Ion o cunoaşte pe Maria. *cunoaste(ion, maria).* % Ipoteza 3
- Ion o cunoaşte pe Ana. *cunoaste(ion, ana).* % Ipoteza 4
- Mihai o place pe Maria. *place(mihai, maria).* % Ipoteza 5
- Dacă două persoane X şi Y se cunosc şi persoana X este bună şi persoana Y este frumoasă, atunci cele două persoane, X şi Y, se plac.

```
place(X, Y) :-                    % Ipoteza 6
        bun(X),
        cunoaste(X, Y),
        frumoasa(Y).
```

Enunțul de la *iptoteza 6* defineşte o *regulă* Prolog. Relația **place(Cine, PeCine)**, fiind definită atât printr-un fapt (Ipoteza 5) cât şi printr-o regulă (Ipoteza 6). În condițiile existenței regulilor în baza de cunoştințe Prolog, satisfacerea unui scop se face printr-un procedeu similar cu cel prezentat în *secțiunea b)*, dar unificarea scopului se încearcă atât cu fapte din baza de cunoştințe, cât şi cu antetul regulilor din bază. La unificarea unui scop cu antetul unei reguli, pentru a putea satisface acest scop trebuie satisfăcută regula. Aceasta revine la a satisface toate faptele din corpul regulii, deci conjuncția de scopuri. Scopurile din corpul regulii devin subscopuri a căror satisfacere se va încerca printr-un mecanism similar cu cel al satisfacerii scopului inițial.

Pentru baza de cunoştințe descrisă mai sus, satisfacerea scopului

**?-** place(ion, ana).

se va astfel: scopul unifică cu antetul regulii (6) şi duce la instanțierea variabilelor din regula (6): **X = ion** şi **Y = ana**. Pentru ca acest scop să fie îndeplinit, trebuie îndeplinită regula, deci fiecare subscop din corpul acesteia. Aceasta revine la îndeplinirea scopurilor **bun(ion)**, care reuşeşte prin unificare cu faptul (2), **cunoaste(ion, ana)**, care reuşeşte prin unificare cu faptul (4), şi a scopului **frumoasa(ana)**, care reuşeşte prin unificare cu faptul (1). În consecință, regula a fost îndeplinită, deci şi întrebarea inițială este adevarată, iar sistemul răspunde **yes**.

Mai departe să vedem ce se întîmplă dacă se pune întrebarea:

**?-** place(X, Y).

Prima soluție a acestui scop este dată de unificarea cu faptul (5), iar răspunsul este:

**X = mihai, Y = maria**

Sistemul Prolog va pune un marcaj în dreptul faptului (5) care a satisfăcut scopul. Următoarea soluție a scopului **place(X, Y)** se obține începând căutarea de la acest marcaj în continuare în baza de cunoştințe. Scopul unifică cu antetul regulii (6) şi se vor fixa trei noi subscopuri de îndeplinit, **bun(X)**, **cunoaste(X, Y)** şi **frumoasa(Y)**. Scopul **bun(X)** este satisfăcut de faptul (2) şi variabila **X** este instanțiată cu valoarea **ion**, *X=ion*. Se încearcă acum satisfacerea scopului **cunoaste(ion, Y)**, care este satisfăcut de faptul (3) şi determină instanțierea **Y = maria**. Se introduce în baza de cunoştințe un marcaj asociat scopului **cunoaste(ion, Y)**, care a fost satisfăcut de faptul (3). Se încearcă apoi satisfacerea scopului **frumoasa(maria)**. Acesta eşuează. În acest moment sistemul intră într-un proces de *backtracking* în care se încearcă resatisfacerea scopului anterior satisfăcut, **cunoaste(ion, Y)**, în speranța că o noua soluție a acestui scop va putea satisface şi scopul curent care a eşuat, **frumoasa(Y)**. Resatisfacerea scopului **cunoaste(ion, Y)** se face pornind căutarea de la marcajul asociat scopului în jos, deci de la faptul (3) în jos. O nouă soluție (resatisfacere) a scopului **cunoaste(ion, Y)** este dată de faptul (4) care determină instanțierea **Y = ana**. În acest moment se încearcă satisfacerea scopului **frumoasa(ana)**. Cum este vorba de un nou scop, căutarea se face de la începutul bazei de cunoştințe şi scopul **frumoasa(ana)** este satisfăcut de faptul (1). În consecință a doua soluție a scopului **place(X, Y)** este obținută şi sistemul răspunde:

***X = ion, Y = ana,*** urmând un mecanism de *backtracking*, descris intuitiv în figura 1 prin prezentarea arborilor de deducție construiți de sistemul Prolog.

