

Laborator 1

2019-2020

Programare Logică

De ce programare logica (PL)?

- Programarea logică este utilă în strategii de căutare, prototipuri, rezolvare de puzzle-uri etc.
- Idei "declarative" apar în multe domenii din informatică:
 - concepte din PL sunt utile în inteligență artificială și baze de date
 - demonstratoare automate, *model-checking*, *constraint programming*
 - devin importante în analiza programelor, semantica Web
- Învățând o metoda foarte diferită "de a gândi probleme", deveniți programatori mai buni :)

Prolog

- Prolog este cel mai cunoscut limbaj de programare logică.
 - bazat pe logica clasică de ordinul I (cu predicate)
 - funcționează pe bază de unificare și căutare
- Multe implementări îl transformă în limbaj de programare "matur"
 - I/O, operații implementate deja în limbaj etc.

Prolog

- Vom folosi implementarea SWI-Prolog
 - gratuit
 - folosit des pentru predare
 - conține multe librării
 - <http://www.swi-prolog.org/>
- Varianta online SWISH a SWI-Prolog
 - <http://swish.swi-prolog.org/>

Laboratorul 1

TODO

- Cum arată un program în Prolog?
- Cum punem întrebări în Prolog?

Mai multe detalii

- Capitolul 1 din *Learn Prolog Now!*.

Sintaxă: atomi

Atomi:

- secvențe de litere, numere și `_`, care încep cu o literă mică
- șiruri între apostrofuri `'Atom'`
- anumite simboluri speciale

Exemplu

- `sansa`
- `jon_snow`
- `'Ser Gregor Clegane'`
- `'(@ *+'`
- `+`

```
?- atom('(@ *+ ').  
true.
```

`atom/1` este un predicat predefinit

Sintaxă: constante și variabile

Constante:

- `atomi`: sansa, 'I am an atom'
- `numere`: 2, 2.5, -33

Variabile:

- secvențe de litere, numere și `_`, care încep cu o literă mare sau cu `_`
- Variabilă specială: `_` este o **variabilă anonimă**
 - două apariții ale simbolului `_` sunt variabile diferite
 - este folosită când nu vrem detalii despre variabila respectivă

Exemplu

- `X`
- `Arya`
- `_cersei`

Sintaxă: termeni compuși

Termni compuși:

- au forma $p(t_1, \dots, t_n)$ unde
 - p este un atom,
 - t_1, \dots, t_n sunt termeni.

Exemplu

- `dislike(cersei,tyrion)`
- `dislike(cersei,X)`
- Un termen compus are
 - un **nume** (**functor**): `dislike` în `dislike(cersei,tyrion)`
 - o **aritate** (numărul de argumente): 2 în `dislike(cersei,tyrion)`

kb1: Un prim exemplu

Un **program** Prolog definește o bază de cunoștințe.

Exemplu

```
stark(eddard).  
stark(jon_snow).  
stark(sansa).  
  
lannister(tyrion).  
lannister(cersei).  
  
dislike(cersei,tyrion).
```

Predicate

O bază de cunoștințe este o mulțime de **predicate** prin care definim **lumea**(universul) programului respectiv.

Exemplu

```
stark(eddard).  
stark(jon_snow).  
stark(sansa).  
  
lannister(tyrion).  
lannister(cersei).  
  
dislike(cersei,tyrion).
```

Acest program conține trei predicate:
stark/1, lannister/1, dislike/2.

Definirea predicatelor

- Predicate cu același nume, dar cu arități diferite, sunt predicate diferite.
- Scriem `foo/n` pentru a indica că un predicat `foo` are aritatea `n`.
- Predicatele pot avea aritatea 0 (nu au argumente); sunt predefinite în limbaj (`true`, `false`).

kb2: Un exemplu cu fapte și reguli

- O **regulă** este o afirmație de forma **Head** :- **Body**.
- Un **fapt** (*fact*) este o regulă fără Body.

Exemplu

```
eating(joffrey).  
deceased(robert).  
dislike(cersei,tyrion).
```

```
happy(cersei) :- happy(joffrey).  
happy(ser_jamie) :- happy(cersei), deceased(robert).  
happy(joffrey) :- dislike(joffrey,sansa).  
happy(joffrey) :- eating(joffrey).
```

Reguli

O **regulă** este o afirmație de forma **Head** :- **Body**. unde

- Head este un predicat (termen complex)
- Body este o secvență de predicate, separate prin virgulă.

Exemplu

```
happy(ser_jamie) :- happy(cersei), deceased(robert).
```

Interpretarea:

- :- se interpretează drept **implicație** (\leftarrow)
- , se interpretează drept **conjuncție** (\wedge)

Astfel, din punct de vedere al logicii, putem spune că `happy(ser_jamie)` este echivalent cu `happy(cersei) \wedge deceased(robert)`.

Definirea predicatelor

Mai multe reguli care au același Head trebuie gândite că au **sau** între ele.

