

## Laborator 1

### Limbajul Prolog

Începutul programării logice poate fi atribuit lui R. Kowalski și A. Colmerauer și se situează la începutul anilor '70. Kowalski a plecat de la o formulă logică de tipul:

$$S_1 \wedge S_2 \wedge \dots S_n \rightarrow S$$

care are, în logica cu predicate de ordinul întâi semnificația declarativă conform căreia  $S_1 \wedge S_2 \wedge \dots S_n$  implică  $S$ , adică dacă  $S_1$  și  $S_2 \dots$  și  $S_n$  sunt fiecare adevărate atunci și  $S$  este adevărat, și a propus o interpretare procedurală asociată. Conform acestei interpretări, formula de mai sus poate fi scrisă sub forma:

$$S \text{ dacă } S_1 \text{ și } S_2 \dots \text{ și } S_n$$

și poate fi executată ca o procedură a unui limbaj de programare recursiv, unde  $S$  este antetul procedurii și  $S_1, S_2, \dots S_n$  corpul acesteia. Deci, pe lângă interpretarea declarativă, logică, a unei astfel de formule, formula poate fi interpretată procedural astfel: pentru a executa  $S$  se execută  $S_1$  și  $S_2 \dots$  și  $S_n$ .

În aceeași perioadă, A. Colmerauer și colectivul lui de cercetare de la Universitatea din Marsilia au dezvoltat un limbaj de implementare a acestei abordări, pe care l-au denumit Prolog, abreviere de la "**P**rogrammation et **L**ogique". De atunci și până în prezent, limbajul Prolog s-a impus ca cel mai important limbaj de programare logică și s-au dezvoltat numeroase implementări, atât ale unor interpretoare, cât și ale unor compilatoare ale limbajului. Limbajul Prolog a fost dezvoltat ca un instrument pentru programarea și rezolvarea de probleme care implicau reprezentări simbolice de obiecte și relații între obiecte, fiind considerat un limbaj cu fundament matematic solid.

Limbajul Prolog este un limbaj declarativ susținut de o componentă procedurală. Spre deosebire de limbajele procedurale, cum ar fi C sau Pascal, în care rezolvarea problemei este specificată printr-o serie de pași de execuție sau acțiuni, într-un limbaj declarativ problema este specificată prin descrierea universului problemei și a relațiilor sau funcțiilor existente între obiecte din acest univers. Exemple de astfel de limbaje sunt cele funcționale, de exemplu Lisp, Scheme, ML, și cele logice, de exemplu Prolog. Soluția problemei nu mai reprezintă o execuție pas cu pas a unei secvențe de instrucțiuni. Modul de execuție a programului depinde în primul rând de definițiile relațiilor, de modul de inferență și de controlul explicit realizat interactiv de către programator.

Deși inițial a fost gândit pentru un set restrâns de probleme, Prolog a devenit cu timpul un limbaj de uz general, fiind o unealtă importantă în aplicațiile de inteligență artificială. Pentru multe probleme, un program Prolog are cam de 10 ori mai puține linii decât echivalentul lui în Pascal.

În 1983, cercetătorii din Japonia au publicat un plan ambițios de creare a unor calculatoare de generația a 5-a pentru care Prolog era limbajul de asamblare. Planul nu a reușit, dar acest proiect a marcat o dezvoltare deosebită a interpretoarelor și compilatoarelor de Prolog, precum și o creștere mare a numărului de programatori în acest limbaj.

Multe clase de probleme pot fi rezolvate în Prolog, existând anumite categorii care sunt rezolvabile mult mai ușor în Prolog decât în orice alt limbaj procedural. Astfel de probleme sunt în principal cele dedicate prelucrării simbolice sau care necesită un proces de căutare a soluției într-un spațiu posibil de transformări ale problemei. De asemenea, limbajul Prolog are aplicabilitate în gestiunea bazelor de date relaționale, inteligența artificială, logica matematică, demonstrarea teoremelor, sistemele expert, ingineria arhitecturală etc.