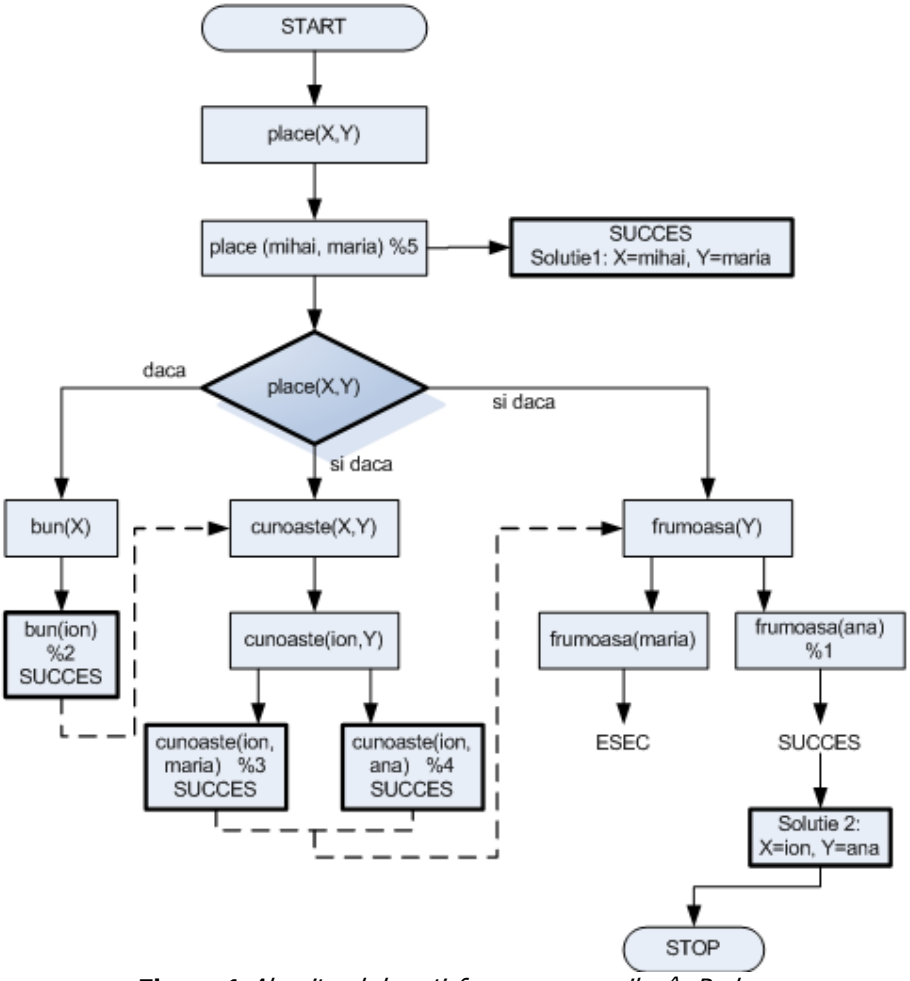


**Figura 1** *Algoritmul de satisfacere a scopurilor în Prolog*

La încercarea de resatisfacere a scopului **place(X, Y)**, printr-un mecanism similar, se observă că nu mai există alte solutii. În concluzie, fiind dată baza de cunoştinţe Prolog anterioară, răspunsul sistemului Prolog la întrebarea **place(X, Y)** este:

**?-** place(X, Y).

**X = mihai, Y = maria**;

**X = ion, Y = ana**;

**no**

**Observații**:

- La satisfacerea unei conjuncții de scopuri în Prolog, se încearcă satisfacerea fiecărui scop pe rând, de la stânga la dreapta. Prima satisfacere a unui scop determină plasarea unui marcaj în baza de cunoştințe în dreptul faptului sau regulii care a determinat satisfacerea scopului.
- Dacă un scop nu poate fi satisfăcut (eşuează), sistemul Prolog se întoarce şi încearcă resatisfacerea scopului din stânga, pornind căutarea în baza de cunoştințe de la marcaj în jos. Înainte de resatisfacerea unui scop se elimină toate instanțierile de variabile determinate de ultima satisfacere a acestuia. Dacă cel mai din stânga scop din conjuncția de scopuri nu poate fi satisfăcut, întreaga conjuncție de scopuri eşuează.
- Această comportare a sistemului Prolog în care se încearcă în mod repetat satisfacerea şi resatisfacerea scopurilor din conjuncțiile de scopuri se numeşte *backtracking*.

**Aplicatii in Prolog:**

1. În cele ce urmează prezentăm un program Prolog care descrie relațiile existente într-o familie. Spre exemplu, alegem trei predicate de bază: *bărbat, femeie, parinte* şi în funcție de acestea vom descrie relațiile din familia prezentată în figura 2.

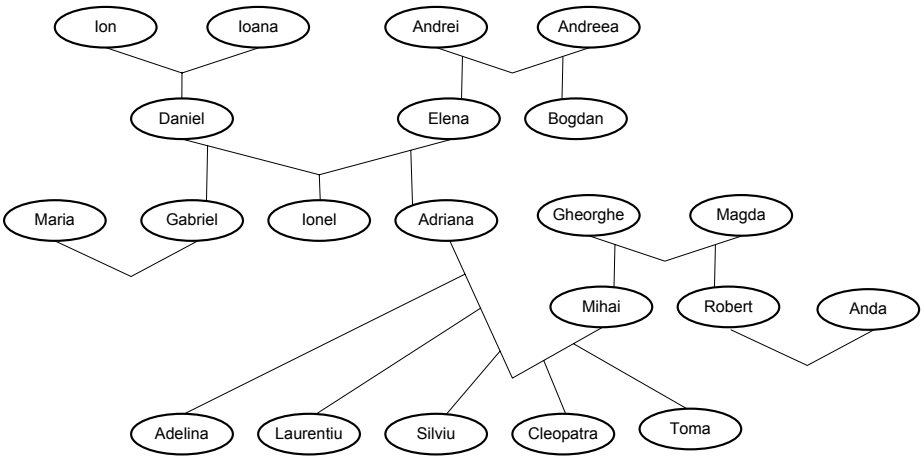


**Figura 2** *Exemplu de familie (arbore genealogic)*

Codul sursă (SWI-Prolog) al aplicației este următorul:

```
barbat(ion).
barbat(andrei).
barbat(daniel).
barbat(bogdan).
barbat(gheorghe).
barbat(gabriel).
barbat(ionel).
barbat(mihai).
barbat(robert).
barbat(laurentiu).
barbat(silviu).
barbat(toma).

femeie(ioana).
femeie(andreea).
femeie(elena).
femeie(magda).
femeie(maria).
femeie(adriana).
femeie(anda).
femeie(adelina).
femeie(cleopatra).
```

Mai departe, utilizând predicatul *cuplu_casatorit,* vom identifica membrii arborelui genealogic care sunt căsătoriți.

```
cuplu_casatorit(ion,ioana).
cuplu_casatorit(andrei,andreea).
cuplu_casatorit(daniel,elena).
cuplu_casatorit(gabriel,maria).
cuplu_casatorit(gheorghe,magda).
cuplu_casatorit(mihai,adriana).
cuplu_casatorit(robert,anda).
```

Observație: faptele construite cu același numede relație (cum ar fi: *cuplu_casatorit*) se află unul lângă altul în fișierul sursă. Aceste fapte formează definiția relației respective.

