

Laborator 1

Logică matematică și computațională

- Prolog este cel mai cunoscut limbaj de programare logică.
 - bazat pe logica clasică de ordinul I (cu predicate), despre care se vor învăța mai multe la curs și seminar
 - funcționează pe bază de unificare, rezoluție și *backtracking* (pe aceste lucruri le vom vedea doar în practică, pe parcursul laboratorului)
- Multe implementări îl transformă în limbaj de programare matur
 - I/O, operații implementate deja în limbaj etc.

- O implementare cunoscută este **SWI-Prolog**:
 - gratuită
 - folosită des cu scop pedagogic
 - conține multe biblioteci
 - <http://www.swi-prolog.org/>
- Vom folosi pentru început varianta online **SWISH** a SWI-Prolog:
 - <http://swish.swi-prolog.org/>

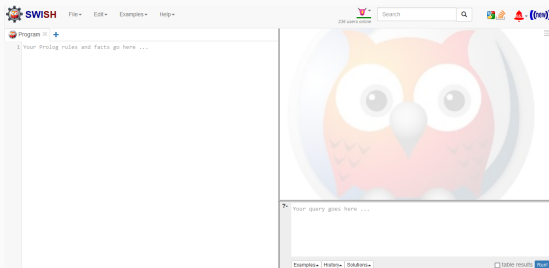
Întrebări:

- Cum arată un program în Prolog?
- Cum interogăm un program în Prolog (și ce înseamnă aceasta)?

Interfața SWISH

Interfața mediului SWISH conține trei componente:

- programul Prolog – în stânga
- interogarea – în dreapta-jos
- rezultatul interogărilor – în dreapta-sus



Când ne uităm la câmpul de interogare, vedem că apare semnul ?- în stânga sa, lucru care indică faptul că, atunci când vom avea de-a face în acest suport cu un șir de caractere care are ?- la început, restul textului va trebui scris în acel câmp.

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'`
- `'(@ *+'`
- `+`

Interogați:

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

```
true.
```

Aici, `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 să folosim variabila respectivă

Exemplu

- X
- Animal
- `_x`
- X_1_2

Un mic exercițiu sintactic

Care din următoarele șiruri de caractere sunt **constante** și care sunt **variabile** în Prolog?

- vINCENT
- Footmassage
- variable23
- Variable2000
- big_kahuna_burger
- 'big kahuna burger'
- big kahuna burger
- 'Jules'
- _Jules
- '_Jules'

Un mic exercițiu sintactic

Care din următoarele șiruri de caractere sunt **constante** și care sunt **variabile** în Prolog?

- vINCENT – constantă
- Footmassage – variabilă
- variable23 – constantă
- Variable2000 – variabilă
- big_kahuna_burger – constantă
- 'big kahuna burger' – constantă
- big kahuna burger – nici una, nici alta
- 'Jules' – constantă
- _Jules – variabilă
- '_Jules' – constantă

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** (numit **functor**): `is_bigger` în `is_bigger(horse, X)`
 - o **aritate** (numărul de argumente): 2 în `is_bigger(horse, X)`

Când vorbim despre un program, marcăm aritatea unui functor cu o bară după nume, cum ar fi `is_bigger/2` în exemplul de mai sus. Functorul poate fi un **predicat** sau un **simbol de funcție**, după cum vom vedea și înțelege în exemple concrete.

Ideea de programare logică

- Un **program logic** este o colecție de proprietăți (scrise sub formă de formule logice) presupuse despre lume (sau mai degrabă despre lumea programului).
- Programatorul furnizează și o proprietate (scrisă tot ca o formulă logică) care poate să fie sau nu adevărată în lumea respectivă (**interogare, query**).
- Sistemul determină dacă proprietatea aflată sub semnul întrebării este o consecință a proprietăților presupuse în program.

Exemplu de program logic

Acest exemplu este scris în limbajul logicii propoziționale:

```
oslo → windy
oslo → norway
norway → cold
cold ∧ windy → winterIsComing
oslo
```

Exemplu de program logic

Acest exemplu este scris în limbajul logicii propoziționale:

```
oslo → windy
oslo → norway
norway → cold
cold ∧ windy → winterIsComing
oslo
```

Exemplu de interogare

Este adevărat `winterIsComing`?

