Prolog: Introducere

- Responsabil: Vlad Talmaciu [mailto:vlad.talmaciu@gmail.com]

- Data publicării: 22.04.2019
- Data ultimei modificări: 22.04.2019

Obiective

Scopul acestui laborator este introducerea în programarea logică și învățarea primelor noțiuni despre Prolog.

Aspectele urmărite sunt:

- diferențierea dintre aceasta paradigmă și cele studiate anterior
- familiarizarea cu entitățile limbajului: **fapte**, **reguli**, **scopuri**
- sintaxa **Prolog**
- structuri de date
- backtracking
- unificare

SWI-Prolog

Folosim SWI-Prolog, așa cum este detaliat aici.

În cadrul laboratorului, recomandăm folosirea **swipl** în modul următor:

- se rulează comanda `swipl` în terminal, sau direct `swipl fisier.pl` pentru a încărca un fișier existent
  - alternativ se poate folosi comanda `prolog`
- se salvează faptele și regulile într-un fișier cu extensia `.pl`
- pentru a invoca editorul prologului, se folosește comanda:
  - `edit.`, dacă a fost deja încărcat un fișier;
  - `edit(file('new.pl')).`, pentru a crea un fișier nou;
  - `edit('new.pl').` dacă fișierul a fost creat anterior.
- dacă fișierul nu a fost încărcat prin pasarea ca argument în linia de comandă, se încarcă folosind comanda `consult('file.pl').`
- pentru a reîncărca toate fișierele ce au fost modificate de la ultima încărcare se folosește `make.`

Comentarii

Simbolul `%` transformă restul rândului într-un comentariu.

Entitățile limbajului

Limbajul Prolog (al cărui nume provine de la Programmable Logic) este un limbaj **logic**, **descriptiv**, care permite specificarea problemei de rezolvat în termenii unor **fapte** cunoscute despre obiectele universului problemei și ai relațiilor existente între aceste obiecte.

Tot ceea ce nu este cunoscut sau nu poate fi demonstrat este considerat a fi fals (**ipoteza lumii închise**).

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, numite și **scopuri**, despre obiecte și relațiile dintre ele, la care programul răspunde pe baza faptelor și regulilor existente

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.

Exemple de fapte:

```
papagal(coco).
iubește(mihai, maria).
iubește(mihai, ana).
frumoasă(ana).
bun(gelu).
deplasează(cub, camera1, camera2).
```

Structuri

Structurile au aceeași sintaxă cu faptele, dar apar ca argumente ale predicatelor.

Exemplu de structură:

```
are(ion,carte(aventuri,2002)).
```

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, **true**, sau negativ, **false** (în alte versiuni de Prolog răspunsul poate fi **yes** sau **no**; sau **true** sau **fail**).

Se va vedea mai târziu că programul Prolog, în cazul unui răspuns afirmativ la o întrebare, **poate furniza şi alte informaţii** din baza de cunoștințe.

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

```
?- iubeste(mihai, maria).
true.
?- papagal(coco).
true.
?- papagal(mihai).
false.
?- inalt(gelu).
false.
```

Variabile

În exemplele prezentate până acum, argumentele **faptelor** și **întrebărilor** au fost 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** 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.

Semnul _ (underscore) desemnează o variabila a cărei valoare nu interesează.

```
?- papagal(coco).
true.
?- papagal(CineEste).
CineEste = coco
?- deplaseaza(_, DeUnde, Unde).
DeUnde = camera1, Unde = camera2
```

La fixarea unui **scop** Prolog care conține **variabile**, acestea sunt neinstanțiate 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.

În exemplul de mai jos exista mai multe răspunsuri posibile. Prima soluție este dată de prima unificare și există atâtea soluții câte unificări diferite există.

```
?- iubeste(mihai, X).
```

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 acelui 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 **false**.

In exemplul de mai jos, tastând caracterul “;” și Enter, cerem o nouă soluție.

```
?- iubeste(mihai, X).
X = maria;
X = ana;
false.
?- iubeste(Cine, PeCine).
Cine = mihai, PeCine = maria;
Cine = mihai, PeCine = ana;
false.
```

Reguli

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

```
S :- S1, S2, ..., Sn.
```

Fiecare `Si`, `i = 1,n` și `S` au forma faptelor Prolog, deci sunt predicate, cu argumente constante, variabile sau structuri. Faptul S care definește regula, se numește **antet de regulă**, iar `S1, S2,…, Sn` formează corpul regulii și reprezintă conjuncția de scopuri care trebuie satisfăcute pentru ca antetul regulii să fie satisfăcut.

Fie următoarea bază de cunoștințe:

```
frumoasa(ana).                                      %1
bun(vlad).                                          %2
cunoaste(vlad, maria).                              %3
cunoaste(vlad, ana).                                %4
iubeste(mihai, maria).                              %5
iubeste(X, Y):- bun(X), cunoaste(X, Y), frumoasa(Y).     %6
```

Se observă definirea atât printr-un fapt(linia 5), cât și printr-o regulă (linia 6) a predicatului `iubeste(?Cine, ?PeCine)`.

