

Laborator 1

Fundamentele Limbajelor de Programare

- 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.

- 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/>

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 apostrofuli `'Atom'`
- anumite simboluri speciale

Exemplu

- `elefant`
- `abcXYZ`
- `'Acesta este un atom'`
- `'(@ *+'`
- `+`

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

`atom/1` este un predicat predefinit

Sintaxă: constante și variabile

Constante:

- `atomi`: `a`, `'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`
- `Animal`
- `_x`
- `X_1_2`

Sintaxă: termeni compuși

Termeni compuși:

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

Exemplu

- `is_bigger(horse, X)`
- `is_bigger(horse, dog)`
- `f(g(X, _), 7)`
- Un termen compus are
 - un **nume** (functor): `is_bigger` în `is_bigger(horse, X)`
 - o **aritate** (numărul de argumente): 2 în `is_bigger(horse, X)`

kb1: Un prim exemplu

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

Exemplu

```
bigger(elephant, horse).
```

```
bigger(horse, donkey).
```

```
bigger(donkey, dog).
```

```
bigger(donkey, monkey).
```

```
is_bigger(X, Y) :- bigger(X, Y).
```

```
is_bigger(X, Y) :- bigger(X, Z), is_bigger(Z, Y).
```


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

Exemplu

```
bigger(elephant, horse).  
bigger(horse, donkey).  
bigger(donkey, dog).  
bigger(donkey, monkey).
```

```
is_bigger(X, Y) :- bigger(X, Y).  
is_bigger(X, Y) :- bigger(X, Z), is_bigger(Z, Y).
```

Acest program conține două predicate:

```
bigger/2, is_bigger/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`).
- Predicate predefinite: $X=Y$ (este adevărat dacă X poate fi unificat cu Y); $X \neq Y$ (este adevărat dacă X nu poate fi unificat cu Y);

Un exemplu cu fapte și 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

```
is_smaller(X, Y) :- is_bigger(Y, X).  
aunt(Aunt, Child) :- sister(Aunt, Parent),  
                        parent(Parent, Child).
```

- Un **fapt** (*fact*) este o regulă fără Body.

Exemplu

```
bigger(whale, _).  
life_is_beautiful.
```

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

Exemplu

```
is_bigger(X, Y) :- bigger(X, Y).
```

```
is_bigger(X, Y) :- bigger(X, Z), is_bigger(Z, Y).
```

Interpretarea:

- Mai multe reguli care au același Head trebuie gândite că au **sau** între ele.
- **:-** se interpretează drept **implicație** (\leftarrow)
- **,** se interpretează drept **conjunție** (\wedge)

Astfel, din punct de vedere al logicii, putem spune că **is_bigger(X, Y)** este adevarat dacă **bigger(X, Y) \vee bigger(X, Z) \wedge is_bigger(Z, Y)** este adevarat.

Definirea predicatelor

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

Exemplu

```
is_bigger(X, Y) :- bigger(X, Y).  
is_bigger(X, Y) :- bigger(X, Z), is_bigger(Z, Y).
```

Ele se pot uni folosind ;.

Exemplu

```
is_bigger(X, Y) :-  
    bigger(X, Y);  
    bigger(X, Z), is_bigger(Z, Y).
```

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:

```
bigger(elephant, horse).  
bigger(horse, donkey).  
bigger(donkey, dog).  
bigger(donkey, monkey).  
is_bigger(X, Y) :-  
    bigger(X, Y).  
is_bigger(X, Y) :-  
    bigger(X, Z),  
    is_bigger(Z, Y).
```

Inc corect:

```
bigger(elephant, horse).  
bigger(horse, donkey).  
is_bigger(X, Y) :-  
    bigger(X, Y).  
bigger(donkey, dog).  
bigger(donkey, monkey).  
is_bigger(X, Y) :-  
    bigger(X, Z),  
    is_bigger(Z, Y).
```

- 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*).

Exemple de întrebări și răspunsuri

Exemplu

```
bigger(elephant, horse).  
bigger(horse, donkey).  
bigger(donkey, dog).  
bigger(donkey, monkey).
```

```
is_bigger(X, Y) :-  
    bigger(X, Y).  
is_bigger(X, Y) :-  
    bigger(X, Z),  
    is_bigger(Z, Y).
```

```
?- is_bigger(elephant, horse).  
true
```

```
?- bigger(donkey, dog).  
true
```

```
?- is_bigger(elephant, dog).  
true
```

```
?- is_bigger(monkey, dog).  
false
```

```
?- is_bigger(X, dog).  
X = donkey ;  
X = elephant ;  
X = horse
```


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

```
bigger(elephant, horse).  
bigger(horse, donkey).  
bigger(donkey, dog).  
bigger(donkey, monkey).  
is_bigger(X, Y) :- bigger(X, Y).  
is_bigger(X, Y) :- bigger(X, Z), is_bigger(Z, Y).
```

```
/** <examples>
```

```
?- is_bigger(elephant, horse).  
?- bigger(donkey, dog).  
?- is_bigger(elephant, dog).  
?- is_bigger(monkey, dog).  
?- is_bigger(X, dog).  
*/
```

Un exemplu cu date și reguli ce conțin variabile

Exemplu

```
?- is_bigger(X, Y), is_bigger(Y,Z).  
X = elephant,  
Y = horse,  
Z = donkey  
X = elephant,  
Y = horse,  
Z = dog  
X = elephant,  
Y = horse,  
Z = monkey  
X = horse,  
Y = donkey,  
Z = dog  
.....
```

- 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'].
```

```
?- consult('...cale.../nume.pl').
```

Exercițiul 1

Încercați să răspundeți la următoarele întrebări, verificând în interpretor.