Exemplul de mai sus în SWI-Prolog

Program:

```
windy :- oslo.  
norway :- oslo.  
cold :- norway.  
winterIsComing :- windy, cold.  
oslo.
```

Interogare:

```
?- winterIsComing.  
true
```

Un exemplu mai complex

Iată un alt program Prolog:

```
father(peter,meg).  
father(peter,stewie).
```

```
mother(lois,meg).  
mother(lois,stewie).
```

```
griffin(peter).  
griffin(lois).
```

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

Un program în Prolog poate fi privit ca o **bază de cunoștințe** (Knowledge Base).

Program în Prolog = mulțime de reguli

Practic, gândim un program în Prolog ca pe o mulțime de **reguli** cu ajutorul cărora descriem *lumea* (*universul*) programului respectiv.

Exemplu

```
father(peter,meg).  
father(peter,stewie).
```

```
mother(lois,meg).  
mother(lois,stewie).
```

```
griffin(peter).  
griffin(lois).
```

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

Predicate:

father/2

mother/2

griffin/1

Program

Fapte + Reguli

- Un **program** în Prolog este format din **reguli** de forma

Head :- Body.

- **Head** este un predicat aplicat unui număr de termeni egal cu aritatea sa, iar **Body** este o secvență de predicate (din nou, fiecare dintre ele aplicat unui număr de termeni egal cu aritatea sa), separate prin virgulă.
- Regulile fără Body se numesc **fapte**.

- Un **program** în Prolog este format din **reguli** de forma

Head :- Body.

- **Head** este un predicat aplicat unui număr de termeni egal cu aritatea sa, iar **Body** este o secvență de predicate (din nou, fiecare dintre ele aplicat unui număr de termeni egal cu aritatea sa), separate prin virgulă.
- Regulile fără Body se numesc **fapte**.

Exemplu

- Exemplu de regulă: `griffin(X) :- father(Y,X), griffin(Y).`
- Exemplu de fapt: `father(peter,meg).`

- operatorul `:-` modelează implicația logică \leftarrow

Exemplu

`comedy(X) :- griffin(X)`

dacă `griffin(X)` este adevărat, atunci `comedy(X)` este adevărat.

- operatorul `:-` modelează implicația logică \leftarrow

Exemplu

```
comedy(X) :- griffin(X)
```

dacă `griffin(X)` este adevărat, atunci `comedy(X)` este adevărat.

- virgula `,` modelează conjuncția \wedge

Exemplu

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

dacă `father(Y,X)` și `griffin(Y)` sunt adevărate,
atunci `griffin(X)` este adevărat.

Interpretarea din punctul de vedere al logicii

- mai multe reguli cu același Head servesc la definirea aceluiași predicat, între definiții considerându-se a fi un sau logic.

Exemplu

```
comedy(X) :- family_guy(X).  
comedy(X) :- south_park(X).  
comedy(X) :- disenchantment(X).
```

dacă

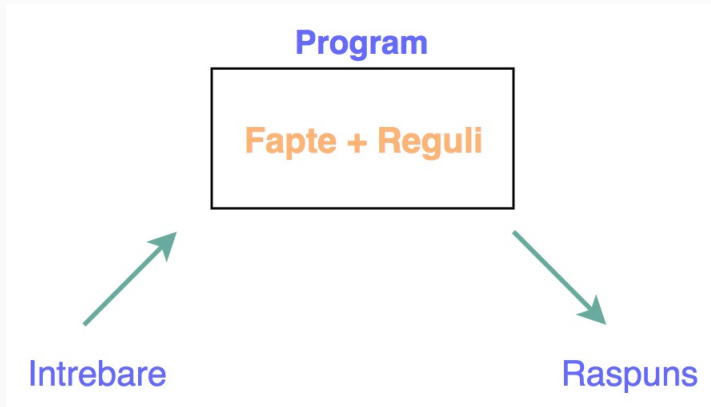
family_guy(X) este adevărat sau south_park(X) este adevărat sau
disenchantment(X) este adevărat,

atunci

comedy(X) este adevărat.



Cum folosim un program în Prolog?



Interogări în Prolog

- Prolog poate răspunde la interogări legate de consecințele relațiilor descrise într-un program în Prolog.

- **Interogările** sunt de forma:

`?- predicat1(...), ..., predicatn(...).`

- Prolog verifică dacă interogarea este o consecință a relațiilor definite în program.
- Dacă este cazul, Prolog caută valori pentru variabilele care apar în interogare astfel încât instanțierea interogării cu acele valori să fie o consecință a relațiilor din program.
- Un predicat care este analizat pentru a se răspunde la o interogare se numește **țintă** (*goal*).