Backtracking

Pentru a prezenta mecanismul de backtracking în Prolog, vom folosi baza de fapte dată ca exemplu mai sus.

Vom urmari în continuare pașii realizați pentru satisfacerea scopului:

```
?- iubeste(X, Y).
```

Așa cum am precizat anterior, căutarea pornește de la începutul bazei de cunoștințe. Prima potrivire va fi la linia 5, cu faptul:

```
iubeste(mihai, maria).  %5
```

În urma unificării, variabilele `X` si `Y` se vor instanția la constantele `mihai` și `maria` și se va realiza un marcaj la linia respectivă. Pentru a încerca resatisfacerea scopului, căutarea va începe acum după marcajul făcut anterior. Urmatoarea potrivire va fi la linia 6, cu regula:

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

La unificarea scopului 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 identic cu cel al satisfacerii scopului inițial.

În continuare se va încerca satisfacerea scopului `bun(X).`

Fiind vorba de un nou scop, căutarea va avea loc de la începutul bazei de cunoștințe. Acest scop va unifica cu predicatul de la linia 2:

```
bun(vlad).          %2
```

În urma unificării, varibila `X` se va instanția la valoarea `vlad`.

În continuare se va încerca satisfacerea scopului `cunoaste(vlad, Y).` care va unifica cu faptul de la linia 3 și va duce la legarea variabilei `Y` la `maria`:

```
cunoaste(vlad, maria).  %3
```

Ultimul scop care mai trebuie satisfăcut este `frumoasa(maria)`, dar acesta eșuează deoarece nu unifica cu niciunul dintre faptele sau regulile din baza de cunoștințe.

Aici intervine mecanismul de **backtracking**. Se va reveni la scopul satisfăcut anterior, `cunoaste(vlad, Y).` și se va încerca resatisfacerea acestuia. Dacă aceasta ar eșua, ne-am întoarce la scopul dinaintea sa ș.a.m.d. Acest lucru se va repeta pană când se epuizează toate scopurile din corpul regulii, caz în care unificarea scopului inițial cu antetul regulii ar eșua, iar răspunsul ar fi `false`.

În cazul nostru, însă, este necesar un singur pas de întoarcere deoarece `cunoaste(vlad, Y).` poate fi resatisfăcut prin unificarea cu faptul de la linia 4:

```
cunoaste(vlad, ana).    %4
```

În urma unificării, variabila `Y` va fi legată la `ana`, iar următorul scop ce va trebui satisfăcut este `frumoasa(ana)`. Acesta va reuși deoarece unifică cu faptul de la linia 1:

```
frumoasa(ana).     %1
```

Astfel am obținut o noua soluție pentru scopul `iubeste(X, Y).`, avand `X = vlad` si `Y = ana`.

Procesul nu se oprește însă pana când nu s-au încercat toate unificările posibile pentru satisfacerea scopului. Tot prin intermediul mecanismului de backtracking, se va reveni la scopurile anterioare și se va observa dacă acestea pot fi resatisfăcute. În exemplul nostru acest lucru nu este posibil, deci răspunsul va fi `false`.

Prin urmare, rezultatul final va fi:

```
?- iubeste(X, Y).
X = mihai, Y = maria;
```

```
X = vlad, Y = ana;
false.
```

Operatori

- Aritmetici: + - * /
- Relaționali: = \= < > =< >= =:=

La scrierea expresiei `1+2*(X/Y)`, valoarea acesteia nu este calculată, ci expresia este reținută ca atare. Se poate observa că operatorii `=:=` și `is` forțează evaluarea unei expresii, pe când `=` verifica doar egalitatea structurală.

De asemenea, `is` și `=` pot primi variabile neinstanțiate pe care le instanțiază.

```
?- 1 + 2 =:= 2 + 1.
true.
```

```
?- 1 + 2 = 2 + 1.
false.
```

```
?- X = 2 + 1.
X = 2+1.
```

```
?- X is 2 + 1.
X = 3.
```

```
?- X =:= 2 + 1.
ERROR: =:=/2: Arguments are not sufficiently instantiated
```

Liste

- Lista vida: `[]`
- Lista cu elementele a, b, c: `[a,b,c]`
- Lista nevida: `[Prim|Rest]` – unde variabila `Prim` unifică cu primul element al listei, iar variabila `Rest` cu lista fără acest prim element
- Lista care începe cu n elemente `X1, X2, …, XN` și continua cu o alta lista `Rest`: `[X1,X2,…,XN|Rest]`

Documentarea predicatelor și a argumentelor

Pentru claritate, antetele predicatelor se scriu sub forma:

- **predicat/nrArgumente**
- predicat(+Arg1, -Arg2, ?Arg3, …, +ArgN)