Exemple de medii de programare ce utilizează limbajul Prolog (care rulează pe microcalculatoare de tip IBM/PC, fie sub sistemul de operare DOS, fie sub sistemul Windows) sunt: Turbo Prolog (produs al firmei Borland), ARITY Prolog, Visual Prolog, Amzi Prolog, SWI-Prolog etc. Trebuie amintit și faptul că există și variante de sisteme Prolog sub Unix, de exemplu, GNU Prolog, SWI-Prolog sau sub X Windows.

În cadrul acestui laborator se va studia Visual Prolog, care este considerat succesorul Turbo Prolog-ului și PDC Prolog. Visual Prolog este un mediu de programare complet, bazat pe

limbajul de programare Prolog. Visual Prolog conține tot ce este necesar pentru implementarea aplicațiilor: mediu grafic de dezvoltare, compilator, link-editor și depanator. Visual Prolog include o bibliotecă ce oferă acces la API, punând la dispoziția programatorului instrumente pentru: interfața utilizator grafică, baze de date ODBC/OCI, precum și Internet (socket, ftp, http, cgi etc.).

Visual Prolog este recomandat pentru implementarea sistemelor expert, planificare, precum și altor probleme de inteligență artificială.

O caracteristică generală a programelor ce utilizează Prolog-ul este aceea că programatorul specifică fapte și relații despre obiecte, iar sistemul Prolog găsește (prin inferență) noi relații despre obiecte.

### Entitățile limbajului Prolog

Limbajul Prolog este un limbaj logic, descriptiv, care permite specificarea problemei de rezolvat în termenii unor fapte cunoscute despre obiectele universului problemei și a relațiilor existente între aceste obiecte. Execuția unui program Prolog constă în deducerea implicațiilor acestor fapte și 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 și relațiile existente între aceste obiecte;
- *reguli* despre obiecte și relațiile dintre ele, care permit deducerea (inferarea) de noi fapte pe baza celor cunoscute;
- *întrebări* (interogări), numite și *scopuri*, despre obiecte și relațiile dintre ele, la care programul răspunde pe baza faptelor și regulilor existente.

#### 1.1 Fapte

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

Interpretarea particulară a predicatului și a argumentelor acestuia depinde de programator. Ordinea argumentelor, odată fixată, este importantă și trebuie păstrată la orice altă utilizare a faptului, cu aceeași semnificație. Mulțimea faptelor unui program Prolog formează *baza de cunoștințe Prolog*. Se va vedea mai târziu că în baza de cunoștințe a unui program Prolog sunt incluse și regulile Prolog.

#### 1.2 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 cunoștințe Prolog. Scopurile sunt predicate pentru care se dorește aflarea valorii de adevăr în contextul faptelor existente în baza de cunoștințe. 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**. Se va vedea mai târziu că programul Prolog, în cazul unui răspuns afirmativ la o întrebare, poate furniza și alte informații din baza de cunoștințe.

#### 1.3 Variabile

În Prolog, argumentele faptelor și întrebărilor pot fi obiecte particulare, numite și constante sau atomi simbolici. Predicatele Prolog, ca orice predicate în logica cu predicate de ordinul I, admit ca argumente și obiecte generice numite *variabile*. În Prolog, prin convenție, numele argumentelor variabile începe cu literă mare (sau cu liniuța de subliniere), 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 știe încă ce obiect va desemna variabila.

Variabilele Prolog sunt locale, nu globale, astfel încât dacă două clauze conțin fiecare câte o variabilă numită X, atunci vor exista, de fapt, două variabile distincte X.

La fixarea unui scop Prolog care conține variabile, acestea sunt neinstantiate, iar sistemul încearcă satisfacerea acestui scop căutând printre faptele din baza de cunoștințe un fapt care 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 cunoștințe.

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

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

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

Nou pentru Visual Prolog: dacă o variabilă apare doar o dată într-o clauză și numele începe cu liniuță de subliniere, atunci compilatorul Visual Prolog consideră această variabilă ca fiind o variabilă anonimă.