Interogări în Prolog

Prolog poate da 2 tipuri de răspunsuri:

- **false** – în cazul în care interogarea nu este o consecință a programului.
- **true** sau **valori pentru variabilele din interogare** în cazul în care ea este o consecință a programului.

Interogări în Prolog

Prolog poate da 2 tipuri de răspunsuri:

- **false** – în cazul în care interogarea nu este o consecință a programului.
- **true** sau **valori pentru variabilele din interogare** în cazul în care ea este o consecință a programului.

Exemplu

```
?- griffin(meg)
```

```
true
```

```
?- griffin(glenn)
```

```
false
```

```
?- griffin(X)
```

```
X = peter ;
```

```
X = lois ;
```

```
X = meg ;
```

```
X = stewie ;
```

```
false
```

Cum găsește Prolog răspunsul

Pentru a găsi un răspuns, Prolog încearcă regulile în ordinea apariției lor.

Exemplu

Să presupunem că avem programul:

```
foo(a). foo(b). foo(c).
```

și că interogăm:

```
?- foo(X).
```

```
X = a.
```

Pentru a răspunde la interogare se caută o potrivire (unificator) între scopul `foo(X)` și baza de cunoștințe. Răspunsul este substituția care realizează potrivirea, în cazul nostru `X = a`.

Cum găsește Prolog răspunsul

Pentru a găsi un răspuns, Prolog încearcă regulile în ordinea apariției lor.

Exemplu

Să presupunem că avem programul:

```
foo(a). foo(b). foo(c).
```

și că punem următoarea întrebare:

```
?- foo(X).
```

```
X = a.
```

```
?- foo(d).
```

```
false
```

Dacă nu se poate face potrivirea, răspunsul este **false**.

Cum găsește Prolog răspunsul

Pentru a găsi un răspuns, Prolog încearcă regulile în ordinea apariției lor.

Exemplu

Să presupunem că avem programul:

```
foo(a). foo(b). foo(c).
```

și că punem următoarea întrebare:

```
?- foo(X).
```

```
X = a.
```

Cum găsește Prolog răspunsul

Pentru a găsi un răspuns, Prolog încearcă regulile în ordinea apariției lor.

Exemplu

Să presupunem că avem programul:

```
foo(a). foo(b). foo(c).
```

și că punem următoarea întrebare:

```
?- foo(X).
```

```
X = a.
```

Dacă dorim mai multe răspunsuri, apăsăm **Next**

```
?- foo(X).
```

```
X = a ;
```

```
X = b ;
```

```
X = c.
```

Cum găsește Prolog răspunsul

Pentru a găsi un răspuns, Prolog încearcă regulile în ordinea apariției lor.

Exemplu

Să presupunem că avem programul:

foo(a).

foo(b).

foo(c).

și că punem următoarea întrebare:

?- foo(X).

```
?- trace.  
true.  
  
[trace] ?- foo(X).  
  Call: (8) foo(_4556) ? creep  
  Exit: (8) foo(a) ? creep  
X = a ;  
  Redo: (8) foo(_4556) ? creep  
  Exit: (8) foo(b) ? creep  
X = b ;  
  Redo: (8) foo(_4556) ? creep  
  Exit: (8) foo(c) ? creep  
X = c.
```


Cum găsește Prolog răspunsul

Pentru a găsi un răspuns, **Prolog redenumbește variabilele**.

Exemplu

Să presupunem că avem programul:

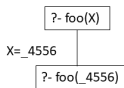
`foo(a).`

`foo(b).`

`foo(c).`

și că punem următoarea întrebare:

`?- foo(X).`



Cum găsește Prolog răspunsul

Pentru a găsi un răspuns, Prolog încearcă regulile în ordinea apariției lor.

Exemplu

Să presupunem că avem programul:

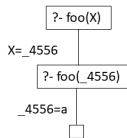
```
foo(a).
```

```
foo(b).
```

```
foo(c).
```

și că punem următoarea întrebare:

```
?- foo(X).
```



În acest moment, a fost găsită prima soluție: `X=_4556=a`.

Cum găsește Prolog răspunsul

Pentru a găsi un răspuns, Prolog încercă clauzele în ordinea apariției lor.

