

## Laborator 2

### Tipuri de obiecte Visual Prolog

Programele Visual Prolog conțin *obiecte* și *relații* dintre aceste obiecte. Obiectele sunt entități distincte ale lumii reale (obiecte reale – fizice) sau abstracte (obiecte abstracte). În Prolog, obiectele sunt reprezentate prin nume simbolice. Regulile sintactice aplicate la formarea numelor simbolice pentru obiecte permit definirea mai multor tipuri de obiecte:

#### 1. tipul elementar:

##### a. standard

- i. caracter - char: un caracter încadrat între două caractere apostrof;
- ii. întreg - integer: întreg cuprins în intervalul [-32768, 32767];
- iii. real - real: număr real, echivalent cu double din C și al cărui modul este în intervalul [1e-307, 1e308]
- iv. șir - string: secvență de caractere, ce este implementată ca un pointer la un tablou terminat prin zero, ca și în C. Pentru șiruri sunt permise două formate:
  - iv.1. secvență de litere, numere și liniuță de subliniere, în care primul caracter este minuscul (de ex., numar\_telefon);
  - iv.2. secvență de caractere încadrată între ghilimele, maxim 250 caractere (de exemplu, „șir de caractere”);
- v. simbol - symbol: analog ca la tipul string, cu diferența că primul caracter este literă mică (în reprezentarea internă string-ul și simbolul gen string nu sunt identice; pentru simbol se construiește un tabel, el este unic și regăsit la fiecare citare - practic se lucrează cu adresa lor, deci se constată o mare rapiditate de acces; în cazul string-urilor, acestea sunt tratate caracter cu caracter)
- vi. fișier - file: nume simbolic de fișier
- vii. număr de referință - ref: pentru termen dintr-o bază externă
- viii. selector de bază de date externă - db\_selector: nume simbolic prin care se face referință la o bază externă de cunoștințe; acest nume simbolic este extern programului
- ix. selector de arbore binar - bt\_selector: selectorul unui arbore este un număr natural, cu valoarea minimă 0, prin care se face referință în program la arborele binar; baza externă de cunoștințe are atașat un arbore binar sau mai mulți arbori, prin care se asigură căutarea în baza de cunoștințe
- x. domeniu - place: locul ocupat de un fișier
- xi. localizare

##### b. definit de utilizator

#### 2. complex

##### a. lista

- i. de tipuri elementare
- ii. de tipuri compuse

##### b. compus din:

- i. nume simbolic
- ii. diferite tipuri

În afara celor menționate anterior, mai trebuie amintite și următoarele tipuri de date, pentru care s-a notat în paranteză intervalul: short (intervalul +32768, 32767), ushort (valori în intervalul 0, 65535), long (intervalul -2147483648, 2147483647), ulong (0, 4294967295), unsigned (valori în intervalul 0, 4294967295), byte (0, 255), sbyte (-128, 127), word (0, 65535), dword (0, 4294967295).

O valoare întreagă poate fi precedată de 0x sau 0o, indicând sintaxa hexazecimală, respectiv octală.

Trebuie specificat faptul că în declarațiile domain, cuvintele cheie **signed** și **unsigned** pot fi utilizate împreună cu **byte**, **word**, și **dword**.

Un obiect sau o variabilă care este de un tip, nu poate apare în locuri (pe poziții) unde se cere alt tip. Excepție fac câteva tipuri standard pentru care există conversie automată. Conversiile automate admise sunt:

- între string și simbol;
- între integer, char și real (char e reprezentat ca un cod ASCII - deci numeric; analog, un integer poate fi considerat un cod ASCII; orice întreg e un real).

Niciodată tipurile utilizator nu pot fi echivalate unul cu altul.

Din cele prezentate mai sus, rezultă o primă utilitate a declarării de tipuri, aceea de a declara tipuri specifice, care pe de o parte fac ca programul să fie mai lizibil, pe de altă parte definesc clase peste datele utilizate. O altă utilitate este aceea de a defini - cum se va vedea mai târziu - tipuri complexe.