Următorul pas constă în identificarea relațiilor dintre oameni. Singura relație necesară este reprezentarea părinților pentru fiecare individ. De aici, vom putea deduce alte relații, cum ar fi bunici, nepoți,etc.

```
tata(ion,daniel).
tata(andrei,elena).
tata(andrei,bogdan).
tata(daniel,gabriel).
tata(daniel,ionel).
tata(daniel,adriana).
tata(gheorghe,mihai).
tata(gheorghe,robert).
tata(mihai,adelina).
tata(mihai,laurentiu).
tata(mihai,silviu).
tata(mihai,cleopatra).
tata(mihai,toma).

mama(ioana,daniel).
mama(andreea,elena).
mama(andreea,bogdan).
mama(elena,gabriel).
mama(elena,ionel).
mama(elena,adriana).
mama(magda,mihai).
mama(magda,robert).
mama(adriana,adelina).
mama(adriana,laurentiu).
mama(adriana,silviu).
mama(adriana,cleopatra).
mama(adriana,toma).
```

Introduceți acest program folosind editorul SWI Prolog (prezentat în figura 3) şi salvați-l într-un fişier cu extensia *pl*, de exemplu *familie.pl*. Compilați aplicația (tasta F9). Realizați această *compilare* ori de câte ori modificați ceva în program.

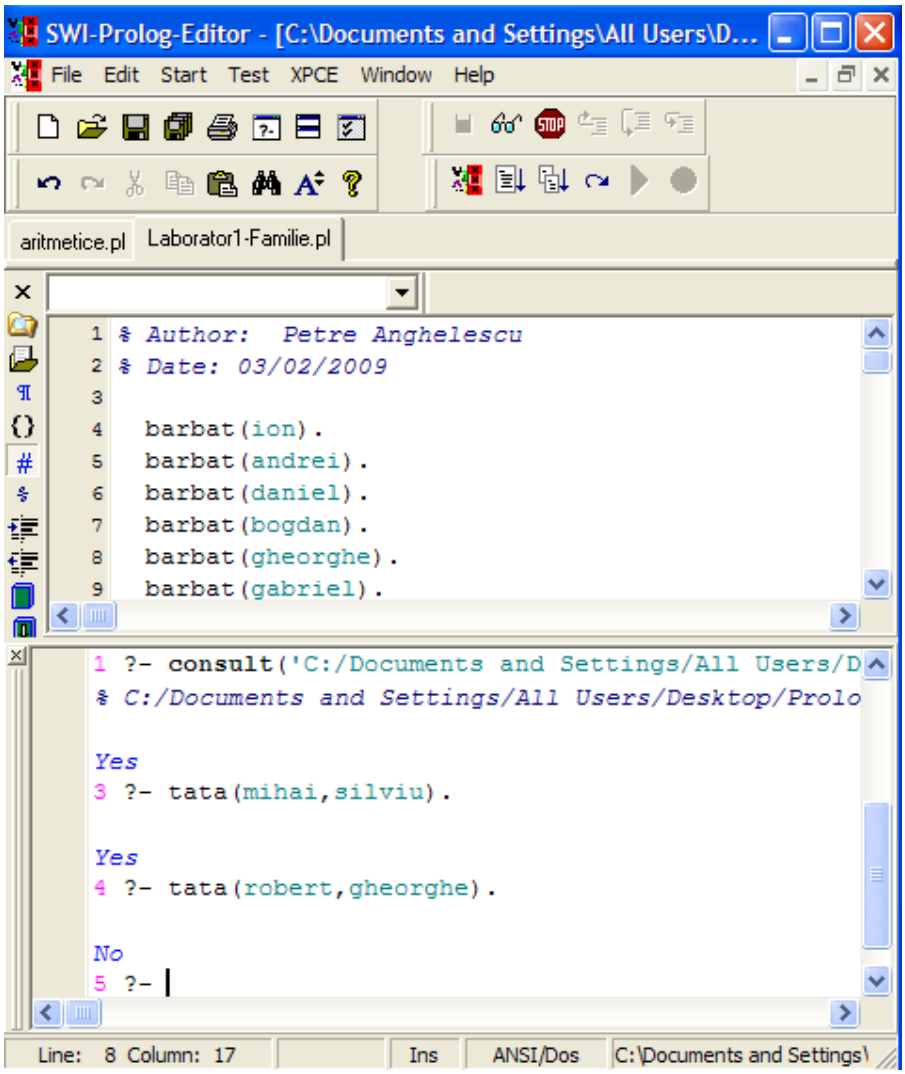


**Figura 3** *Editorul SWI Prolog*

În acest moment putem formula întrebări ce forma: este Mihai tatăl lui Silviu? sau este Robert tatăl lui Gheorghe? (figura 3).