Exemplu

```
happy(joffrey) :- dislike(joffrey,sansa).  
happy(joffrey) :- eating(joffrey).
```

Dacă `dislike(joffrey,sansa)` este adevărat sau `eating(joffrey)` este adevărat, atunci `happy(joffrey)` este adevărat. Mai multe reguli cu aceeași parte stângă se pot uni folosind `;`.

Exemplu

```
happy(joffrey) :- dislike(joffrey,sansa) ;  
eating(joffrey).
```

Astfel, din punct de vedere al logicii, putem spune că `happy(joffrey)` este echivalent cu `dislike(joffrey,sansa) ∨ eating(joffrey)`.

Sintaxă: program

Un **program** în Prolog este o colecție de fapte și reguli.

Faptele și regulile trebuie grupate după atomii folosiți în Head.

Exemplu

Corect:

```
stark(eddard).  
stark(jon_snow).  
stark(sansa).  
  
lannister(tyrion).  
lannister(cersei).  
  
dislike(cersei,tyrion).
```

Incorrect:

```
stark(eddard).  
dislike(cersei,tyrion).  
stark(sansa).  
  
lannister(tyrion).  
stark(jon_snow).  
  
lannister(cersei).
```

Întrebări, răspunsuri, ținte

- O **întrebare** (*query*) este o secvență de forma
$$?- p_1(t_1, \dots, t_n), \dots, p_n(t_1', \dots, t_n').$$
- Fiind dată o întrebare (deci o țintă), Prolog caută **răspunsuri**.
 - **true**/**false** dacă întrebarea nu conține variabile;
 - dacă întrebarea conține variabile, atunci sunt căutate valori care fac toate predicatele din întrebare să fie satisfăcute; dacă nu se găsesc astfel de valori, răspunsul este **false**.
- Predicatele care trebuie satisfăcute pentru a răspunde la o întrebare se numesc **ținte** (*goals*).

kb2: Exemple de întrebări și răspunsuri

Exemplu

```
eating(joffrey).  
deceased(rickard).  
dislike(cersei,tyrion).
```

```
happy(cersei) :- happy(joffrey).  
happy(ser_jamie) :- happy(cersei),  
                    deceased(robert).  
happy(joffrey) :-  
                    dislike(joffrey,sansa).  
happy(joffrey) :- eating(joffrey).
```

```
?- happy(joffrey).  
true
```

```
?- happy(cersei).  
true
```

```
?- happy(ser_jamie).  
false
```

```
?- happy(X).  
X = cersai ;  
X = joffrey.
```

În varianta online, puteți adăuga întrebări la finalul programului ca în exemplul de mai jos. Întrebările vor apărea în lista din *Examples* (partea dreaptă).

Exemplu

```
eating(joffrey).
deceased(rickard).
dislike(cersei,tyrion).

happy(cersei) :- happy(joffrey).
happy(ser_jamie) :- happy(cersei), deceased(robert).
happy(joffrey) :- dislike(joffrey,sansa).
happy(joffrey) :- eating(joffrey).

/** <examples>

?- happy(joffrey).
?- happy(cersei).
?- happy(ser_jamie).
?- happy(X).
*/
```

kb3: Un exemplu cu date și reguli ce conțin variabile

Exemplu

```
father(eddard,sansa).  
father(eddard,jon_snow).
```

```
mother(catelyn,sansa).  
mother(wylla,jon_snow).
```

```
stark(eddard).  
stark(catelyn).
```

```
stark(X) :- father(Y,X),  
            stark(Y).
```

Pentru orice X, Y , dacă $\text{father}(Y,X)$ este adevărat și $\text{stark}(Y)$ este adevărat, atunci $\text{stark}(X)$ este adevărat.

Adică, pentru orice X , dacă tatăl lui X este stark, atunci și X este stark.

kb3: Un exemplu cu date și reguli ce conțin variabile

Exemplu

```
?- stark(jon_snow).  
true
```

```
?- stark(X).  
X = eddard  
X = catelyn  
X = sansa  
X = jon_snow  
false
```

```
?- stark(X), mother(Y,X), stark(Y).  
X = sansa,  
Y = catelyn  
false
```

- Un program în Prolog are extensia `.pl`

- Comentarii:

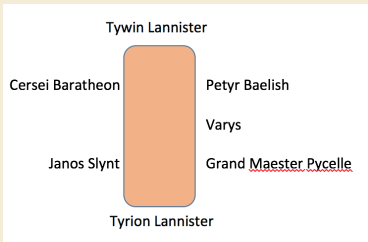
```
% comentează restul liniei
```

```
/* comentariu  
pe mai multe linii */
```

- Nu uitați să puneți `.` la sfârșitul unui fapt sau al unei reguli.
- un program(o bază de cunoștințe) se încarcă folosind:
?- [nume].
?- ['...cale.../nume.pl'].

Exercițiul 1: consiliu

Imaginea de mai jos arată cum sunt așezați membrii consiliului lui Joffrey:



Definiți predicatul `sits_right_of/2` pentru a reprezenta cine lângă cine stă. `sits_right_of(X,Y)` trebuie să fie adevărat dacă X este la dreapta lui Y.

Exercițiul 1 (cont.)