Pentru a diferenția intrările (+) de ieșiri(-), se prefixează argumentele cu indicatori. Acele argumente care pot fi fie intrări, fie ieșiri se prefixează cu '?'. Instanțierea parametrilor ține de specificarea acestora:

- Arg1 va fi instanțiat atunci când se va încerca satisfacerea p/3
- Arg2 se va instanția odată cu satisfacerea p/3
- Arg3 va putea fi instanțiat sau nu atunci când se va satisface p/3

Puterea generativa a limbajului

Așa cum am văzut anterior scopurile pot fi privite ca întrebări ale căror răspunsuri sunt `true` sau `false`. În plus, acest răspuns poate fi însoțit de instanțierile variabilelor din cadrul scopului. Acest mecanism ne ajută să folosim scopurile pentru a obține rezultate de orice tip.

De exemplu, pentru a obține lungimea unei liste putem folosi:

```
% lungime(+Lista,-Lungime)
lungime([],0).
lungime([_|R], N):- lungime(R,N1), N is N1 + 1.
```

```
?- lungime([1,2,3],N).
N = 3.
```

În exemplul de mai sus se va încerca satisfacerea scopului `lungime([1,2,3],N).` prin instanțierea convenabilă a variabilei `N`. În acest caz soluția este unică, dar așa cum am văzut anterior, putem avea situații în care există mai multe unificări posibile. Putem folosi faptul că se încearcă resatisfacerea unui scop în mod exhaustiv pentru a genera multiple rezultate.

În exemplul de mai jos vom defini predicatul **membru(?Elem,+List)** care verifică apartenența unui element la o listă:

```
% membru(?Elem,+Lista)
membru(Elem,[Elem|_]).
membru(Elem,[_|Rest]) :- membru(Elem,Rest).
```

Putem folosi acest predicat pentru a obține un răspuns:

```
?- membru(3,[1,2,3,4,5]).
true .
```

Sau putem să îl folosim pentru a genera pe rând toate elementele unei liste:

```
?- membru(N,[1,2,3]).
N = 1 ;
N = 2 ;
N = 3 ;
false.
```

Inițial, scopul `membru(N,[1,2,3]).` va unifica cu faptul `membru(Elem,[Elem|_]).`, în care `Elem = N` și `Elem = 1`, din care rezultă instanțierea `N = 1`. Apoi se va încerca unificarea cu antetul de regulă `membru(Elem,[_|Rest])`, în care `Elem = N`, iar `Rest = [2,3]`. Acest lucru implică satisfacerea unui nou scop, `membru(N,[2,3]).`. Noul scop va unifica, de asemenea, cu faptul de la linia 1, `membru(Elem,[Elem|_]).`, din care va rezulta `N = 2`. Asemănător se va găsi și soluția `N = 3`, după care nu va mai reuși nicio altă unificare.

Pentru a exemplifica utilizarea acestui mecanism, vom considera următorul exemplu în care dorim generarea, pe rând, a tuturor permutărilor unei liste:

```
% remove(+Elem,+Lista,-ListaNoua)
remove(E,[E|R],R).
remove(E,[F|R],[F|L]):- remove(E,R,L).

% perm(+Lista,-Permutare)
perm([],[]).
perm([F|R],P):- perm(R,P1), remove(F,P,P1).
```

Observați ca am definit predicatul **remove(+Elem,+Lista,-ListaNoua)**, care șterge un element dintr-o listă. Rolul acestuia în cadrul regulii `perm([F|R],P):- perm(R,P1), remove(F,P,P1).` este, de fapt, de a insera elementul `F` în permutarea `P1`. Poziția pe care va fi inserat elementul va fi diferită la fiecare resatisfacere a scopului `remove(F,P,P1)`, ceea ce ne ajută sa obținem permutările.

```
?- remove(3,L,[1,1,1]).
L = [3, 1, 1, 1] ;
L = [1, 3, 1, 1] ;
L = [1, 1, 3, 1] ;
L = [1, 1, 1, 3] ;
false.
```

```
?- perm([1,2,3],Perm).
Perm = [1, 2, 3] ;
Perm = [2, 1, 3] ;
Perm = [2, 3, 1] ;
Perm = [1, 3, 2] ;
Perm = [3, 1, 2] ;
Perm = [3, 2, 1] ;
false.
```

Resurse

- Exemple
- Cheatsheet

- Exerciții

Referințe

- Learn prolog now! [http://www.learnprolognow.org/]
- Logic, Programming, and Prolog [http://www.ida.liu.se/~ulfni53/lpp/bok/bok.pdf]
- Built-in Predicates [http://www.swi-prolog.org/pldoc/doc_for?object=section%281,%274%27,swi%28%27/doc/Manual/builtin.html%27%29%29]
- Red Cut vs Green Cut [http://en.wikipedia.org/wiki/Cut_%28logic_programming%29#Red_Cut]