Se reamintește faptul că în Prolog numele obiectelor încep cu literă mică, putând fi urmată de cifre, litere sau semnul "\_".

## Variabilele

Variabilele pot avea un conținut aleator în timpul prelucrării. Numele de variabile este constituit dintr-o succesiune de caractere (litere, cifre, caracterul "\_"), în care primul caracter este o literă mare sau "\_". De asemenea, se întâlnește și variabila reprezentată prin "\_", denumită *variabilă anonimă* care, practic, specifică doar prezența unei variabile, fără ca aceasta să fie utilizată. Utilizarea unei variabile anonime semnifică faptul ca nu interesează valoarea la care se va instanția acea variabila.

Variabilele prezente în structura clauzelor au efect local în cadrul respectivei clauze, fiecărei variabile legându-i-se o valoare, pentru ca ieșirea din respectiva clauză să determine dezlegarea ei de valoare. Două clauze pot folosi același nume pentru o variabilă, neexistând nici un fel de legătură între ele. Inițial, o variabilă nu are valoare (este nelegată), în cursul execuției programului, variabila putând lua valori.

Prolog se bazează pe "clauzele Horn" care formează un subset simplificat al unui sistem formal, numit "logica predicatelor".

Logica predicatelor a fost dezvoltată pentru o mai ușoară exprimare în forma scrisă a ideilor bazate pe logica.

În principiu, un enunț exprimat printr-o propoziție în limbaj natural poate fi transpus folosind sintaxa din logica predicatelor, astfel:

- se elimina cuvântul mai puțin important dintr-o propoziție;
- se transforma propoziția, punând pe prima poziție "relația" și grupând obiectele după relație. Astfel, obiectele devin argumentele asupra cărora acționează relația respectivă.

*Exemplu:*

## Propoziții limbaj natural

*Bunicul lui Mircea este Aurel.  
Casa este înalta.  
Lui Marius îi place o masina  
daca masina este verde.*

## Sintaxa-logica predicatelor

bunic(mircea, aurel).  
înalt(casa).  
place(marius, Masina) if  
verde(Masina).

Un program Visual Prolog este o mulțime de clauze Visual Prolog. Așa după cum s-a precizat, acestea sunt de trei tipuri: fapte, reguli și scopuri. Se prezintă, pe scurt, fiecare dintre acestea.

**Faptele** sunt declarații de predicate în care toate argumentele sunt obiecte cunoscute (nu conțin variabile). Conceptual, faptele se referă fie la caracteristicile obiectelor, fie la relațiile dintre mai multe obiecte. Faptele sunt reprezentate prin următoarea structură sintactică:

*nume\_predicat(arg<sub>1</sub>, arg<sub>2</sub>, ..., arg<sub>n</sub>).*

unde,

1. *nume\_predicat* este identificatorul unui atribut, format dintr-o succesiune de caractere (litere, cifre, liniuța de subliniere "\_"), dar cu primul caracter fiind o literă mică;
2. *arg<sub>1</sub>, arg<sub>2</sub>, ..., arg<sub>n</sub>* - identifică argumentele predicatului care, din punct de vedere simbolic pot fi nume de obiecte sau de variabile. Se poate ca un predicat să nu aibă nici un argument, dar utilizarea acestor predicate este limitată.

Se face precizarea că faptele se termină cu punct. Se recomandă gruparea faptelor la început.

Exemple:

papagal(coco). - predicat de aritate 1.

deplaseaza(cub, camera1, camera2). - este predicat de aritate 3.

Deci, se poate spune că faptele sunt relații sau proprietăți pe care programatorul le cunoaște ca fiind adevărate.