#### Observație

Compilatorul Visual Prolog nu face distincție între literele mici și mari care apar în nume, cu excepția primei litere. Deci, următoarele două nume pot fi folosite pentru a accesa aceeași variabilă: `SourceCode` și `SOURCECODE`.

### 1.4 Reguli

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

$$S :- S_1, S_2, \dots, S_n.$$

cu semnificația prezentată la începutul acestui capitol. Fiecare  $S_i$ ,  $i = 1, n$  și  $S$  au forma faptelor Prolog, deci sunt predicate, cu argumente constante, variabile sau structuri. Așa după cum s-a precizat, faptul  $S$  care definește regula, se numește *antet* de regulă, iar  $S_1, S_2, \dots, S_n$  formează *corpul* regulii și reprezintă conjuncția de scopuri care trebuie satisfăcute pentru ca antetul regulii să fie satisfăcut.

În condițiile existenței regulilor în baza de cunoștințe Prolog, satisfacerea unui scop se face printr-un procedeu similar cu cel prezentat în secțiunea 1.2, dar unificarea scopului se încearcă atât cu fapte din baza de cunoștințe, cât și 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.

Ținând cont de cele prezentate până acum, se poate defini un program Prolog, conform următoarei definiții generale:

#### Definiție

Un program Prolog (pur) constă din:

- a. o mulțime de fapte, cu forma generală

P.

unde P este un literal pozitiv.

- b. o mulțime finită de clauze program (definite), cu forma generală:

$S: - S_1, S_2, \dots, S_k$

unde:

- $P, S_1, S_2, \dots, S_k$  sunt literali pozitivi,
- simbolul „-” reprezintă o implicație dreapta-stânga,
- semnul „,” înlocuiește conjuncția.

Astfel, se obține clauza:

$$S_1 \wedge S_2 \wedge \dots S_n \rightarrow S$$

sau, mai precis:  $\neg S_1 \vee \neg S_2 \vee \dots \vee \neg S_n \vee S$

Acestea sunt numite și clauze procedură, clauze suplimentare sau reguli, unde  $P$  este antetul (capul) procedurii, lista  $S_1, S_2, \dots, S_k$  este corpul procedurii, fiecare  $S_i$  fiind un apel de procedură.

c. o clauză scop (interogare) cu forma generală:

$$? - S_1, S_2, \dots, S_k$$

care reprezintă, de asemenea, o clauză:

$$S_1 \wedge S_2 \wedge \dots S_n \rightarrow 0$$

sau,  $\neg S_1 \vee \neg S_2 \vee \dots \vee \neg S_n$ . Pentru  $n = 0$ , se obține clauza vidă.

Mulțimea faptelor sau mulțimea clauzelor procedură pot coincide cu mulțimea vidă, dar nu ambele în același timp.

În concluzie, programul logic este rulat (activat) pentru precizarea clauzei scop. Se pornește cu clauza de bază și apoi literalii pozitivi sunt eliminați, producându-se noi clauze scop. Se mai spune că pentru a executa o procedură, trebuie să fie satisfăcut (îndeplinit) antetul procedurii, iar pentru a îndeplini un antet de procedură  $S$ , trebuie să se execute toate apelurile de procedură  $S_i$ , prezente în corpul acesteia. Astfel, un scop reprezintă și o secvență de apeluri de procedură care apar în program. Rezultatul dorit se obține după execuția cu succes a tuturor acestor apeluri de procedură (prin execuția anumitor clauze procedură).

## Concluzii

În concluzie, ținând cont de cele prezentate, se pot enumera, pe scurt, principalele caracteristici ale limbajului Visual Prolog:

- este un limbaj declarativ;
- folosește fapte și reguli;
- poate face deducții;
- execuția unui program Visual Prolog este controlată automat;
- are o sintaxă simplă;
- este un limbaj puternic;
- permite dezvoltarea programelor interactive;
- are interfață grafică plăcută;
- poate lucra în condiții de incertitudine;
- este eliminat cazul de căutare ce nu duce nicăieri (ex. bucle infinite), Prolog-ul raportând cazul în care nu există soluții, putând să ofere soluții parțiale.

