Curs 2

2021-2022

Fundamentele Limbajelor de Programare

Cuprins

Programare logică & Prolog

Program în Prolog = mulțime de predicate

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

Example

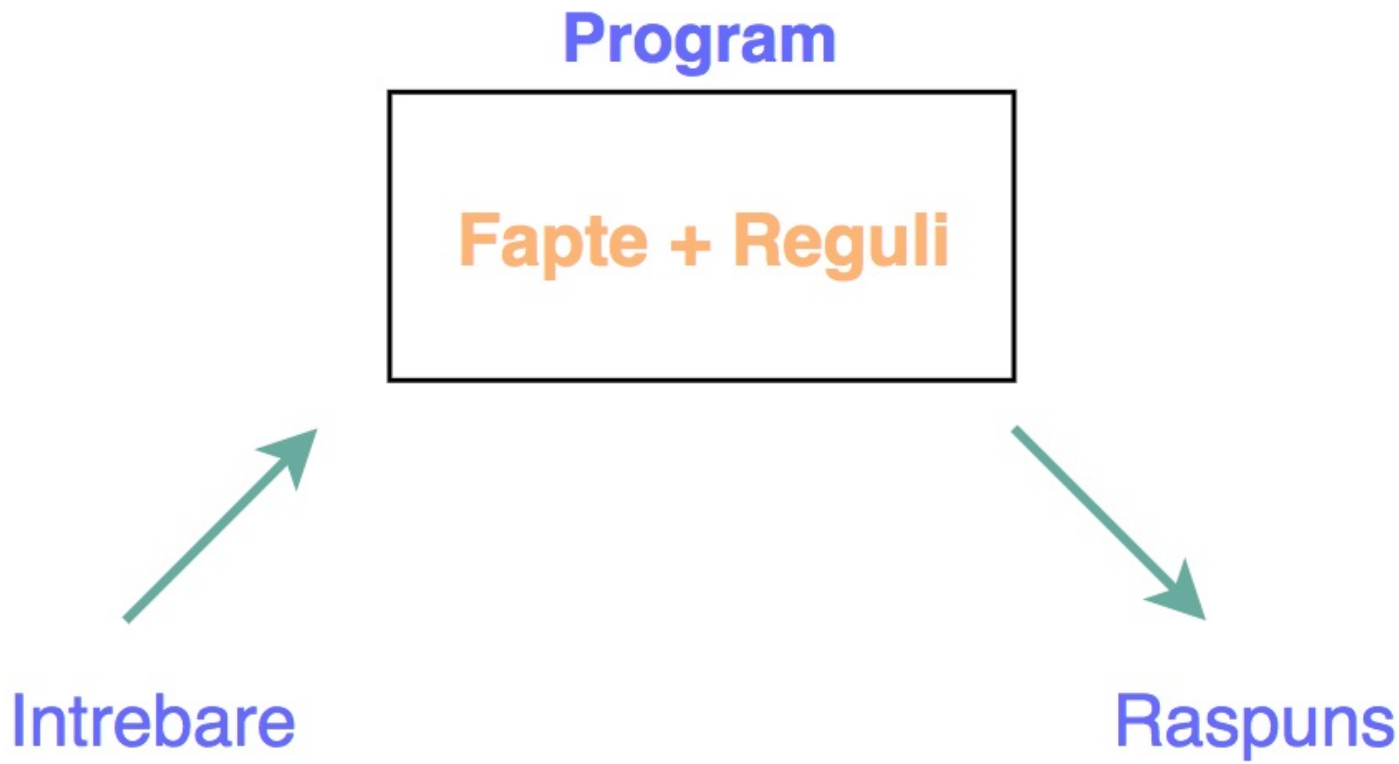
```
father(peter,meg).
father(peter,stewie).

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

griffin(peter).
griffin(lois).

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

Predicate:
father/2
mother/2
griffin/1

Un program în Prolog

Cum folosim un program în Prolog?

Întrebări în Prolog



Întrebări în Prolog

Prolog poate da 2 tipuri de răspunsuri:

- `false` – în cazul în care întrebarea nu este o consecință a programului.
- `true` sau valori pentru variabilele din întrebare în cazul în care întrebarea este o consecință a programului.

Example