1. Care dintre următoarele expresii sunt atomi?
f, loves(john, mary), Mary, _c1, 'Hello'
2. Care dintre următoarele expresii sunt variabile?
a, A, Paul, 'Hello', a_123, _, _abc

Exercițiul 2

Fișierul `ex2.pl` conține o bază de cunoștințe reprezentând un arbore genealogic.

- Definiți următoarele predicate, folosind `male/1`, `female/1` și `parent/2`:
 - `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)`
- Verificați predicate definite punând diverse întrebări.

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 3: negația

Folosind baza anterioară (arbore genealogic) testați predicatul

`not_parent`:

`?- not_parent(bob,juliet).`

`?- not_parent(X,juliet).`

`?- not_parent(X,Y).`

Ce observați? Încercați să analizați răspunsurile primite.

Exercițiul 3: negația

Folosind baza anterioară (arbore genealogic) testați predicatul

`not_parent`:

```
?- not_parent(bob,juliet).
```

```
?- not_parent(X,juliet).
```

```
?- 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.

Aritmetica în Prolog

Exemplu

```
?- 3+5 = +(3,5).
```

```
true
```

```
?- 3+5 = +(5,3).
```

```
false
```

```
?- 3+5 = 8.
```

```
false
```

Explicații:

- $3+5$ este un termen.
- Prolog trebuie anunțat explicit pentru a îl evalua ca o expresie aritmetică, folosind predicate predefinite în Prolog, cum sunt `is/2`, `:=/2`, `>/2` etc.

Exercițiu. Analizați următoarele exemple:

```
?- 3+5 is 8.
```

```
false
```

```
?= X is 3+5.
```

```
X = 8
```

```
?- 8 is 3+X.
```

```
is/2: Arguments are not sufficiently instantiated
```

```
?- X=4, 8 is 3+X.
```

```
false
```

Exercițiu. Analizați următoarele exemple:

?- X is 30-4.

X = 26

?- X is 3*5.

X = 15

?- X is 9/4.

X = 2.25

Operatorul `is`:

- Primește două argumente
- Al doilea argument trebuie să fie o expresie aritmetică validă, cu toate variabilele inițializate
- Primul argument este fie un număr, fie o variabilă
- Dacă primul argument este un număr, atunci rezultatul este `true` dacă este egal cu evaluarea expresiei aritmetice din al doilea argument.
- Dacă primul argument este o variabilă, răspunsul este pozitiv dacă variabila poate fi unificată cu evaluarea expresiei aritmetice din al doilea argument.

Totuși, nu este recomandat să folosiți `is` pentru a compara două expresii aritmetice, ci operatorul `==`.

Exercițiu. Analizați următoarele exemple:

`?- 8 > 3.`

`true`

`?- 8+2 > 9-2.`

`true`

`?- 8 < 3.`

`false`

`?- 8 >= 3.`

`true`

`?- 8 == 3.`

`false`

`?- 8 \= 3.`

`true`

Operatorii aritmetici predefiniți în Prolog sunt de două tipuri:

- funcții
- relații

- Adunarea și înmulțirea sunt exemple de funcții aritmetice.
- Aceste funcții sunt scrise în mod uzual și în Prolog.

Exemplu

$$2 + (-3.2 * X - \max(17, X)) / 2 ** 5$$

- $2**5$ înseamnă 2^5
- Exemple de alte funcții disponibile:
`min/2`, `abs/1` (modul), `sqrt/1` (radical), `sin/1` (sinus)
- Operatorul `//` este folosit pentru împărțire întreagă.
- Operatorul `mod` este folosit pentru restul împărțirii întregi.

Relații

- Relațiile aritmetice sunt folosite pentru a compara evaluarea expresiilor aritmetice (e.g, $X > Y$)
- Exemple de relații disponibile:
 $<$, $>$, $=<$, $>=$, $=\backslash=$ (diferit), $==$ (aritmetic egal)
- **Atenție** la diferența dintre $==$ și $=$:
 - $==$ compară două expresii aritmetice
 - $=$ caută un unificator

Exemplu

```
?- 2 ** 3 == 3 + 5.
```

```
true
```

```
?- 2 ** 3 = 3 + 5.
```

```
false
```

Exercițiul 4: distanța dintre două puncte

Definiți un predicat `distance/3` pentru a calcula distanța dintre două puncte într-un plan 2-dimensional. Punctele sunt date ca perechi de coordonate.

Exemple:

```
?- distance((0,0), (3,4), X).
```

```
X = 5.0
```

```
?- distance((-2.5,1), (3.5,-4), X).
```

```
X = 7.810249675906654
```

- <http://www.learnprolognow.org>
- <http://cs.union.edu/~striegnk/courses/esslli04prolog>
- U. Endriss, Lecture Notes. An Introduction to Prolog Programming, ILLC, Amsterdam, 2018.

Pe data viitoare!