`?- tata(mihai,silviu).` Sistemul va consulta baza de fapte de care dispune şi va răspunde cu YES.

La întrebarea: `?- tata(robert,gheorghe).`, sistemul va răspunde cu NO, deoarece în baza de cunoştinţe nu există nici un fapt care să corespundă acestei întrebări.

Dacă dorim să aflăm care sunt copii lui Mihai, trebuie să introducem noțiunea de *variabilă*. O variabilă poate fi văzută ca o locație de memorie care primește în timpul procesului de raționament o valoare. Odată asignată, valoarea unei variabile nu mai poate fi schimbată. În Prolog, numele unei variabile începe întotdeauna cu literă mare.

Variabilele pot să apară şi în cadrul faptelor dintr-un program pentru a desemna adevăruri general valabile. Spre exemplu, dacă adăugăm la exemplul nostru faptul *place(Persona, handbal)*, aceasta înseamnă că tuturor persoanelor le place să joace handbal. Cu alte cuvinte, oricare ar fi *Persoana*, are loc relația *place(Persona, handbal)*.

`?- tata(mihai,Copil).` *Copil* reprezintă o variabilă. Sistemul va căuta în baza de cunoştințe un fapt care să corespundă interogării şi se va opri la *tata(mihai, adelina)*.

Pentru a obține toți copiii lui *mihai*, vom construi o interogare mai complicată:

`?- tata(mihai,Copil), writeln(Copil),fail.` În această interogare, *virgula* desemnează operatorul ”SI logic”, *writeln* afişează valoarea din paranteză şi trece la linia următoare, iar *fail* este o relație (predicat) predefinită care eşuează întotdeauna determinând căutarea de noi soluții.
Răspunsul la interogarea de mai sus va fi:

*adelina, laurentiu, silviu, cleopatra, toma,* `No`.

Până acum am definit relațiile ca mulțimi de fapte. O relație poate fi definită şi în funcție de alte relații existente în program. Pentru aceasta vom folosi construcții sintactice mai complicate numite *reguli*.

În general, o regulă are următoarea sintaxă:

*nume_rel(arg1, ..., argN) :- nume_rel_1(...), ..., nume_rel_M(...).*

Exemplu: putem defini predicatul *sot* astfel – Barbat este sot dacă este barbat şi este casatorit cu o femeie (Femeie). Această definiție va fi codificată prin următoarea regulă care va fi adăugată la programul prezentat mai sus. Operatorul *:-* are semnificația *dacă*. Partea sin stânga operatorului *:-* este numită *capul regulii (head)*, iar cea din dreapta *corpul regulii (body)*.

Observații:

- Un fapt poate fi văzut ca o regulă fără corp (fără condiție).
- Regulile şi faptele care definesc aceeaşi relație (au acelaşi nume de relație în cap) trebuie grupate în sursa programului.
- În Prolog, relațiile sunt numite de obicei predicate.

```
sot(Barbat, Femeie):-
        barbat(Barbat), cuplu_casatorit(Barbat, Femeie).
```

Similar,

```
sotie(Femeie, Barbat):-
        femeie(Femeie), cuplu_casatorit(Barbat, Femeie).
```

După adăugarea celor două reguli, recompilați programul și introduceți interogarea:

```
?- sot(Barbat,Femeie), writeln(Barbat),fail.
```

Răspunsul va fi:

*ion, andrei, gheorghe, gabriel, mihai, robert.*

Pentru a putea urmări modul în care este rezolvată interogarea anterioară se introduce comanda `?- trace.` și apoi se repetă interogarea precedentă. Pe ecran vor apare diferite subscopuri care trebuie rezolvate de programul Prolog pentru a obține răspunsul al interogare. Pentru a trece de la un subscop la altul se apasă tasta ENTER.