Adăugați următoarele predicate:

- `sits_left_of/2`: `sits_left_of(X,Y)` trebuie să fie adevărat dacă X este la stânga lui Y.
- `are_neighbors_of/3`: `are_neighbors_of(X,Y,Z)` trebuie să fie adevărat dacă X este la stânga lui Z și Y este la dreapta lui Z.
- `next_to_each_other/2`: `next_to_each_other(X,Y)` trebuie să fie adevărat dacă X este lângă Y.

Exercițiul 1 (cont.)

Testați implementarea voastră punând următoarele întrebări:

- ☐ Este Petyr Baelish la dreapta lui Cersei Baratheon?
- ☐ Este Petyr Baelish la dreapta lui Varys?
- ☐ Cine este la dreapta lui Janos Slynt?
- ☐ Cine stă doua scaune la dreapta lui Cersei Baratheon?
- ☐ Cine stă între Petyr Baelish și Grand Master Pycelle?

Exercițiul 2: arbore genealogic

Aici găsiți poza marită.
Folosiți baza de cunoștințe "kb4.pl" ca punct de plecare.

Exercițiul 2 (cont.)

Folosind predicatele `male/1`, `female/1` și `parent_of/2`, adăugați reguli pentru următoarele predicate:

- ☐ `father_of(Father, Child)`
- ☐ `mother_of(Mother, Child)`
- ☐ `grandfather_of(Grandfather, Child)`
- ☐ `grandmother_of(Grandmother, Child)`
- ☐ `sister_of(Sister, Person)`
- ☐ `brother_of(Brother, Person)`
- ☐ `aunt_of(Aunt, Person)`
- ☐ `uncle_of(Uncle, Person)`

Exercițiul 2 (cont.)

Verificați predicate definite punând diverse întrebări.

Folosiți acest program în Prolog pentru a afla dacă Daenerys Targaryen este matușa lui Jon Snow.

Exercițiul 3: recursie/arbore genealogic

Folosind baza de cunoștințe anterioară, scrieți un predicat `ancestor_of` (`Ancestor`, `Person`) care să verifice dacă al primul argument este strămoș al celui de-al doilea.

Exemplu

```
?- ancestor_of(rickardStark,jonSnow).  
true
```

Negația

În Prolog există predicatul predefinit `not` cu următoarea semnificație:
`not(goal)` este true dacă `goal` nu poate fi demonstrat în baza de date curentă.

Atenție: `not` nu este o negație logică, ci exprimă imposibilitatea de a face demonstrația (sau instanțierea) conform cunoștințelor din bază ("closed world assumption"). Pentru a marca această distincție, în variantele noi ale limbajului, în loc de `not` se poate folosi operatorul `\+`.

Exemplu

```
not_parent(X,Y) :- not(parent_of(X,Y)). % sau  
not_parent(X,Y) :- \+ parent_of(X,Y).
```

Exercițiul 4: negația

Folosind baza anterioară (arbore genealogic) testați predicatul `not_parent`:

```
?- not_parent(rickardStark,viserysTargaryen).  
?- not_parent(X,jonSnow).  
?- not_parent(X,Y).
```

Ce observați? Încercați să analizați răspunsurile primite.

Exercițiul 4: negația

Folosind baza anterioară (arbore genealogic) testați predicatul `not_parent`:

```
?- not_parent(rickardStark,viserysTargaryen).  
?- not_parent(X,jonSnow).  
?- not_parent(X,Y).
```

Ce observați? Încercați să analizați răspunsurile primite.

- Corectați `not_parent` astfel încât să dea răspunsul corect la toate întrebările de mai sus.
- Scrieți un predicat `ancestor_not_parent(X,Y)` care să verifice dacă `X` este strămoș dar nu este părinte al lui `Y`.

Practică

În GOT apare următoarea ghicitoare:

<https://goodriddlesnow.com/posts/view/riddle-from-game-of-thrones>

Varys - "Power is a curious thing, my lord. Are you fond of riddles?"

Tyrion - "Why? Am I about to hear one?"

Varys - "Three great men sit in a room, a king, a priest and the rich man. Between them stands a common sellsword. Each great man bids the sellsword kill the other two. Who lives? Who dies?"

Tyrion - "Depends on the sellsword."

Vom descrie această problemă în Prolog, făcând urmă toarele presupuneri:

- dacă mercenarul este un om religios, atunci nu va ucide preotul,
- dacă mercenarul este un om care respectă autoritatea, atunci nu va ucide regele,
- dacă mercenarul este un om care dorește bani, atunci nu va ucide omul bogat.

Exercițiul 5: ghicitoare

În fișierul kb5.pl este începută o modalitate de a descrie ghicitoarea de mai sus. Completați baza de cunoștințe și scrieți un predicat

```
is_killed(C,X,Y)
```

care verifică faptul că X și Y sunt uciși de alegerea C a mercenarului.

```
?- is_killed(god, X,Y).
```

```
X = king,
```


```
Y = richMan
```

Bibliografie

- <http://www.learnprolognow.org>
- <http://cs.union.edu/~striegnk/courses/esslli04prolog>



Pe data viitoare!



`http://tinyurl.com/y4u96fv2`