Exemplu

Să presupunem că avem programul:

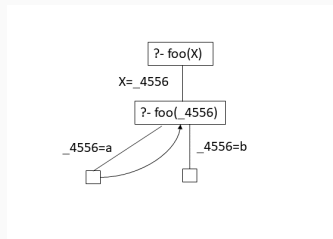
```
foo(a).
```

```
foo(b).
```

```
foo(c).
```

și că punem următoarea întrebare:

```
?- foo(X).
```



Dacă se dorește încă un răspuns, atunci se face un pas înapoi în **arborele de căutare** și se încercă satisfacerea țintei cu o nouă valoare.

Cum găsește Prolog răspunsul

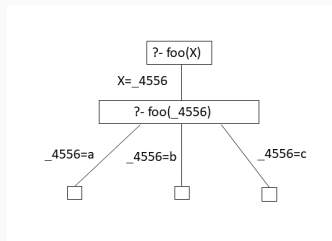
Pentru a găsi un răspuns, Prolog încearcă clauzele în ordinea apariției lor.

Exemplu

Să presupunem că avem programul:

```
foo(a).  
foo(b).  
foo(c).
```

și că punem următoarea întrebare:
`?- foo(X).`



arborele de căutare

Cum găsește Prolog răspunsul

Exemplu

Să presupunem că avem programul:

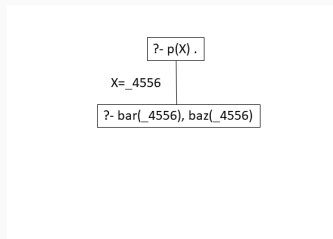
`bar(b).`

`bar(c).`

`baz(c).`

și că punem următoarea întrebare:

`?- bar(X), baz(X).`



Cum găsește Prolog răspunsul

Prolog se întoarce la ultima alegere dacă o sub-țintă eșuează.

Exemplu

Să presupunem că avem programul:

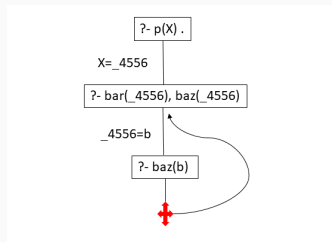
`bar(b).`

`bar(c).`

`baz(c).`

și că punem următoarea întrebare:

`?- bar(X), baz(X).`



Cum găsește Prolog răspunsul

Prolog se întoarce la ultima alegere dacă o sub-țintă eșuează.

Exemplu

Să presupunem că avem programul:

`bar(b).`

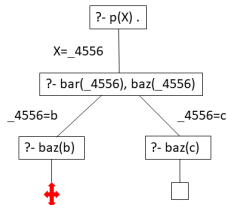
`bar(c).`

`baz(c).`

și că punem următoarea întrebare:

`?- bar(X), baz(X).`

Soluția găsită este: `X=_4556=c.`



Cum găsește Prolog răspunsul

Ce se întâmplă dacă schimbăm ordinea regulilor?

Exemplu

Să presupunem că avem programul:

```
bar(c).
```

```
bar(b).
```

```
baz(c).
```

și că punem următoarea întrebare:

```
?- bar(X),baz(X).
```


Cum găsește Prolog răspunsul

Ce se întâmplă dacă schimbăm ordinea regulilor?

Exemplu

Să presupunem că avem programul:

```
bar(c).
```

```
bar(b).
```

```
baz(c).
```

și că punem următoarea întrebare:

```
?- bar(X),baz(X).
```

```
X = c ;
```

```
false
```

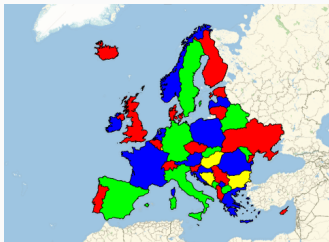
Vă explicați ce s-a întâmplat? Desenați arborele de căutare!

Un program mai complicat

Problema colorării hărților

Să se coloreze o hartă dată cu un număr minim de culori astfel încât oricare două țări vecine să fie colorate diferit.

Exemplu



Sursa imaginii

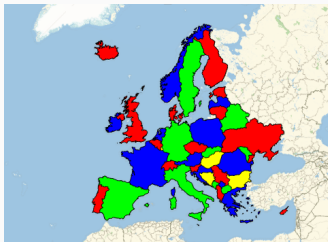
Un program mai complicat

Problema colorării hărților

Să se coloreze o hartă dată cu un număr minim de culori astfel încât oricare două țări vecine să fie colorate diferit.

Cum modelăm această problemă în Prolog?