## Instalarea Visual Prolog-ului

Visual Prolog-ul este o implementare a Prolog-ului pentru calculatoarele compatibile IBM PC. Mai trebuie precizat și faptul că este un compilator rapid. Trebuie specificat faptul că Visual Prolog permite lucrul sub Windows 3-x/95/98/NT/2000, OS/2 și oferă suport în mod text pentru DOS, Linux și SCO UNIX.

## Prezentarea mediului de programare Visual Prolog

Pentru lansare, se va activa meniul Start, Programs, și se va selecta Visual Prolog 5.2 Personal Edition, Vip32. La lansare pe ecran este afișată o fereastră al cărei meniu principal (meniul bară) constă din submeniurile:

File  
Edit  
Project

Options  
Resource  
Window  
Help

Selectarea unuia dintre submeniuri se face apăsând simultan tastele Alt+litera marcată (highlight) sau apăsând tasta funcțională F10 și cu ajutorul tastelor săgeți, alegând opțiunea dorită. Meniurile sunt de tip pull-down, în continuare fiind prezentate pe scurt meniurile și o serie de opțiuni utilizate frecvent:

- File – oferă opțiunile pentru gestiunea fișierelor, astfel:
  - o New (F7) – permite crearea unui nou fișier;
  - o Open (F8) – permite încărcarea unui fișier existent;
  - o Close – închide fișierul curent;
  - o Save (F2) – permite salvarea fișierului curent;
  - o Save As... – permite salvarea fișierului curent sub alt nume (deci, practic, modificarea numelui unui fișier);
  - o Print – permite listarea la imprimantă;
  - o Exit (Alt + x) – ieșirea totală în sistemul de operare (închidere VIP).
- Project
  - New Project (Ctrl + F7) – permite crearea unui proiect nou;
  - Open Project (Ctrl + F8) – permite încărcarea/crearea unui proiect;
  - Close Project – permite închiderea unui proiect;
  - Save Project (Ctrl+S) – permite salvarea unui proiect;
  - Compile Module (Ctrl+F9) – utilizat pentru compilarea unui modul;
  - Build (Alt + F9) – folosit pentru compilarea și link-editarea unui proiect;
  - Rebuild All (Ctrl + Alt + F9) – permite compilarea și link-editarea tuturor fișierelor din cadrul unui proiect, chiar dacă s-au operat sau nu modificări de la ultima compilare (spre deosebire de comanda Build, care realizează o verificare a timpului fișierului și recompilează doar fișierele care au fost modificate de la ultima compilare);
  - Link Only (Shift + F9) – utilizat pentru link-editarea fișierelor obiect și librăriilor în vederea obținerii modulului executabil sau DLL. În acest caz, nu se verifică dacă există vreun fișier care ar trebui compilat sau recompilat;
  - Run (F9) – pentru compilarea, link-editarea unui proiect în vederea obținerii executabilului, și lansarea în execuție a respectivului proiect;
  - Test Goal (Ctrl + G) – se poate utiliza pentru lansarea în execuție a unui program (nu a unui proiect). Practic, din fișierul curent deschis în editare se generează un executabil special TestGoal și se rulează acest executabil. Programul TestGoal încearcă să satisfacă secțiunea **goal**, în cazul în care toate scopurile au fost executate cu succes, atunci programul TestGoal se termină cu succes, fiind afișate toate soluțiile găsite, precum și numărul de soluții pentru toate variabilele.

Trebuie menționat că în orice moment se poate apela Help-ul, apăsând tasta funcțională F1.