```
?- griffin(meg)
true
?- griffin(glenn)
false
```

```
?- griffin(X)
X = petr ;
X = lois ;
X = meg ;
X = stewie ;
false
```

Cum găsește Prolog răspunsul

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

Example

Să presupunem că avem programul:

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

și că punem următoarea întrebare:

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

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

Răspunsul la întrebare este găsit prin unificare!

Cum găsește Prolog răspunsul

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

Example

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 raspuns, Prolog încearcă regulile în ordinea apariției lor.

Example

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 raspuns, Prolog încearcă regulile în ordinea apariției lor.

Example

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, tastăm `;`

```
?- foo(X).
X = a ;
X = b ;
X = c.
```

Cum găsește Prolog răspunsul

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

Example

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 raspuns, Prolog redenumește variabilele.

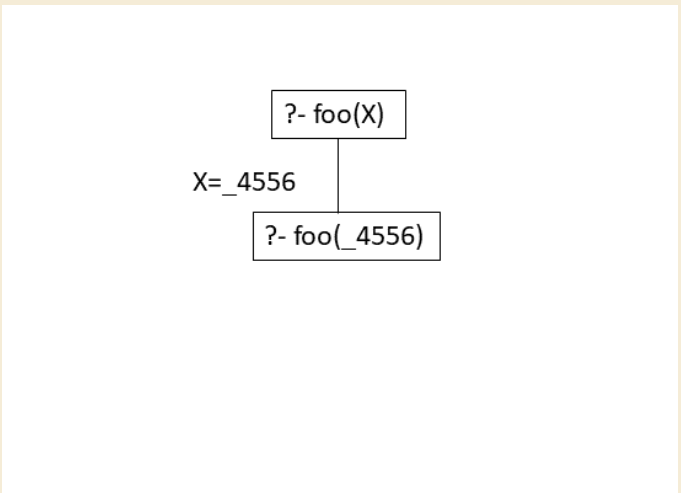
Example

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 raspuns, Prolog încearcă regulile în ordinea apariției lor.

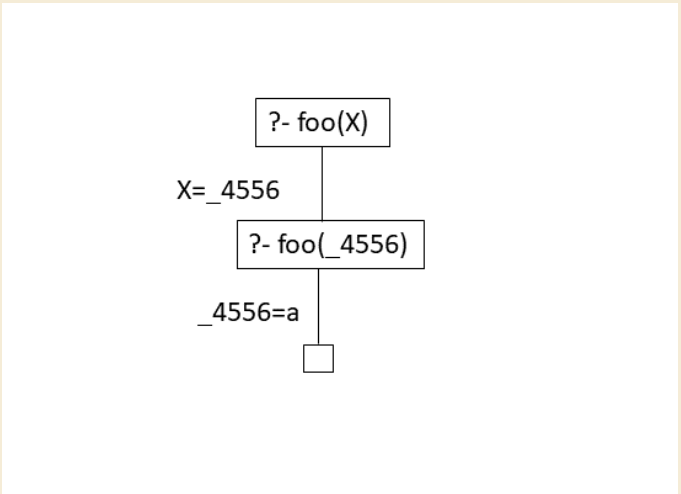
Example

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 raspuns, Prolog încearcă clauzele în ordinea apariției lor.

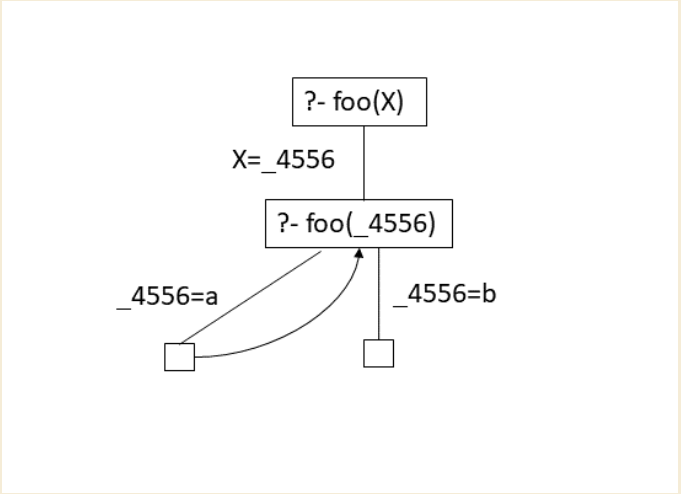
Example

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 încearcă satisfacerea țintei cu o nouă valoare.

Cum găsește Prolog răspunsul

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

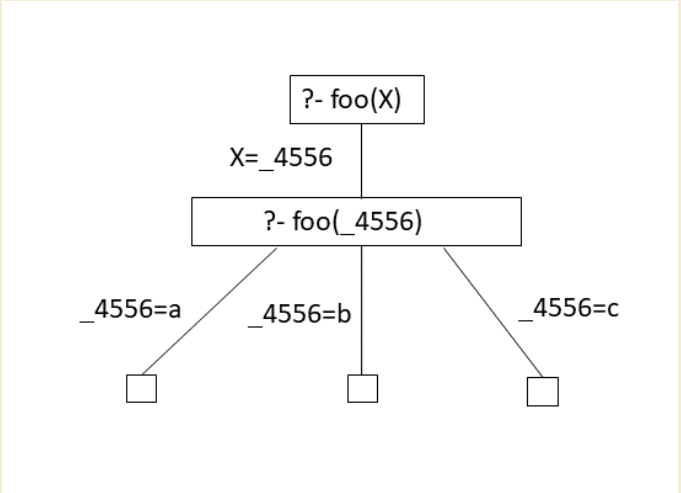
Example

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

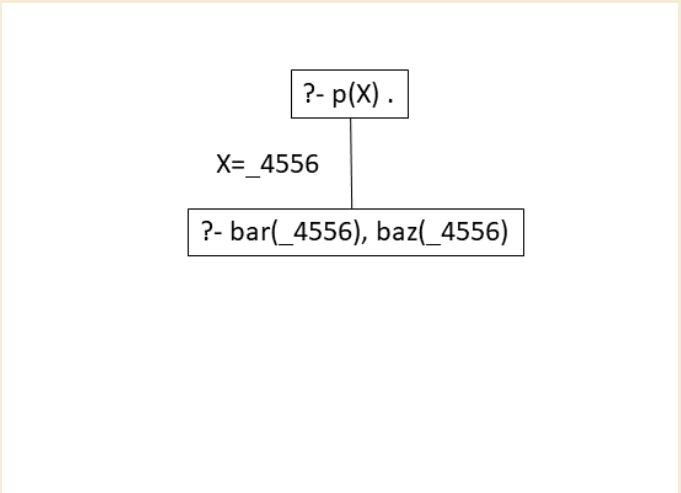
Example

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.

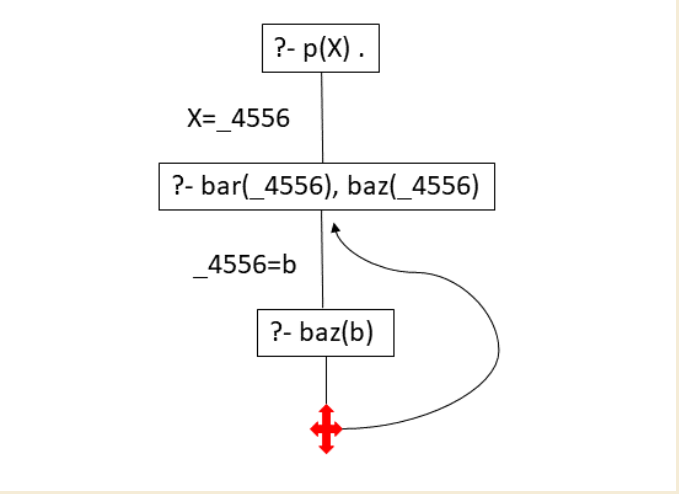
Example

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.

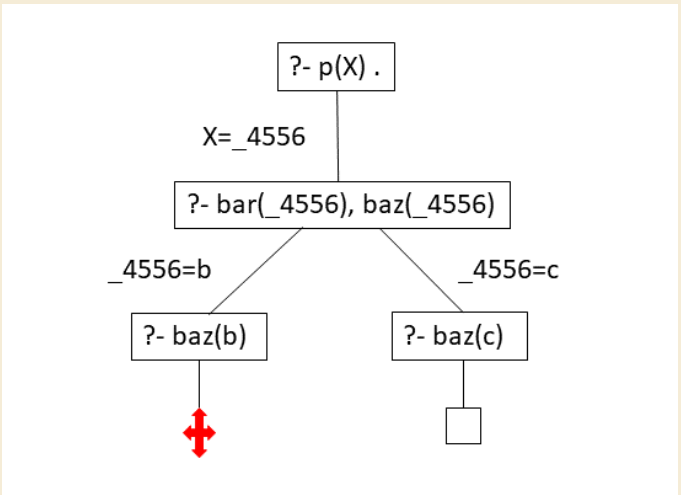
Example

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?

Example

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?

Example

Să presupunem că avem programul:

```
bar(c).
bar(b).

baz(c).
```

și că punem următoarea întrebare:

```
?- bar(X),baz(X).
X = c ;
false
```

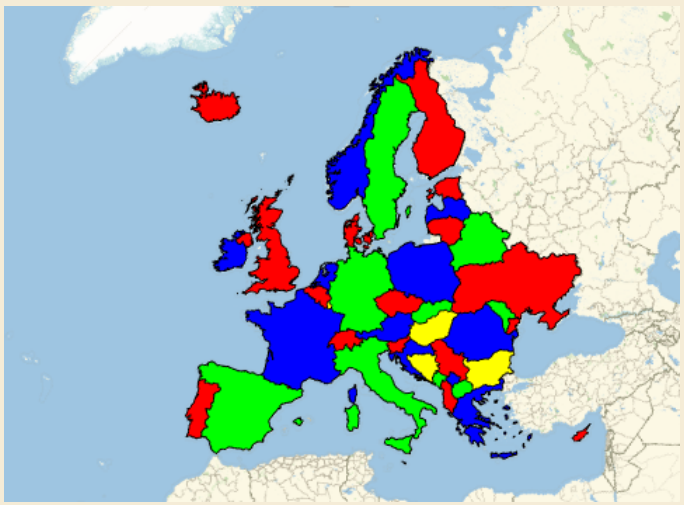
Vă explicați ce s-a întâmplat? Desenați arborele de căutare!

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.

Example



Sursa imaginii

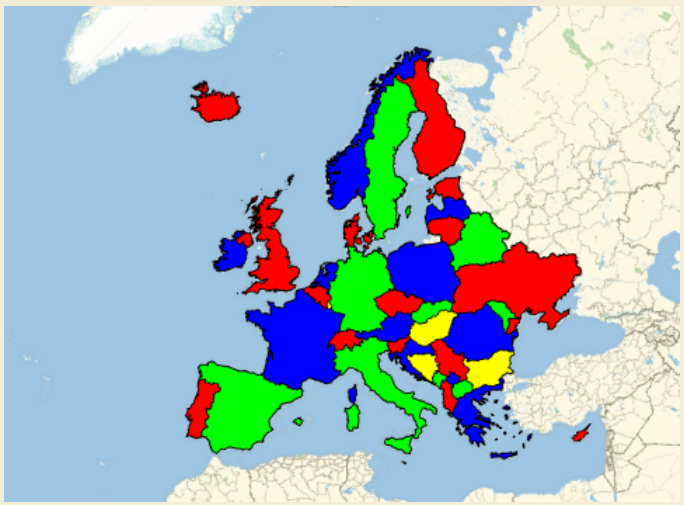
Un program mai complicat

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?

Example



Sursa imaginii

Un program mai complicat

Problema colorării hărților

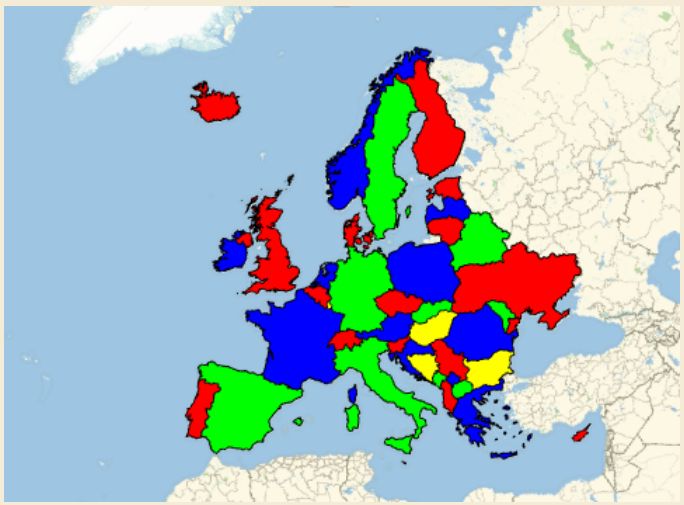
Să se coloreze o hartă dată cu un număr minim de culori astfel încât oricare două țări vecine să fie colorate diferit.

Cum modelăm această problemă în Prolog?

Example

Trebuie să definim:

- culorile
- harta
- constrângerile



Sursa imaginii

Problema colorării hărților

Definim culorile

Example

```
culoare(albastru).
culoare(rosu).
culoare(verde).
culoare(galben).
```

Problema colorării hărților

Definim culorile, harta

Example

```
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.