Exemplu



Sursa imaginii

Un program mai complicat

Problema colorării hărților

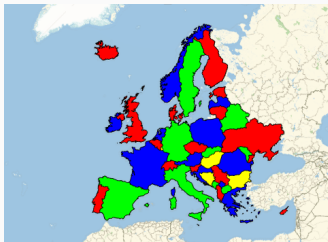
Să se coloreze o hartă dată cu un număr minim de culori astfel încât oricare două țări vecine să fie colorate diferit.

Cum modelăm această problemă în Prolog?

Exemplu

Trebuie să definim:

- culorile
- harta
- constrângerile



Sursa imaginii

Problema colorării hărților

Definim culorile

Exemplu

```
culoare(albastru).
```

```
culoare(rosu).
```

```
culoare(verde).
```

```
culoare(galben).
```

Problema colorării hărților

Definim culorile, harta

Exemplu

```
culoare(albastru).
```

```
culoare(rosu).
```

```
culoare(verde).
```

```
culoare(galben).
```

```
harta(RO,SE,MD,UA,BG,HU) :- vecin(RO,SE), vecin(RO,UA),  
                               vecin(RO,MD), vecin(RO,BG),  
                               vecin(RO,HU), vecin(UA,MD),  
                               vecin(BG,SE), vecin(SE,HU).
```

Problema colorării hărților

Definim culorile, harta și constrângerile.

Exemplu

```
culoare(albastru).
```

```
culoare(rosu).
```

```
culoare(verde).
```

```
culoare(galben).
```

```
harta(RO,SE,MD,UA,BG,HU) :- vecin(RO,SE), vecin(RO,UA),  
                               vecin(RO,MD), vecin(RO,BG),  
                               vecin(RO,HU), vecin(UA,MD),  
                               vecin(BG,SE), vecin(SE,HU).
```

```
vecin(X,Y) :- culoare(X),  
              culoare(Y),  
              X \== Y.
```

Problema colorării hărților

Definim culorile, harta și constrângerile. Cum punem întrebarea?

Exemplu

```
culoare(albastru).
```

```
culoare(rosu).
```

```
culoare(verde).
```

```
culoare(galben).
```

```
harta(RO,SE,MD,UA,BG,HU) :- vecin(RO,SE), vecin(RO,UA),  
                               vecin(RO,MD), vecin(RO,BG),  
                               vecin(RO,HU), vecin(UA,MD),  
                               vecin(BG,SE), vecin(SE,HU).
```

```
vecin(X,Y) :- culoare(X),  
              culoare(Y),  
              X \== Y.
```


Problema colorării hărților

Definim culorile, harta și constrângerile. Cum punem întrebarea?

Exemplu

```
culoare(albastru).
```

```
culoare(rosu).
```

```
culoare(verde).
```

```
culoare(galben).
```

```
harta(RO,SE,MD,UA,BG,HU) :- vecin(RO,SE), vecin(RO,UA),  
                               vecin(RO,MD), vecin(RO,BG),  
                               vecin(RO,HU), vecin(UA,MD),  
                               vecin(BG,SE), vecin(SE,HU).
```

```
vecin(X,Y) :- culoare(X),  
              culoare(Y),  
              X \== Y.
```

```
?- harta(RO,SE,MD,UA,BG,HU).
```

Problema colorării hărților

Ce răspuns primim?

Exemplu

```
culoare(albastru).
```

```
culoare(rosu).
```

```
culoare(verde).
```

```
culoare(galben).
```

```
harta(RO,SE,MD,UA,BG,HU) :- vecin(RO,SE), vecin(RO,UA),  
                               vecin(RO,MD), vecin(RO,BG),  
                               vecin(RO,HU), vecin(UA,MD),  
                               vecin(BG,SE), vecin(SE,HU).
```

```
vecin(X,Y) :- culoare(X),  
              culoare(Y),  
              X \== Y.
```

```
?- harta(RO,SE,MD,UA,BG,HU).
```

Problema colorării hărților

Exemplu

```
culoare(albastru).
```

```
culoare(rosu).
```

```
culoare(verde).
```

```
culoare(galben).
```

```
harta(RO,SE,MD,UA,BG,HU) :- vecin(RO,SE), vecin(RO,UA),  
                               vecin(RO,MD), vecin(RO,BG),  
                               vecin(RO,HU), vecin(UA,MD),  
                               vecin(BG,SE), vecin(SE,HU).
```

```
vecin(X,Y) :- culoare(X),  
              culoare(Y),  
              X \== Y.
```

```
?- harta(RO,SE,MD,UA,BG,HU).
```

```
RO = albastru,
```

```
SE = UA, UA = rosu,
```

```
MD = BG, BG = HU, HU = verde ■
```