### Ferestrele Prolog-ului

Mediul de programare Visual Prolog pune la dispoziția utilizatorului o serie de ferestre :

- fereastra proiectului
- fereastra de editare - este cea în care se încarcă sau se editează programele.
- fereastra de dialog - este fereastra implicită a intrărilor și ieșirilor din program.
- fereastra mesajelor - este o fereastră a ieșirilor (mesajelor) produse de sistem. Se referă la comenzile de compilare, rulare, încărcare, salvare, mesaje de eroare etc.

- fereastra de depanare a programelor - se utilizeaza pentru rulara pas cu pas, a programului

## Exemple de programe Visual Prolog

Pentru lansarea în execuție a acestor programe trebuie parcurși următorii pași:

1. din meniul Project se alege opțiunea New Project...
2. în cutia de dialog care este afișată pe ecran, la General se specifică numele proiectului și se setează opțiunea Multiprogrammer Mode;
3. în sub-fereastra Target, se alege tipul proiectului (pentru exemplele considerate se selectează la Platform, DOS);
4. se apasă butonul Create;
5. se editează fisierul cu extensia \*.pro și se scrie programul;
6. se apasă tasta F9 pentru lansarea în execuție a proiectului.

1. Afișarea unui text pe ecran.

goal

```
write ("Hello !").
```

unde:

- goal – reprezintă scopul programului
- write (Variabila | constanta) – permite scrierea pe ecran a valorii variabilei sau constantei.

2. Afișarea unui text într-o fereastră utilizator.

goal

```
makewindow(1, 7, 7, "Visual Prolog", 2, 10, 20, 60),  
nl, write ("Primul program!\nSisteme expert").
```

unde:

- goal – reprezintă, așa cum s-a specificat anterior, scopul programului
- makewindow (NrFereastra, AtrEcran, AtrCadru, Titlu, Rind, Coloana, Latime, Inaltime) – este un predicat predefinit care creează o fereastră utilizator. Parametrii reprezintă: numărul ferestrei, attributele (culoarea fondului și culoarea utilizată pentru afișare) ferestrei, attributele chenarului ferestrei, titlul ferestrei (centrat sus), coordonatele colțului stânga – sus, lățimea și înălțimea ferestrei. Atributul de culoare reprezintă un număr întreg obținut prin adunarea unui cod corespunzător culorii fondului și a unui cod corespunzător culorii pentru afișare. Dacă se dorește să se obțină un efect de clipire a ecranului, se adaugă 128 la codul obținut anterior. Codurile culorilor se găsesc în tabelul de la sfârșitul exercițiului. Colțul stânga – sus al ecranului are coordonatele 0, 0.
- nl – predicat care întoarce true și care are ca efect trecerea la rând nou.
- write (Variabila | constanta) – așa cum s-a specificat anterior, permite, printre altele, scrierea unui text pe ecran. Trebuie precizat faptul că se permite utilizarea secvenței escape "\n", care are același efect ca și nl.

Codurile culorilor

Culoare text	Cod	Culoare fond	Cod
Negru	0	negru	0
Albastru	1	albastru	16
Verde	2	verde	32



Bleu	3	bleu	48
Roșu	4	roșu	64
Mov	5	mov	80
Maro	6	maro	96
Alb	7	alb	112
Gri	8		
albastru deschis	9		
verde deschis	10		
Bleu deschis	11		
Roșu deschis	12		
Mov deschis	13		
Galben	14		
alb intens	15		

### 3. Citirea și afișarea unui șir de caractere în Visual Prolog.

goal

```
makewindow(1, 7, 7, "Al doilea program", 2, 10, 20, 60),
nl, write("Introduceti numele dvs.\nși apasati <CR>\n"),
cursor(3, 3), readln(Nume), nl,
write("Bine ai venit, ", Nume).
```

unde, în plus față de exemplul anterior s-au utilizat:

- cursor(Rind, Coloana) – determină poziționarea cursorului în cadrul ferestrei pe linia Rind, coloana indicată de al doilea parametru.
- readln(StringVar) – permite citirea unei variabile de tip șir de caractere (s-a specificat anterior faptul că variabilele în Visual Prolog se scriu cu literă majusculă).