Următoarea regulă definește dacă o persoană este părintele unui copil. În această regulă, "*punctul și virgula*" (;) desemnează operatorul logic SAU.

```
parinte(Parinte,Copil):-
      tata(Parinte,Copil); mama(Parinte,Copil).
```

Mai departe putem utiliza regula *parinte* pentru a crea o nouă regulă ce testează dacă o persoană este copilul unei persoane date.

```
copil(Copil,Parinte):-
      parinte(Parinte,Copil).
```

Până acum am definit reguli focalizate pe relații directe dintre persoanele ce alcătuiesc baza de cunoștințe. În continuare vom defini două reguli care prezintă relații indirecte între persoanele din baza de cunoștințe. Vom defini regula *bunic* și regula *nepot* utilizând relațiile prezentate în figura 4.

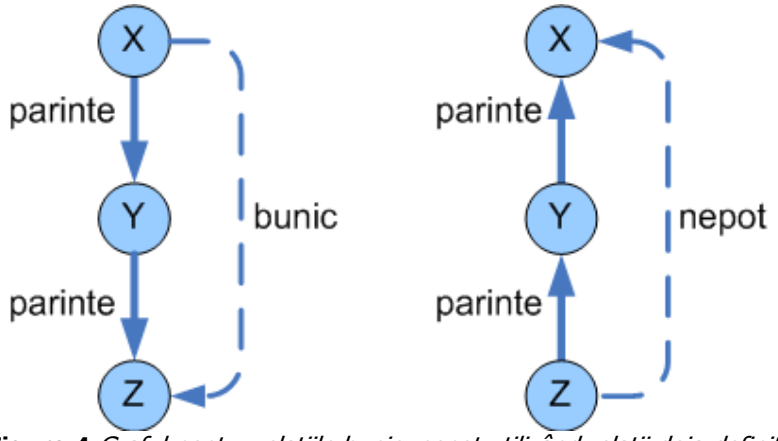


**Figura 4** *Graful pentru relațiile bunic, nepot utilizând relații deja definite*

Pentru a afla bunicul Z unei persoane *X* procedăm astfel:

- Cine este părintele lui X? (presupunem că este Y).
- Cine este părintele lui Y? (presupunem că este Z).
- Deci, Z va fi bunicul lui X.

Interogarea în Prolog va fi:

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

Similar se construiește interogarea pentru *nepot*.

```
nepot(X, Z):-
        bunic(Z,X).
```

În figura 5 prezentăm relația *sora*.

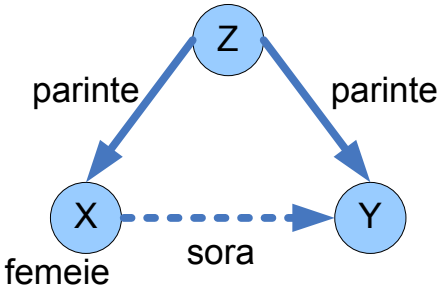


**Figura 5** *Definirea relației sora*

Algoritmul este următorul:

Pentru orice X şi Y,

X este sora lui Y dacă

(1) X şi Y au acelaşi părinte, şi

(2) X este femeie.

O posibilă implementare în Prolog a algoritmului prezentat mai sus este următoarea:

```
sora(X,Y):-
        parinte(Z,X), parinte(Z,Y), femeie(X), X\=Y.
```

Menționăm faptul că modalitatea în care X şi Y au acelaşi părinte a fost prezentată anterior. În Prolog, operatorul `\=` are sensul de *diferit*.