Compararea termenilor: =, \=, ==, \==

- $T = U$ reușește dacă există o potrivire (termenii se unifică)
- $T \backslash = U$ reușește dacă nu există o potrivire
- $T == U$ reușește dacă termenii sunt identici
- $T \backslash == U$ reușește dacă termenii sunt diferiți

Compararea termenilor: $=, \backslash=, ==, \backslash==$

$T = U$	reușește dacă există o potrivire (termenii se unifică)
$T \backslash= U$	reușește dacă nu există o potrivire
$T == U$	reușește dacă termenii sunt identici
$T \backslash== U$	reușește dacă termenii sunt diferiți

Exemplu

?- $X = Y$.

$X = Y$.

?- $X == Y$.

false

?- $p(X, q(Z)) = p(Y, X)$.

$X = Y, Y = q(Z)$.

?- $p(X, Y) == p(X, Y)$.

true

?- $2 = 1 + 1$

false

?- $2 == 1 + 1$

false

- În exemplul de mai sus, $1+1$ este privită ca o expresie, nu este evaluată. Există și predicate care forțază evaluarea (e.g., $:=$).

Exemplu de program și interogări cu recursie

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
```

Acest program conține două predicate:

bigger/2, is_bigger/2.

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  
.....
```

Negarea unui predicat: `\+ pred(X)`

Exemplu

```
animal(dog). animal(elephant). animal(sheep).
```

```
?- animal(cat).
```

```
false
```

```
?- \+ animal(cat).
```

```
true
```


Negarea unui predicat: `\+ pred(X)`

Exemplu

```
animal(dog). animal(elephant). animal(sheep).
```

```
?- animal(cat).
```

```
false
```

```
?- \+ animal(cat).
```

```
true
```

- Clauzele din Prolog dau doar condiții suficiente (dar nu și necesare) pentru ca un predicat să fie adevărat.
- Pentru a da un răspuns pozitiv la o țintă, Prolog trebuie să construiască o „demonstrație” pentru a arată că mulțimea de fapte și reguli din program implică acea țintă.
- Astfel, un răspuns **false** nu înseamnă neapărat că ținta nu este adevărată, ci doar că **Prolog nu a reușit să găsească o demonstrație.**

- Operatorul predefinit \+ (scris și not) se folosește pentru a nega un predicat.
- O țintă \+ Goal reușește dacă Prolog nu găsește o demonstrație pentru Goal. Negația din Prolog este definită ca incapacitatea de a găsi o demonstrație.
- Semantica operatorului \+ se numește **negation as failure**.

Negația ca eșec ('negation as failure')

Exemplu

Să presupunem că avem o listă de fapte cu perechi de oameni căsătoriți între ei:

```
married(peter, lucy).  
married(paul, mary).  
married(bob, juliet).  
married(harry, geraldine).
```

Negația ca eșec

Exemplu (cont.)

Putem să definim un predicat `single/1` care reușește dacă argumentul său nu este nici primul nici al doilea argument în faptele pentru `married`.

```
single(Person) :-  
    \+ married(Person, _),  
    \+ married(_, Person).  
  
?- single(mary).    ?- single(anne).    ?- single(X).  
false              true                false
```

Răspunsul la întrebarea `?- single(anne)`. trebuie gândit astfel:

*Presupunem că Anne este single,
deoarece **nu am putut demonstra** că este căsătorită.*

- Comentarii:

```
% comentează restul liniei
```

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

- Nu uitați să puneți punct la sfârșitul fiecărei reguli.

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.

Exercițiul 3: negația

Definim predicatul `not_parent` folosind operatorul de negație introdus mai devreme:

```
not_parent(X,Y) :- not(parent(X,Y)).
```

sau

```
not_parent(X,Y) :- \+ parent(X,Y).
```

Testați-l, folosind baza anterioară de cunoștințe (arbore genealogic):

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

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

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

Ce observați? Încercați să analizați răspunsurile primite.

Exercițiul 3: negația

Definim predicatul `not_parent` folosind operatorul de negație introdus mai devreme:

```
not_parent(X,Y) :- not(parent(X,Y)).
```

sau

```
not_parent(X,Y) :- \+ parent(X,Y).
```

Testați-l, folosind baza anterioară de cunoștințe (arbore genealogic):

```
?- 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.

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