**Regulile** sunt predicate definite condițional, a căror formă sintactică este:

*nume\_predicat(arg<sub>1</sub>, arg<sub>2</sub>, ..., arg<sub>n</sub>) if*

*nume\_predicat<sub>1</sub>(arg<sub>11</sub>, arg<sub>12</sub>, ..., arg<sub>1l</sub>)*

**[and**

*nume\_predicat<sub>i</sub>(arg<sub>i1</sub>, arg<sub>i2</sub>, ..., arg<sub>ik</sub>)]*

**[or**

*nume\_predicat(arg<sub>1</sub>, arg<sub>2</sub>, ..., arg<sub>n</sub>) if*

*nume\_predicat<sub>m</sub>(arg<sub>m1</sub>, arg<sub>m2</sub>, ..., arg<sub>mq</sub>)*

**[and**

*nume\_predicat<sub>p</sub>(arg<sub>p1</sub>, arg<sub>p2</sub>, ..., arg<sub>pk</sub>)]*

**or ...**

**]**

Argumentele pot fi variabile.

Se observă că un predicat este adevărat dacă sunt îndeplinite simultan una sau mai multe condiții, sau dacă se verifică alt(e) set(uri) de condiții. Se mai spune că predicatul este definit prin clauzele ce urmează după **if**. Deci, dacă sunt adevărate condițiile, se poate deduce că și capul regulii este adevărat. Capul regulii este concluzia trasă când toate condițiile sunt adevărate. În

capul regulii este doar un singur predicat. Rezultă că în Visual Prolog, regulile sunt prezente numai în forma regulilor inductive, în care părțile regulii - concluzia și premisele - sunt separate prin "dacă" (if). Un fapt poate fi privit și ca o regulă cu corpul vid.

Așa după cum se poate observa și din sintaxă, o regulă poate da mai multe variante de evaluare (legate prin **or**), care vor fi în mod obligatoriu grupate. În cadrul aceleiași variante condițiile ce trebuie îndeplinite simultan sunt legate prin **"and"**. În Visual Prolog, există următoarele notații echivalente pentru simbolurile **if**, **and**, **or**:

":- "    pentru **if**,  
" ,"    pentru **and**,  
"; "    pentru **or**.

Se face precizarea că regulile se termină cu punct, acesta fiind folosit ca element lexical de precizare a sfârșitului regulii.

Dacă în scrierea unei reguli, o variabilă apare o singură dată (semnificând faptul că valoarea ei nu este folosită), compilatorul de Visual Prolog afișează un mesaj de avertisment:

"... Unused variable"

caz în care se recomandă utilizarea unei variabile anonime, care se poate lega la orice valoare (dar valoarea ei nu se poate folosi).

Regulile sunt reguli de raționare (deducție), dar și de execuție în măsura în care sunt legate variabile și se fac anumite evaluări. Dacă regula este de execuție (oarecum procedurală), predicatul poate să nu aibă nici un argument.

Deci, se poate spune că regulile sunt relații de dependență, care permit ca Prolog-ul să obțină (prin inferență) o informație din alta, o regulă fiind adevărată dacă setul de condiții dat se dovedește a fi adevărat.

**Scop** sau goal reprezintă ceea ce se caută.

## Structura programelor Visual Prolog

Un program Prolog conține următoarele secțiuni:

- **domains**;
- **predicates**;
- **goal**;
- **clauses**;
- **constants**;
- **databases**.

Se face precizarea că nu toate secțiunile sunt obligatorii.

**Clauses** conține fapte și reguli (care au fost prezentate mai sus), sau mai precis în această secțiune se definesc predicatele. Argumentele unei clauze sunt în ordine bine definită, fiecare argument având un anumit tip. Secțiunea CLAUSES trebuie să înceapă cu cuvântul cheie CLAUSES. Așa cum s-a specificat, atât faptele, cât și regulile trebuie să se termine cu punct (.).

De exemplu, afirmația "Popescu lucrează la biroul aprovizionare" poate fi exprimată în Prolog astfel:

lucreaza('Popescu', biroul\_aprovizionare)

Modul de formulare al clauzei sau mai precis, ordinea argumentelor este în totalitate la latitudinea programatorului. Astfel, afirmația anterioară poate fi exprimată și astfel:

lucreaza(biroul\_aprovizionare, 'Popescu')

Oricare dintre aceste două variante este corectă pentru Visual Prolog, singurul criteriu de selecție fiind doar percepția utilizatorului. O formă odată folosită trebuie respectată cu consecvență în întregul program.