2. Simulare funcționare circuite hardware digitale. Presupunem următoarea schemă digitală electronică:

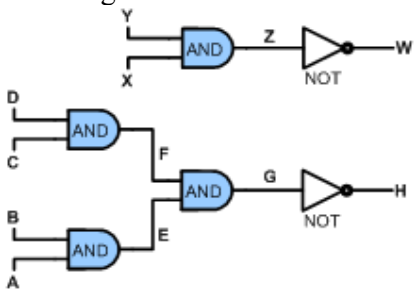


**Figura 6** *Circuite electronice digitale*

Circuitele fundamentale (porțile logice) se descriu uzual prin tabela de adevăr. De exemplu, pentru circuitele din figura 6, se folosesc pentru modelare funcții logice de tip AND, respectiv NOT, corespunzătoare porților logice elementare care îl compun.

Funcțiile globale, corespunzătoare celor două circuite din figura 6, sunt următoarele:

| X | Y | Z | W |
|---|---|---|---|
| 0 | 0 | 0 | **1** |
| 0 | 1 | 0 | **1** |
| 1 | 0 | 0 | **1** |
| 1 | 1 | 1 | **0** |

| A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | **1** |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | **1** |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | **1** |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 | **1** |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | **1** |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | **1** |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | **1** |
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | **1** |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | **1** |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | **1** |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 | **1** |
| 1 | 0 | 1 | 1 | 0 | 1 | 0 | **1** |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | **1** |
| 1 | 1 | 0 | 1 | 1 | 0 | 0 | **1** |
| 1 | 1 | 1 | 0 | 1 | 0 | 0 | **1** |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | **0** |

Baza de cunoştinţe Prolog va trebui să cuprindă predicate care să modeleze blocurile operaționale logice ale circuitelor cu una, respectiv două intrări şi o ieşire, având argumente de tip binar.

Codul sursă (SWI-Prolog) al aplicației este următorul:

```
siLogic2(0,0,0).
siLogic2(0,1,0).
siLogic2(1,0,0).
siLogic2(1,1,1).
inv(0,1).
inv(1,0).
```

```
circuit1(X,Y,Iesire1):-
                        siLogic2(X,Y,Z),
                        inv(Z,Iesire1).
circuit2(A,B,C,D,Iesire2):-
                              siLogic2(A,B,E),
                              siLogic2(C,D,F),
                              siLogic2(E,F,G),
                              inv(G,Iesire2).
```

Baza de cunoştinţe, în acest caz, constituie clauzele (faptele) specifice sistemului , formate din stările (cunoscute) pe care acesta le poate avea.

3. În acest program ne propunem să determinăm dacă două persoane de sunt parteneri în afaceri. Vom defini regulile de aşa natură încât să eliminăm situația în care o persoană este partener cu el însuşi. În acest scop vom utiliza operatorul `\=`. Prezentăm mai jos premisele (ipotezele) de la care se pleacă:

- Robert este corect.
- Andrei nu este partener cu Robert.
- Dacă două persoane X şi Y sunt corecte, atunci X este partener cu Y.
- Dacă Andrei nu este corect, atunci Ion este corect.
- Dacă o persoană X este partener cu Y, atunci şi Y este partener cu X.

Codul sursă (SWI-Prolog) al aplicației este următorul:

```
% Premisa 1
corect(robert).

% Premisa 2
not_partener(andrei, robert).

% Premisa 3
partener(X, Y) :- corect(X), corect(Y), X \= Y.

% Premisa 4, se foloseşte şi ipoteza 3, inversată conform
principiului reducerii la absurd: formula p→q este
echivalentă cu !q→!p.
corect(ion) :- not_corect(andrei).
% din premisa 3 avem:
not_corect(X) :- not_ambii_corecti(X, Y), corect(Y).
not_ambii_corecti(X, Y) :- not_partener(X, Y).

% Premisa 5, este prezentată în premisa 3 (definiţia pentru
partener) care include şi simetria.
```

