Documentatie laborator 12 – Introducere in Prolog

Documentatia este un rezumat din indrumarul de Prolog, pe care il puteti consulta pentru a aprofunda aceste cunostinte. Pentru a incepe sa programati in Prolog, downloadati programul de la http://www.swi-prolog.org/Download.html.

Entitatile limbajului

Limbajul Prolog este un limbaj logic, descriptiv, care permite specificarea problemei de rezolvat în termenii unor fapte cunoscute despre obiectele universului problemei şi a relaţiilor existente între aceste obiecte. Execuţia unui program Prolog constă în deducerea implicaţiilor acestor fapte şi relaţii, programul definind astfel o mulţime de consecinţe ce reprezintă înţelesul sau semnificaţia declarativă a programului.

Un program Prolog conţine următoarele entităţi:

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

Fapte

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

Exemple:

Fapt:	**Aritate:**
papagal(coco).	1
iubeşte(mihai, maria).	2
iubeşte(mihai, ana).	2
frumoasă(ana).	1
bun(gelu).	1
deplasează(cub, camera1, camera2).	3

Structuri

Structurile au aceeasi sintaxa cu faptele.

Exemplu:

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

Scopuri

Obţinerea consecinţelor sau a rezultatului unui program Prolog se face prin fixarea unor scopuri care pot fi adevărate sau false, în funcţie de conţinutul bazei de cunoştinţe Prolog. Scopurile sunt predicate pentru care se doreşte aflarea valorii de adevăr în contextul faptelor existente în baza de cunoştinţe. Cum scopurile pot fi văzute ca întrebări, rezultatul unui program Prolog este răspunsul la o întrebare (sau la o conjuncţie 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 şi alte informaţii din baza de cunoştinţe.

Exemplu

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

?- iubeste(mihai, maria).
yes deoarece acest fapt există în baza de cunoştinţe
?- papagal(coco).
yes
?- papagal(mihai).
no deoarece acest fapt **nu** există în baza de cunoştinţe
?- inalt(gelu).
no

Variabile

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 convenţie, numele argumentelor variabile începe cu literă mare iar numele constantelor simbolice începe cu literă mică. O variabilă poate fi *instanţiată* (legată) dacă există un obiect asociat acestei variabile, sau *neinstanţiată* (liberă) dacă nu se ştie încă ce obiect va desemna variabila. Semnul _ (underscore) desemneaza o variabila a carei valoare nu intereseaza.

La fixarea unui scop Prolog care conţine variabile, acestea sunt neinstanţiate iar sistemul încearcă satisfacerea acestui scop căutând printre faptele din baza de cunoştinţe un fapt care poate identifica cu scopul, printr-o instanţiere 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 cunoştinţe.

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

Exemple:

?- papagal(CineEste).
CineEste = coco
?- deplaseaza(Ce, DeUnde, Unde).
Ce = cub, DeUnde = camera1**, Unde = camera**2
?- deplaseaza(Ce, Aici, Aici).
no

Cum se comportă sistemul Prolog în cazul în care există mai multe fapte în baza de cunoştinţe care unifică cu întrebarea pusă? În acest caz există mai multe răspunsuri la întrebare, corespunzând mai multor soluţii ale scopului fixat. Prima soluţie este dată de prima unificare şi există atâtea soluţii câte unificări diferite există. La realizarea primei unificări se marchează faptul care a unificat şi care reprezintă prima soluţie. La obţinerea următoarei soluţii, căutarea este reluată de la marcaj în jos în baza de cunoştinţe. Obţinerea primei soluţii este de obicei numită *satisfacerea scopului* iar obţinerea altor soluţii, *resatisfacerea scopului*. La satisfacera unui scop căutarea se face întotdeauna de la începutul bazei de cunoştinţe. 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 obţinerea 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:

?- iubeste(mihai, X).
X = maria; tastând caracterul ";" şi Enter, cerem o nouă soluţie
X = ana;
no
?- iubeste(Cine, PeCine).
Cine = mihai, PeCine = maria;
Cine = mihai, PeCine = ana;
no

Reguli

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

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

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 conjuncţia de scopuri care trebuie satisfăcute pentru ca antetul regulii să fie satisfăcut.