În clauza următoare, primul argument este o variabilă:  
lucreaza(Persoana, biroul\_aprovizionare)

**Predicates** este acea secțiune care conține fiecare predicat ce apare citat în clauze și precizează tipul argumentelor. Se spune că reprezintă descrierea "ideală" a clauzelor.

Deci **predicates** este un fel de catalog al tuturor predicatelor utilizate, care arată ce nume au aceste predicate, fiecare predicat ce argumente are: câte, în ce ordine, de ce tip (ca un fel de secțiune a declarațiilor de funcții în C). Fiecare predicat când e apelat în clauze, trebuie să respecte numele și argumentele unui predicat descris în **predicates**. Dacă nu respectă tipul și ordinea argumentelor, se obțin erori. Totuși, se admit ca două nume de predicat identice să aibă un număr diferit de argumente. În acest caz, se recomandă ca cele două declarații să fie una după alta, atât în **predicates**, cât și în **clauses**. Se spune că un predicat are *aritate* (numărul de argumente) *multiplă*. Visual Prolog va accepta aritatea multiplă, în interior va considera totuși că e vorba de predicate diferite, cu nume identic.

Numele predicatului este un șir de caractere care poate conține litere, cifre și semnul "\_", dar care începe obligatoriu cu o literă (preferabil, literă mică). Dacă un nume de predicat este compus din mai multe cuvinte, acestea se pot lega între ele folosind semnul "\_". Lungimea maximă a numelui de predicat este de 250 caractere.

În următorul exemplu, primul argument este o variabilă anonimă:  
lucreaza(\_, biroul\_aprovizionare)

Mai trebuie precizat faptul că predicatele standard nu se definesc în această secțiune, altfel, Visual Prolog avertizează printr-un mesaj de eroare:

"... This name is reserved for a standard predicate (on the selected platform)"

În concluzie, predicatele trebuie declarate în secțiunea de predicate, definite în secțiunea de clauze, iar la apel, în funcție de numărul și tipul argumentelor se va alege definiția corespunzătoare.

În secțiunea **domains** se poate realiza redenumirea unor domenii standard pentru a le da o semantică mai precisă, precum și declararea unor noi structuri de date. Domeniile de bază au fost prezentate la începutul acestei lucrări (char, integer, real, string, symbol).

Secțiunile **domains** și **predicates** pot fi declarate globale, astfel:

global domains  
global predicates

Odată scrise definițiile, ele pot fi incluse în unul sau mai multe programe cu:

include *nume\_fis*

O altă secțiune este **databases**, care reprezintă o secțiune specială pentru declararea unor fapte care se doresc a fi o parte a unei baze de date dinamice. Astfel, uneori este necesară actualizarea faptelor din program (modificare, ștergere, adăugare) în timpul rulării programului. Într-un astfel de caz, faptele constituie o bază de date dinamică sau internă. Visual Prolog pune la dispoziție mai multe predicate pentru gestiunea bazelor de date (assertz, asserta etc.). Mai trebuie menționat că și această secțiune poate fi declarată global.

Secțiunea **goal** trebuie să apară în program - în acest caz fiind vorba despre varianta compilator, la rulare programul execută acest goal, caută prima soluție și nu afișează variabilele interne de lucru (**goal** declarat). Când apare în program, în general, secțiunea **goal** este după secțiunile **predicates** și **clauses**. Un program Visual Prolog poate conține doar o singură secțiune goal. Secțiunea goal conține o clauză sau un șir de clauze conectate logic (cu **and** și/sau **or**): **goal** complex (compus din mai multe subgoal-uri legate logic). Acest **goal** poate fi un predicat cu sau fără argumente, iar argumentele, dacă apar, pot fi variabile (variabile anonime sau nu) sau obiecte. Acest **goal** se termină cu punct.