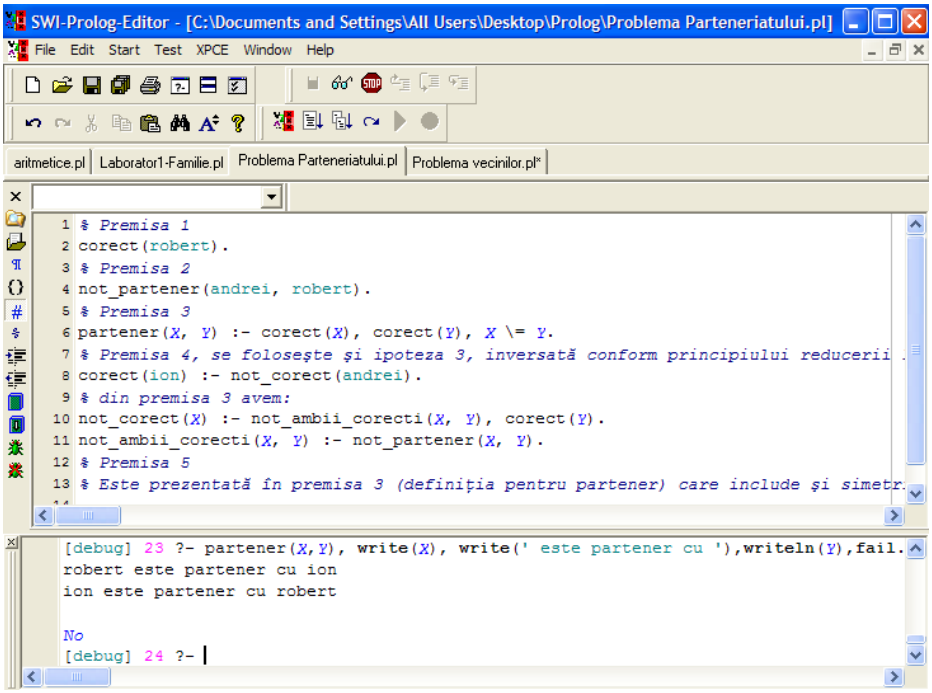


**Figura 7** *Problema parteneriatului*

## Desfăşurarea lucrării:

1. Se va citi breviarul teoretic. Se atrage atenția asupra faptului că toate cunoştințele din această lucrare vor fi necesare şi în derularea celorlalte lucrări.
2. Se vor studia problemele rezolvate (problemele prezentate pe parcursul acestui laborator precum şi cele oferite ca exemple în kitul SWI-Prolog), încercând găsirea altor posibilități de soluționare a acestora. Utilizați şi alte scopuri (interogări) pentru a testa definițiile predicatelor introduse.
3. Să se scrie ipotezele de mai jos sub formă de clauze Prolog şi să se demonstreze concluziile prin formularea de interogări.

   Ipoteze:
   - Ion are mere.
   - Ana are mere şi iepuri.
   - Dan are prune.

   Cerințe:
   - Cine are mere?
   - Ce are Ana?

- Cine nu are fructe (not fruct(X))?
- Cine are la fel ca Ion?
- Are cineva mere ( _ )?
- Are Dan ceva?
- Ce are Ana şi nu este fruct?
- Cine are ce?

4. Să se descrie în Prolog circuitele electronice de bază utilizate pentru modelarea operatorilor logici: INVERSOR(NOT), SI (AND), SI-NU (NAND), SAU(OR), SAU-NU(NOR), SAU EXCLUSIV(XOR), SAU EXCLUSIV NEGAT(XNOR).
5. Să se completeze aplicația 1 cu enunțurile de mai jos şi să se urmărească execuția corectă a acestora.
   - Definiți relațiile *fiu* şi *fiica*.
   - Modificați regula *sora* pentru a obține regula pentru relația *frate*.
   - Definiți relațiile *soacra*, *cumnat*, *ginere*.
   - Să se afişeze numele familiilor fără copii.
   - Construiți o interogare prin intermediul căreia să afişați toate mamele.
   - Construiți o interogare prin intermediul căreia să afişați părinții lui Ionel.
   - Să se afişeze toți nepoții (fete) care provin de la fetele lui Andrei.
   - Definiți relația *nepot* utilizând relația *parinte*.
   - Definiți relația *matusa(X,Y)* utilizând relațiile *parinte* şi *sora*.
   - Definiți relația *fericit*, ştiind că oricine are minim doi copii este fericit.
   - Pentru orice X, dacă X are un copil ce are o soră, atunci X are doi copii (introduceți relația *aredoicopii*).