Example

```
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?

Example

```
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?

Example

```
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?

Example

```
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

Example

```
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 \= U`	reușește dacă nu există o potrivire
`T == U`	reușește dacă termenii sunt identici
`T \== U`	reușește dacă termenii sunt diferiți

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

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

Example

```
?- X = Y.
X = Y.

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

?- 2 = 1 + 1
false
```

```
?- X == Y .
false

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

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

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

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

Example

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

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

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

Example

```
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 \+

- Negarea unei ținte se poate defini astfel:

```
neg(Goal) :- Goal, !, fail.
neg(Goal)
```

unde `fail/0` este un predicat care eșuează întotdeauna.

Operatorul \+

- Negarea unei ținte se poate defini astfel:

```
neg(Goal) :- Goal, !, fail.
neg(Goal)
```

  unde `fail/0` este un predicat care eșuează întotdeauna.
- În PROLOG acest predicat este predefinit sub numele \+.
- Operatorul \+ se foloseste pentru a nega un predicat.
- !(cut) este un predicat predefinit (de aritate 0) care restricționează mecanismul de backtracking: execuția subțintei ! se termină cu succes, deci alegerile (instanțierile) făcute înainte de a se ajunge la ! nu mai pot fi schimbate.
- 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")

Example

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

Example (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).
false
```

```
?- single(anne).
true
```

```
?- single(X).
false
```

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

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

Predicatul -> /2 (if-then-else)

- if-then

```
If -> Then :- If, !, Then.
```

Predicatul -> /2 (if-then-else)

- if-then

```
If -> Then :- If, !, Then.
```

- if-then-else

```
If -> Then; _Else :- If, !, Then.
If -> Then; Else :- !, Else.
```

Se încearcă demonstrarea predicatului If. Dacă întoarce true atunci se încearcă demonstrarea predicatului Then, iar dacă întoarce false se încearcă demonstrarea predicatului Else.

```
max(X,Y,Z) :- (X =< Y) -> Z = Y ;  Z = X

?- max(2,3,Z).
Z = 3.
```

Predicatul -> /2 (if-then-else)

- if-then

```
If -> Then :- If, !, Then.
```

- if-then-else

```
If -> Then; _Else :- If, !, Then.
If -> Then; Else :- !, Else.
```

  Se încearcă demonstrarea predicatului If. Dacă întoarce true atunci se încearcă demonstrarea predicatului Then, iar dacă întoarce false se încearcă demonstrarea predicatului Else.

```
max(X,Y,Z) :- (X =< Y) -> Z = Y ;  Z = X

?- max(2,3,Z).
Z = 3.
```

Observăm că `If -> Then` este echivalent cu `If -> Then ; fail`.

Liste și recursie

Listă `[t1,...,tn]`

- O listă în Prolog este un șir de elemente, separate prin virgulă, între paranteze drepte:

```
[1,cold, parent(jon),[winter,is,coming],X]
```

- O listă poate conține termeni de orice fel.
- Ordinea termenilor din listă are importanță:

```
?- [1,2] == [2,1] .
false
```

- Lista vidă se notează `[]`.
- Simbolul `|` desemnează coada listei:

```
?- [1,2,3,4,5,6] = [X|T].
X = 1, T = [2, 3, 4, 5, 6].

?- [1,2,3|[4,5,6]] == [1,2,3,4,5,6].
true.
```

Listă [t1,...,tn]==[t1 |[t2,...,tn]

- Simbolul | desemnează coada listei:

```
?- [1,2,3,4,5,6] = [X|T].
X = 1,
T = [2, 3, 4, 5, 6].
```

- Variabila anonimă _ este unificată cu orice termen Prolog:

```
?- [1,2,3,4,5,6] = [X|_].
X = 1.
```

- Deoarece Prologul face unificare poate identifica șabloane mai complicate:

```
?- [5,1,1,3,2]=[_|[X|[X|_]]].
X = 1.
?- [5,1,4,3,2]=[_|[X|[X|_]]].
false.
```

Liste

Exercițiu

- Definiți un predicat care verifică că un termen este lista.