Dându-se fapte, reguli și unul sau mai multe goal-uri, pentru fiecare goal, Visual Prolog execută următoarele operații:

- îl caută printre fapte;
- dacă nu-l găsește ca fapt, caută pentru el regula de deducție și verifică condițiile regulii, considerând pe rând variantele (dacă există);
- fiecare condiție devine un subgoal.

Goal-ul este demonstrat dacă toate subgoal-urile sunt demonstrate.

Secțiunea **constants** permite declararea unor constante (constante simbolice). Secțiunea poate să apară oriunde în program, cu condiția să fie declarată constanta înainte de utilizarea ei. Constanta e vizibilă în toate clauzele, dar valoarea constantei nu se poate modifica în timpul execuției programului. Forma generală:

#### **constants**

*nume \_constantă=valoare*

În Visual Prolog sunt constante numele de obiecte (șiruri de caractere) și numerele (întregi sau reale). Relativ la numele constantei, se face precizarea că nu contează dacă ea este declarată cu litere mici sau mari, spre deosebire de toate celelalte date din Visual Prolog. Astfel pi, Pi sau PI vor referi aceeași constantă. Ca regulă generală se cere ca în secțiunea de constante, constantele să fie declarate cu literă mare, dar în secțiunea de clauze, acestea trebuie să fie scrise cu literă mică, pentru a se distinge de variabile. Valoare poate fi orice obiect, de orice tip definit.

#### **Example:**

constants

```
Titlu="SALARII LUNARE"  
Expr=l+7*(47-12)  
Pi=3.14  
G=9,8  
Computer=mcintosh
```

Se mai precizează faptul că pentru constante nu sunt permise definiții recursive, de exemplu  $x = 2 * x / 2$ .

Într-un program pot fi mai multe secțiuni de declarații pentru constante, dar constantele trebuie să fie declarate înainte de a fi utilizate. Constantele declarate au durata de viață începând din momentul declarației până la sfârșitul fișierului sursă, precum și în orice fișier inclus după declarații.

Alte secțiuni vor fi prezentate mai târziu.

De asemenea, se prezintă comentariul, folosit pentru o mai mare claritate a programului. Simbolul "%" introduce un comentariu care se termină la sfârșitul liniei. Un text considerat comentariu și care ocupă mai multe linii se va include între caracterele /\* și \*/.

## Exemplu:

```
/*
Sisteme expert
*/
```

Exemple de reguli:

În continuare sunt prezentate o serie de reguli, precum și implementarea lor în Visual Prolog:

1. "Dacă o persoană (X) are sex masculin și este adult, atunci X este un bărbat." - în acest caz premisa este o clauză compusă, formată din două clauze simple asociate prin conectorul "și".

barbat(X):- sex\_masculin(X), adult(X).

2. Noțiunea de părinte poate fi definită astfel:

parinte(X, Y):- tata(X, Y); mama(X, Y)

Deoarece Prolog permite și utilizarea variabilelor anonime, noțiunea de părinte poate fi formulată astfel:

parinte(X, \_):- tata(X, \_); mama(X, \_)

care are următoarea interpretare: "o persoană este părinte dacă este fie tatăl, fie mama cuiva".

3. "Dacă tatăl lui Z este Y, iar părintele lui X este Z, atunci bunicul lui X este Y" - variabila sau variabilele cuprinse în concluzie trebuie să se regăsească și în premise, dar premisele pot conține și variabile suplimentare, care nu se regăsesc în concluzie, așa cum se poate observa din implementare:

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

Deoarece Prolog permite și utilizarea variabilelor anonime, noțiunea de părinte poate fi formulată astfel:

bunic(X, Y):- tata(\_, Y), parinte(X, \_).