Fie următoarea bază de cunoştinţe Prolog:

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

Se observă că enunţul (6) defineşte o regulă Prolog; relaţia **iubeste(Cine, PeCine)**, fiind definită atât printr-un fapt (5) cât şi printr-o regulă (6).

În condiţiile existenţei regulilor în baza de cunoştinţe Prolog, satisfacerea unui scop se face printr-un procedeu similar cu cel prezentat în Secţiunea 1.2, dar unificarea scopului se încearcă atât cu fapte din baza de cunoştinţe, 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 conjuncţia de scopuri. Scopurile din corpul regulii devin subscopuri a căror satisfacere se va încerca printr-un mecanism similar cu cel al satisfacerii scopului iniţial.

Pentru baza de cunoştinţe descrisă mai sus, satisfacerea scopului

?- iubeste(vlad, ana).

se va face în următorul mod. Scopul unifică cu antetul regulii (6) şi duce la instanţierea variabilelor din regula (6): **X = vlad** 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(vlad)**, care reuşeşte prin unificare cu faptul (2), **cunoaste(vlad, ana)**, care reuşeşte prin unificare cu faptul (4), şi a scopului **frumoasa(ana)**, care reuşeşte prin unificare cu faptul (1). În consecinţă, regula a fost îndeplinită, deci şi întrebarea iniţială este adevarată, iar sistemul răspunde **yes**.

Ce se întîmplă dacă se pune întrebarea:

?- iubeste(X, Y).

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

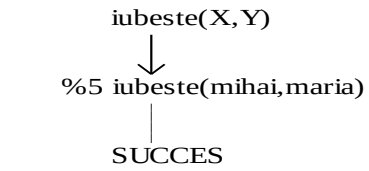
X = mihai, Y = maria

Sistemul Prolog va pune un marcaj în dreptul faptului (5) care a satisfăcut scopul. Următoarea soluţie a scopului **iubeste(X, Y)** se obţine începând căutarea de la acest marcaj în continuare în baza de cunoştinţe. 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 instanţiată cu valoarea **vlad**, $X = \text{vlad}$. Se încearcă acum satisfacerea scopului **cunoaste(vlad, Y)**, care este satisfăcut de faptul (3) şi determină instanţierea **Y = maria**. Se introduce în baza de cunoştinţe un marcaj asociat scopului **cunoaste(vlad, 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(vlad, Y)**, în speranţa că o noua soluţie a acestui scop va putea satisface şi scopul curent care a eşuat, **frumoasa(Y)**. Resatisfacerea scopului **cunoaste(vlad, Y)** se face pornind căutarea de la marcajul asociat scopului în jos, deci de la faptul (3) în jos. O nouă soluţie (resatisfacere) a scopului **cunoaste(vlad, Y)** este dată de faptul (4) care determină instanţierea **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 cunoştinţe şi scopul **frumoasa(ana)** este satisfăcut de faptul (1). În consecinţă a doua soluţie a scopului **iubeste(X, Y)** este obţinută şi sistemul răspunde:

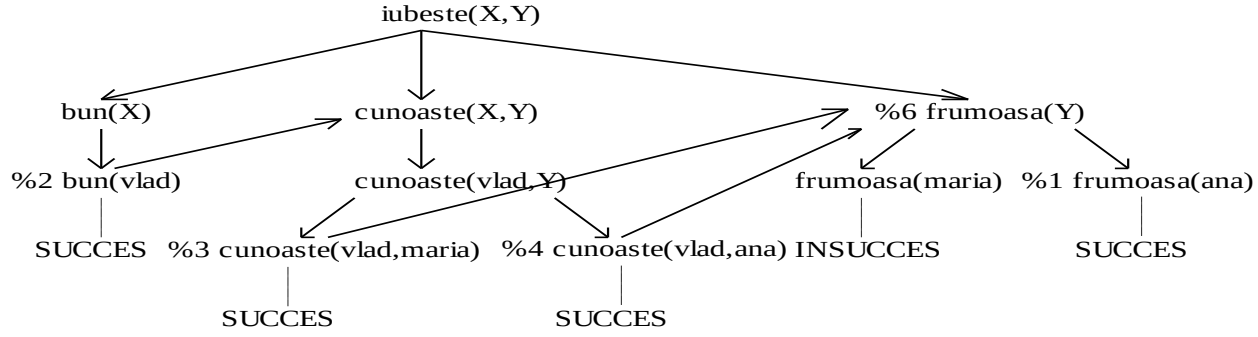
X = vlad, Y = ana

urmând un mecanism de backtracking, descris intuitiv în figura de mai jos prin prezentarea arborilor de deducţie construiţi de sistemul Prolog.

La încercarea de resatisfacere a scopului **iubeste(X, Y)**, printr-un mecanism similar, se observă că nu mai există alte solutii. În concluzie, fiind dată baza de fapte şi reguli Prolog anterioară, comportarea sistemului Prolog este:



solutie 1: X=mihai, Y=maria



solutie 2: X=vlad, Y=ana

?- iubeste(X, Y).

X = mihai, Y = maria;

X = vlad, Y = ana;

no

Observaţii:

- La satisfacerea unei conjuncţii 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 cunoştinţe 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 cunoştinţe de la marcaj în jos. Înainte de resatisfacerea unui scop se elimină toate instanţierile de variabile determinate de ultima satisfacere a acestuia. Dacă cel mai din stânga scop din conjuncţia de scopuri nu poate fi satisfăcut, întreaga conjuncţie de scopuri eşuează.
- Această comportare a sistemului Prolog în care se încearcă în mod repetat satisfacerea şi resatisfacerea scopurilor din conjuncţiile de scopuri se numeşte *backtracking*.

Operatori

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

La scrierea expresiei 1+2*(X/Y), valoarea acesteia nu este calculata, ci expresia este retinuta ca atare. Evaluarea se poate forta folosind operatorul **is**.

Exemplu:

```
X is 1+2
```

Din acest motiv, operatorul = testeaza echivalenta structurala, iar operatorul =:= testeaza echivalenta valorilor.

Exemple:

```
1+2 =:= 2+1.
yes
1+2 = 2+1.
No
```

Liste

- Lista vida: **[]**
- Lista cu elementele a,b,c: **[a,b,c]**
- Lista nevida: **[Prim|Rest]** – unde variabila Prim unifica cu primul element al listei, iar variabila Rest cu lista fara acest prim element
- Lista care incepe cu 2 elemente X, Y si continua cu o alta lista Rest: **[X,Y|Rest]**

Mecanismele cut si fail

Predicatul **cut**, notat cu !, este un predicat fara argumente care se indeplineste intotdeauna si nu poate fi resatisfacut. Cand se intalneste cut, toate selectiile facute intre antet si cut sunt inghetate (nu mai pot fi gasite noi solutii pe aceasta portiune; dar mai pot fi gasite solutii noi prin resatisfacerea scopurilor aflate dupa cut in acea clauza). Daca o clauza cu cut reuseste sa ajunga pana la cut, toate clauzele urmatoare cu acelasi antet sunt ignorate.

Predicatul **fail** este un predicat fara argumente care esueaza intotdeauna. Este de obicei precedat de cut, pentru a preveni backtrackingul pe scopurile care il preced (inutil pentru ca oricum se va ajunge la fail deci la esec). Combinatia !,fail are efect de negatie, ca si predicatul not din Prolog.

Comentarii

Simbolul % transforma restul randului intr-un comentariu.

Consultati fisierul lab12-doc.pl pentru cateva exemple.