Liste

Exercițiu

- Definiți un predicat care verifică că un termen este lista.

```
is_list([]).
is_list([_|_]).
```

Liste

Exercițiu

- Definiți un predicat care verifică că un termen este lista.

```
is_list([]).
is_list([_|_]).
```

- Definiți predicate care verifică dacă un termen este primul element, ultimul element sau coada unei liste.

Liste

Exercițiu

- Definiți un predicat care verifică că un termen este lista.

```
is_list([]).
is_list([_|_]).
```

- Definiți predicate care verifică dacă un termen este primul element, ultimul element sau coada unei liste.

```
head([X|_],X).

last([X],X).
last([_|T],Y):- last(T,Y).

tail([],[]).
tail([_|T],T).
```

Liste

Exercițiu

- Definiți un predicat care verifică dacă un termen aparține unei liste.

Liste

Exercițiu

- Definiți un predicat care verifică dacă un termen aparține unei liste.

```
member(H, [H|_]).
member(H, [_|T]) :- member(H,T).
```

Liste

Exercițiu

- Definiți un predicat care verifică dacă un termen aparține unei liste.

```
member(H, [H|_]).
member(H, [_|T]) :- member(H,T).
```

- Definiți un predicat `append/3` care verifică dacă o listă se obține prin concatenarea altor două liste.

Liste

Exercițiu

- Definiți un predicat care verifică dacă un termen aparține unei liste.

```
member(H, [H|_]).
member(H, [_|T]) :- member(H,T).
```

- Definiți un predicat `append/3` care verifică dacă o listă se obține prin concatenarea altor două liste.

```
append([],L,L).
append([X|T],L, [X|R]) :- append(T,L,R).
```

Liste

Exercițiu

- Definiți un predicat care verifică dacă un termen aparține unei liste.

```
member(H, [H|_]).
member(H, [_|T]) :- member(H,T).
```

- Definiți un predicat `append/3` care verifică dacă o listă se obține prin concatenarea altor două liste.

```
append([],L,L).
append([X|T],L, [X|R]) :- append(T,L,R).
```

Există predicatele predefinite `member/2` și `append/3`.

Liste append/3

- Funcția append/3:

```
?- listing(append/3).
append([],L,L).
append([X|T],L, [X|R]) :- append(T,L,R).

?- append(X,Y,[a,b,c]).
X = [],
Y = [a, b, c] ;
X = [a],
Y = [b, c] ;
X = [a, b],
Y = [c] ;
X = [a, b, c],
Y = [] ;
false
```

- Funcția astfel definită poate fi folosită atât pentru verificare, cât și pentru generare.

Liste

Exercițiu

- Definiți un predicat `elim/3` care verifică dacă o listă se obține din alta prin eliminarea unui element.

Liste

Exercițiu

- Definiți un predicat `elim/3` care verifică dacă o listă se obține din alta prin eliminarea unui element.

```
elim(X, [X|T], T).
elim(X, [H|T], [H|L]) :- elim(X,T,L).
```

Liste

Exercițiu

- Definiți un predicat `elim/3` care verifică dacă o listă se obține din alta prin eliminarea unui element.

```
elim(X, [X|T], T).
elim(X, [H|T], [H|L]) :- elim(X,T,L).
```

- Definiți un predicat care `perm/2` care verifică dacă două liste sunt permutări.

Liste

Exercițiu

- Definiți un predicat `elim/3` care verifică dacă o listă se obține din alta prin eliminarea unui element.

```
elim(X, [X|T], T).
elim(X, [H|T], [H|L]) :- elim(X,T,L).
```

- Definiți un predicat care `perm/2` care verifică dacă două liste sunt permutări.

```
perm([],[]).
perm([X|T],L) :- elim(X,L,R), perm(R,T).
```

Liste

Exercițiu

- Definiți un predicat `elim/3` care verifică dacă o listă se obține din alta prin eliminarea unui element.

```
elim(X, [X|T], T).
elim(X, [H|T], [H|L]) :- elim(X,T,L).
```

- Definiți un predicat care `perm/2` care verifică dacă două liste sunt permutări.

```
perm([],[]).
perm([X|T],L) :- elim(X,L,R), perm(R,T).
```

Predicatele predefinite `select/3` și `permutation/2` au aceeași funcționalitate.

Quiz time!

https://www.questionpro.com/t/AT4NiZrPCD

Pe săptămâna viitoare!