## Temă

1. Se va urmări programul masini.pro, în care s-a declarat o bază de date pentru o firmă care se ocupă cu vânzarea mașinilor.

```
/*
Visual Prolog
*/
```

### predicates

```
%marca, km, vechimea, culoarea
nondeterm masina(symbol,real,integer,symbol)
nondeterm camion(symbol,real,integer,symbol)
```

### clauses

```
masina(chrysler, 130000, 3, rosu).
masina(ford, 90000, 4, argintiu).
masina(datsun, 8000, 1, rosu).
camion(ford, 80000, 6, verde).
camion(datsun, 50000, 5, roz).
camion(toyota, 25000, 2, negru).
```

```
/*
Se va rula aplicatia si se vor specifica la goal, urmatoarele:
Pentru a determina masina cu 130000km si 3 ani vechime:
masina(X, 130000, 3, _)
Pentru a determina toate masinile rosii:
```



```
masina(X, _, _, rosu)
*/
```

2. Se va urmări programul bunici.pro:

```
/* Program cu parinti, mame si bunici */
predicates
    nondeterm masc(symbol)
    nondeterm fem(symbol)
    nondeterm parinte(symbol, symbol) % primul este parintele pentru al doilea
    nondeterm mama(symbol, symbol)
    nondeterm bunica(symbol, symbol)

clauses
    masc(vasile).
    masc(alex).
    fem(elena).
    fem(ioana).
    parinte(vasile, alex).
    parinte(elena, alex).
    parinte(alex, ioana).

    mama(Persoana, Copil) if
        parinte(Persoana, Copil) and
        fem(Persoana).
    bunica(Bunica, Nepot) if
        parinte(Persoana, Nepot) and
        mama(Bunica, Persoana).
```

Se face observația că o variantă a ultimei reguli este:

```
bunica(Bunica, Nepot) if
    parinte(Persoana, Nepot) and
    parinte(Bunica, Persoana) and
    fem(Bunica).
```

Se va nota răspunsul primit în cazul următoarelor scopuri :

```
parinte(elena, alex)
parinte(Parinte, alex)
mama(Parinte, alex)
bunica(Bunica, ioana)
parinte(Tata, Copil) and masc(Tata)
```

Adăugați noi fapte în program pentru a exprima următoarele:

Părinții lui Vasile sunt Gheorghe și Maria.

Părinții Ioanei sunt Alex și Sanda.

Cuvintul cheie **nondeterm** specifică faptul că la un moment dat pot exista mai multe instanțe ale unui fapt fact\_N.

Cuvintul cheie **determ** specifică faptul că la un moment dat poate exista o singură instanță a unui fapt fact\_N.

Cuvintul cheie **single** specifică faptul că întotdeauna va exista o singură instanță a unui fapt fact\_N.



### Directive de compilare

Dacă se dorește să se urmărească pas cu pas execuția unui program, atunci din meniul Project, se va alege opțiunea Debug (Ctrl + Shift + F9).

În continuare se consideră directiva **include**, care este folosită atunci când se dorește concatenarea mai multor fișiere spre a fi executate în Visual Prolog. Trebuie specificat faptul că procedurile din fișierele incluse trebuie să fie declarate o singură dată și trebuie să existe cel mult o secțiune goal.

3. Se consideră fișierul tata.pro:

```
domains
    pers=symbol
predicates
    nondeterm tata(pers)
    nondeterm tata(pers, pers)
clauses
    tata(X) if
        tata(X, _).
        tata(alex).
        tata(ion, vasile).
        tata(george, ana).

goal
    tata(X),
    write(X).

/*
Tema
1. Rulați programul tata.pro
*/
```

### Observații

1. S-a declarat un predicat cu arități diferite, cu următoarea semantică intuitivă:
  - tata (pers) - înseamnă că pers este tată (al cuiva);
  - tata(pers1, pers2) - înseamnă că pers1 este tatăl lui pers2 (s-a notat pers1 și pers2 pentru a se putea face distincția).
2. S-a redenumit domeniul standard symbol pentru a se sugera că argumentele vor fi persoane.
3. Scopul intern tata(X) se va evalua folosind prima clauză.